

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторная работа №1

GUI-ИНТЕРФЕЙС ДЛЯ ПАКЕТА NEURAL NETWORKS TOOLBOX ПРОГРАММНОЙ СРЕДЫ MATLAB . НАЗНАЧЕНИЕ И ОБЗОР

Цель работы: изучение основных свойств и основ работы с GUI – интерфейсом пакета Neural Networks Toolbox в программной среде MatLab .

Окно GUI-интерфейса пакета нейронных сетей

GUI-интерфейс – это специальное инструментальное средство организации диалога с пользователем. Это, например, пакет по нейронным сетям, в состав которого входит инструментальное средство *NNTool*. Этот графический интерфейс позволяет, не обращаясь к командному окну системы MatLab, выполнять создание, обучение, моделирование, а также импорт и экспорт нейронных сетей и данных, используя только инструментальные возможности GUI-интерфейса. Однако такие инструменты наиболее эффективны лишь на начальной стадии работы с пакетом, поскольку имеют определенные ограничения. В частности, интерфейс *NNTool* допускает работу только с простейшими однослойными и двухслойными нейронными сетями, но при этом пользователь выигрывает во времени и эффективности решения прикладных задач [39].

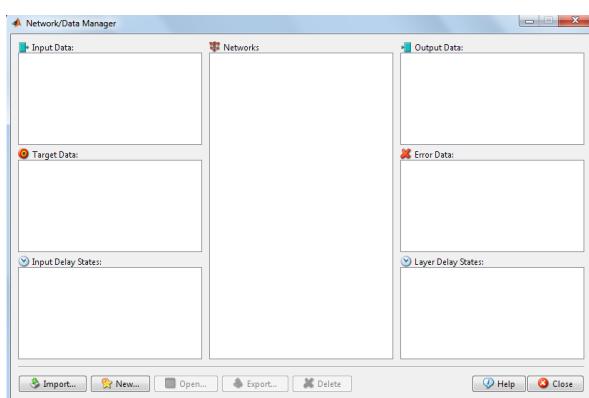


Рисунок 1.1. Окно управления
сетью/данными

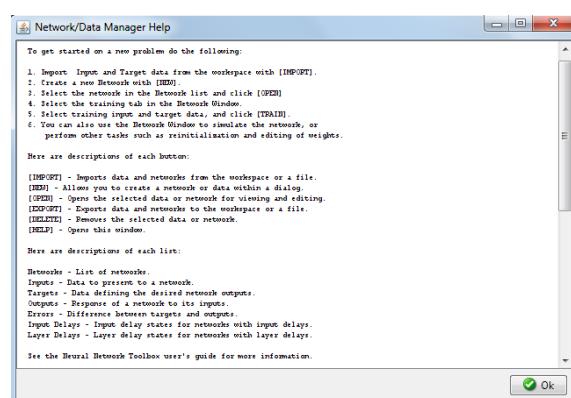


Рисунок 1.2. Окно подсказки

Вызов GUI-интерфейса *NNTool* осуществляется командой **nntool** из командной строки. После вызова появляется окно *Network/Data Manager* (Управление сетью/данными) – рисунок 1.1. Здесь имеются следующие области и кнопки:

Input Data – последовательность входов;

Target data – последовательность целей;

Input Daley States – начальные условия линии задержки входов;

Networks – список нейронных сетей;

Output Data – последовательность выходов;

Error Data – последовательности ошибок сети;

Layer Delay States – начальные условия линии задержки слоя;

Help – кнопка вызова окна подсказки (рис. 1.2);

New ... – кнопка вызова окна формирования данных и создания новой нейронной сети (рис. 1.3, 1.4);

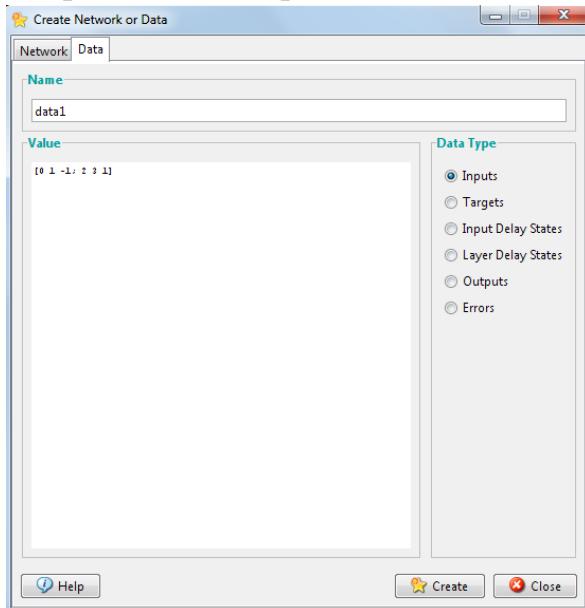


Рисунок 1.3. Окно формирования
данных

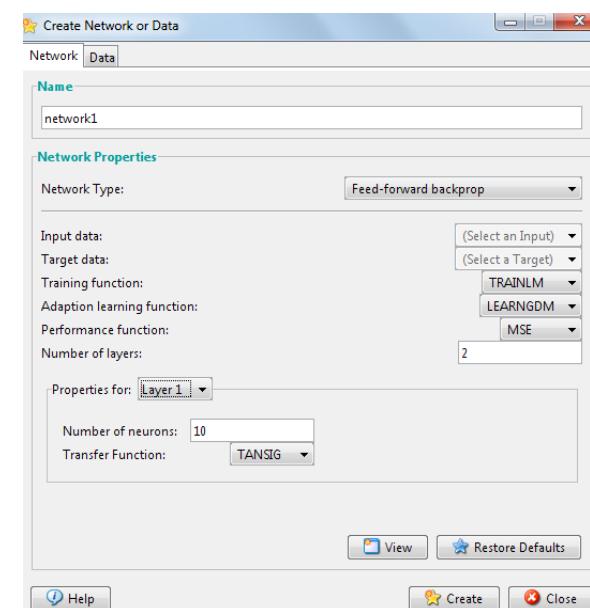


Рисунок 1.4. Окно создания новой
нейронной сети

Import... – кнопка вызова окна импорта или загрузки данных;

Export... – кнопка вызова окна экспорта или загрузки данных в файл.

Кнопки *View*, *Delete* становятся активными только после создания и активизации данных, относящихся к последовательностям входа, цели, выхода или ошибок сети. Кнопка *View* позволяет просмотреть, а кнопка *Delete* удалить активизированные данные.

Кнопки *View*, *Delete*, *Initialize...*, *Simulate...*, *Train...*, *Adapt...* становятся активными после создания и активации самой нейронной сети. Они позволяют просмотреть, удалить, инициализировать, промоделировать, обучить или адаптировать нейронную сеть.

Работа с инструментальными средствами GUI

Окно подсказки (*Network/Data Manager Help*) показано на рисунке 1.2 и описывает правила работы диспетчером *Network/Data Manager* при создании нейронной сети.

При создании нейронной сети, необходимо выполнить следующие операции:

- 1) сформировать последовательность входов и целей (кнопка *New Data*) либо загрузить их из рабочей области системы MatLab или из файла (кнопка *Import*);
- 2) создать новую нейронную сеть (кнопка *New Network*) либо загрузить ее из рабочей области системы MatLab или из файла (кнопка *Import*);
- 3) выбрать тип нейронной сети и нажать кнопку *Train...*, чтобы открыть окно для задания параметров процедуры обучения;
- 4) открыть окно *Network* для просмотра, обучения, моделирования и адаптации сети.

Окно формирования данных (*Create New Data*) показанное на рисунке 1.3, содержит две области редактирования текста для записи имени вводимых данных (область *Name*) и ввода самих данных (область *Value*), а также 6 кнопок для указания типа вводимых данных:

Inputs (Входы) – последовательность значений входов;

Targets (Цели) – последовательность значений целей;

Input Delay States (Состояния линии задержки (ЛЗ) входа) – начальные условия линии задержки на входе;

Layer Delay States (Состояния ЛЗ слоя) – начальные условия линии задержки в слое;

Outputs – последовательность значений выходов сети;

Errors – разность значений целей и выходов.

Окно создания новой нейронной сети (*Create New Network*) показано на рисунке 1.4 и включает поля для задания параметров создаваемой сети. В зависимости от типа сети количество полей и их названия изменяются.

Приведем описания полей.

Network Name (Имя сети) – стандартное имя сети, присваиваемое GUI-интерфейсом *NNTool*; в процессе создания новых сетей порядковый номер будет изменяться автоматически.

Network Type (Тип сети) – список сетей, доступных для работы с интерфейсом *NNTool*. Для удобства этот список повторен в табл. 1.1. Интерфейс *NNTool* позволяет создавать нейронные сети только с одной или двумя слоями.

Input ranges (Диапазон входа) – допустимые границы входов, которые либо назначаются пользователем, либо определяются автоматически по имени входной последовательности, выбираемой из списка Get from Inp...

Training function (Функция обучения) – список обучающих функций.

Performance function (Функция качества обучения) – список функций оценки качества обучения.

Number of layers (Количество слоев) – количество слоев нейронной сети.

Properties for (Свойства) – список слоев.

Number of neurons (Количество нейронов) – количество нейронов в слое.

Transfer function (Функция активации) – функции активации слоя.

Окно для импорта и загрузки данных показано на рисунке 1.5.

Source (Источники) – поле для выбора источника данных. Это либо рабочая область системы MatLab (кнопка выбора *Input from Matlab Workspace*), либо файл (кнопка выбора *Load from disk file*).

Таблица 1.1. Типы сетей, доступных с интерфейсом *NNTool*

№	Тип сети	Название сети	Число слоев
1	Competitive	Конкурирующая сеть	1
2	Cascade-forward backprop	Каскадная сеть с прямым распространением сигнала и обратным распространением ошибки	2
3	Elman backprop	Сеть Элмана с обратным распространением ошибки	2
4	Feed-forward backprop	Сеть с прямым распространением сигнала и обратным распространением ошибки	2
5	Time delay backprop	Сеть с запаздыванием и обратным распространением ошибки	2
6	Generalized regression	Обобщенная регрессионная сеть	2
7	Hopfield	Сеть Хопфилда	1
8	Linear layer (design)	Линейный слой (создание)	1
9	Linear layer (train)	Линейный слой (обучение)	1
10	LVQ	Сеть для классификации входных векторов	2
11	Perceptron	Персептрон	1
12	Probabilistic	Вероятностная сеть	2
13	Radial basis (exact fit)	Радиально базисная сеть с нулевой ошибкой	2
14	Radial basis (fewer neurons)	Радиально базисная сеть с минимальным числом нейронов	2
15	Self organizing map	Самоорганизующаяся карта Кохонена	1

Примечание:

- 1) для сетей 2, 3, 7 в данной версии *NNTool* не обеспечивается просмотр структурных схем;
- 2) сети 5, 9 допускают введение линий задержек на входе;
- 3) сети 3 допускают введение линий задержек в слое;
- 4) сети с двумя слоями имеют последовательную структуру, когда выход первого слоя служит входом второго слоя. Исключение составляют сети 3, которые допускают наличие обратной связи в первом слое и передачу входного сигнала на входы обоих слоев.

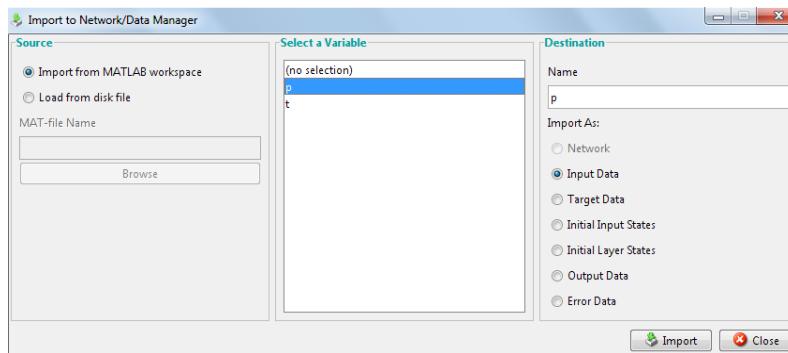


Рисунок 1.5. Окно для импорта и загрузки данных

Если выбрана первая кнопка, то в поле *Select a Variable* можно увидеть все переменные рабочей области, и, выбрав одну из них, например x , можно передать ее в поле *Destination* (Назначение) как последовательность входа *Inputs* (Входы).

При выборе кнопки *Load from disk file* активизируется поле *MAT-file Name* и кнопка *Browse*, что позволяет начать поиск и загрузку файла из файловой системы.

Окно для экспорта или записи данных в файл (*Export or Save from Network/Data Manager*) показано на рисунке 1.6 и позволяет передавать данные из рабочей области GUI-интерфейса *NNTool* в рабочую область системы MatLab или записать их в виде файла на диске.

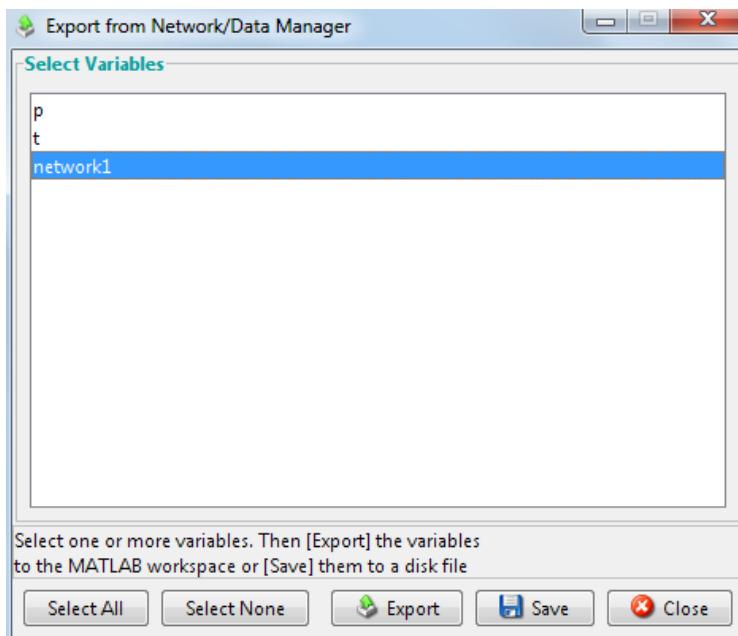


Рисунок 1.6. Окно для экспорта или записи данных в файл

Диалоговая панель *Network* показана на рисунке 1.7.

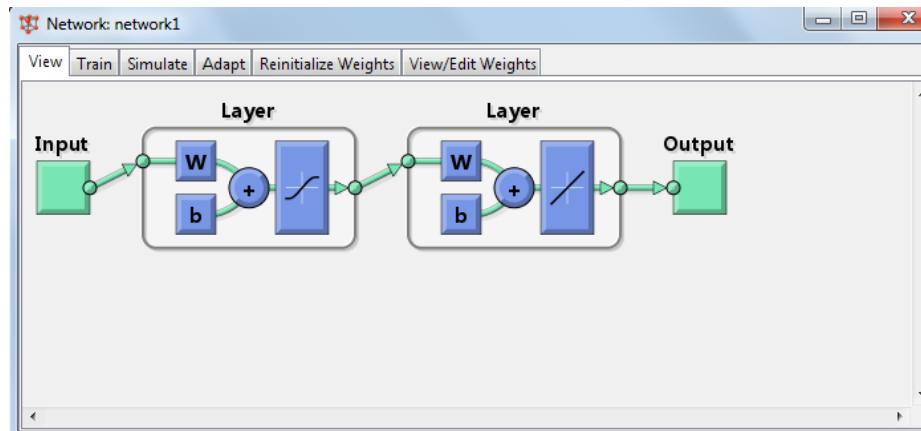


Рисунок 1.7. Диалоговая панель Network

Данная диалоговая панель открывается только в том случае, когда в окне *Network/Data Manager* выделена созданная сеть и становятся активными кнопки *View, Train, Simulate, Adapt, Reinitialize Weights, View/Editor Weights*.

Панель имеет 6 закладок:

- *View* (Просмотреть) – структура сети;
- *Train* (Обучение) – обучение сети;
- *Simulate* (Моделирование) – моделирование сети;
- *Adapt* (Адаптация) – адаптация и настройка параметров сети;
- *Reinitialize Weights* (реинициализация) – возвращение значений весов, смещений и входного диапазона к значениям последней инициализации;
- *View/Editor Weights* (Просмотреть/Редактор весов) – просмотреть/Редактор весов.

Пример: создадим, используя графический интерфейс пользователя, нейронную сеть для вычисления функции $z = 2 \cdot x^2 - y^3$.

p = [-1 -0.7 -0.6 -0.4 0 0.2 0.3 0.6 0.8 1;-0.9 -0.8 -0.5 -0.3 -0.1 0.1 0.3 0.5 0.7 0.9]; % векторов входа
t = [2.729 1.492 0.845 0.347 0.001 0.079 0.153 0.595 0.937 1.271] % вектор цели

Откроем с помощью функции ***nntool*** основное окно интерфейса, затем сформируем последовательность входов и целей в рабочей области GUI-интерфейса, используя окно *Create New Data*.

С этой целью сначала нажмем кнопку *New ...* и далее – в поле *Name* окна *Create New Data* – введем сначала имя переменной *p*, затем – в области значений *Value* – вектор значений $[-1 \ -0.7 \ -0.6 \ -0.4 \ 0 \ 0.2 \ 0.3 \ 0.6 \ 0.8 \ 1; -0.9 \ -0.8 \ -0.5 \ -0.3 \ -0.1 \ 0.1 \ 0.3 \ 0.5 \ 0.7 \ 0.9]$ и, используя кнопку *Inputs* (в правой части окна), укажем тип переменных (*Inputs* - Входы). Ввод завершим нажатием кнопки *Create* (Создать).

Аналогичную операцию проделаем для вектора *t*, с указанием (с помощью кнопки *Targets*), что это – вектор целевых данных.

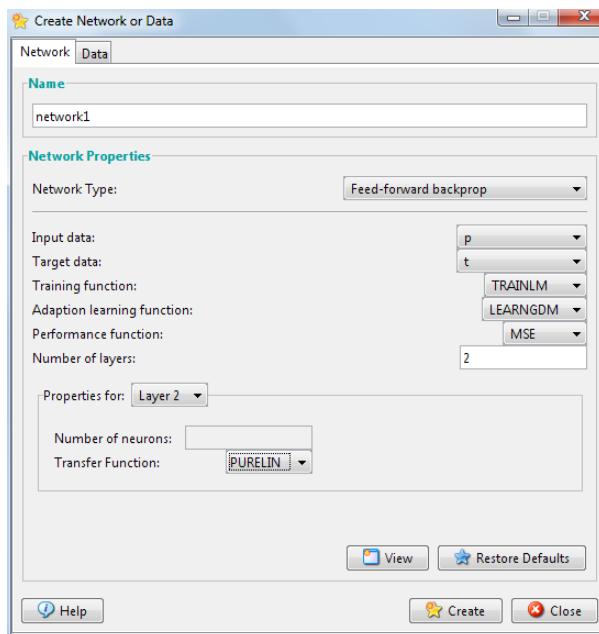


Рисунок 1.8 – Окно создания нейронной сети

Создадим новую нейронную сеть. Для этого перейдем на вкладку *Network* и выберем нейронную сеть типа *Feed-forward backprop* с прямой передачей сигнала и обратным распространением ошибки. При создании сети сохраним ей имя, даваемое по умолчанию (*network1*), Input data , Target data выбрать из списка (p, t) а количество нейронов (*Number of neurons*) первого слоя (*Layer 1*) установим равным двум, функция активации во

втором слое – линейная (PURELIN). Остальные установки при создании сети оставим по умолчанию (Рисунок 1.8). Создание сети завершим нажатием кнопки *Create*.

После этого в окне *Network/Data Manager*, в области *Networks* появится имя новой созданной сети – *network1*. Выберем это имя с помощью мышки, что ведет к активации всех кнопок указанного окна.

Обучение нейронной сети в GUI

Затем выполняется обучение сети, для чего выбирается закладка *Train* и открывается диалоговая панель, показанная на рисунке 1.9.

Панель имеет две закладки:

- 1) *Training info* (Информация об обучающих последовательностях) – Рисунок 1.9;
- 2) *Training Parameters* (Параметры обучения) – рисунок 1.10.

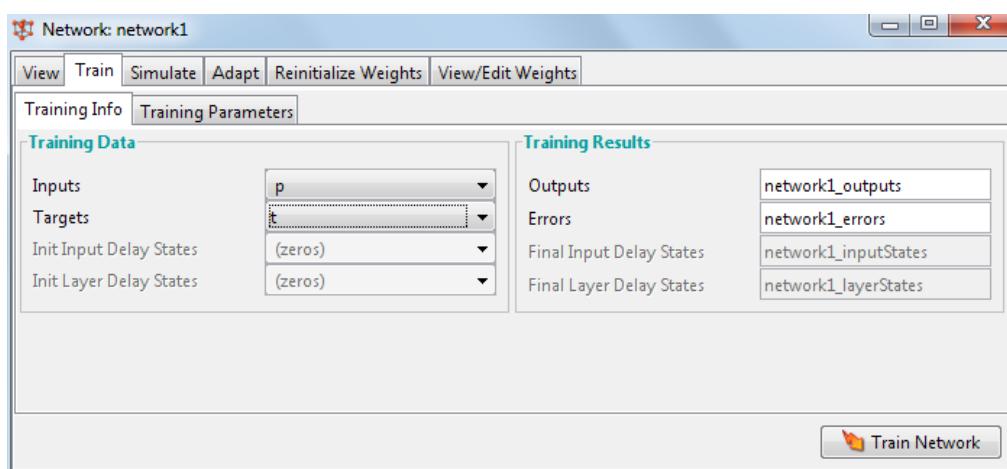


Рисунок 1.9. Окно информации об обучающих последовательностях

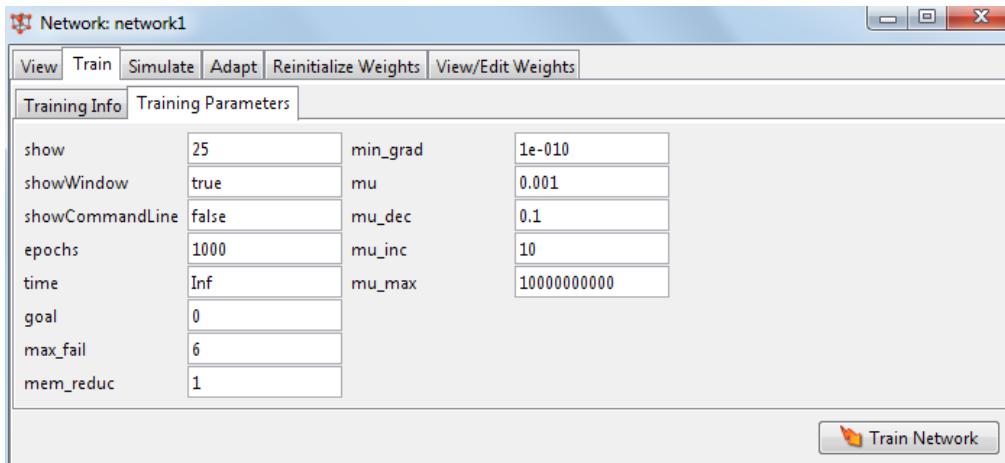


Рисунок 1.10. Окно с информацией о параметрах обучения

Применяя эти закладки, можно установить имена последовательностей входа и цели (на вкладке *Training Info* – в левой ее части необходимо указать p и t), а также значения параметров процедуры обучения (на вкладке *Training Parameters*; в условиях примера сохраняем значения по умолчанию).

Теперь нажатие кнопки *Train Network* вызывает обучение сети. Качество обучения сети на выбранной обучающей последовательности поясняется на рисунках 1.11-1.13. Видно, что к концу процесса обучения ошибка становится очень малой (вид данного рисунка при повторе вычислений может отличаться от приведенного).

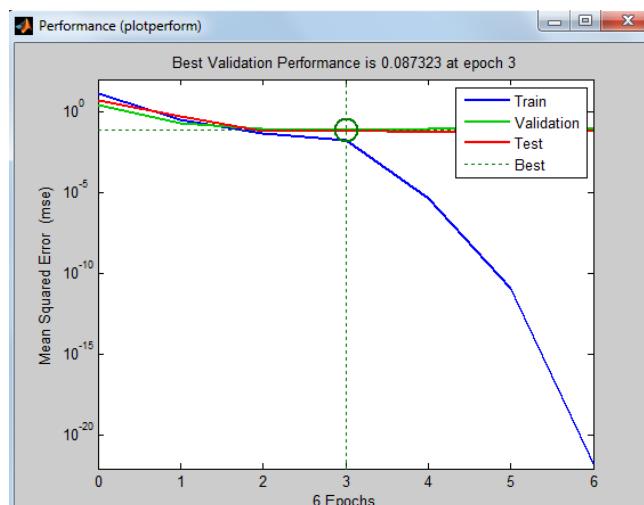


Рисунок 1.11. Изменение ошибки сети в процессе обучения

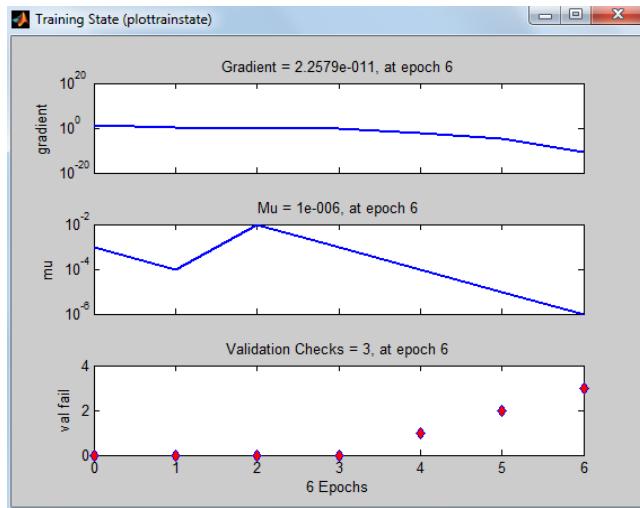


Рисунок 1.12. Окно состояния обучения

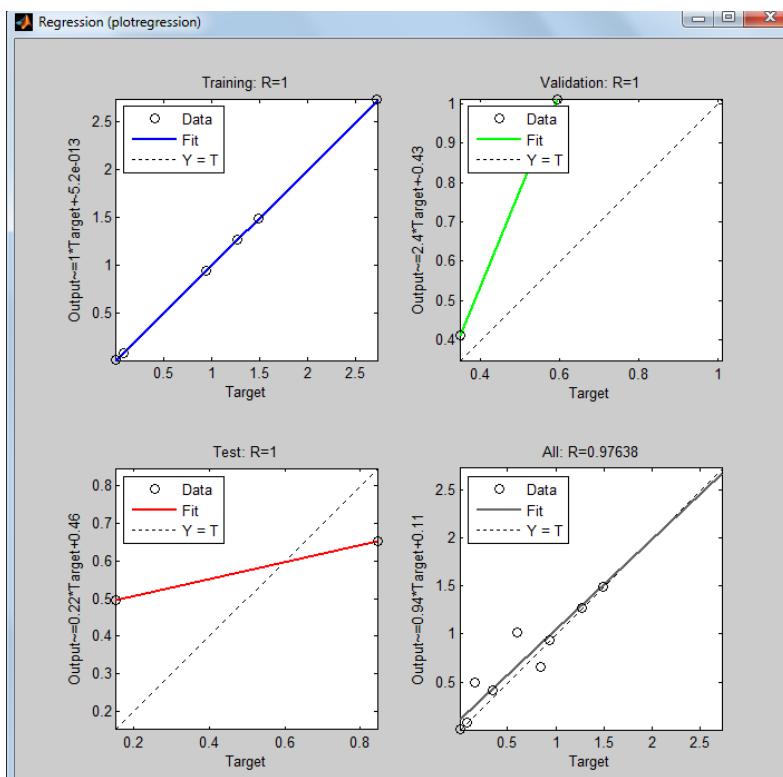


Рисунок 1.13. Окно линейной регрессии между выходом НС и целями

Результаты обучения можно просмотреть в окне *Network/Data Manager*, активизируя щелчком левой кнопки мыши имена последовательностей выходов *network1_outputs* или ошибок *network1_errors*(рисунок 1.15).

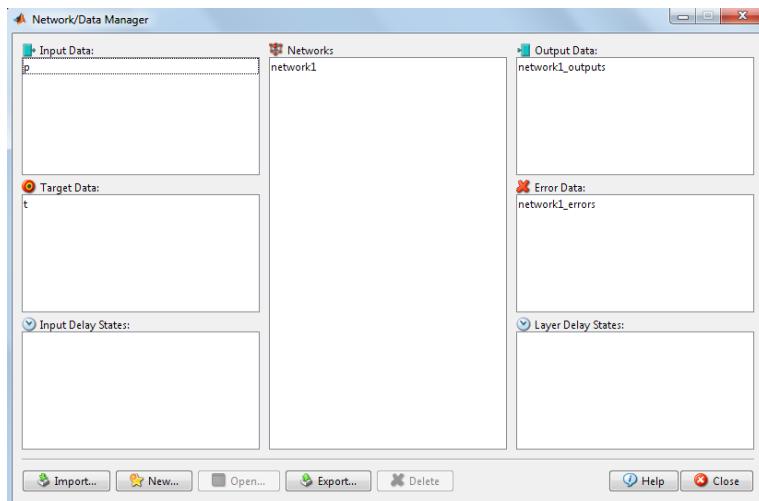


Рисунок 1.14. Окно Network/Data Manager

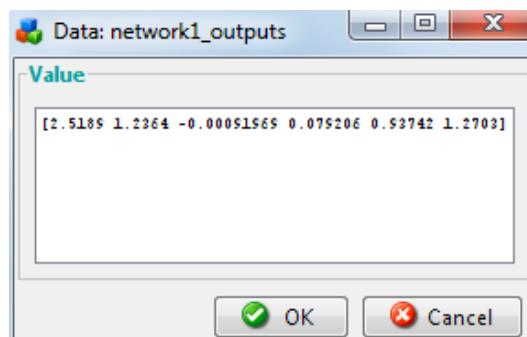


Рисунок 1.15. Значения выходов сети

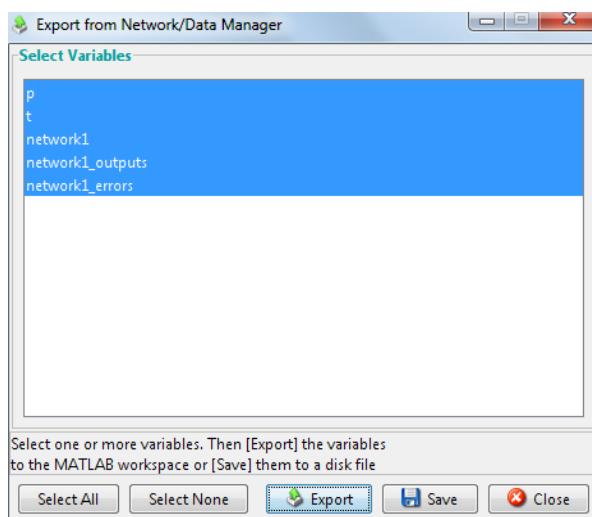


Рисунок 1.16. Окно для экспорта или записи данных в файл

При необходимости можно экспорттировать созданную нейронную сеть в рабочую область системы MatLab (нажав кнопку *Export* и далее, в открывшемся окне *Export from Network/Data Manager* – кнопки *Select All* (Выбрать все) и *Ex-*

port (Рисунок 1.16)) и получить информацию о весах и смещениях непосредственно в рабочем окне системы, выполнив команду:

```
network1.IW{1,1}, network1.b{1};
```

```
network1.IW{2,1}, network1.b{2};
```

Построить модель НС в среде Simulink и отобразить ее схему можно командой **gensim(network1)** (рисунок 1.17). Эти схемы могут быть применены для моделирования нейронной сети.

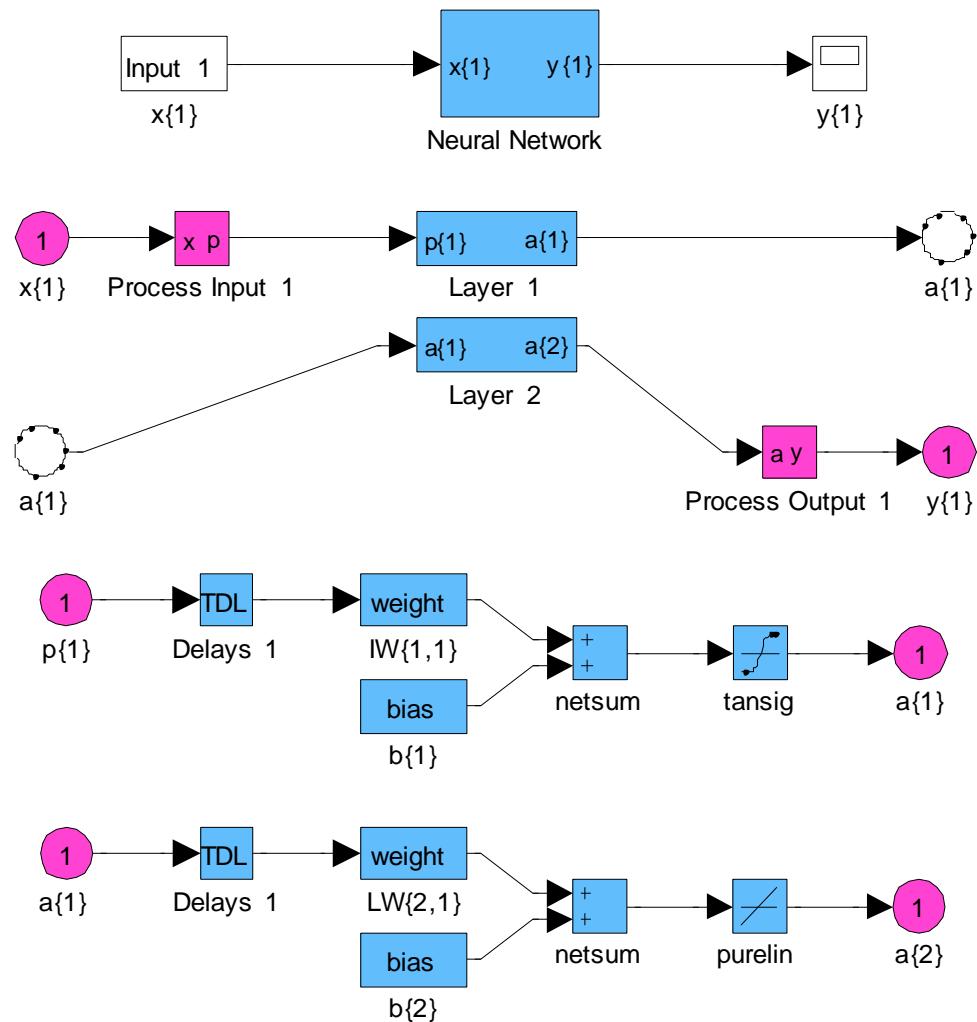


Рисунок 1.17. Структурные схемы, созданной нейронной сети в среде Simulink

Программа работы и методические указания

Создайте, используя графический интерфейс пользователя, нейронную сеть для аппроксимации функции (табл. 1.2), векторы входа и цели студент формирует самостоятельно.

Таблица 1.2. Функции двух переменных

№ варианта	Функция
1.	$y = 2 \sin^2(x_1) \cdot \cos(x_2), \quad x_1, x_2 \in [-1;1]$
2.	$y = 4 \frac{\cos(x_1)}{x_2}, \quad x \in [1;\pi]$
3.	$y = x_1 \cdot \cos(x_2), \quad x_1, x_2 \in [-1;1]$
4.	$y_1 = \frac{x_1 - x_2}{x_1 + x_2}, \quad x_1, x_2 \in [1;5]$
5.	$y = \cos(x_2) \cdot \sin(x_1), \quad x_1, x_2 \in [-1;1]$
6.	$y = x_1 \cdot \sin(x_2), \quad x_1, x_2 \in [-1;1]$
7.	$y_1 = x_1 \cdot x_2 + \sin x_2, \quad x_1, x_2 \in [-\pi;1]$
8.	$y_1 = 1,5 \cdot x_1 + x_1^3, \quad x_1, x_2 \in [-1;1]$
9.	$y_1 = \sqrt{x_1^2 + x_2^2}, \quad x_1, x_2 \in [-1;1]$
10.	$y_1 = 2,3 \cdot x_1 \cdot x_2 - 0,5 \cdot x_1^2 + 1,8 \cdot x_2^2, \quad x_1, x_2 \in [1;10]$
11.	$y = 6 \cdot x_1 + 5 \cdot x_1 \cdot x_2 + x_2^2, \quad x_1, x_2 \in [-5;5]$
12.	$y = \sin(x_1) \cdot \cos(x_2), \quad x_1, x_2 \in [-\pi;\pi]$
13.	$y = 2 \cdot x_1 \cdot \cos x_2, \quad x_1 \in [0;1], \quad x_2 \in [-\pi;\pi]$
14.	$y = 2 \cdot x_1^2 + \cos(x_2), \quad x_1, x_2 \in [-1;1]$
15.	$y = 2 \cdot x_2 \cdot \sin^2(x_1), \quad x_1, x_2 \in [-1;1]$
16.	$y = 4 \frac{\sin(x_2)^2}{\cos(x_1)}, \quad x_1, x_2 \in [-\pi;\pi]$
17.	$y = \operatorname{arctg} \frac{x_2}{x_1}, \quad x_1, x_2 \in [-1;1]$
18.	$y = (3 \cdot x_1^2 - x_2^2)^3, \quad x_1, x_2 \in [-1;1]$
19.	$y = 2 \cdot x_2 \cdot \sin^2(x_1), \quad x_1, x_2 \in [-1;1]$
20.	$y = 2 \cdot \operatorname{tg}(x_2)^2 \cdot \cos(x_1), \quad x_1, x_2 \in [-1;1]$

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Лабораторная работа № 2

ИЗУЧЕНИЕ СВОЙСТВ ЛИНЕЙНОГО НЕЙРОНА И ЛИНЕЙНОЙ НЕЙРОННОЙ СЕТИ

Цель работы: изучить свойства линейного нейрона.

1. Краткие теоретические сведения

Нейрон, используемый в модели персептрана, имеет ступенчатую функцию активации hardlim с жесткими ограничениями (рисунок 2.1).

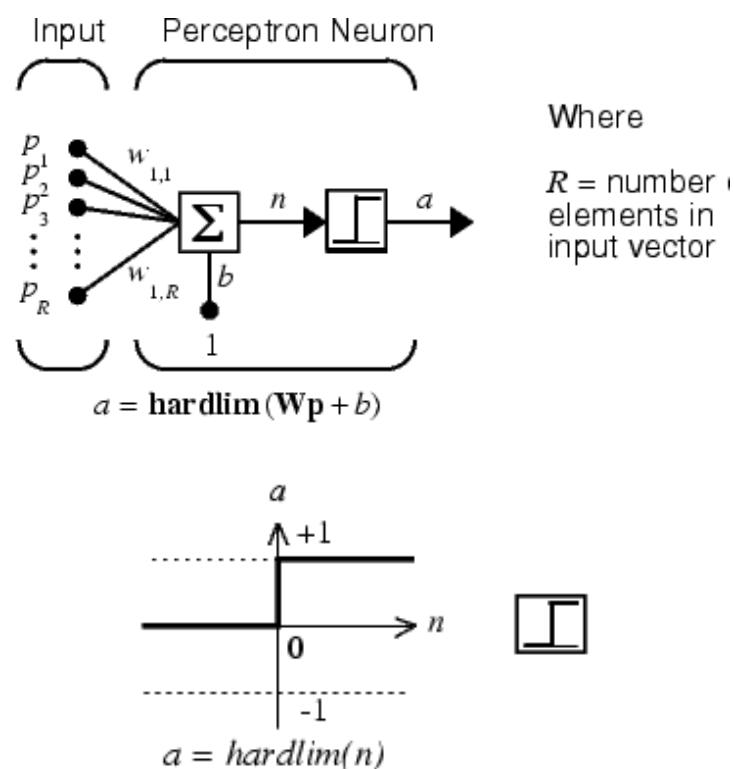


Рисунок 2.1. Структура порогового нейрона

Каждое значение элемента вектора входа персептрана умножено на соответствующий вес w_{1j} , и сумма полученных взвешенных элементов является входом функции активации.

Если вход функции активации $n \geq 0$, то нейрон персептрана возвращает 1, если $n < 0$, то 0.

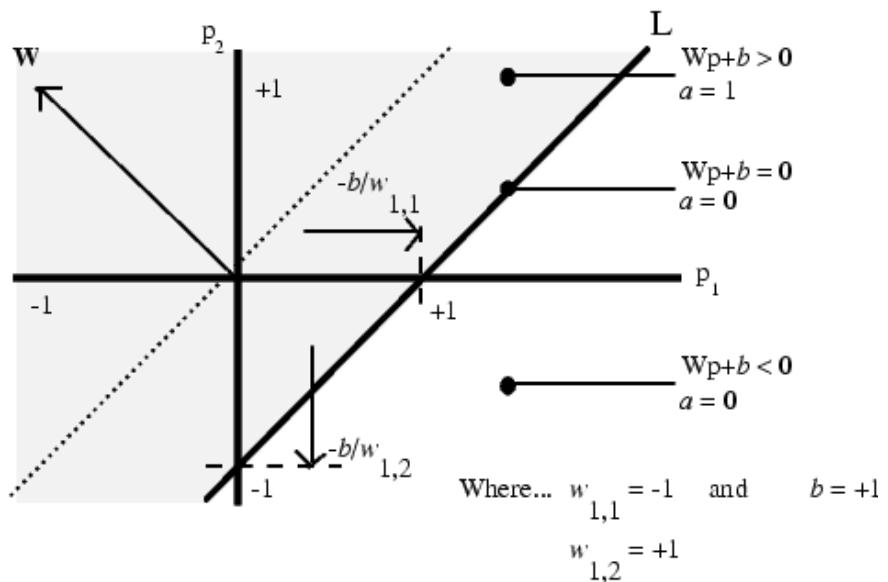


Рисунок 2.2. Разделение нейроном пространства входа на две области

Функция активации с жесткими ограничениями придает персептрону способность классифицировать векторы входа, разделяя пространство входов на две области, как это показано на рисунке 2.2, для персептрана с двумя входами и смещением.

Пространство входов делится на две области разделяющей линией L , которая для двумерного случая задается уравнением

$$W^T p + b = 0. \quad (2.1)$$

Эта линия перпендикулярна к вектору весов w и смещена на величину b . Векторы входа выше линии L соответствуют положительному потенциалу нейрона, и, следовательно, выход персептрана для этих векторов будет равен 1; векторы входа ниже линии L соответствуют выходу персептрана, равному 0. При изменении значений смещения и весов граница линии L из меняет свое положение.

Персептрон без смещения всегда формирует разделяющую линию, проходящую через начало координат; добавление смещения формирует линию, которая не проходит через начало координат, как это показано на рисунке 2.2.

В случае, когда размерность вектора входа превышает 2, т. е. входной вектор p имеет более 2 элементов, разделяющей границей будет служить гиперплоскость.

Для того чтобы создать нейрон, используют функцию *newp*, имеющую следующий синтаксис [41, 42]:

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

где PR – матрица минимальных и максимальных R входных элементов, S – количество нейронов (при создании одного нейрона $S = 1$), TF – функция активации (transfer function), LF – имя функции обучения нейрона.

В случае если параметры функции *newp* не заданы, их значения определяются посредством ввода значений в диалоговые окна. Построенный нейрон характеризуется функциями весов (*weight function*), входов сети (*net input function*) и определенной функцией активации. Функция весов – это умножение весов на входной сигнал, функция входов сети – их сумма. Веса задаются как для входов нейрона, так и для фиксированного входа, задающего порог срабатывания (*bias*). Вектор весов инициализируется нулями. Для обучения используются функции, рассмотренные ниже.

Функция *learnp* настраивает веса нейрона. Синтаксис функции обучения довольно сложен:

```
[dW, LS] = leamp(W, P, Z, N, A, T, E, gW, gA, D, LP, LS),  
[db, LS] = leamp(b, ones(l,Q), Z, N, A, T, E, gW, gA, D, LP, LS),  
Info = learnp(code)
```

Функция *learnp* ($W, P, Z, N, A, T, E, gW, gA, D, LP, LS$) имеет несколько входов, где вектор W – вектор весов; P – вектор входов; Z – вектор взвешенных входов; N – вектор сети; A – вектор выхода; T – вектор желаемых выходов; E – вектор ошибок; gW – вектор изменения весов; gA – изменения выходов. Функция возвращает значения: dW – изменения матрицы весов; LS – новый уровень обученности.

Функция *learnp* может быть использована с параметрами по умолчанию:

```
dW = learnp([ ], p, [],[],[],[], e, [],[],[],[],[]).
```

Использование пустого списка [] означает параметр по умолчанию.

Функция learnpr вычисляет изменение весов dW для заданного нейрона в соответствии с правилом обучения персептрона:

$$dw = \begin{cases} 0, & \text{если ошибка } e = 0, \\ p', & \text{если ошибка } e = 1, \\ -p', & \text{если ошибка } e = -1, \end{cases}$$

т.е. $dw = e \cdot p'$

Тогда новый вектор весов примет вид: $w = w + dw$.

Функция обучает нормализованные веса:

[dW, LS] = learnpn{W, P, Z, N, A, T, E, gW, gA, D, LP, LS). Функция learnpn вычисляет изменение весов dW для данного нейрона и его входа P и ошибки E в соответствии с нормализованным правилом обучения персептрона:

$$pn = \sqrt{(1 + p(1)^2 + \dots + p(R)^2)},$$

$$dw = \begin{cases} 0, & \text{если ошибка } e = 0, \\ np', & \text{если ошибка } e = 1, \\ -np', & \text{если ошибка } e = -1, \end{cases}$$

т.е. $dw = e \cdot np'$

Линейный нейрон имеет одно существенное ограничение. Входные векторы должны быть линейно сепарабельны. Если векторы невозможно отделить прямой или гиперплоскостью, то персептрон не способен решить задачу классификации.

Функция adapt адаптирует нейрон к условиям задачи:

[net, Y, E, Pf, Af] = adapt(net, P, T, P_i, A_i).

Параметры функции: net – имя сети, P – входы сети, T – желаемый выход, P_i – исходные условия задержки, A_i – исходные условия задержки для слоя.

Функция возвращает параметры адаптированной сети net.adaptParam: net – измененная сеть, Y – выход сети, E – ошибки сети, Pf – условия задержки входов, Af – условия задержки слоя. Параметры P_i и Pf необязательные и необходимы только для сетей, имеющих задержки на входах и слое.

Функция *train* также обучает нейронную сеть и использует следующий синтаксис:

$[net, tr] = \text{train}(\text{NET}, P, T, P_i, A_i)$

$[net, tr] = \text{train}(\text{NET}, P, T, P_i, A_i, W, TV)$.

Функция $\text{train}(net, P, T, P_i, A_i)$ имеет следующие параметры: net – сеть, P – входы сети, T – желаемый выход, P_i – исходные условия задержки входа, A_i – исходные условия задержки слоя.

Функция эмулирует нейронную сеть:

$[Y, Pf, Af] = \text{sim}(net, P, P_i, A_i)$,

где net – сеть;

P – входы сети;

P_i – исходные условия задержки входов сети;

A_i – исходные условия задержки слоя.

Функция возвращает

Y – выходы сети;

Pf – окончательные условия задержки входов;

Af – окончательные условия задержки слоя.

Ниже представлены назначения этих функций.

Таблица 2.1. Функции активации

Функция	Назначение
<code>hardlim(N)</code>	Возвращает 1, если N положительное и 0 в противном случае
<code>tansig(N)</code>	Вычисляет гиперболический тангенс от входа
<code>purelin</code>	Вычисляет выход слоя от сетевого входа

Таблица 2.2. Функции графического интерфейса и вспомогательные функции

Функция	Назначение
<code>axis([Xmin Xmax Ymin Ymax])</code>	Устанавливает диапазоны координатных осей
<code>title('строка')</code>	Выводит в графическое окно рисунков заголовок графика
<code>rand(M, N)</code>	Возвращает матрицу размерности M на N со случайными значениями
<code>xlabel('строка')</code> <code>ylabel('строка')</code>	Подписывают наименование координатных осей
<code>cla reset</code>	Очищает координатную сетку в окне рисунков

Функция	Назначение
hold on и hold off	Включают и отключают режим добавления графиков на координатную сетку
text(X, Y, 'строка')	Выводит строку, начиная с указанных координат в поле рисунков
pause (n)	Ожидает пользовательского ответа n секунд
plot(X, Y, 'цвет и символ')	Изображает на координатной сетке точки с координатами, заданными векторами X, Y, с помощью указанного символа и цвета ¹ .
plotpv(P, T)	Изображает точки P указанными маркерами T, где P – матрица входных векторов размерностью R на Q (R должен быть 3 или меньше), T – матрица двоичных векторов размерностью S на Q (S должен быть 3 или меньше)
plotes(WV, BV, ES, V)	Изображает поверхность ошибки на отдельном входе, где WV – вектор строк значений весов W размерности N, BV – вектор строк значений порогов В размерности M, ES – матрица ошибки размерности L/на N, V – угол зрения по умолчанию [-37,5, 30]
plotsom(POS)	Изображает позицию нейрона красной точкой, связывая синей линией нейрона, находящиеся друг от друга на расстоянии 1. POS – матрица S N-размерных нейронов
ind2vec и vec2ind	Позволяют представить индексы либо собственно значениями индексов, либо векторами, строки которых содержат 1 в позиции индекса ²
full	Преобразует разреженную матрицу в полную
maxlinlr(P)	Функция возвращает максимальный уровень обученности линейного слоя без bias, который обучался только на векторе P
trainlm	Выполняет обучение многослойной НС методом Левенберга–Марквардта
netprod	Входная сетевая функция, которая вычисляет выход сетевого слоя, умножая входной вектор на веса и прибавляя bias
init	Итеративно инициализирует НС

¹ plot{X, Y, 'g+!') изображает точки зеленого цвета с помощью символа «+».

² для четырех векторов (содержащих только одну 1 в каждой строке) можно найти индексы единиц: vec = [1 0 0 0; 0 0 1 0; 0 1 0 1]; ind = vec2ind(vec).

Структура данных описания нейронных сетей. Структура данных net – это описание обученной НС. Обучение осуществляется в соответствии со следующими параметрами, значения которых либо устанавливаются пользователем, либо по умолчанию (табл. 2.3).

Таблица 2.3. Структура данных описания нейронных сетей

Структура данных	Комментарий
net.trainParam.epochs 100	максимальное количество эпох обучения
net.trainParam.goal 0	целевое значение ошибки
net.trainParam.max_fail 5	максимальное значение ошибки
net.trainParam.mem_reduc 1	фактор оптимизации процесса обучения: оптимизация использования памяти или времени процессора
net.trainParam.min_grad le-10	минимальное значение градиента
net.trainParam.show 25	количество эпох между показами
net.trainParam.time inf	максимальное время обучения в секундах
TR	структура данных, содержащая значения об обученности НС в текущую эпоху
TR.epoch	номер эпохи
TR.perf	уровень обученности (Trainingperformance)
TR.vperf	степень качества (Validationperformance)
TR.tperf	результативность обработки теста (Testperformance)
TR.mu	значение адаптивности

Структура данных описания адаптированной НС net.adaptfcn включает в себя следующие поля net.adapt.param:

NET – адаптированная НС;

Y – выходы НС;

E – ошибки НС;

Pf – окончательные входные значения задержек;

Af – окончательные выходные задержки;

TR – результат обучения (эпохи и целевая ошибка).

Проведем в среде MatLab toolbox эксперименты, используя рассмотренные функции.

2. Создание нейрона, выполняющего функцию логического И

Создадим нейрон с одним двухэлементным входом (интервалы первого и второго элементов [0; 1]). Определим два первых параметра функции *newp*, а в качестве значений третьего и четвертого параметра (типа функции активации и имени процедуры обучения нейрона) воспользуемся значениями по умолчанию.

net = newp([0 1; 0 1], 1); % создание нейрона с одним двухэлементным входом (интервал первого элемента [0; 1] и второго элемента [0; 1]).

Для того чтобы исследовать поведение нейрона, необходимо имитировать его работу с помощью функции *sim*. Для определения последовательности значений входа создадим последовательность Р1.

P1 = {[0; 0] [0; 1] [1; 0] [1; 1]}; % создание последовательности значений входа

Y = sim(net, P1); % имитация работы нейрона net на последовательности входов Р1 желаемых выходов – Т, которая позволит нам провести адаптацию нейрона (обучить его) через 20 проходов.

T1 = {0, 0, 0, 1}; % создание последовательности выходов

net.adaptParam.passes = 20; % установка количества проходов (циклов) адаптации

net = adapt(net, P1, T1); % адаптация нейрона net для обучающей выборки <P1;T>

Y = sim(net, P1); % симуляция работы нейрона net на последовательности входов Р1

Y =

[0] [0] [0] [1]

В результате мы получим нейрон, выполняющий функцию конъюнкции.

3. Обучение нейрона выполнению функции логического ИЛИ

Для переобучения нейрона на выполнение функции ИЛИ переопределим входы Р и выходы Т.

P2 = [0 0 1 1; 0 1 0 1]; % создание последовательности входов

T2 = [0, 1, 1, 1]; % создание последовательности выходов (реакций) для нейрона, выполняющего функцию логического ИЛИ

Инициализируем нейрон, обучим его на 20 проходах (эпохах).

net = init(net); % инициализация нейрона net

Y = sim(net, P2); % имитация работы нейрона net на последовательности входов Р2

net.trainParam.epochs = 20; % установка количества проходов

```
net = train(net, P2, T2); % обучение нейрона net на обучающей выборке
<P2, T2>
```

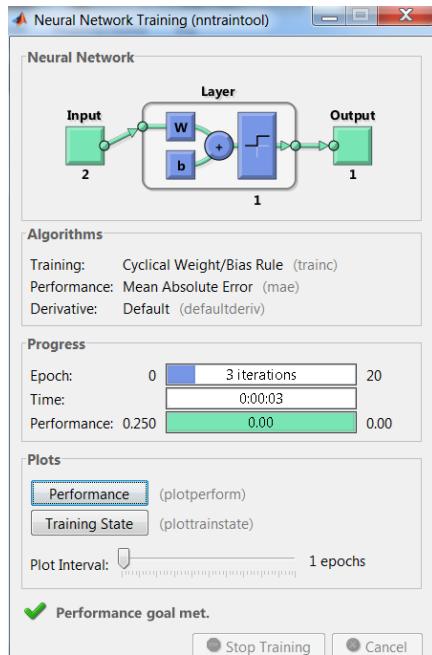


Рисунок 2.3. Окно обучения нейронной сети

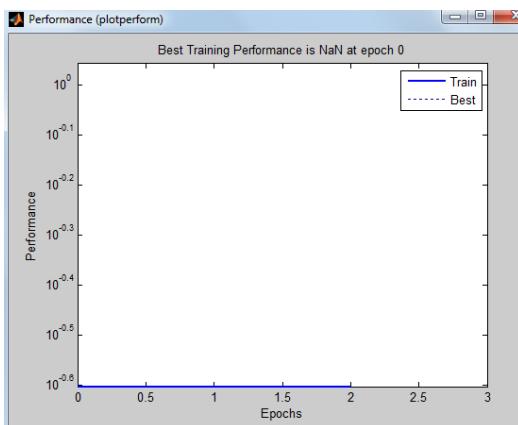


Рисунок 2.4. Изменение ошибки сети в процессе обучения

Y = sim(net, P2) % имитация работы нейрона net на последовательности входов P2

Y =

0 1 1 1

Для случайного изменения весов и порога срабатывания используем функцию `init`. По умолчанию для создаваемого нейрона указана функция `hardlim`.

4. Определение входов со случайными значениями

Определим случайный вход p и ошибку e для нейрона с двухэлементным входом.

```
p = rand(2,1); % определение входа p как вектора двух случайных чисел из интервала [0;1]
```

```
e = rand(0,1); % определение входа e как случайного числа из интервала [0;1]
```

5. Построение графика функции активации hardlim

График функции активации для интервала $[-3; +3]$ с шагом 0.1.

```
n = -3:0.1:3; % определение переменной n со значениями из интервала [-3; 3] с шагом 0.1
```

```
b = hardlim(n); % вычисление пороговой функции от переменной n  
plot(n, b); % изображение графика функции b = hardlim(n)
```

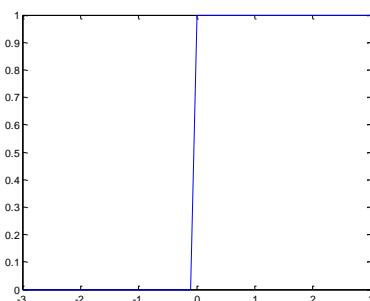


Рисунок 2.5. График функции активации hardlim

6. Имитация работы линейного нейрона

Создадим с помощью newp нейрон с двухэлементным входом (значения входа в интервале $[0;1]$):

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

Функция sim имитирует работу нейрона для отдельного вектора и для трех векторов.

```
p1 = [.3; .7]; % определение двухэлементного вектора
```

a1 = sim(net, p1); % имитация работы нейрона net на входном векторе
результат имитации – вектор a1

p2 = [.3 .5 .2; .8 .5 .4]; %определение матрицы входов размерностью 3 на 2
(три двухэлементных вектора)

a2 = sim(net, p2); % имитация работы нейрона net на входном векторе p2,
результат имитации – вектор a2

7. Обучение нейрона классификации векторов на две категории

Начнем с классификации векторов на основе двухвходового нейрона. Будем использовать функцию *newpr* для создания нейрона, *sim* для имитации его работы, *adapt* для адаптации (обучения) нейронной сети. Обучим двухвходовой нейрон классифицировать входные векторы на две категории. Определим четыре входных вектора нейрона.

P = [-0.5 -0.5 +0.4 -0.2; -0.5 +0.5 -0.4 +1.0]; % определение четырех двух-
элементных входов

Зададим желаемые выходы нейрона для определенных векторов.

T = [1 1 0 0]; % определение выходов нейрона – реакций на входы P.

Изобразим входные векторы (точки) на плоскости:

plotpv(P, T)

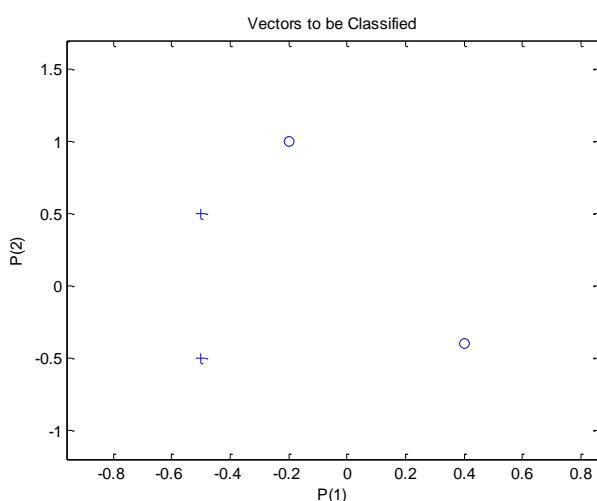


Рисунок 2.6. Изображение входных векторов на плоскости

Каждый из четырех входных векторов на плоскости Р определяется двумя координатами, представленными как двухэлементные столбцы в матрице Р.

Создадим один линейный нейрон с двумя входами, значения которых лежат в интервале [-1,1].

```
net = newp([-1 1; -1 1], 1); % создание линейного нейрона с двумя входами из интервала [-1, 1]
```

Нейрон по умолчанию имеет функцию активации *hardlim* и такой нейрон разделяет входные векторы прямой линией.

Определим координаты линии классификации: веса (IW) и порог срабатывания нейрона (b).

```
linehandle = plotpc(net.IW{1}, net.b{1}); % получение управляющей структуры linehandle для изображения разделяющей линии в координатах весов (IW) и порога срабатывания нейрона (b).
```

```
plotpc(net.IW{1}, net.b{1}); % изображение разделяющей прямой
```

Если исходным весам задать нулевые значения, то любые входы дадут одинаковые выходы, и линия классификации не будет видна на плоскости. Проведем обучение:

```
clf; % очистка координатных осей
```

```
plotpv(P, T); % изображение входных векторов двух категорий, категория задается элементами вектора Т
```

```
linehandle = plotpc(net.IW{1}, net.b{1}); % получение управляющей структуры linehandle
```

```
E = 1; % присвоение начального значения ошибки
```

```
net = init(net); % инициализация нейрона
```

```
linehandle = plotpc(net.IW{1}, net.b{1}); % получение управляющей структуры linehandle
```

```
while(mse (E)); % организация цикла пока ошибка не равна 0
```

```
[net, Y, E] = adapt(net, P, T); % адаптация нейрона net на обучающей выборке <P, T>, функция возвращает адаптированный нейрон net, выход Y, ошибку E
```

```

linehandle = plotpc(net.IW{1}, net.b{1}, linehandle); % изображение разде-
ляющей прямой нейрона после адаптации
drawnow; % очистка окна графиков
end; % конец цикла while

```

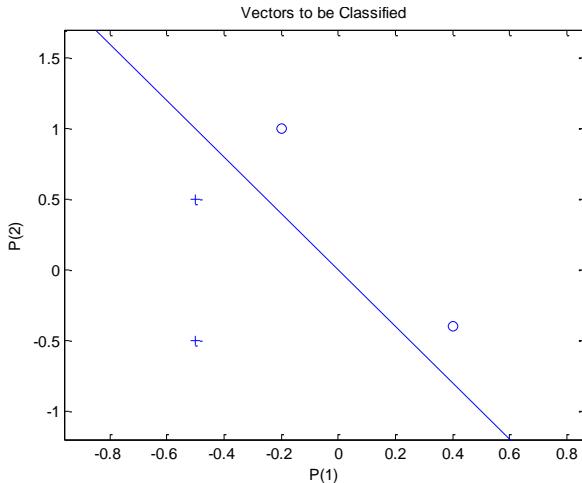


Рисунок 2.7. Изображение разделения пороговым нейроном входных векторов

Функция `adapt` возвращает новый объект – сеть, которая выполняет классификацию, выход сети и ошибку.

Проведем классификацию нового вектора с помощью обученного нейрона на основе функции `sim`. Определим новый вектор P .

$P = [0.6; 1.1];$ % определение вектора P

$a = sim(net, P);$ % имитация работы нейрона net , получение отклика нейрона a

`plotpv(P, a);` % изображение входа P , отнесенного нейроном к категории a

Обученный нейрон можно использовать для классификации любого вектора.

`hold on;` % включить режим добавления графиков в графическом окне

`plotpv(P, T);` % изображение входных точек в соответствии с категориями T

`plotpc(net.IW {1}, net.b{1});` % изображение разделяющей поверхности

`hold off;` % отключение режима добавления графиков

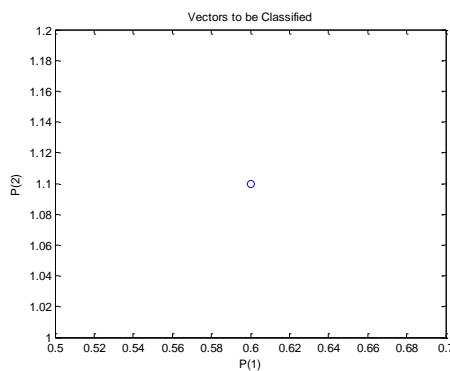


Рисунок 2.8. Изображение входа Р, отнесенного нейроном к категории а

Нейрон классифицирует новую точку, как принадлежащую категории «0» (представлена кружком), а не категории «1» (представлена +).

8. Создание слоя линейных нейронов

Рассмотрим последовательность из 10 шагов (для выхода Т1, который известным образом зависит от входов Р1):

P1 = [-1 0 0 0 1 1 -1 0 -1 1]; % последовательность входов

T1 = [-1 -1 1 0 1 2 0 -1 -1 1]; % последовательность выходов

Используем функцию *newlin*, чтобы создать нейрон со значениями входа в интервале [-1;1], задержками входа от 0 до 1 и уровнем обучения 0,1.

net = newlin([-1 1], 1, [0 1], 0.1); % создание линейного слоя из одного нейрона со значениями входа в интервале [-1; 1], задержками входа от 0 до 1 и уровнем обучения 0,1.

Адаптируем нейрон к задаче одним проходом через последовательность входа. Измерим среднюю квадратичную ошибку с помощью функции *mse(e)*.

[net, y, e, pf] = adapt(net, P1, T1); % адаптация нейрона к последовательности Р1

mse(e) % измерение ошибки

ans =

1.1000

Получим довольно большую ошибку. Вновь адаптируем сеть на 10 шагах последовательности, используя предыдущее значение P_f как новое исходное значение задержки:

p2 = [1 -1 -1 1 1 -1 0 0 0 1];

t2 = [2 0 -2 0 2 0 -1 0 0 1];

[net, y, e, pf] = adapt(net, p2, t2, pf); % адаптация с начальным вектором задержки pf

mse(e)

ans =

0.6476

Адаптируем сеть на 100-разовом прогоне последовательности:

p3 = [P1 p2]; % формирование новой последовательности входов

t3 = [T1 t2]; % формирование новой последовательности выходов

net.adaptParam.passes = 1000; % установка количества проходов

[net, y, e] = adapt(net, p2, t2); % адаптация нейрона

mse(e);

Получим приемлемую ошибку, значит сеть обучена зависимости выходов от входов.

9. Изучение возможности линейного нейрона решать линейно несепарельные задачи

Проведем эксперимент с линейным нейроном по классификации пяти входных векторов на две категории, которые линейно несепарабельны.

Определим входы линейного нейрона:

P = [-0.5 -0.5 +0.4 -0.1 -0.9;-0.5 +0.5 -0.5 +1.0 +0.0]; % входы линейного нейрона

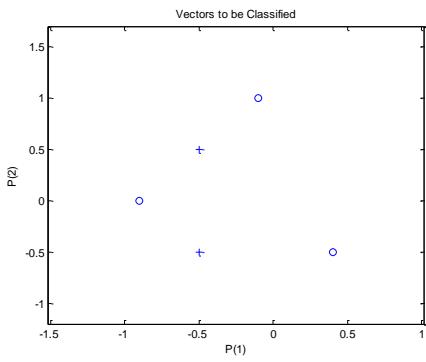


Рисунок 2.9. Изображение входных векторов на плоскости

$T = [1 \ 1 \ 0 \ 0 \ 0]$; % выходы – реакции нейрона на входы Р

Поскольку векторы разных видов нельзя отделить прямой линией, их классификация невозможна.

```
net = newp([-2 2; -2 2],1); % создание нейрона
plotpv(P,T); % изображение обучающей выборки
linehandle = plotpc(net.IW{1}, net.b{1}); % изображение разделяющей
прямой
```

Обучим нейрон классификации векторов.

```
for a = 1:20; % организация цикла обучения нейрона
[net, Y, E] = adapt (net, P, T); % адаптация нейрона
linehandle = plotpc(net.IW{1}, net.b{1}, linehandle); % получение управ-
ляющей структуры изображения для новой разделяющей прямой
drawnow; % изображение разделяющей прямой
end; % конец цикла
```

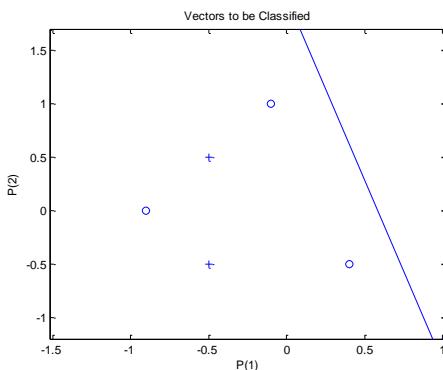


Рисунок 2.10. Изображение невозможности разделения пороговым нейроном линейно
неразделимых векторов

Цикл дает возможность адаптировать нейрон при одном проходе, изобразить линию классификации и остановиться. Нулевая ошибка, а следовательно, и классификация никогда не будет получена. Фундаментальное свойство линейного нейрона – ограничение линейно сепарабельными данными.

10. Процесс обучения линейного нейрона

Для выполнения примера будем использовать функции: *newlin* – для создания нейрона, *train* – для обучения, *sim* – для имитации поведения нейрона. Определим входы нейрона и желаемые выходы:

p = [1.0 -1.4]; % определение входов нейрона

T = [0.5 1.0]; % определение выходов нейрона

Вектор *p* задает два входных элемента. Используем линейный нейрон с одним входом для решения задачи.

w = -1:0.1:1; % установка интервала весов

b = -1:0.1:1; % установка интервала пороговых значений

Вычислить и изобразить ошибку нейрона можно следующим образом:

ES = errsurf(p, T, w, b, 'purelin'); % формирование поверхности ошибки в координатных осях – весов и пороговых значений

plotes(w, b, ES); % изображение поверхности ошибки

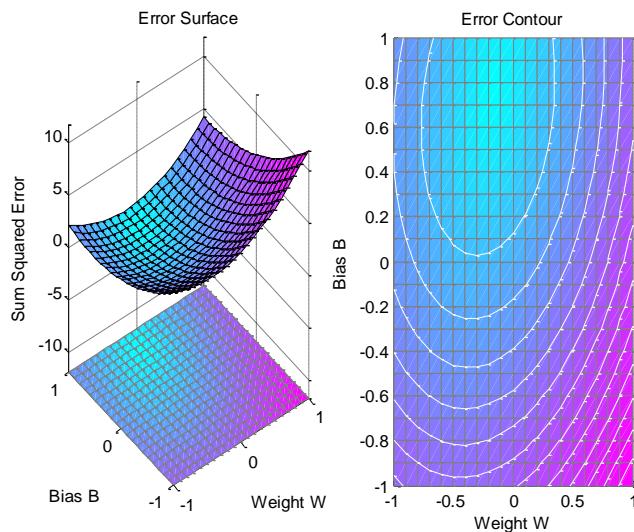


Рисунок 2.11. Изображение поверхности ошибки в координатных осях – весов и пороговых значений

Лучший вес и значение порога – это самые низкие точки на поверхности ошибки. Переопределим уровень обученности и создадим нейрон.

maxlr = 0.20*maxlinlr(p, 'bias'); % возвращение максимального уровня обученности для линейного слоя с bias, для обучения на основе p

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

Функция *maxlinlr* находит значение, соответствующее самому быстрому уровню обученности линейной нейронной сети. Возьмем 20% от вычисленного уровня.

Функция *newlin* содержит четыре параметра:

- матрицу интервалов входов размерностью Rx2;
- количество элементов в выходном векторе;
- вектор задержек входов;
- уровень обучения.

Установим целевое значение ошибки:

net.trainParam.goal = .001; % установка целевого значения ошибки

Получим поверхность ошибки.

ES = errsurf(p, T, w, b, 'purelin'); % формирование поверхности ошибки

plotes(w, b, ES); % изображение поверхности ошибки

subplot (1, 2, 2); % создание 2 координатных осей в графическом окне и установка первых осей активными

Произвольно изменим веса нейрона в установленных интервалах:

net.IW{1, 1} = 0;

net.b{1} = 0.1; % произвольное изменение весов нейрона

[net, tr] = train(net, p, T); % эпоха обучения нейрона, формирование истории обучения tr

Функция *train* выводит на экран историю обучения обученной сети (tr).

График показывает падение ошибки в ходе обучения:

plotperf(tr, net.trainParam.goal); % изображение истории обучения на поверхности ошибки

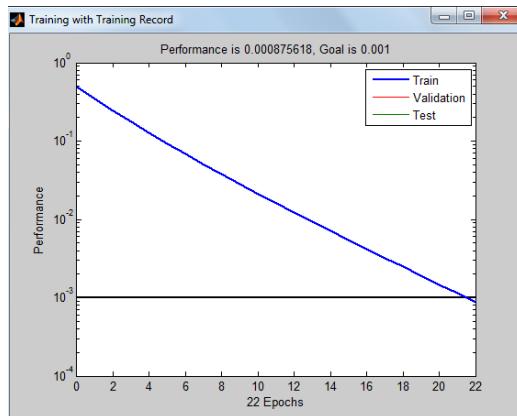


Рисунок 2.12. Изменение среднеквадратической ошибки сети в процессе обучения

p = -1.1;

a = sim(net, p); % формирование выхода обученного нейрона для входа p

Результат очень близок к 1.

Программа работы и методические указания

Изучите возможности нейрона выполнять логические функции двух переменных: И-НЕ (NAND), ИЛИ (OR), ИЛИ-НЕ (NOR), равнозначность (XOR), неравнозначность (NXOR), представленные в таблице 2.1.

Таблица 2.1. Логические функции двух переменных

Входы		Логические функции					
x_2	x_1	AND	NAND	OR	NOR	XOR	NXOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Лабораторная работа № 3

ИЗУЧЕНИЕ МНОГОСЛОЙНОГО НЕЛИНЕЙНОГО ПЕРСЕПТРОНА И АЛГОРИТМА ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Цель работы: изучить возможности многослойного персептрана как универсального аппроксиматора и классификатора.

1. Краткие теоретические сведения

В лабораторной работе рассматривается нейронная сеть с прямой передачей сигнала (с прямой связью), то есть сеть, в которой сигналы передаются только в направлении от входного слоя к выходному, и элементы одного слоя связаны со всеми элементами следующего слоя. Важнейшим для реализации нейронных сетей является определение алгоритма обучения сети [42].

В настоящее время одним из самых эффективных и обоснованных методов обучения нейронных сетей является *алгоритм обратного распространения ошибки*, который применим к *однонаправленным многослойным сетям*. В многослойных нейронных сетях имеется множество скрытых нейронов, входы и выходы которых не являются входами и выходами нейронной сети, а соединяют нейроны внутри сети, то есть *скрытые нейроны*. На рисунке 3.1 показана архитектура нейронной сети с прямой передачей сигнала.

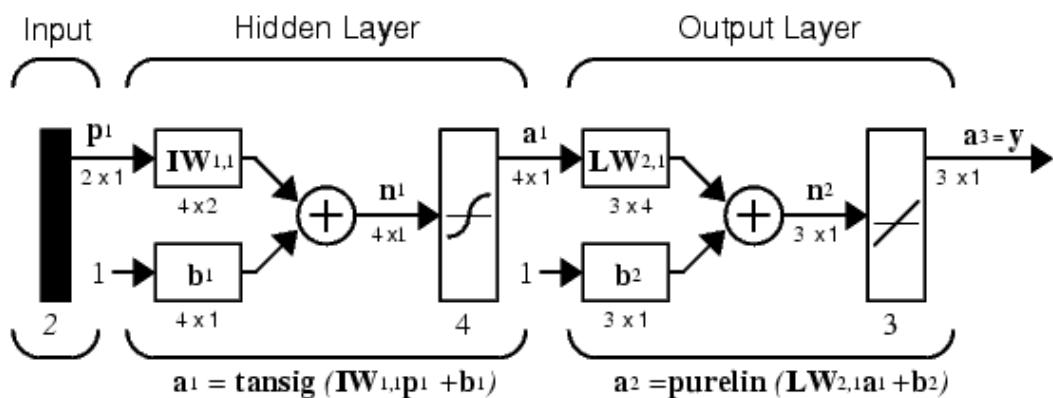


Рисунок 3.1. Схема архитектуры нейронной сети с прямой передачей сигнала

Здесь приняты обозначения: p^i – вектор входа, $IW^{i,j}$, $LW^{i,j}$ – матрицы весов входа и выхода, b^i – смещение, a^i – выход слоя, y – выход сети, *tansig* (гиперболическая тангенциальная), *purelin* (линейная) – соответствующие функции активации.

Веса и смещения определяются с помощью алгоритма обратного распространения ошибок.

Обучение сети обратного распространения требует выполнения следующих операций:

1. Выбрать очередную обучающую пару из обучающего множества; подать входной вектор на вход сети.
2. Вычислить выход сети.
3. Вычислить разность между выходом сети и требуемым выходом (целевым вектором обучающей пары).
4. Скорректировать веса сети так, чтобы минимизировать ошибку.
5. Повторять шаги с 1 по 4 для каждого вектора обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемого уровня.

Функция *newff* создает нейронную сеть прямого распространения сигнала, обучаемую с помощью алгоритма обратного распространения ошибки:

```
net = newff(PR, [S1 S2 SNI], {TF1 TF2 TFNI}, BTF, BLF, PF).
```

Рассмотрим параметры функции *newff*: PR – матрица интервалов значений для R входных элементов, задаваемых минимальным и максимальным значениями; S_i – размер i -го слоя, для N слоев; TF_i – функция активации i -го слоя, по умолчанию используется функция *tansig* – гиперболический тангенс; BTF – функция обучения сети методом обратного распространения ошибки, по умолчанию используется функция *traingdx*; BLF – функция изменения весов при обучении, по умолчанию используется *learngdm*; PF – функция измерения ошибки, по умолчанию *mse*. Функция *newff* возвращает многослойную нейронную сеть прямого и обратного распространения сигнала и ошибки соответ-

ственno. Функции активации могут быть выбраны из следующего перечня: гиперболический тангенс *tansig*; логистическая сигмоида *logsig* или линейная функция *purelin*.

2. Создание и обучение нейронной сети с помощью алгоритма обратного распространения ошибки

Создать нейронную сеть, чтобы обеспечить следующее отображение последовательности входа Р в последовательность целей Т:

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

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

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

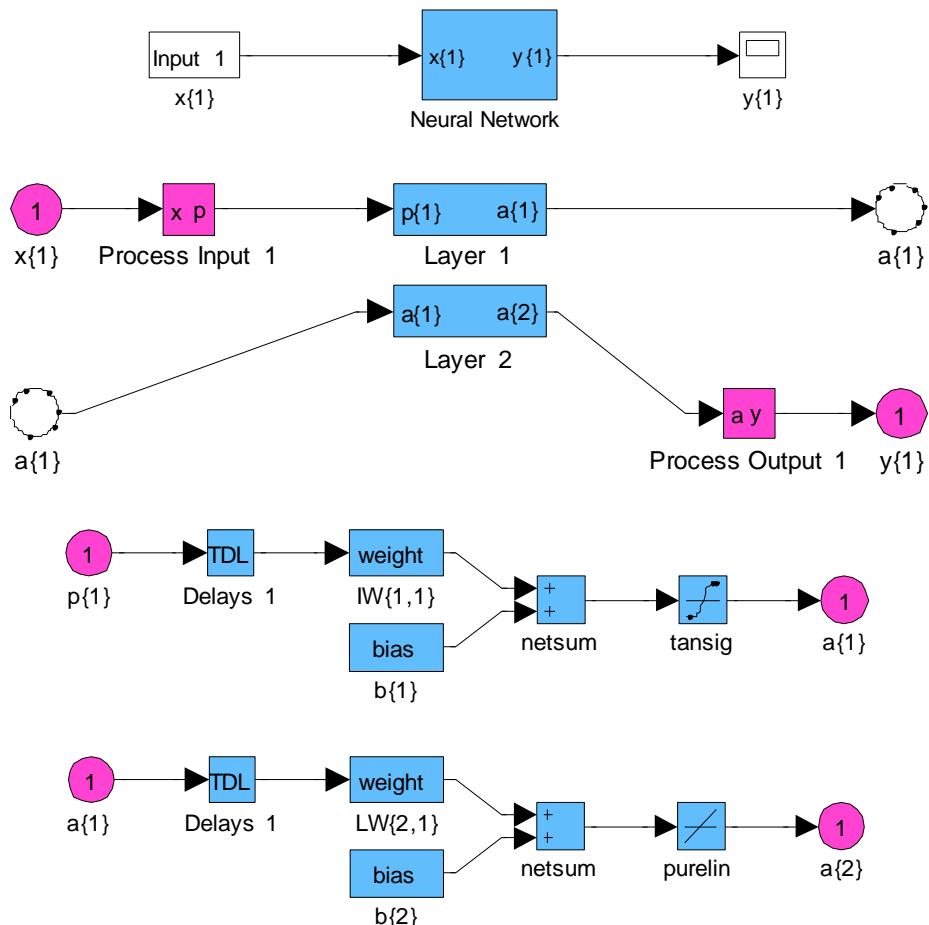


Рисунок 3.2. Структурная схема нейронной сети

Выполним моделирование сети и построим графики сигналов выхода и цели:

```
Y = sim(net,P); figure(1), clf  
plot(P, T, P, Y, 'o'), grid on
```

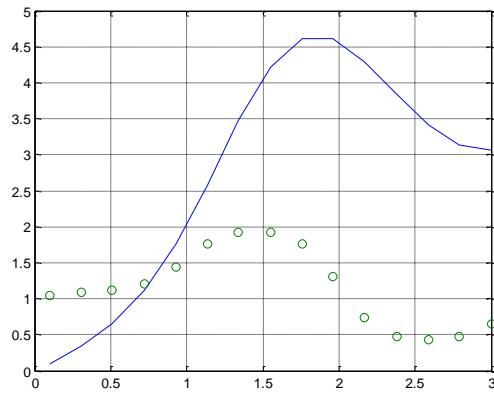


Рисунок 3.3. Графики сигналов выхода и цели

Обучим сеть в течение 50 циклов:

```
net.trainParam.epochs = 50;  
net = train(net, P, T);
```

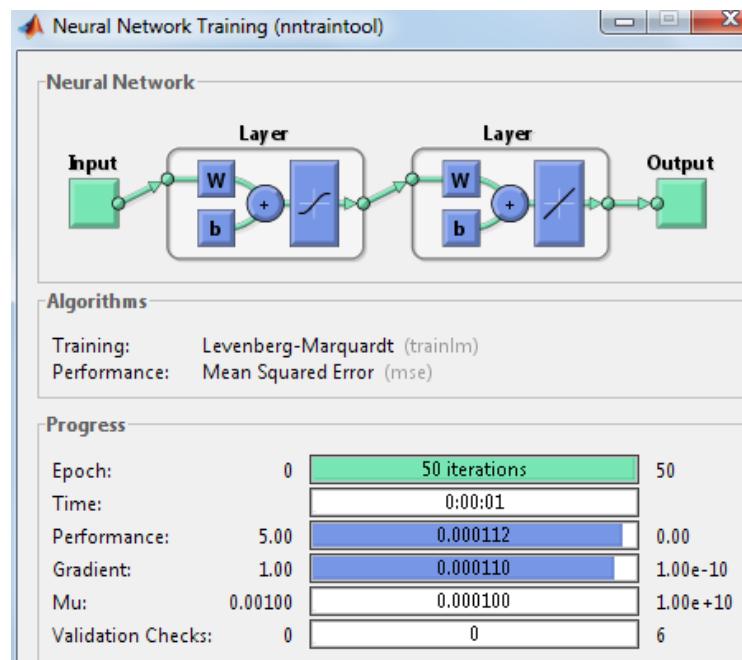


Рисунок 3.4. Окно обучения нейронной сети

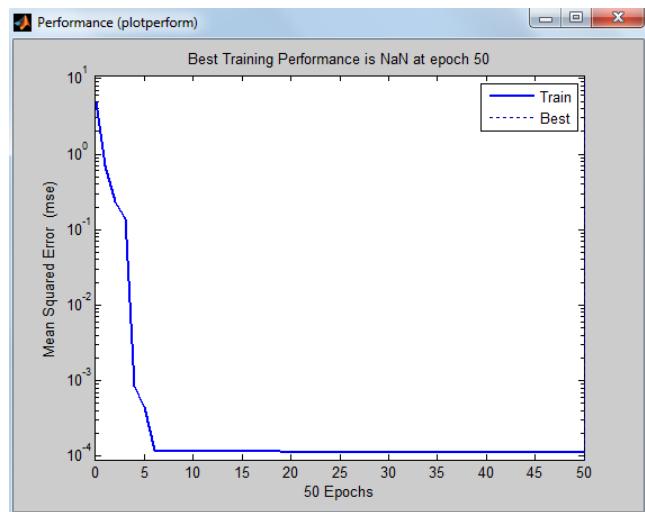


Рисунок 3.5. Изменение ошибки сети в процессе обучения

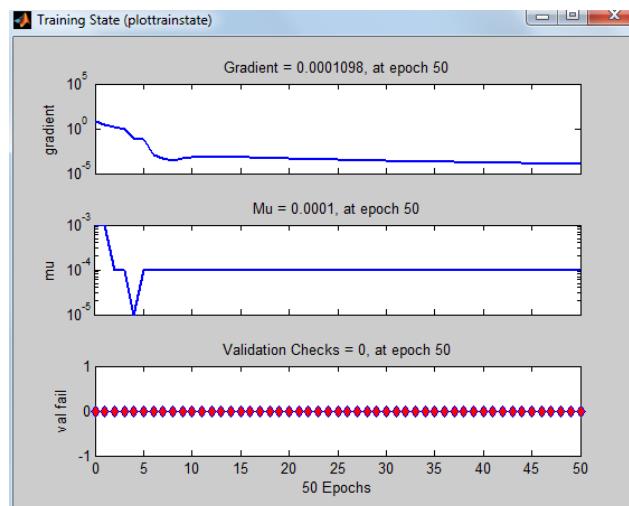


Рисунок 3.6 – Окно состояния обучения

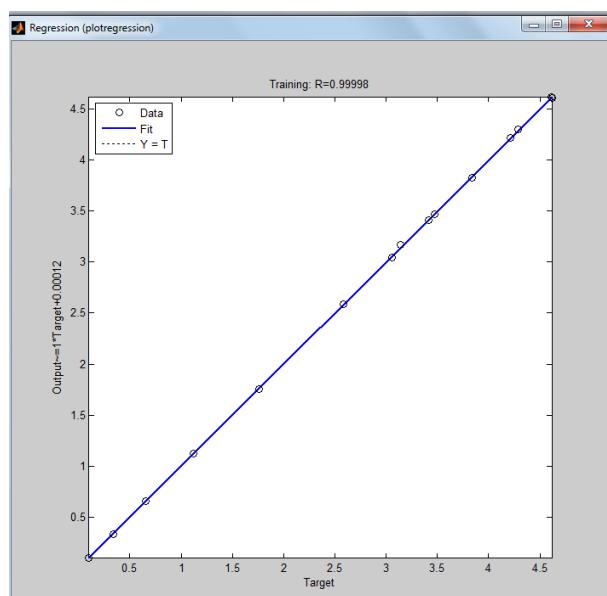


Рисунок 3.7 – Окно линейной регрессии между выходом НС и целями

Выполним моделирование сформированной двухслойной сети, используя обучающую последовательность входа

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

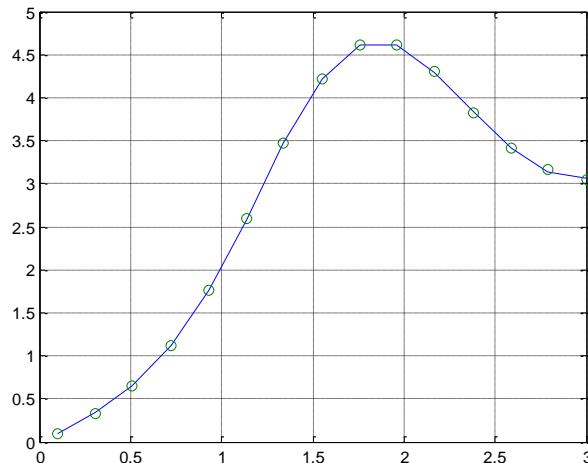


Рисунок 3.8. Результат аппроксимации векторов двухслойным персепtronом

Для исследования работы алгоритма обратного распространения ошибки воспользуемся примером, встроенным в MatLab toolbox, набрав команду *demos*.

В появившемся диалоговом окне необходимо последовательно выбирать пункты меню: *toolbox*, *neural networks*, *Other Neural Network Design textbook demos*, *Table of Contents*, *10–13, Backpropagation Calculation*.

В примере используется двухслойный персепtron с двумя нелинейными нейронами в первом слое и одним во втором. Действие алгоритма обратного распространения ошибки разбито на следующие шаги: назначение входа и желаемого выхода, прямой проход входного сигнала до выхода, обратное распространение ошибки, изменение весов. Переменные, позволяющие проследить работу алгоритма обратного распространения ошибки, обозначены следующим образом:

P – входной сигнал;

$W_1(i)$ – вектор весов первого слоя, $W_1(1)$ – вес связи, передающий входной сигнал на первый нейрон, а $W_1(2)$ – на второй;

$W_2(i)$ – вектор весов второго слоя, $W_2(1)$ – вес связи, передающий входной сигнал с первого нейрона во второй слой, а $W_2(2)$ – со второго;

$B_2(i)$ – вектор пороговых значений (bias) нейронов первого слоя, $i = 1,2$;

B_2 – пороговое значение (bias) нейрона второго слоя;

$N_1(i)$ – вектор выходов первого слоя, $i = 1,2$;

N_2 – выход второго слоя;

$A_1(i)$ – вектор выходных сигналов первого слоя после выполнения функции активации (сигмоиды), $i = 1,2$;

A_2 – выход второго слоя после выполнения функции активации (линейной);

L_r – коэффициент обучаемости.

Пусть входной сигнал $P = 1.0$, а желаемый выход $t = 1+\sin(p*\pi/4)=1,707$ (рисунок 3.9):

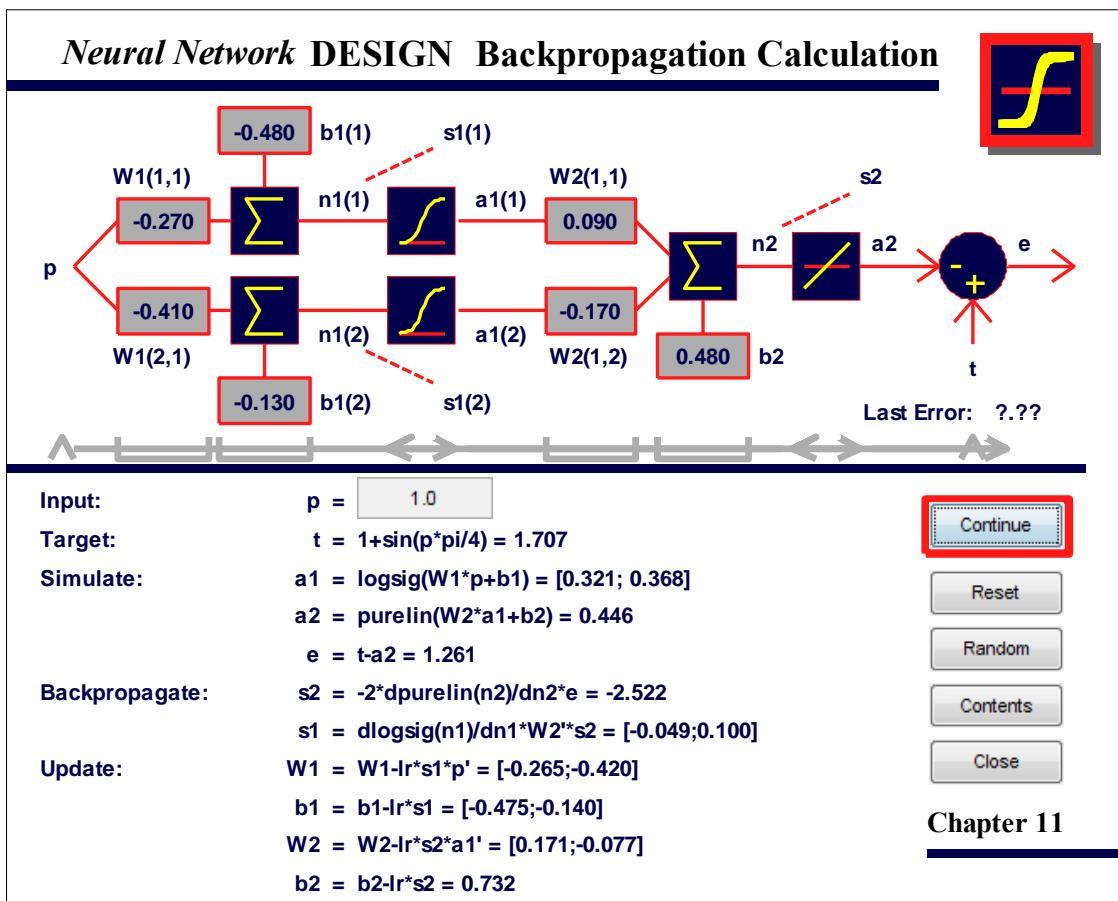


Рисунок 3.9. Demo пример работы алгоритма *Backpropagation*

3. Исследование градиентных алгоритмов обучения нейронных сетей

Алгоритмы обучения, как правило, функционируют пошагово; и эти шаги принято называть *эпохами* или *циклами*. На каждом цикле на вход сети последовательно подаются все элементы обучающей последовательности, затем вычисляются выходные значения сети, сравниваются с целевыми и вычисляется функционал ошибки. Значения функционала, а также его градиента используются для корректировки весов и смещений, после чего все действия повторяются. Начальные значения весов и смещений выбираются случайным образом, а процесс обучения прекращается, когда выполнено определенное количество циклов либо когда ошибка достигнет некоторого малого значения или перестанет уменьшаться [39].

При такой формализации задачи обучения предполагаются известными желаемые (целевые) реакции сети на входные сигналы, что ассоциируется с присутствием учителя, а поэтому такой процесс обучения называют *обучением с учителем*. Для некоторых типов нейронных сетей задание целевого сигнала не требуется, и в этом случае процесс обучения называют *обучением без учителя*.

Алгоритм GD. Алгоритм GD, или алгоритм градиентного спуска, используется для такой корректировки весов и смещений, чтобы минимизировать функционал ошибки, т. е. обеспечить движение по поверхности функционала в направлении, противоположном градиенту функционала по настраиваемым параметрам.

```
net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingd'); % создание двухслойной НС с прямой передачи сигнала с сигмоидальным и линейным слоями для обучения ее на основе метода обратного распространения ошибки
```

```
gensim(net) %
```

Структурная схема модели нейронной сети показана на рис 3.10.

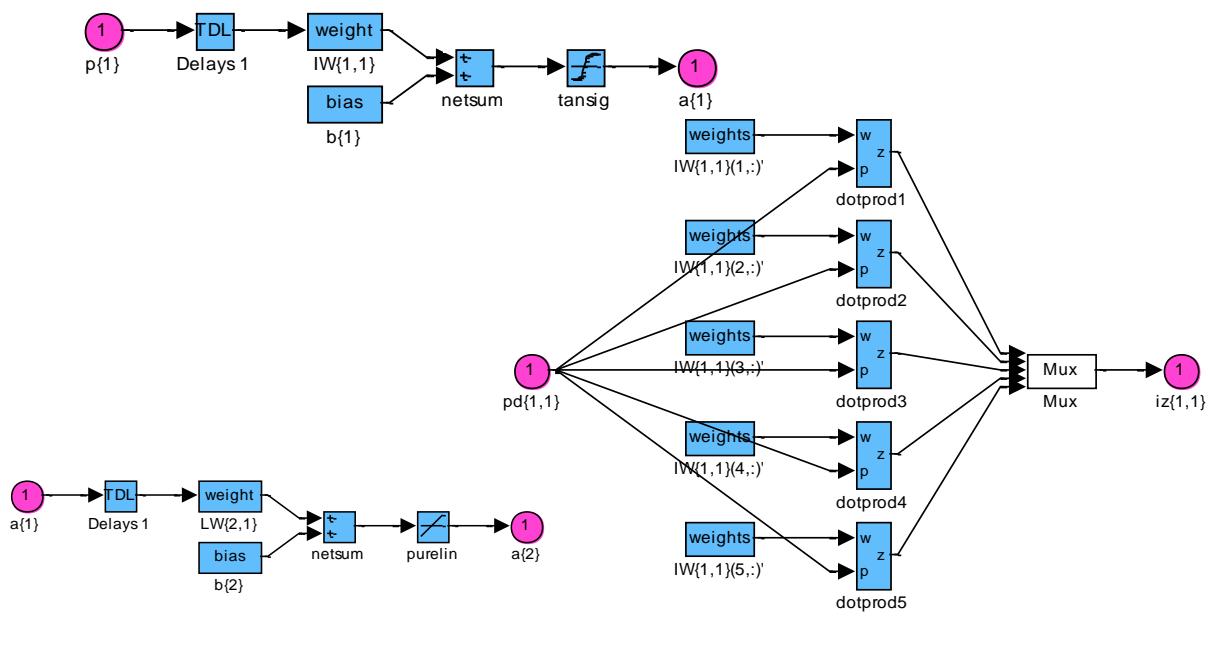
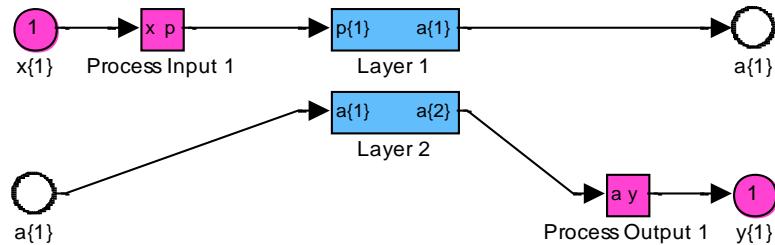
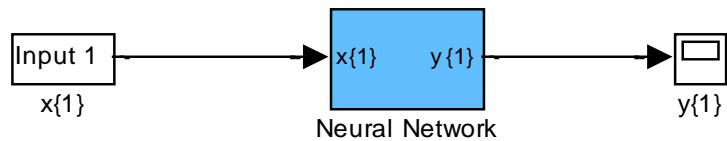


Рисунок 3.10. Структурная схема модели нейронной сети

```
net.biases{1,1}.learnFcn='learngd'; % задание имени функции настройки
learnFcn, соответствующее алгоритму градиентного спуска, в данном случае это M-
функция learngd
```

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

```

net.inputWeights{1,1}.learnFcn='learngd';
net.layerWeights{2,1}.learnParam.lr = 0.2; % установка параметра скорости
настройки
x=-1:0.1:1; % задание множества входов
y=x;
p=[x;y];
p=num2cell(p,1); % поскольку используется последовательный способ обучения,
преобразуем массив входов в массив ячеек
t=sin(x.^2)./cos(y.^2); % задание множества целей
t=num2cell(t,1); % преобразование массива целей в массивы ячеек
net.adaptParam.passes=300; % число проходов при последовательном обуче-
нии
tic, [net, a, e]=adapt(net, p, t); toc % настройка параметров НС, используя
процедуру адаптации
Elapsed time is 18.555497 seconds.
a=sim(net, p) % моделирование НС для проверки качества обучения
a =
[1.2764] [1.1352] [0.8998] [0.5909] [0.2899] [0.0750] [-0.0316] [-0.0387]
[0.0315] [0.1196] [0.1581] [0.1359] [0.0848] [0.0430] [0.0431] [0.1093] [0.2564]
[0.4907] [0.8103] [1.2054] [1.6578]
mse(e) % среднеквадратическая ошибка
ans =
0.0334
[x,y]=meshgrid(x,y);
z=sin(x.^2)./cos(y.^2);
surf(x,y,z) % построение поверхности целевой функции

```

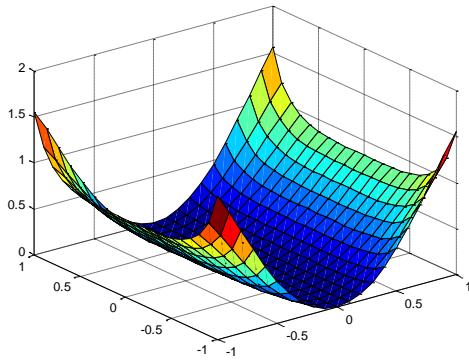


Рисунок 3.11. График поверхности целевой функции

```
a=cat(1,a{:}); % преобразование массива ячеек результата моделирования НС в
массив чисел
```

```
[a]=meshgrid(a);
surf(x,y,a) % построение поверхности функции аппроксимируемой
нейронной сетью
```

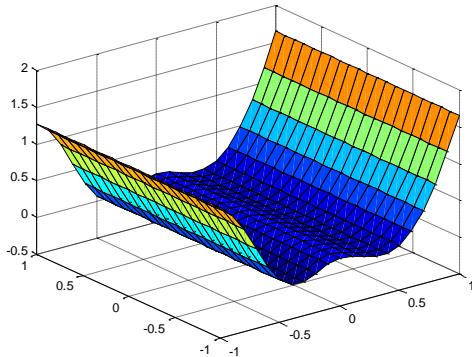


Рисунок 3.12. График поверхности функции аппроксимируемой нейронной сетью

Групповое обучение. Для обучения сети на основе алгоритма GD необходимо использовать М-функцию traingd взамен функции настройки learngd. В этом случае нет необходимости задавать индивидуальные функции обучения для весов и смещений, а достаточно указать одну обучающую функцию для всей сети.

```
net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingd'); % вновь создадим
ту же двухслойную нейронную сеть прямой передачи сигнала с сигмоидальным и ли-
нейным слоями для обучения по методу обратного распространения ошибки
```

```
net.trainParam.show = 50; % show – интервал вывода информации
```

```
net.trainParam.lr = 0.05; % lr – параметр скорости настройки
```

net.trainParam.goal = 1e-005; % goal – предельное значение критерия обучения

```
x=-1:0.1:1;
```

```
y=x;
```

```
p=[x;y];
```

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

```
tic, net=train(net,p,t); toc
```

Elapsed time is 4.796924 seconds.

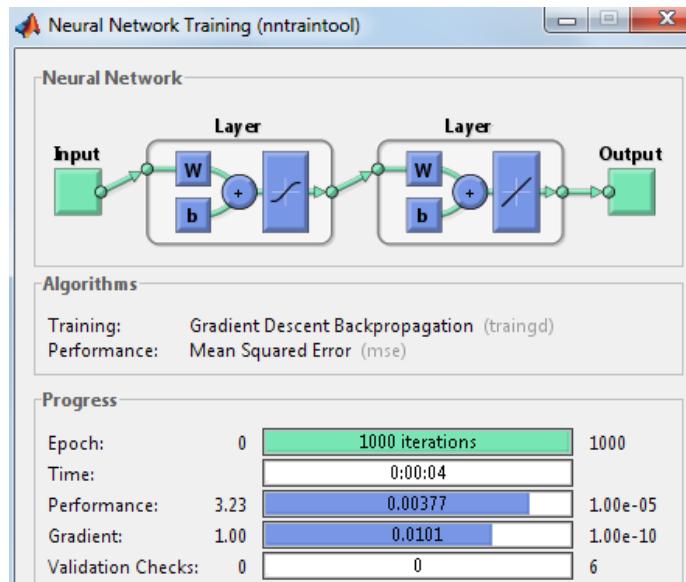


Рисунок 3.13. Окно обучения нейронной сети

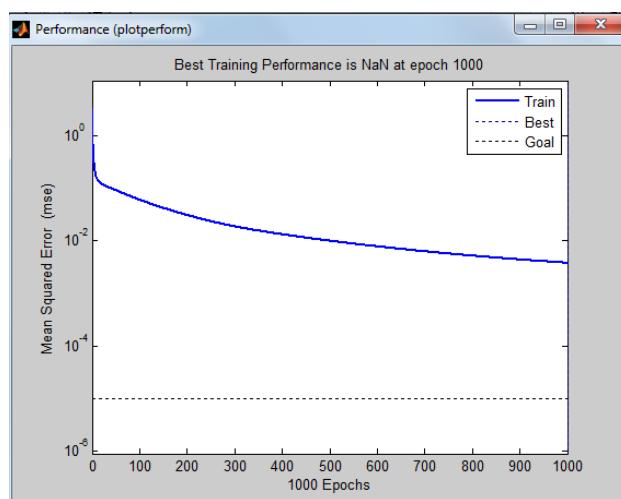


Рисунок 3.14. График изменения ошибки в зависимости от числа выполненных циклов обучения

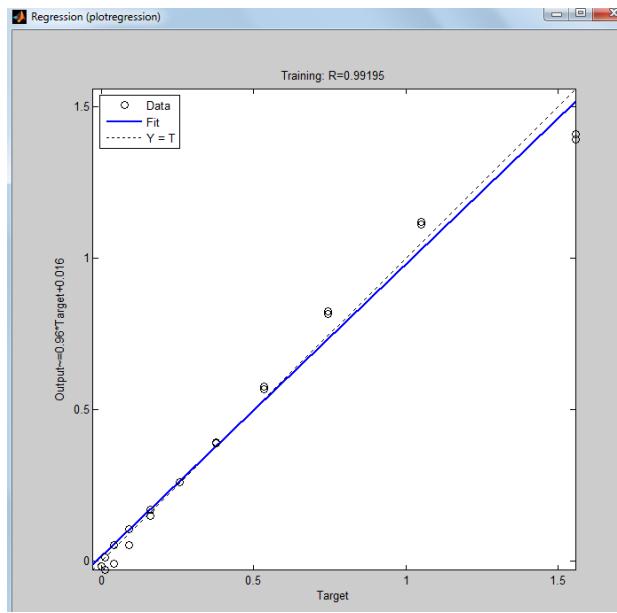


Рисунок 3.15. Окно линейной регрессии между выходом НС и целями

a=sim(net, p)

a =

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

e=t-a

```
0.1487 -0.0695 -0.0723 -0.0333 -0.0132 -0.0052 0.0126 0.0378 0.0488 0.0381
0.0165 -0.0032 -0.0142 -0.0155 -0.0105 -0.0068 -0.0151 -0.0431 -0.0786 -0.0601
0.1657
```

mse(e)

ans =

```
0.0038
```

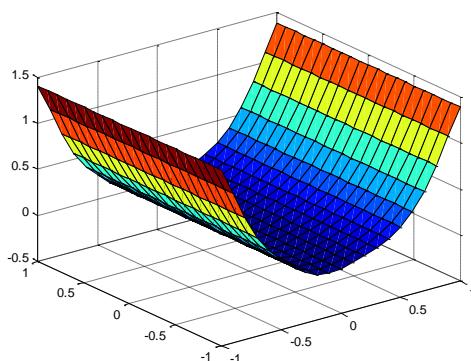


Рисунок 3.16. График поверхности функции аппроксимируемой нейронной сетью

Более тщательно ознакомиться с методом градиентного спуска можно с помощью демонстрационной программы nnd12sd1, которая иллюстрирует работу алгоритма GD.

Алгоритм GDM. Алгоритм GDM, или алгоритм градиентного спуска с возмущением, предназначен для настройки и обучения сетей прямой передачи. Этот алгоритм позволяет преодолевать локальные неровности поверхности ошибки и не останавливаться в локальных минимумах. С учетом возмущения метод обратного распространения ошибки реализует следующее соотношение для приращения вектора настраиваемых параметров:

$$\Delta w_k = mc\Delta w_{k-1} + (1 - mc)lrq_k, \quad (3.1)$$

где Δw_k – приращение вектора весов; mc – параметр возмущения; lr – параметр скорости обучения; g_k – вектор градиента функционала ошибки на k -й итерации.

Если параметр возмущения равен 0, то изменение вектора настраиваемых параметров определяется только градиентом, если параметр равен 1, то текущее приращение равно предшествующему как по величине, так и по направлению.

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

```
net.biases{1,1}.learnFcn='learngdm'; % задание имени функции настройки learnFcn, соответствующее алгоритму градиентного спуска с возмущением – M-функция leargdm
```

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

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

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

```
net.layerWeights{2,1}.learnParam.lr=0.2; % увеличим значение параметра скорости обучения до 0.2
```

```
x=-1:0.1:1;
```

```
y=x;
```

```
p=[x;y];
```

```
p=num2cell(p,1);
```

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

```
t=num2cell(t,1);
```

```

net.adaptParam.passes=300; % значение проходов
tic, [net,a,e]=adapt(net,p,t); toc
Elapsed time is 19.187137 seconds.

a
a =
[0.8683] [1.0634] [1.2363] [1.2927] [1.1903] [0.9339] [0.5665] [0.1546]
[-0.2283] [-0.5175] [-0.6697] [-0.6692] [-0.5286] [-0.2830] [0.0194] [0.3291]
[0.6065] [0.8297] [0.9997] [1.1413] [1.3017]

mse(e)
ans =
0.2196

```

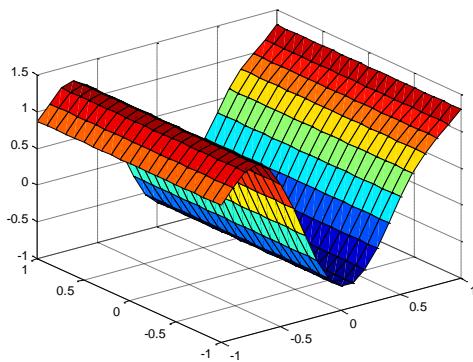


Рисунок 3.17. График поверхности функции аппроксимируемой нейронной сетью

Результаты аппроксимации оказались хуже результатов работы НС с алгоритмом обучения GD и намного дольше.

Групповое обучение. Альтернативой последовательной адаптации является групповое обучение, которое основано на применении функции `train`. В этом режиме параметры сети модифицируются только после того, как реализовано все обучающее множество, и градиенты, рассчитанные для каждого элемента множества, суммируются, чтобы определить приращения настраиваемых параметров.

Для обучения сети на основе алгоритма GDM необходимо использовать M-функцию `traingdm` взамен функции настройки `learningdm`. Различие этих двух функций состоит в следующем. Алгоритм функции `traingdm` суммирует градиенты, рассчитанные на каждом цикле обучения, а параметры модифицируются

только после того, как все обучающие данные будут представлены. Если проведено N циклов обучения и для функционала ошибки оказалось выполненным условие $J_N \geq 1.04J_{N-1}$, то параметр возмущения mc следует установить в 0.

```

net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingdm');
net.trainParam.epochs=1000;
net.trainParam.goal=1e-5;
net.trainParam.lr=0.05;
net.trainParam.mc =0.9; % по сравнению с функцией traingd здесь добавляется только 1 параметр возмущения mc
net.trainParam.show=50;
x=-1:0.1:1;
y=x;
p=[x;y];
t=sin(x.^2)./cos(y.^2);
tic, net=train(net,p,t); toc
Elapsed time is 5.533868 seconds.

```

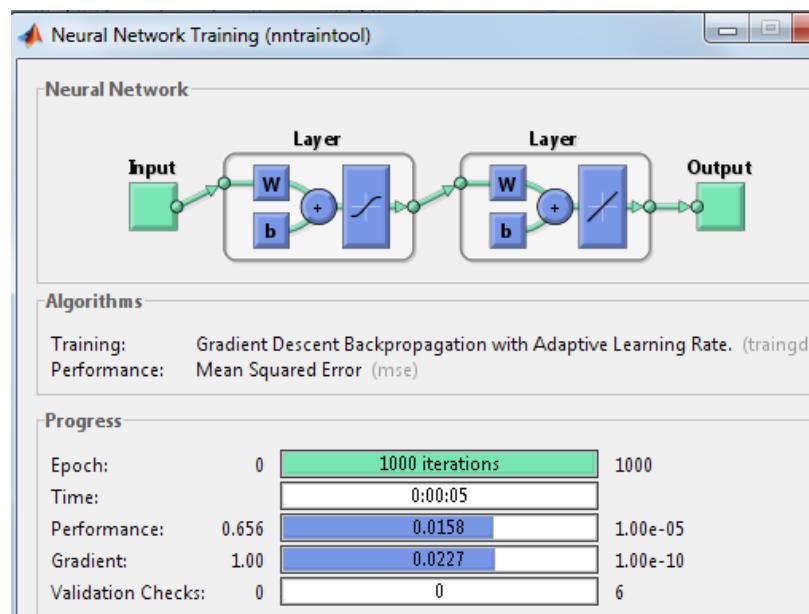


Рисунок 3.18. Окно обучения нейронной сети

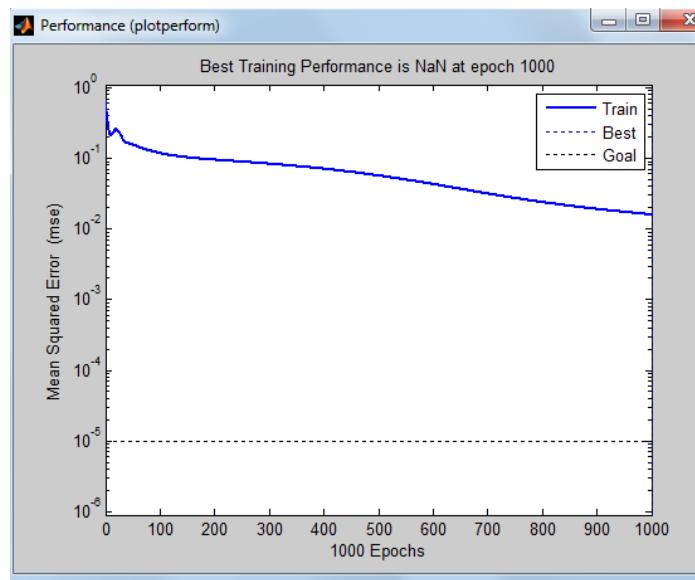


Рисунок 3.19. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

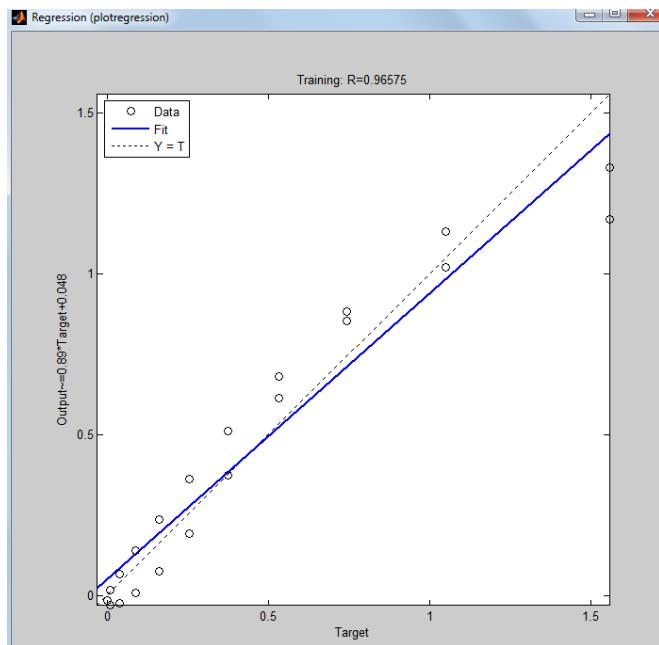


Рисунок 3.20. Окно линейной регрессии между выходом НС и целями

```
a=sim(net,p);
```

```
e=t-a;
```

```
mse(e)
```

```
ans =
```

```
0.0158
```

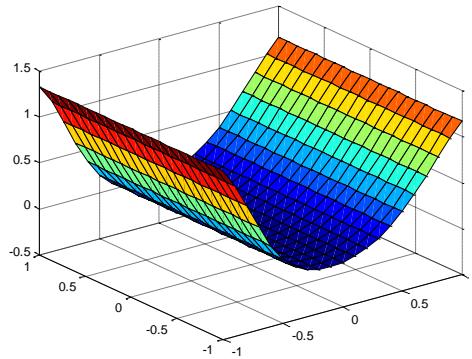


Рисунок 3.21. График поверхности функции аппроксимируемой нейронной сетью

Более тщательно ознакомиться с методом градиентного спуска с возмущением можно с помощью демонстрационной программы nnd12mo, которая иллюстрирует работу алгоритма GDM.

Практика применения описанных выше алгоритмов градиентного спуска показывает, что эти алгоритмы слишком медленны для решения реальных задач. Ниже обсуждаются алгоритмы группового обучения, которые сходятся в десятки и сотни раз быстрее. Ниже представлены 2 разновидности таких алгоритмов: один основан на стратегии выбора параметра скорости настройки и реализован в виде алгоритма GDA, другой – на стратегии выбора шага с помощью порогового алгоритма обратного распространения ошибки и реализован в виде алгоритма Rprop.

Алгоритм GDA. Алгоритм GDA, или алгоритм градиентного спуска с выбором параметра скорости настройки, использует эвристическую стратегию изменения этого параметра в процессе обучения.

Эта стратегия заключается в следующем. Вычисляются выход и погрешность инициализированной нейронной сети. Затем на каждом цикле обучения вычисляются новые значения настраиваемых параметров и новые значения выходов и погрешностей. Если отношение нового значения погрешности к прежнему превышает величину max_perf_inc (по умолчанию 1.04), то новые значения настраиваемых параметров во внимание не принимаются. При этом параметр скорости настройки уменьшается с коэффициентом lr_dec (по умолчанию 0.7). Если новая погрешность меньше

прежней, то параметр скорости настройки увеличивается с коэффициентом lr_inc (по умолчанию 1.05).

Эта стратегия способствует увеличению скорости и сокращению длительности обучения.

Функция traingda характеризуется следующими параметрами, заданными по умолчанию:

net.trainParam

show: 50

showWindow: 1

showCommandLine: 0

epochs: 300

time: Inf

goal: 1.0000e-005

max_fail: 6

lr: 0.0500

lr_inc: 1.0500

lr_dec: 0.7000

max_perf_inc: 1.0400

min_grad: 1.0000e-010

mc: 0.9000

Алгоритм GDA в сочетании с алгоритмом GD определяет функцию обучения traingda, а в сочетании с алгоритмом GDM - функцию обучения traingdx.

Вновь обратимся к той же нейронной сети, но будем использовать функцию обучения traingda:

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

```

x=-1:0.1:1;
y=x;
p=[x;y];
t=sin(x.^2)./cos(y.^2);
tic, net=train(net,p,t); toc

```

Elapsed time is 1.706964 seconds.

На рисунке 3.22 приведен график изменения ошибки обучения в зависимости от числа выполненных циклов.

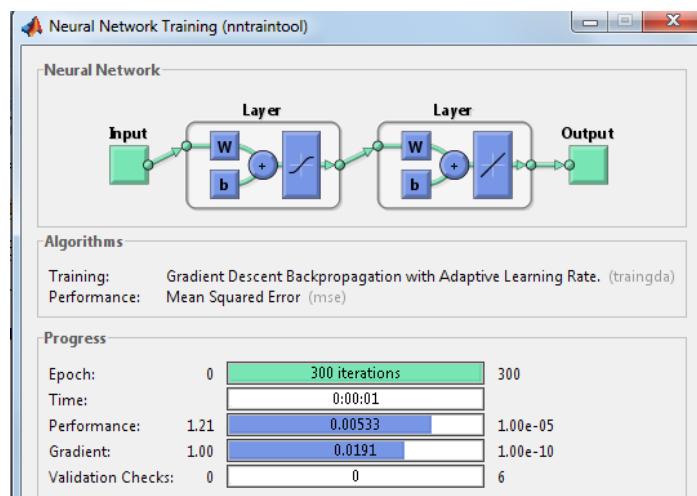


Рисунок 3.22. Окно обучения нейронной сети

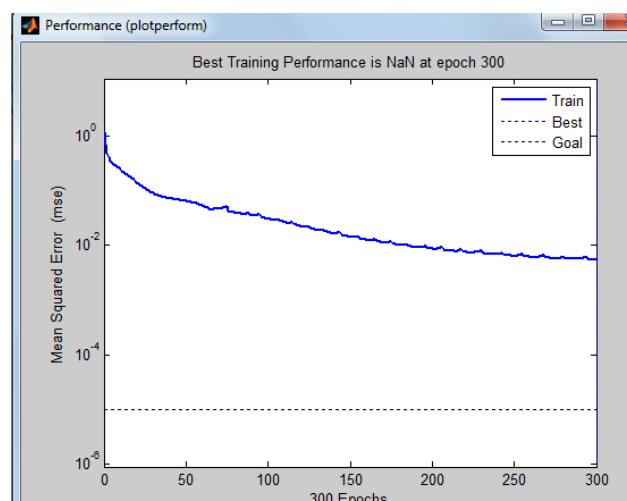


Рисунок 3.23. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

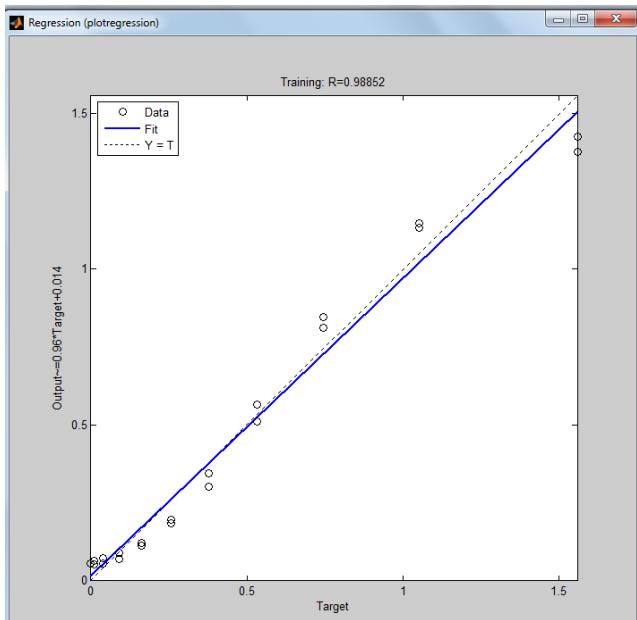


Рисунок 3.24. Окно линейной регрессии между выходом НС и целями

```
a=sim(net,p);
```

```
e=t-a;
```

```
mse(e)
```

```
0.0053
```

Нетрудно заметить, что количество циклов обучения по сравнению с предыдущим примером сократилось практически в 3 раза при уменьшении погрешности обучения.

Демонстрационная программа nnd12vl иллюстрирует производительность алгоритма с переменным параметром скорости настройки.

Алгоритм Rprop. Алгоритм Rprop, или пороговый алгоритм обратного распространения ошибки, реализует следующую эвристическую стратегию изменения шага приращения параметров для многослойных нейронных сетей.

Многослойные сети обычно используют сигмоидальные функции активации в скрытых слоях. Эти функции относятся к классу функций со сжимающим отображением, поскольку они отображают бесконечный диапазон значений аргумента в конечный диапазон значений функции. Сигмоидальные функции характеризуются тем, что их наклон приближается к нулю, когда значения входа нейрона существенно возрастают. Следствием этого является то, что при использовании метода наискорейшего

спуска величина градиента становится малой и приводит к малым изменениям настраиваемых параметров, даже если они далеки от оптимальных значений.

Цель порогового алгоритма обратного распространения ошибки Rprop (Resilient propagation) состоит в том, чтобы повысить чувствительность метода при больших значениях входа функции активации. В этом случае вместо значений самих производных используется только их знак.

Значение приращения для каждого настраиваемого параметра увеличивается с коэффициентом `delt_inc` (по умолчанию 1.2) всякий раз, когда производная функционала ошибки по данному параметру сохраняет знак для двух последовательных итераций. Значение приращения уменьшается с коэффициентом `delt_dec` (по умолчанию 0.5) всякий раз, когда производная функционала ошибки по данному параметру изменяет знак по сравнению с предыдущей итерацией. Если производная равна 0, то приращение остаётся неизменным. Поскольку по умолчанию коэффициент увеличения приращения составляет 20%, а коэффициент уменьшения – 50%, то в случае попеременного увеличения и уменьшения общая тенденция будет направлена на уменьшение шага изменения параметра. Если параметр от итерации к итерации изменяется в одном направлении, то шаг изменения будет постоянно возрастать.

Алгоритм Rprop определяет функцию обучения `trainrp`.

Функция `trainrp` характеризуется следующими параметрами, заданными по умолчанию:

net.trainParam

```
ans =  
show: 25  
showWindow: 1  
showCommandLine: 0  
epoches: 1000  
time: Inf  
goal: 0  
max_fail: 6  
min_grad: 1.0000e-010
```

```
delt_inc: 1.2000  
delt_dec: 0.5000  
delta0: 0.0700  
deltamax: 50
```

Здесь epochs – максимальное количество циклов обучения; show – интервал вывода информации, измеренный в циклах; goal – предельное значение критерия обучения; time – предельное время обучения; min_grad – минимальное значение градиента; max_fail – максимально допустимый уровень превышения ошибки контрольного подмножества по сравнению с обучающим; delt_inc – коэффициент увеличения шага настройки; delt_dec – коэффициент уменьшения шага настройки; delta0 – начальное значение шага настройки; deltamax – максимальное значение шага настройки.

Вновь обратимся к сети, показанной на рисунке 3.10, но будем использовать функцию обучения trainrp:

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

Установим следующие значения этих параметров:

```
net.trainParam.show = 10;  
net.trainParam.epochs = 300;  
net.trainParam.goal = 1e-5;  
x=-1:0.1:1;  
y=x;  
p=[x;y];  
t=sin(x.^2)./cos(y.^2);  
tic, net=train(net,p,t); toc
```

Elapsed time is 1.928695 seconds.

На рисунке 3.26 приведен график изменения ошибки обучения в зависимости от числа выполненных циклов обучения.

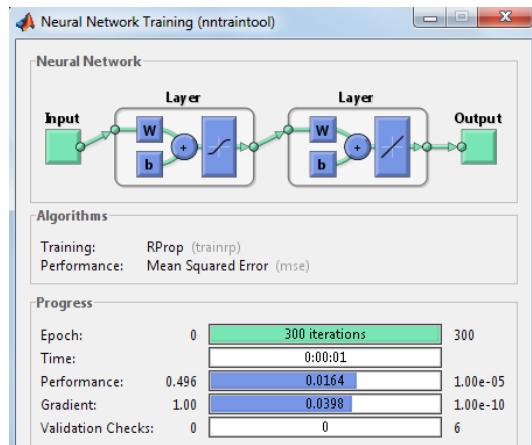


Рисунок 3.26. Окно обучения нейронной сети

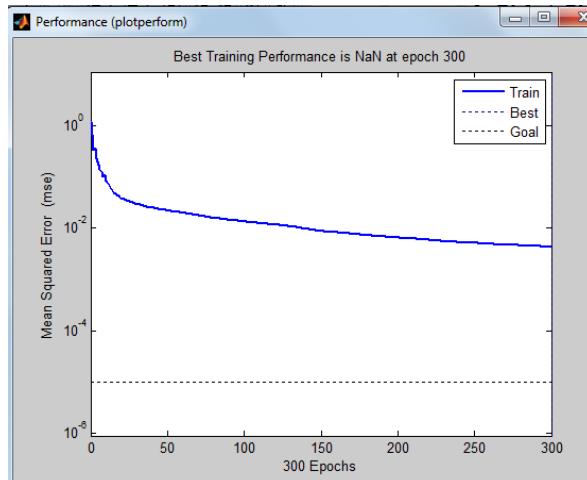


Рисунок 3.26. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

```
a=sim(net,p);
```

```
e=t-a;
```

```
 mse(e)
```

```
0.0043
```

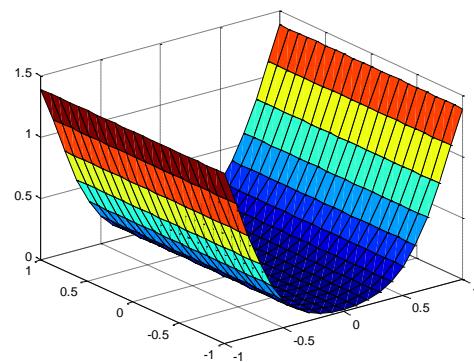


Рисунок 3.27. График поверхности функции аппроксимируемой нейронной сетью

Нетрудно заметить, что количество циклов обучения по сравнению с алгоритмом GDA сократилось практически еще в 3 раза и составило по отношению к алгоритму GD значение, близкое к 9.

Программа работы и методические указания

При заданном преподавателем значении p , рассчитайте в командном окне MatLab прямое распространение входного сигнала, обратное распространение ошибки и изменение весов на первой итерации алгоритма *Backpropagation*. Сравните с результатами, полученными в *Backpropagation Calculation*.

Контрольные вопросы

1. Каким алгоритмом обучаются многослойные нейронные сети?
2. Из каких основных этапов состоит алгоритм обратного распространения ошибки?
3. Почему алгоритм обратного распространения ошибки относится к классу алгоритмов градиентного спуска?
4. Как влияет функция принадлежности на правило изменения весов в обратном алгоритме распространения ошибки?

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- расчеты и выводы по работе.

Лабораторная работа №4

ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ АППРОКСИМАЦИИ ФУНКЦИЙ И ПРЕДСКАЗАНИЯ ВРЕМЕННОГО ПРОЦЕССА

Цель работы: выработать практические навыки при исследовании возможностей применения нейронных сетей для решения прикладных задач при помощи пакета прикладных программ MatLab.

1. Применение нейронных сетей для аппроксимации функций

1.1 Пусть необходимо аппроксимировать полином вида $Y = x_1^2 - x_2^2 + x_1x_2$

Для формирования вектора обучающих примеров в окне команд необходимо набрать следующие команды:

```
x = 1;  
for i = 0:.1:1;  
    for j = 0:.1:1;  
        D(1,x) = i;  
        D(2,x) = j;  
        T(x) = i*j+i*i-j*j;  
        x = x+1;  
    end;  
end;
```

После чего будут сформированы матрица входных данных D и вектор целей T. Импортируем данные из рабочей области: вектор целей T и матрицу входных данных D (рисунок 4.1). Построим при помощи инструментального средства *Neural Network Tools Network/Data Manager* модель нейронной сети. Для этого необходимо выбрать сеть с обратным распространением ошибки (*Feed-forward backprop*), параметры изменения входных сигналов необходимо выбрать из матрицы D, количество нейронов в первом слое 5, во втором 1, функция активации нейронов первого слоя – любая нелинейная, второго – линейная (рисунок 4.3).

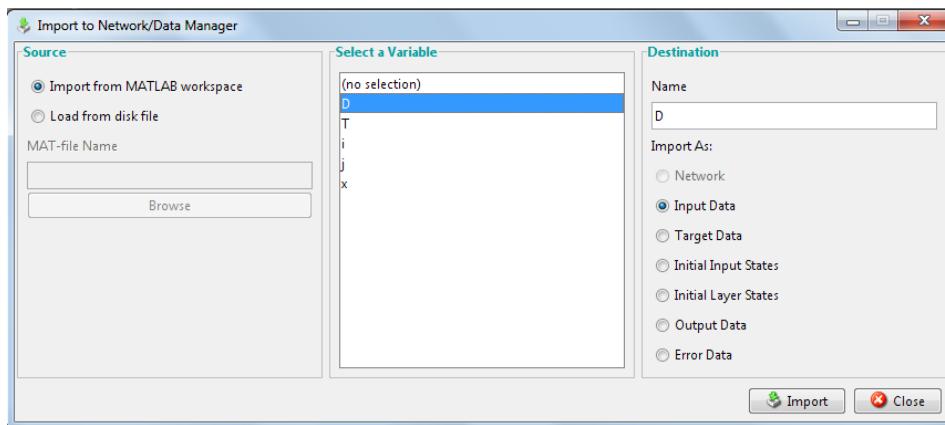


Рисунок 4.1. Окно для импорта и загрузки данных

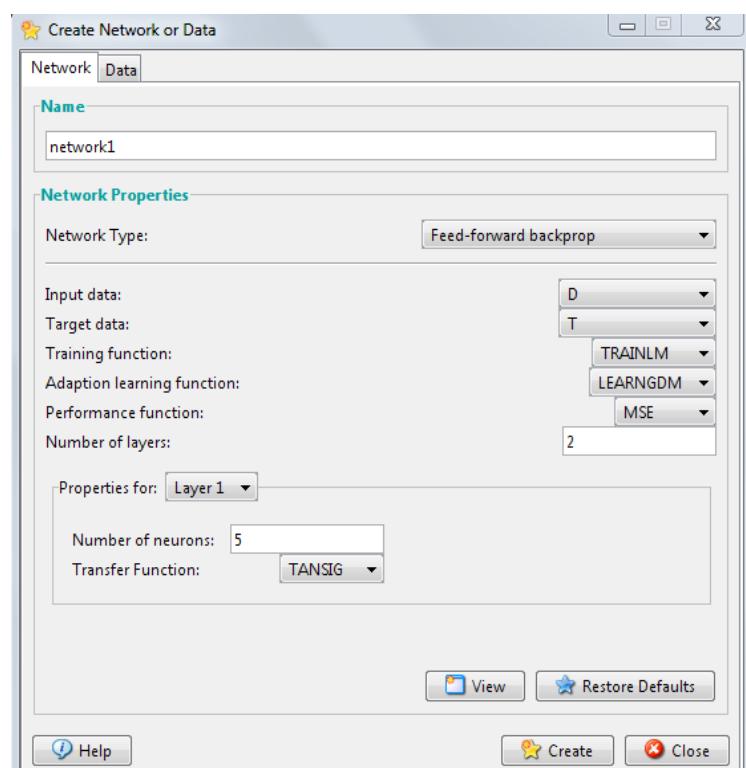


Рисунок 4.2. Окно создания нейронной сети

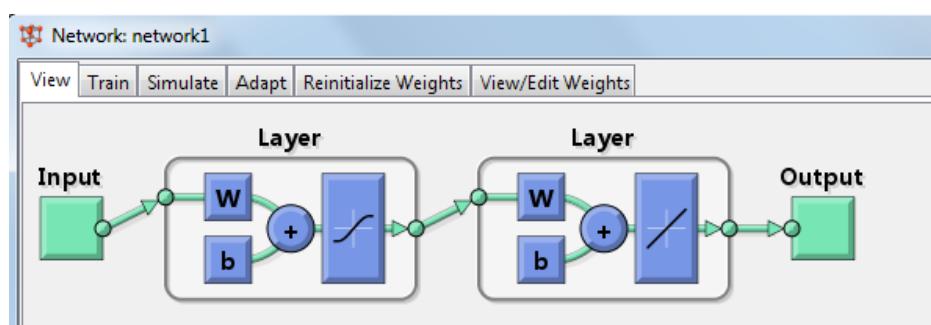


Рисунок 4.3. Диалоговая панель Network

Обучим сеть, при этом в качестве входных данных необходимо использовать матрицу D, а в качестве цели вектор T (рисунок 4.4).

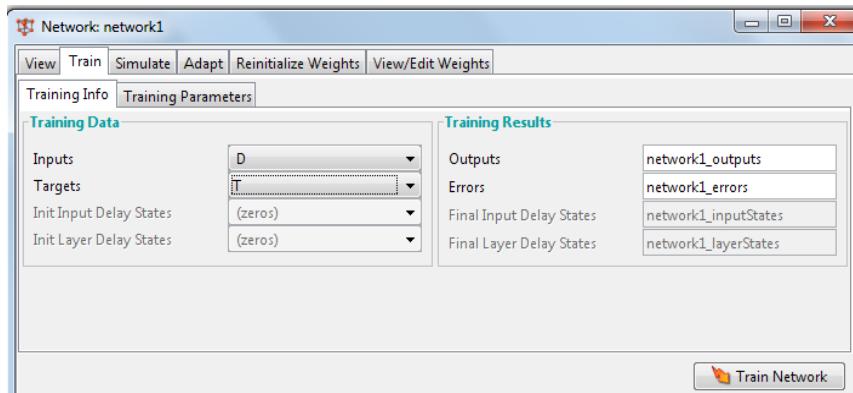


Рисунок 4.4. Окно информации об обучающих последовательностях

Процесс обучения сети представлен на рисунке 5. Сеть обучена за 138 итераций до ошибки 10^{-5} .

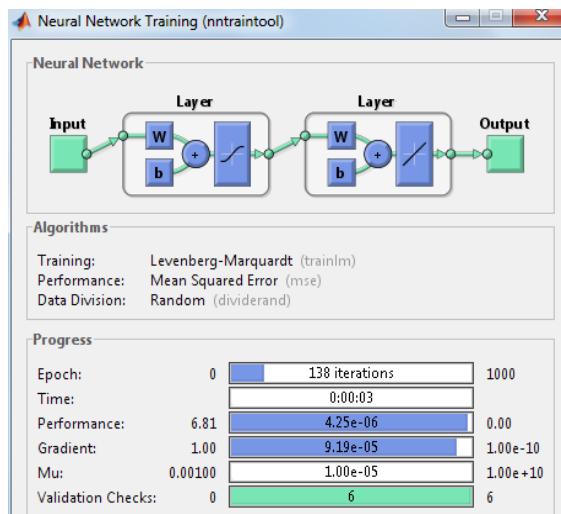


Рисунок 4.5. Окно обучения нейронной сети

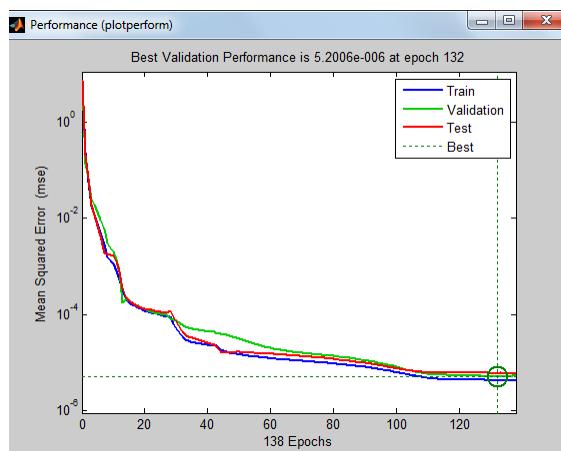


Рисунок 4.6. Изменение ошибки сети в процессе обучения

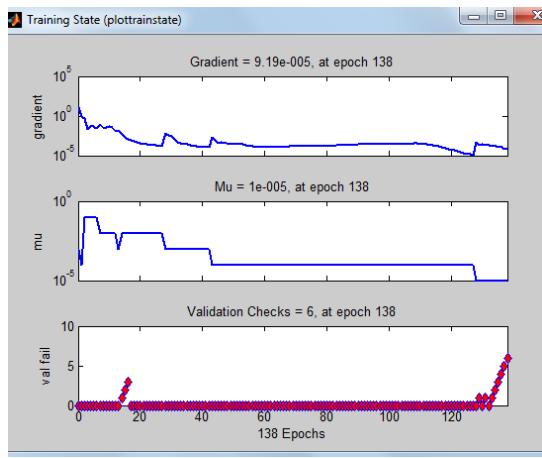


Рисунок 4.7. Окно состояния обучения

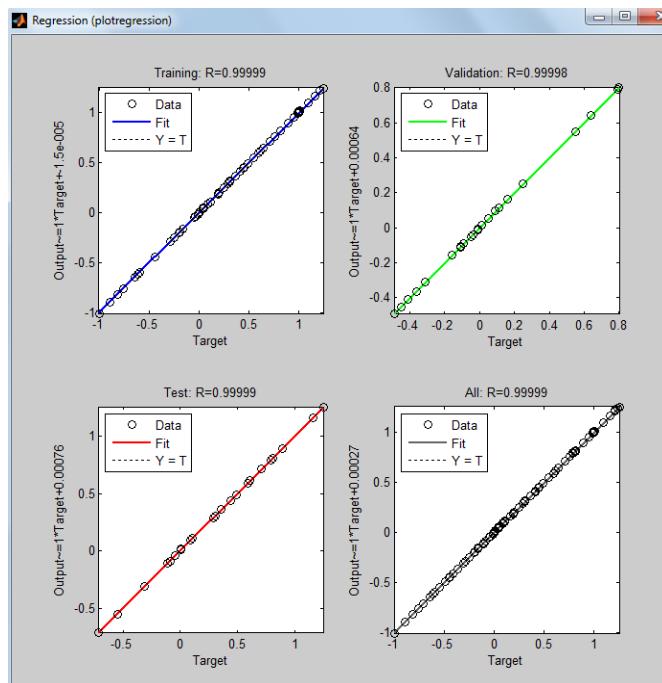


Рисунок 4.8. Окно линейной регрессии между выходом НС и целями

Обученную сеть симулируем вектором данных, получив выход сети network2_outputs на основе входных данных матрицы D (рисунок 4.9).

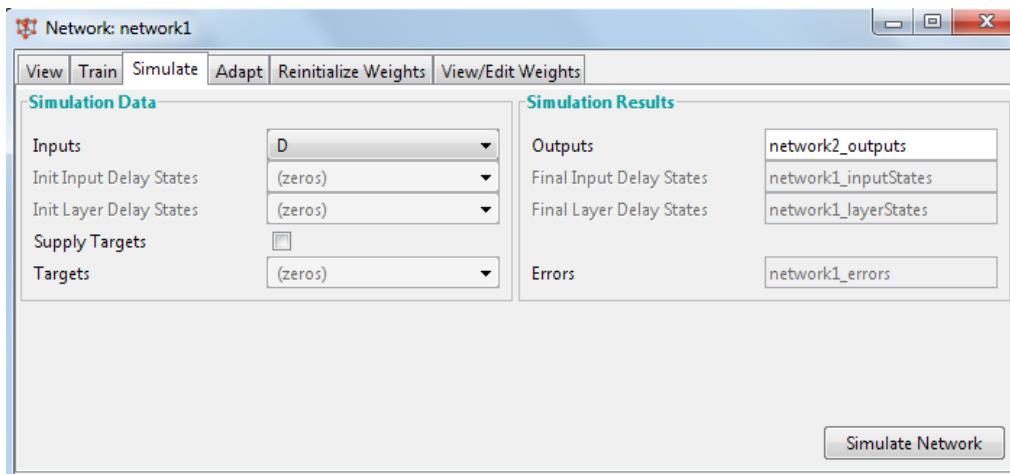


Рисунок 4.9. Окно симуляции нейронной сети

После этого необходимо экспортить вектор `network2_outputs` в рабочую область системы MatLab (рисунок 4.10).

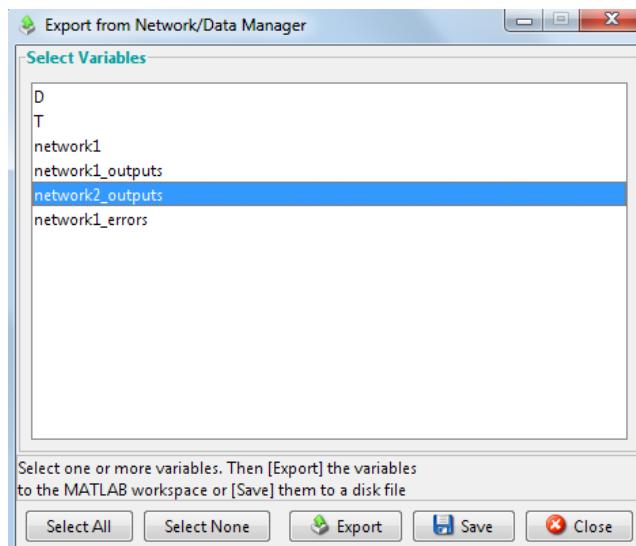


Рисунок 4.10. Окно для экспорта или записи данных в файл

Абсолютную ошибку сети можно найти, получив разность выхода обученной сети при симуляции `network2_outputs` и цели `T` в рабочей области:

error = network2_outputs-T;

График ошибки представлен на рисунке 4.11.

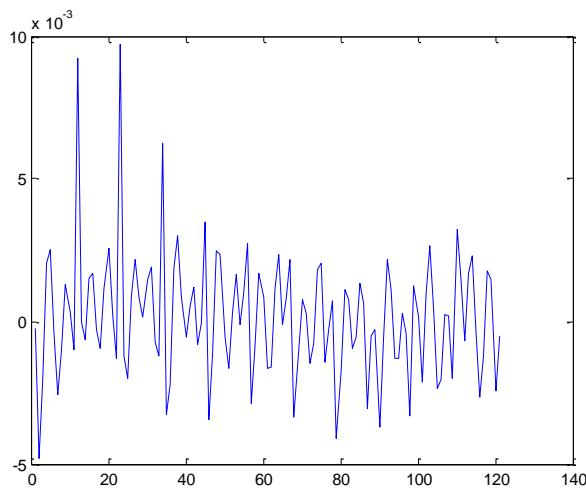


Рисунок 4.11. График ошибки нейронной сети

1.2 Аппроксимация нелинейной функции

Одним из самых замечательных свойств нейронных сетей является способность аппроксимировать и, более того, быть универсальными аппроксиматорами. Сказанное означает, что с помощью нейронных цепей можно аппроксимировать сколь угодно точно непрерывные функции многих переменных. Рассмотрим пример.

Необходимо выполнить аппроксимацию функции следующего вида

$$y = \sin\left(\frac{5 \cdot \pi \cdot x}{N} + \sin\left(\frac{7 \cdot \pi \cdot x}{N}\right)\right),$$

где $x \in 1 \div N$, а N - число точек функции.

Эта кривая представляет собой отрезок периодического колебания с частотой $5\pi/N$, модулированного по фазе гармоническим колебанием с частотой $7\pi N$ (рисунок 4.12).

x = 1:100;

y = sin(5*pi*[1:100]/100+sin(7*pi*[1:100]/100));

plot(x, y)

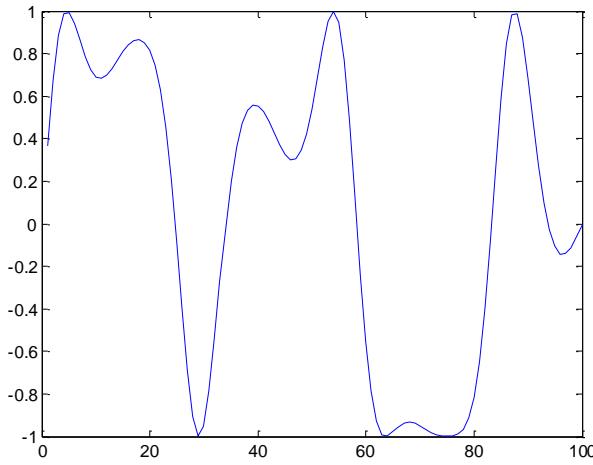


Рисунок 4.12. График функции $y = \sin(5\pi[1:100]/100) + \sin(7\pi[1:100]/100)$

Импортируем входные и целевые данные в nntool (рисунок 4.13)

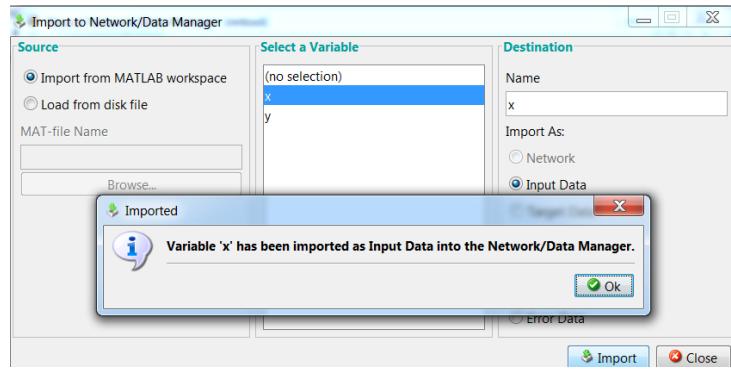


Рисунок 4.13. Окно для импорта и загрузки данных

Выберем персепtron (*Feed-Forward Back Propagation*) с тринадцатью сигмоидными (*TANSIG*) нейронами скрытого слоя и одним линейным (*PURELIN*) нейроном выходного слоя. Обучение будем производить, используя алгоритм Левенберга-Маркардта (*Levenberg-Marquardt*), который реализует функция *TRAINLM*. Функция ошибки – *MSE*.

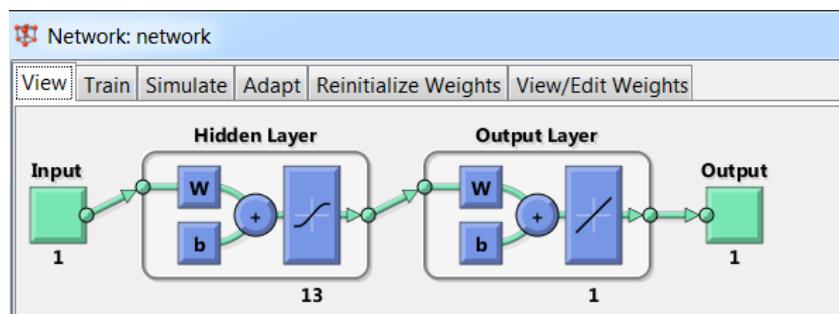


Рисунок 4.14. Архитектура сети для решения задачи аппроксимации

Теперь можно приступить к обучению. Для этого необходимо указать, какие наборы данных должны быть использованы в качестве обучающих и целевых, а затем провести обучение (рис. 4.15–4.19).

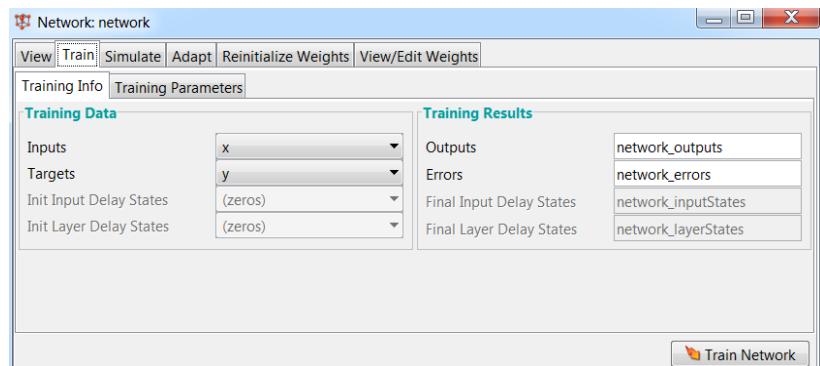


Рисунок 4.15. Окно информации об обучающих последовательностях

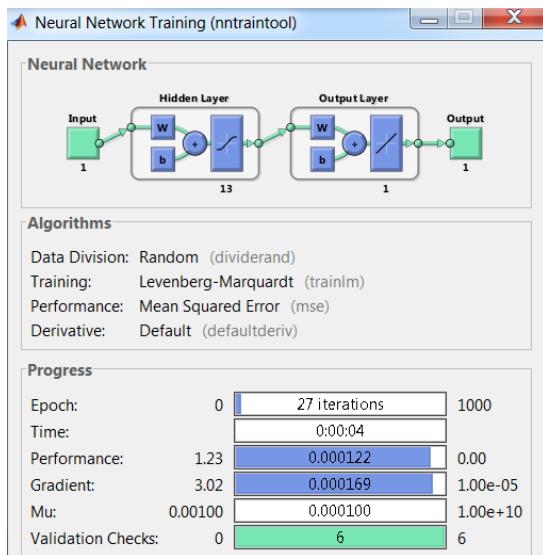


Рисунок 4.16. Окно обучения нейронной сети

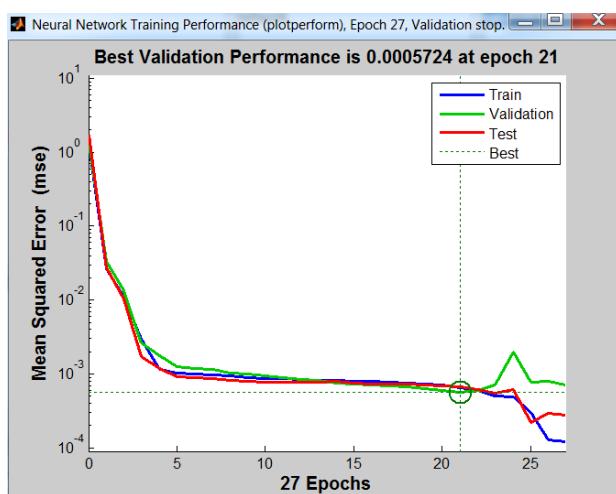


Рисунок 4.17. Изменение ошибки сети в процессе обучения

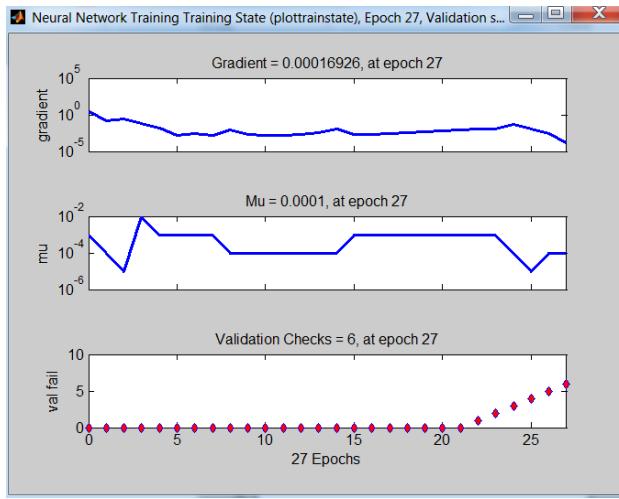


Рисунок 4.18. Окно состояния обучения

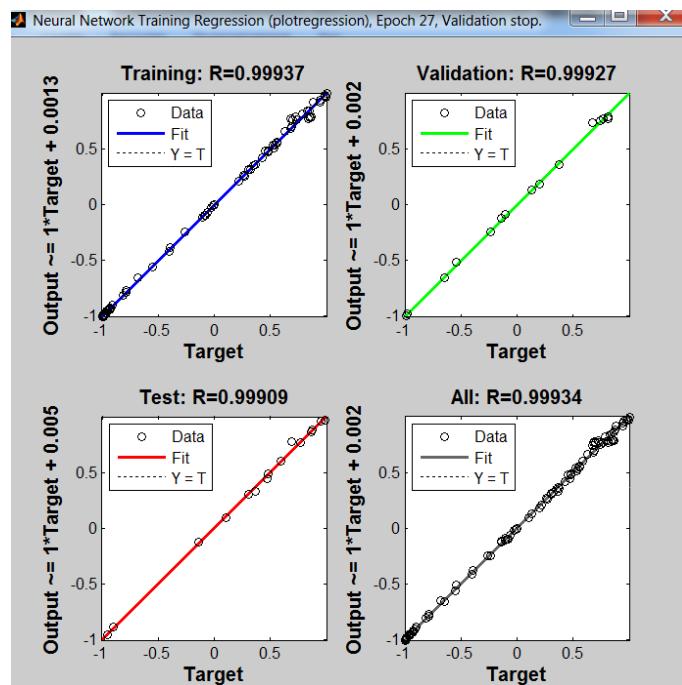


Рисунок 4.19. Окно линейной регрессии между выходом НС и целями

Экспортируем значения выхода нейронной сети в рабочую область:

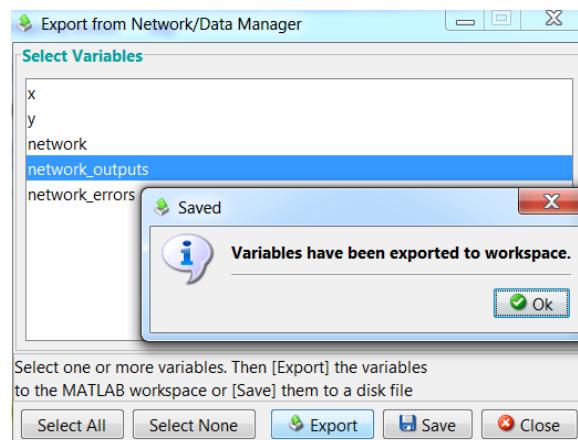


Рисунок 4.20. Окно для экспорта или записи данных в файл

Построим графики функции

$$y = \sin(5\pi[1:100]/100 + \sin(7\pi[1:100]/100));$$

и выхода нейронной сети в одной системе координат:

plot(x,y, x, network_outputs) % Рисунок 4.21

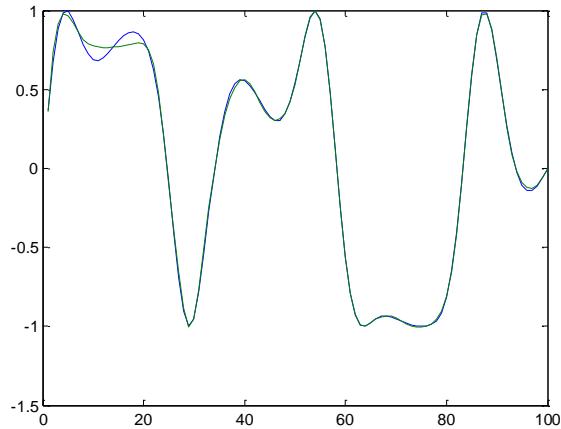


Рисунок 4.21. Кривые целевых данных и выхода нейронной сети

На рисунке 4.22 изображен график ошибки приближения:

error = y-network_outputs;

plot(error)

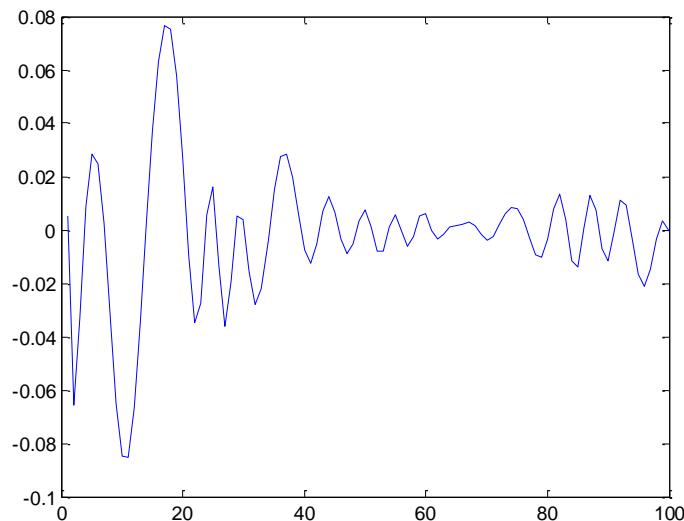


Рисунок 4.22. График величины ошибки

Точность аппроксимации может быть повышена за счет увеличения количества нейронов в скрытом слое.

1.3 Рассмотрим задачу аппроксимации синусоидальной функции, которая зашумлена нормально распределенным шумом:

P = [-1:.05:1];

T = sin(2*pi*P)+0.1*randn(size(P)); % рисунок 4.23

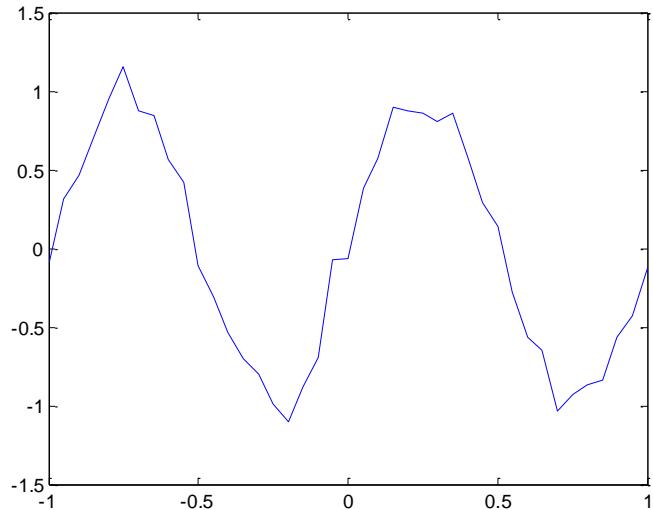


Рисунок 4.23. График синусоидальной функции, которая зашумлена нормально распределенным шумом

Сформируем для решения этой задачи двухслойную нейронную сеть прямой передачи сигнала. Вход сети принимает значения в диапазоне от -1 до 1. Первый слой имеет 20 нейронов с функцией активации *tansig*, второй слой имеет один нейрон с функцией активации *purelin*. В качестве обучающей используем функцию *trainbr*.

Формирование сети

net = newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');

Обучение сети:

net.trainParam.epochs = 50;

net.trainParam.show = 10;

net = train(net,P,T); grid on;

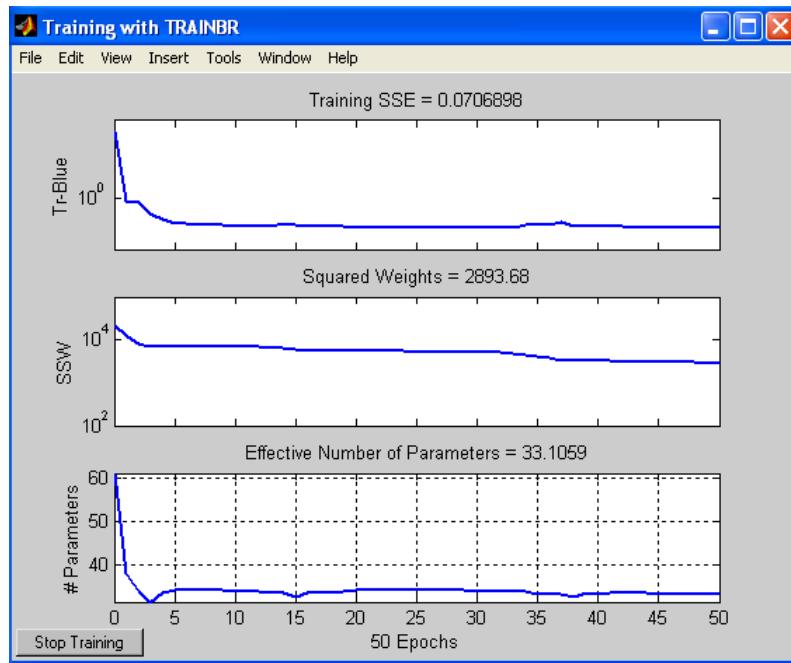


Рисунок 4.24. Окно состояния обучения

Выполним моделирование сети и построим графики исследуемых функций

Y = sim(net,P);

```
figure(2), clf, h1 = plot(P,Y,'LineWidth',2);
hold on, set(h1,'Color',[1/2,1/2,0])
plot(P,T,'+r','MarkerSize',6,'LineWidth',2), grid on
legend('выход','вход') % рисунок 4.25
```

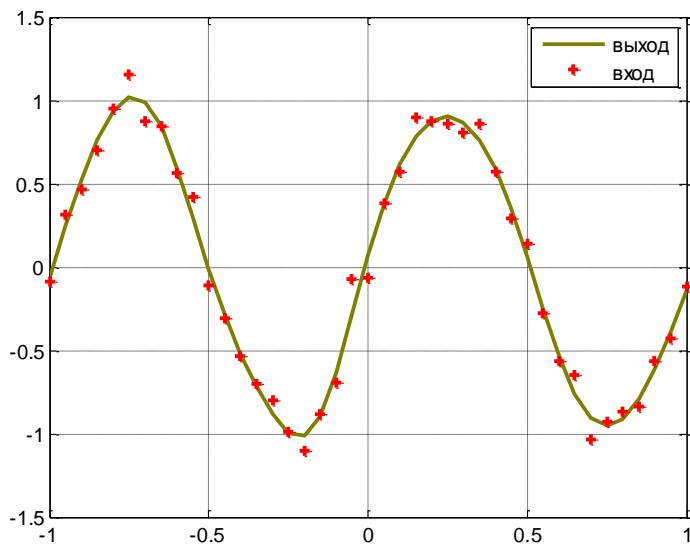


Рисунок 4.25. Графики выхода нейронной сети и фактических значений сигнала

2. Применение нейросетей для предсказания временного процесса

```
time = 1:0.01:2.5;  
X = sin(sin(time).*time*10);  
plot(X) % рисунок 4.26
```

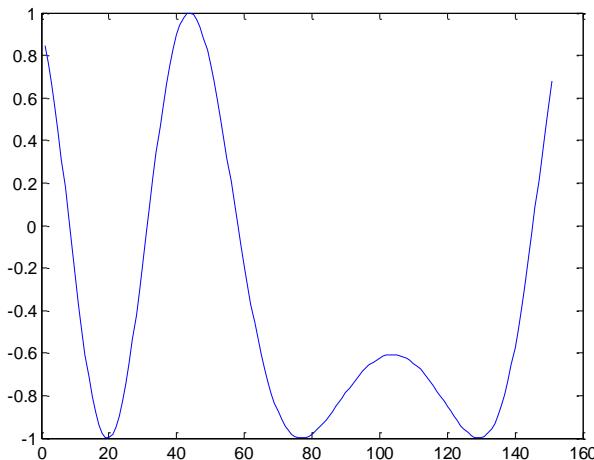


Рисунок 4.26. График временного процесса

Поскольку при синтезе сети будут использоваться адаптивные алгоритмы настройки сформируем обучающую последовательность $\{P, T\}$ в виде массива ячеек, при этом последовательность входов должна совпадать с последовательностью целевых выходов T .

$P = \text{con2seq}(X)$; преобразует числовой массив X размера $Q \times TS$, соответствующий групповому преобразованию данных, в массив ячеек P размера $1 \times TS$, содержащих числовые массивы размера $Q \times 1$ и соответствующий последовательному представлению данных.

```
T = con2seq(2*[0 X(1:(end-1))]+X);  
plot(time,cat(2,P{:}),time,cat(2,T{:}),'--')  
title('Input and Target Signals')  
xlabel('Time')  
ylabel('Input \_\_ Target \_\_')
```

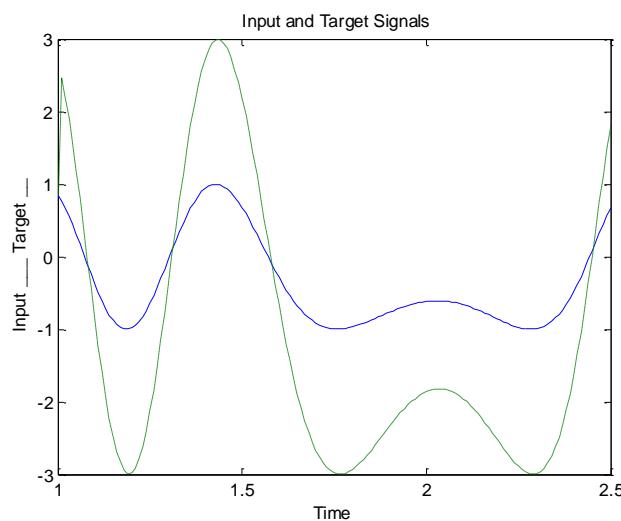


Рисунок 4.27. Графики входного сигнала и цели

```
net = newlin([-3 3], 1, [0 1], 0.1);
[net, y, e, pf] = adapt(net, P, T);
plot(time,cat(2,P{:}),'b', time, cat(2,T{:}),'r', time, cat(2,e{:}),'g')
```

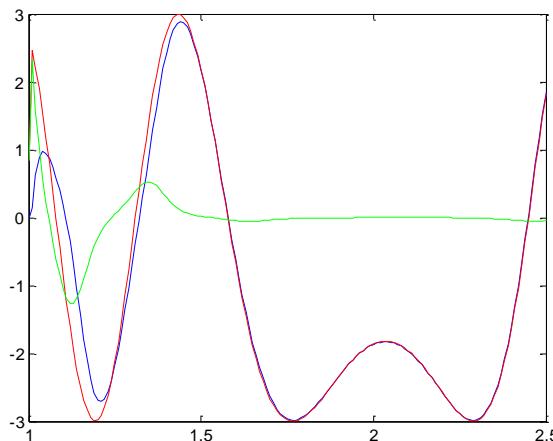


Рисунок 4.28. Графики входного, целевого сигнала и ошибки

3. Этапы реализации нейросетевого подхода решения задачи

Задан массив, состоящий из нескольких значений функции

$y = C \cdot \exp\left(-\frac{(x - A)^2}{S}\right)$ на интервале $(0,1)$, $S > 0$. Создать нейронную сеть такую,

что при вводе этих значений на входы сети на выходах получались бы значения параметров C, A и S [43].

1. Подготовка данных для обучения сети

В первую очередь необходимо определиться с размерностью входного массива. Выберем количество значений функции равным $N = 21$, т.е. в качестве входных векторов массива используем значения функции y в точках $x = 0.05; \dots 1.0$. Для обучения сети необходимо сформировать массив входных векторов для различных наборов параметров C, A и S . Каждый набор этих параметров является вектором-эталоном для соответствующего входного вектора.

Для подготовки входного и эталонного массивов воспользуемся следующим алгоритмом. Выбираем случайным образом значения компонент вектора – эталона C, A, S и вычисляем компоненты соответствующего входного вектора. Повторяем эту процедуру M раз и получаем массив входных векторов в виде матрицы размерностью $N \times M$ и массив векторов – эталонов в виде матрицы размерностью в нашем случае $3 \times M$. Полученные массивы мы можем использовать для обучения сети.

Прежде чем приступить к формированию обучающих массивов необходимо определиться с некоторыми свойствами массивов.

- 1) Выберем диапазоны изменения параметров C, A и S равными $(0.1, 1)$. Значения, близкие к 0 и сам 0 исключим в связи с тем, что функция не определена при $S = 0$. Второе ограничение связано с тем, что при использовании типичных передаточных функций желательно, чтобы компоненты входных и выходных векторов не выходили за пределы диапазона $(-1, 1)$. В дальнейшем мы познакомимся с методами нормировки, которые позволяют обойти это ограничение.
- 2) Количество входных и эталонных векторов выберем равным $M = 100$. Этого достаточно для обучения, а процесс обучения не займет много времени.

Тестовые массивы и эталоны подготовим с помощью программы *mas1*: формирование входных массивов (входной массив P и эталоны T).

P = zeros(100,21);

T = zeros(3,100);

```

x = 0:5.e-2:1;
for i = 1:100
c = 0.9*rand+0.1;
a = 0.9*rand+0.1;
s = 0.9*rand+0.1;
T(1,i) = c;
T(2,i) = a;
T(3,i) = s;
P(i,:) = c*exp(-((x-a).^2/s));
end;
P = P';

```

С помощью этой программы формируется матрица Р из $M = 100$ входных векторов-столбцов, каждый из которых сформирован из 21 точки исходной функции со случайно выбранными значениями параметров C, A, S , и матрица Т эталонов из 100 эталонных векторов-столбцов, каждый из которых сформирован из 3 соответствующих эталонных значений. Матрицы Р и Т будут использованы при обучении сети. Следует отметить, что при каждом новом запуске этой программы будут формироваться массивы с новыми значениями компонентов векторов, как входных, так и эталонных.

2. Создание сети

Вообще, выбор архитектуры сети для решения конкретной задачи основывается на опыте разработчика. Поэтому предложенная ниже архитектура сети является одним вариантом из множества возможных конфигураций.

Для решения поставленной задачи сформируем трехслойную сеть обратного распространения, включающую 21 нейрон во входном слое (по числу компонент входного вектора) с передаточной функцией *logsig*, 15 нейронов во втором слое с передаточной функцией *logsig* и 3 нейрона в выходном слое (по числу компонентов выходного вектора) с передаточной функцией *purelin*. При этом в качестве обучающего алгоритма выбран алгоритм *Levenberg-Marquardt* (*trainlm*). Этот алгоритм обеспечивает быстрое обучение, но требует многоя ре-

сурсов. В случае, если для реализации этого алгоритма не хватит оперативной памяти, можно использовать другие алгоритмы (*trainbfg*, *trainrp*, *trainscg*, *traincgb*, *traincfg*, *traincgp*,*trainoss*,*traingdx*). По умолчанию используется *trainlm*. Указанная сеть формируется с помощью процедуры:

```
net = newff (minmax(P),[21,15,3],{'logsig' 'logsig' 'purelin'},'trainlm');
```

Первый аргумент - матрица Mx2 минимальных и максимальных значений компонент входных векторов вычисляется с помощью процедуры *minmax*.

Результатом выполнения процедуры *newff* является объект – нейронная сеть *net* заданной конфигурации. Сеть можно сохранить на диске в виде *mat*. файла с помощью команды *save* и загрузить снова с помощью команды *load*. Более подробную информацию о процедуре можно получить, воспользовавшись командой *help*.

3. Обучение сети

Следующий шаг – обучение созданной сети. Перед обучением необходимо задать параметры обучения. Задаем функцию оценки функционирования *sse*.

```
net.performFcn = 'sse';
```

В этом случае в качестве оценки вычисляется сумма квадратичных отклонений выходов сети от эталонов. Задаем критерий окончания обучения – значение отклонения, при котором обучение будет считаться законченным:

```
net.trainParam.goal = 0.01;
```

Задаем максимальное количество циклов обучения. После того, как будет выполнено это количество циклов, обучение будет завершено:

```
net.trainParam.epochs = 1000;
```

Теперь можно начинать обучение:

```
[net,tr] = train(net, P, T);
```

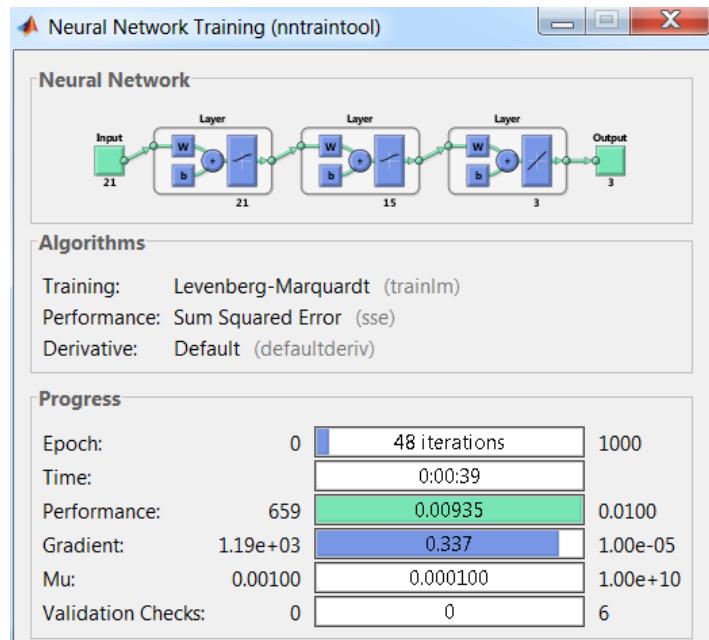


Рисунок 4.29. Окно обучения нейронной сети

Процесс обучения иллюстрируется графиком изменения ошибки сети в процессе обучения (рисунок 4.30).

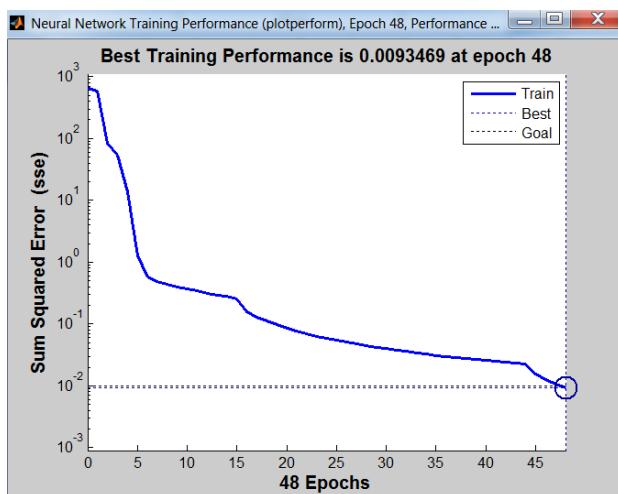


Рисунок 4.30. Изменение ошибки сети в процессе обучения

Таким образом, обучение сети окончено.

4. Тестирование сети

Перед тем, как воспользоваться нейронной сетью, необходимо исследовать степень достоверности результатов вычислений сети на тестовом массиве входных векторов. В качестве тестового массива необходимо использовать массив, компоненты которого отличаются от компонентов массива, использованного для обучения. В нашем случае для получения тестового массива достаточно воспользоваться еще раз программой *mas1*.

Для оценки достоверности результатов работы сети можно воспользоваться результатами регрессионного анализа, полученными при сравнении эталонных значений со значениями, полученными на выходе сети когда на вход поданы входные векторы тестового массива. В среде MatLab для этого можно воспользоваться функцией *postreg*. Следующий набор команд иллюстрирует описанную процедуру *mas1* с соответствующими компонентами выходных векторов сети создание тестового массива.

```
y = sim(net, P); % обработка тестового массива
```

```
[m,b,r] = postreg(y(1,:),T(1,:));
```

```
[m,b,r] = postreg(y(2,:),T(2,:));
```

```
[m,b,r] = postreg(y(3,:),T(3,:)) % регрессионный анализ результатов обра-
```

ботки.

Сравнение компонентов эталонных векторов с соответствующими компонентами выходных векторов сети (рисунок 4.31 (а, б, в)). Видно, что почти все точки легли на прямую, что говорит о правильной работе сети на тестовом массиве.

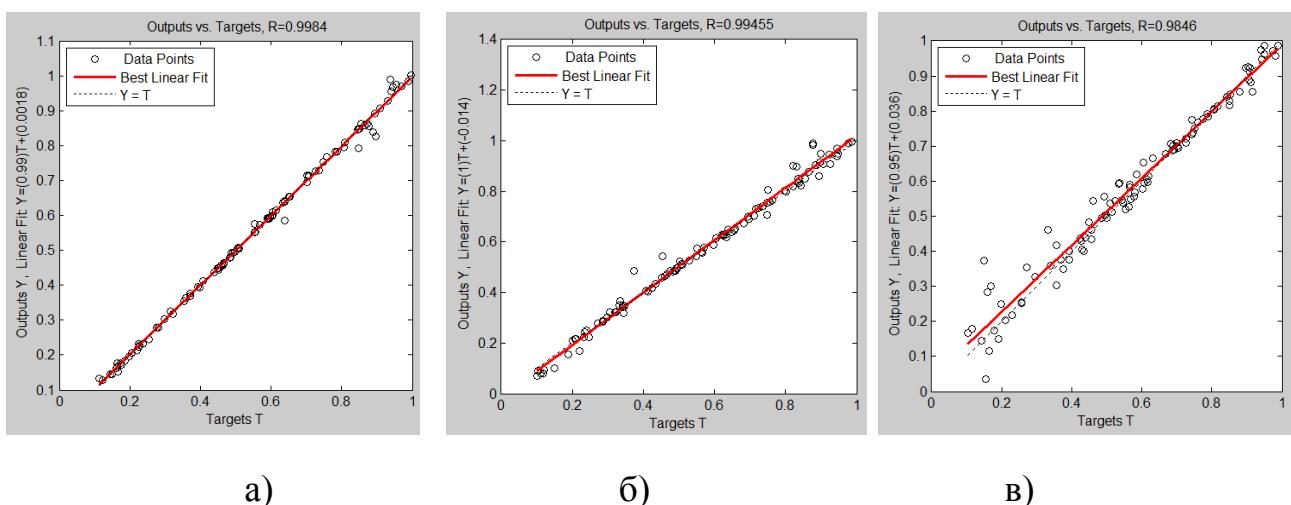


Рисунок 4.31. Сравнение компонентов эталонных векторов с соответствующими компонентами выходных векторов сети

Как видно из рисунков, наша сеть отлично решает поставленную задачу для всех трех выходных параметров.

5. Моделирование сети (использование сети для решения поставленной задачи)

Для того чтобы применить обученную сеть для обработки данных, необходимо воспользоваться функцией sim: $Y = \text{sim}(\text{net}, P)$, где P – набор входных векторов, Y – результат анализа в виде набора выходных векторов.

Например, пусть $C = 0.2$, $A = 0.8$, $S = 0.7$, тогда

p = 0.2*exp(-((x-0.8).^2/0.7));

p = p';

Подставив этот входной вектор в качестве аргумента функции sim:

Y = sim(net, p)

Получим:

Y =

0.2023(C)

0.8030(A)

0.7154(S)

Что весьма близко к правильному результату (0.2; 0.8; 0.7).

Программа работы и методические указания

1. Построить модель нейронной сети, аппроксимирующей полином вида $Y = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$ на интервале $[-1,1]$, коэффициенты полинома заданы в табл. 4.1. Вариант соответствует порядковому номеру по журналу.
2. Построить модель нейронной сети для аппроксимации функции $f(t)$ на промежутке $[0,10]$, заданной согласно номеру варианта (табл. 4.2).

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Таблица 4.1. Коэффициенты полинома

№	a	b	c	d	e	f	№	a	b	c	d	e	f
1.	-2	1	-1	2	1	-2	11.	-1	1	2	0	0	0
2.	-1	3	-1	2	-1	-1	12.	1	1	1	1	1	1
3.	-2	-2	2	2	-2	2	13.	2	1	1	2	2	-8
4.	-3	2	-2	3	-1	1	14.	0	1	0	1	0	-2
5.	-2	2	-2	2	-2	2	15.	1	0	1	0	1	-3
6.	-1	-1	-1	1	1	1	16.	2	1	-2	1	0	-7
7.	0	1	0	2	0	-3	17.	1	2	3	-3	-4	-1
8.	2	0	2	3	1	0	18.	2	1	2	-1	-3	-2
9.	2	-2	4	-4	3	-3	19.	2	1	1	-2	-2	-1
10.	-5	-3	3	3	2	1	20.	-1	-1	-1	-1	2	2

Таблица 4.2. Варианты аппроксимируемых функций

№	$f(t)$	№	$f(t)$
1.	$\cos^2(t) \sin(2 \cdot t)$	2.	$t^2 \cdot \sin(t)$
3.	$\sin(0.5 \cdot t^2)$	4.	$\sin(t^2 - 10 \cdot t)$
5.	$t \cdot \sin(t)$	6.	$\sin(t^2 - 5 \cdot t)$
7.	$\cos(1.4^t)$	8.	$\sin(t^2 - 8 \cdot t)$
9.	$\ln(t) \sin(t)$	10.	$\sin(t^2 - 12 \cdot t)$
11.	$\ln(2 \cdot t) \sin(t^{1.5})$	12.	$\sin(t^2 - 8 \cdot t)$
13.	$0.01 \cdot t^2 - \sin(t)$	14.	$\sin(t^2 - 12 \cdot t)$
15.	$t \cdot \sin(2 \cdot t)$	16.	$\sin(t^2 - 6 \cdot t)$
17.	$t \cdot \sin(3 \cdot t)$	18.	$\sin(t^2 - 4 \cdot t)$
19.	$t \cdot \sin(4 \cdot t)$	20.	$(t^2 - 10 \cdot t) \sin(t^2 - 10 \cdot t)$

Лабораторная работа №5

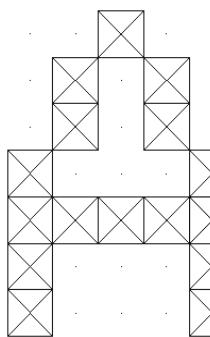
РАСПОЗНАВАНИЕ ОБРАЗОВ

Цель работы: анализ работы многослойной нейронной сети по распознаванию букв методом обратного распространения ошибки.

1. Создание компьютерной модели нейронной сети

Демонстрационная программа пакета *nntool appcr1* иллюстрирует, как распознавание символов может быть выполнено в сети с обратным распространением ошибки. Для распознавания 26 символов латинского алфавита, получаемых, например, с помощью системы распознавания, выполняющей оцифровку каждого символа в ее поле зрения, используется сеть из двух слоев, не считая входного, с n (10) нейронами в скрытом слое и p (26) нейронами в выходном (по одному на букву). Каждый символ представляется шаблоном размера 7×5 , соответствующим пиксельной градации букв.

Постановка задачи. Требуется создать нейронную сеть для распознавания 26 символов латинского алфавита. В качестве датчика предполагается использовать систему распознавания, которая выполняет оцифровку каждого символа, находящегося в поле зрения. В результате каждый символ будет представлен шаблоном размера 5×7 . Например, символ А может быть представлен, как это показано на рисунке 5.1.



0	0	1	0	0
0	1	0	1	0
0	1	0	1	0
1	0	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1

Рисунок 5.1. Цифровое представление символа А в шаблоне 5×7

Однако система считывания символов обычно работает неидеально и отдельные элементы символов могут оказаться искаженными (рисунок 5.2).

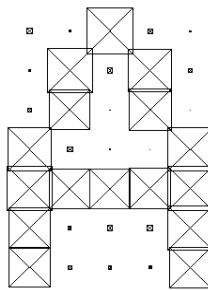


Рисунок 5.2. Искаженное представление символа А в шаблоне 5x7

Проектируемая нейронная сеть должна точно распознавать идеальные векторы вход и с максимальной точностью воспроизводить зашумленные векторы. М-функция *prprob* определяет 26 векторов входа, каждый из которых содержит 35 элементов, этот массив называется алфавитом. М-функция формирует выходные переменные *alphabet* и *targets* которые определяют массивы алфавита и целевых векторов. Массив *targets* определяется как *eye(26)*. Для того чтобы восстановить шаблон для *i*-й буквы алфавита, надо выполнить следующие операторы:

[alphabet, targets] = prprob;

i = 10;

ti = alphabet(:, i);

letter{i} = reshape(ti, 5, 7)';

letter{i}

ans =

0	0	1	0	0
0	1	0	1	0
0	1	0	1	0
1	0	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1

Нейронная сеть. На вход сети поступает вектор входа с 35 элементами; вектор выхода содержит 26 элементов, только один из которых равен 1, а остальные - 0. Правильно функционирующая сеть должна ответить вектором со значением 1 для элемента, соответствующего номеру символа в алфавите. Кро-

ме того, сеть должна быть способной распознавать символы в условиях действия шума. Предполагается, что шум - это случайная величина со средним значением 0 и стандартным отклонением, меньшим или равным 0.2.

Архитектура сети. Для работы нейронной сети требуется 35 входов и 26 нейронов в выходном слое. Для решения задачи выберем двухслойную нейронную сеть с логарифмическими сигмоидальными функциями активации в каждом слое. Такая функция активации выбрана потому, что диапазон выходных сигналов для этой функции определен от 0 до 1, и этого достаточно, чтобы сформировать значения выходного вектора.

Скрытый слой имеет 10 нейронов. Такое число нейронов выбрано на основе опыта и разумных предположений. Если при обучении сети возникнут затруднения, то можно увеличить количество нейронов этого уровня. Сеть обучается так, чтобы сформировать единицу в единственном элементе вектора выхода, позиция которого соответствует номеру символа, и заполнить остальную часть вектора нулями. Однако наличие шумов может приводить к тому, что сеть не будет формировать вектора выхода, состоящего точно из единиц и нулей. Поэтому по завершении этапа обучения выходной сигнал обрабатывается М-функцией *compet*, которая присваивает значение 1 единственному элементу вектора выхода, а всем остальным - значение 0.

Инициализация сети. Вызовем М-файл *prprob*, который формирует массив векторов входа *alphabet* размера 35x26 с шаблонами символов алфавита и массив целевых векторов *targets*:

[alphabet,targets] = prprob;

[R,Q] = size(alphabet);

[S2,Q] = size(targets);

Двухслойная нейронная сеть создается с помощью команды *newff*:

S1 = 10;

net = newff (minmax (alphabet), [S1 S2], {'logsig' 'logsig'}, 'traingdx');

net.LW{2,1} = net.LW{2,1}*0.01;

net.b{2} = net.b{2}*0.01;

Структура нейронной сети, может быть получена при помощи команды **gensim(net);**

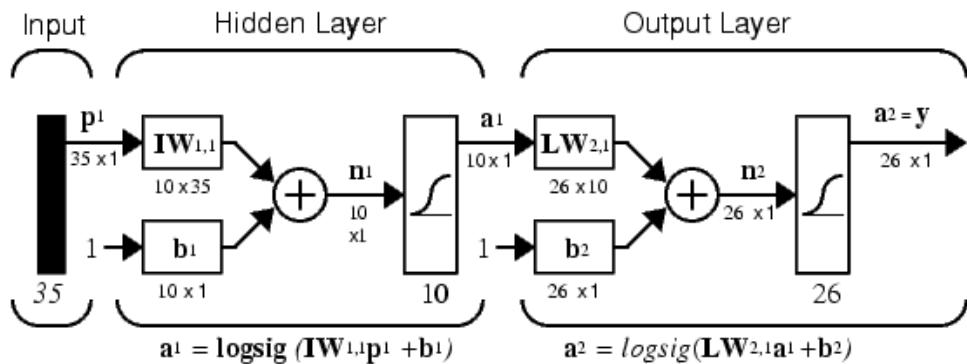


Рисунок 5.3. Архитектура нейронной сети

2. Обучение сети распознаванию образов

Чтобы создать нейронную сеть, которая может обрабатывать зашумленные векторы входа, следует выполнить обучение сети как на идеальных, так и на зашумленных векторах. Сначала сеть обучается на идеальных векторах, пока не будет обеспечена минимальная сумма квадратов погрешностей. Затем сеть обучается на 10 наборах идеальных и зашумленных векторов. Две копии свободного от шума алфавита используются для того, чтобы сохранить способность сети классифицировать идеальные векторы входа. К сожалению, после того, как описанная выше сеть обучилась классифицировать сильно зашумленные векторы, она потеряла способность правильно классифицировать некоторые векторы, свободные от шума. Следовательно, сеть снова надо обучить на идеальных векторах. Это гарантирует, что сеть будет работать правильно, когда на ее вход будет передан идеальный символ. Обучение выполняется с помощью функции **trainbp**, которая реализует метод обратного распространения ошибки с возмущением и адаптацией параметра скорости настройки.

Обучение в отсутствие шума. Сеть первоначально обучается в отсутствие шума с максимальным числом циклов обучения 5000 либо до достижения допустимой средней квадратичной погрешности равной 0.1 (Рисунок 5.4);

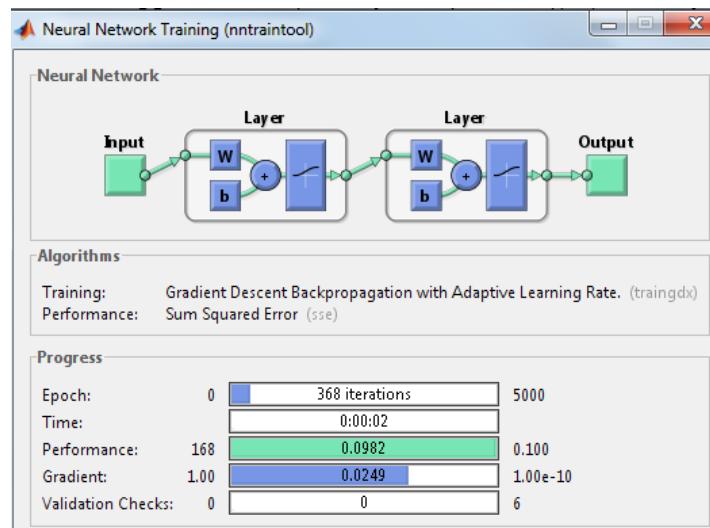


Рисунок 5.4. Окно обучения нейронной сети

P = alphabet;

T = targets;

```
net.performFcn = 'sse'; % Sum-Squared Error performance function
net.trainParam.goal = 0.1; % Sum-squared error goal.
net.trainParam.show = 20; % Frequency of progress displays (in epochs).
net.trainParam.epochs = 5000; % Maximum number of epochs to train.
net.trainParam.mc = 0.95;
[net,tr] = train(net, P, T);
```

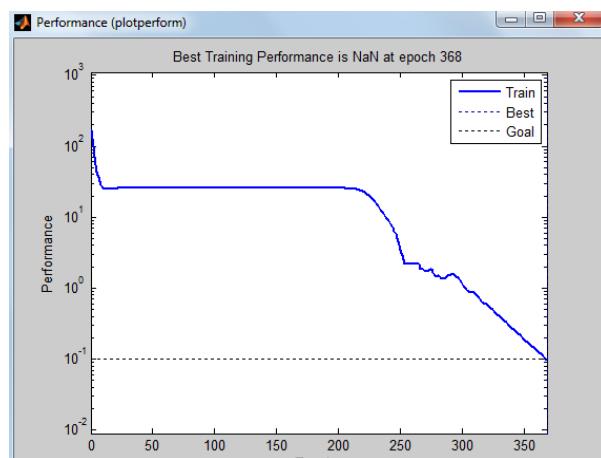


Рисунок 5.5. Изменение ошибки сети в процессе обучения

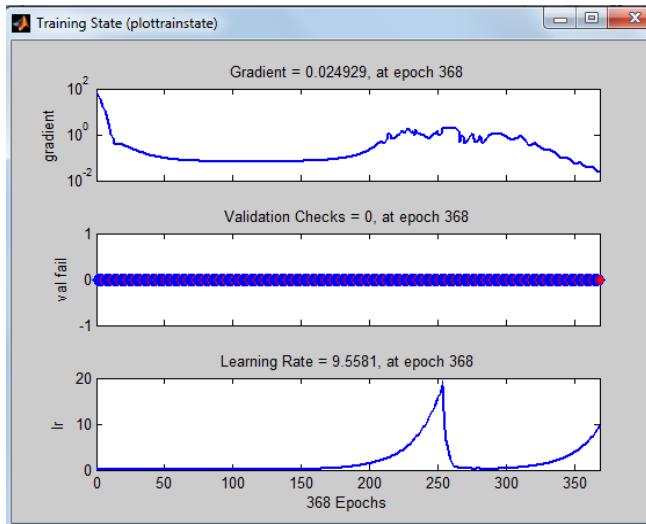


Рисунок 5.6. Окно состояния обучения

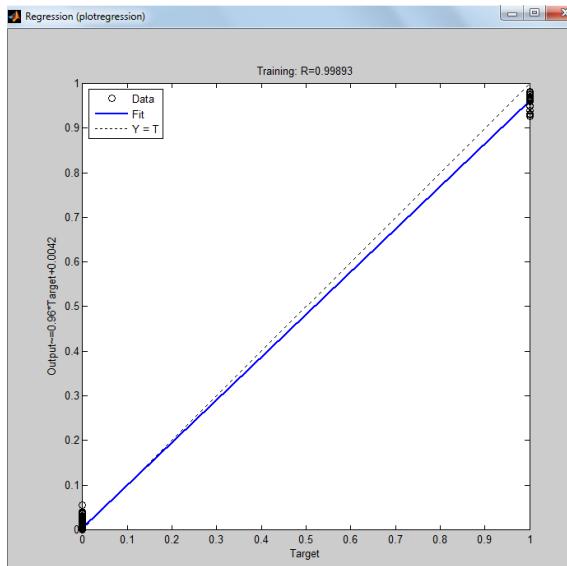


Рисунок 5.7. Окно линейной регрессии между выходом НС и целями

Обучение в присутствии шума. Чтобы спроектировать нейронную сеть, не чувствительную к воздействию шума, обучим ее с применением двух идеальных и двух зашумленных копий векторов алфавита. Целевые векторы состоят из четырех копий векторов. Зашумленные векторы имеют шум со средним значением 0.1 и 0.2. Это обучает нейрон правильно распознавать зашумленные символы и в то же время хорошо распознавать идеальные векторы.

При обучении с шумом максимальное число циклов обучения сократим до 300, а допустимую погрешность увеличим до 0.6:

```
netn = net;
```

netn.trainParam.goal = 0.6; % предельная среднеквадратическая погрешность

netn.trainParam.epochs = 300; % максимальное количество циклов обучения

```
T = [targets targets targets targets];
```

```
for pass = 1:10
```

```
P = [alphabet, alphabet, ...
```

```
(alphabet + randn(R,Q)*0.1), (alphabet + randn(R,Q)*0.2)];
```

```
[netn,tr] = train(netn,P,T);
```

```
end
```

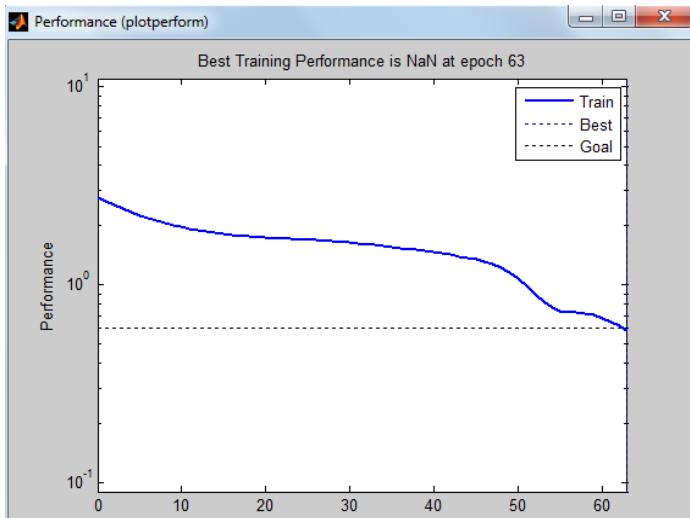


Рисунок 5.8. Изменение ошибки сети в процессе обучения

Повторное обучение в отсутствие шума

Поскольку нейронная сеть обучалась в присутствии шума, то имеет смысл повторить ее обучение без шума, чтобы гарантировать, что идеальные векторы входа классифицируются правильно.

netn.trainParam.goal = 0.1; % предельная среднеквадратичная погрешность

netn.trainParam.epochs = 500; % максимальное количество циклов обучения

```
netn.trainParam.show = 5; % частота вывода результатов на экран
```

```
[netn, tr] = train(netn, P, T);
```

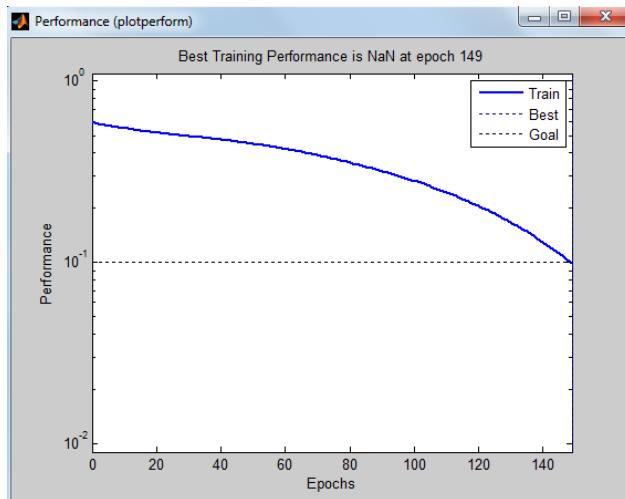


Рисунок 5.9. Изменение ошибки сети в процессе обучения

3. Проверка эффективности функционирования сети

Эффективность нейронной сети будем оценивать следующим образом. Рассмотрим 2 структуры нейронной сети: сеть 1, обученную на идеальных последовательностях, и сеть 2, обученную на зашумленных последовательностях. Проверка функционирования производится на 100 векторах входа при различных уровнях шума.

Приведем фрагмент сценария, который выполняет эти операции:

```

noise_range = 0:.05:.5;
max_test = 100;
network1 = [];
network2 = [];
T = targets;
for noiselevel = noise_range
    sprintf('Уровень шума %.2f.\n',noiselevel);
    errors1 = 0;
    errors2 = 0;
    for i = 1:max_test
        P = alphabet + randn(35,26)*noiselevel;
        % Тест для сети 1
        A = sim(net,P);
    
```

```

AA = compet(A);
errors1 = errors1+sum(sum(abs(AA-T)))/2;
% Тест для сети 2
An = sim(netn,P);
AAn = compet(An);
errors2 = errors2+sum(sum(abs(AAn-T)))/2;
end
network1 = [network1 errors1/26/100]; % среднее значения ошибок (100
последовательностей из 26 векторов)
network2 = [network2 errors2/26/100]; % среднее значения ошибок (100
последовательностей из 26 векторов)
end
Уровень шума 0.00.
Уровень шума 0.05.
Уровень шума 0.10.
Уровень шума 0.15.
Уровень шума 0.20.
Уровень шума 0.25.
Уровень шума 0.30.
Уровень шума 0.35.
Уровень шума 0.40.
Уровень шума 0.45.
Уровень шума 0.50.
plot(noise_range,network1*100,'--',noise_range,network2*100);
title('Процент ошибочных классификаций');
xlabel('Уровень шума');
ylabel('Сеть 1 - - Сеть 2 ---');

```

Тестируемое реализуется следующим образом. Шум со средним значением 0 и стандартным отклонением от 0 до 0.5 с шагом 0.05 добавляется к векторам входа. Для каждого уровня шума формируется 100 зашумленных последо-

вательностей для каждого символа вычисляется выход сети. Выходной сигнал обрабатывается M-функцией *comret* с той целью, чтобы выбрать только один из 26 элементов вектора выхода. После этого оценивался количество ошибочных классификаций и вычислялся процент ошибки. Соответствующий график погрешности сети от уровня входного шума показан на рисунке 5.10.

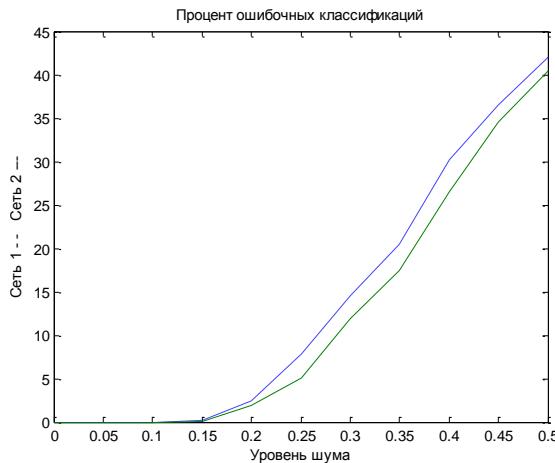


Рисунок 5.10 – График погрешности сети от уровня входного шума

Сеть 1 обучена на идеальных векторах входа, а сеть 2 – на зашумленных. Обучение сети на зашумленных векторах входа значительно снижает погрешность распознавания реальных векторов входа. Сети имеют очень малые погрешности, если среднеквадратичное значение шума находится в пределах от 0.00 до 0.05. Когда к векторам был добавлен шум со среднеквадратичным значением 0.2, в обеих сетях начали возникать заметные ошибки. При этом погрешности нейронной сети, обученной на зашумленных векторах, на 1-4 % ниже, чем для сети, обученной на идеальных входных последовательностях.

Если необходима более высокая точность распознавания, сеть может быть обучена либо в течение более длительного времени, либо с использованием большего количества нейронов в скрытом слое. Можно также увеличить размер векторов, чтобы пользоваться шаблоном с более мелкой сеткой, например 10x14 точек вместо 5x7.

Проверим работу нейронной сети для распознавания символов. Сформируем зашумленный вектор входа для символа J:

```
noisyJ = alphabet(:,10) + randn(35,1) *0.2;
```

```
plotchar(noisyJ); % зашумленный символ J (рисунок 5.11)
```

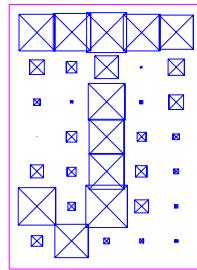


Рисунок 5.11 – Зашумленный символ J

```
A2 = sim(net, noisyJ);  
A2 = compet(A2);  
answer = find(compet(A2) == 1)
```

answer =

10

```
plotchar(alphabet (:,answer)); % распознанный символ J
```

Нейронная сеть выделила сигнал на 10-м выходном нейроне и восстановила символ J без ошибок (рисунок 5.12).

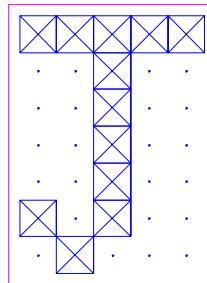


Рисунок 5.12 – Распознанный символ J

Эта задача демонстрирует, как может быть разработана простая система распознавания изображений. Заметим, что процесс обучения не состоял из единственного обращения к обучающей функции. Сеть была обучена несколько раз при различных векторах входа. Обучение сети на различных наборах зашумленных векторов позволило обучить сеть работать с изображениями, иска- женными шумами, что характерно для реальной практики.

Программа работы и методические указания

1. Изучить prprob, appcr1.
2. Построить нейронную сеть для распознавания букв латинского алфавита, используя функцию newff – инициализации сети прямой передачи сигнала.
3. Обучить сеть, используя функцию traingdx.
4. Распознать несколько зашумленных букв.

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Лабораторная работа №6

ИЗУЧЕНИЕ РАДИАЛЬНЫХ БАЗИСНЫХ, СЕТЕЙ РЕГРЕССИИ, ВЕРОЯТНОСТНЫХ НЕЙРОННЫХ СЕТЕЙ

Цель работы: изучить архитектуру радиальной базисной сети, сетей регрессии, вероятностных нейронных сетей, возможность их применения задачи аппроксимации функции и классификации.

1. Краткие теоретические сведения

В общем случае под радиальной базисной нейронной сетью (Radial Basis Function Network, сеть RBF) понимается двухслойная сеть без обратных связей, которая содержит скрытый слой радиально симметричных нейронов.

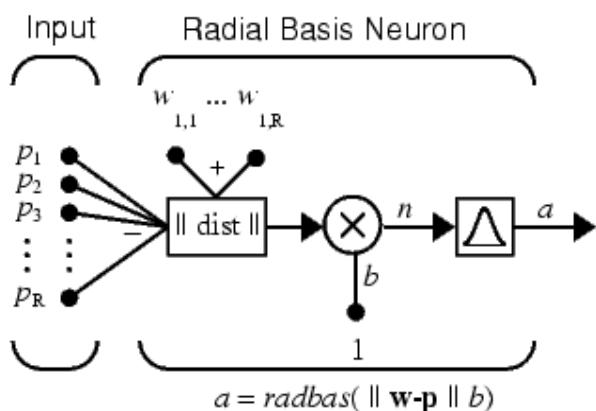


Рисунок 6.1. Радиальный базисный нейрон с R входами

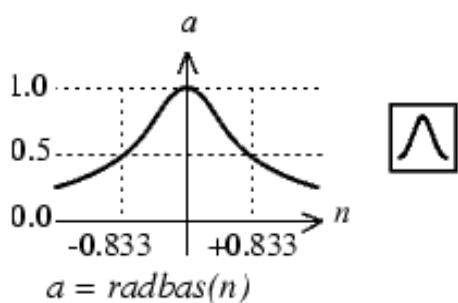


Рисунок 6.2. График функции активации

Радиальные базисные нейронные сети состоят из большего количества нейронов, чем стандартные сети с прямой передачей сигналов и обучением методом обратного распространения ошибки, но на их создание требуется значи-

тельно меньше времени. Эти сети наиболее эффективны, когда доступно большое количество обучающих векторов.

На рисунке 6.1 показан радиальный базисный нейрон с R входами. Функция активации для радиального базисного нейрона имеет вид

$$a = \text{radbas}(n) = \exp(-n^2). \quad (6.1)$$

Вход функции активации определяется как модуль разности вектора весов w и вектора входа p, умноженный на смещение b. Функция активации для радиального базисного нейрона имеет вид: при n=0 a=exp(-0²)=1; при n=0,833 a=exp(-0,833²)=0,5; при n=-0,833 a=exp(0,833²)=0,5 (рисунок 6.2).

Вход функции активации определяется как модуль разности вектора весов w и вектора входа p, умноженный на смещение b:

$$n = \|p - w\| \cdot b. \quad (6.2)$$

Радиальный базисный нейрон формирует значение 1, когда вход p идентичен вектору весов w. Смещение b позволяет корректировать чувствительность нейрона radbas. Например, если нейрон имел смещение 0.1, то его выходом будет 0.5 для любого вектора входа p и вектора веса w при расстоянии между векторами, равном 8.333, или 0.833/b.

Радиальная базисная сеть состоит из двух слоев: скрытого радиального базисного слоя, имеющего S¹ нейронов, и выходного линейного слоя, имеющего S² нейронов (рисунок 6.3).

Входами блока ||dist|| на этом рисунке являются вектор входа p и матрица весов IW^{1,1}, а выходом – вектор, состоящий из S¹ элементов. Выход блока ||dist|| умножается поэлементно на вектор смещения b¹ и формирует вход функции активации.

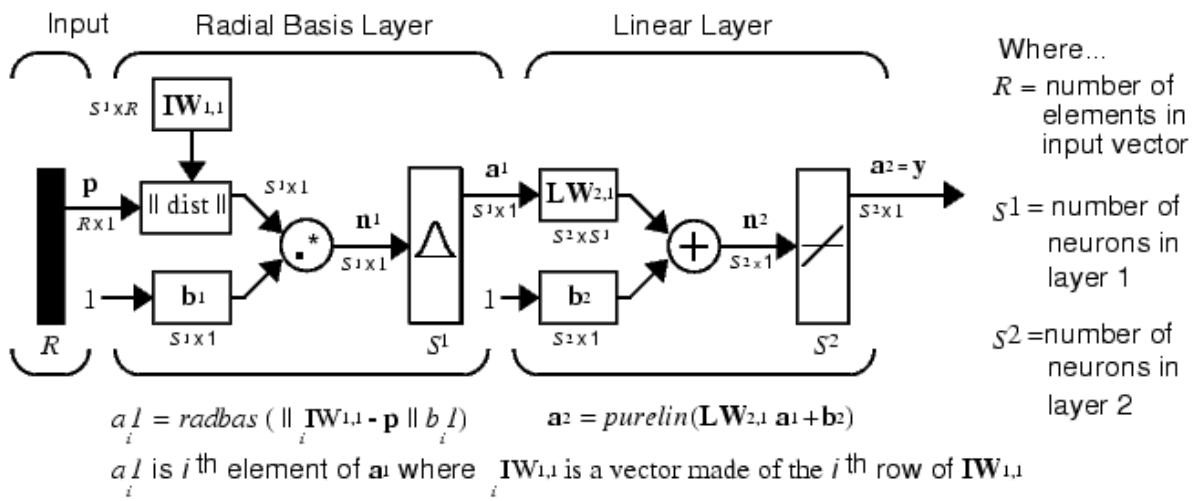


Рисунок 6.3. Схема архитектуры радиальной базисной сети

Радиальные базисные нейронные сети предназначены для аппроксимации функций. Возьмем произвольную непрерывную функцию и представим ее с помощью суммы колоколообразных функций. Аналитически это означает представление $f(x)$ в виде разложения по стандартному набору пространственно локализованных функций:

$$f(x) = \sum_i w_i \varphi(\|\mathbf{x} - \mathbf{c}_i\|), \quad (6.3)$$

где w_i - веса суммирования отдельных откликов, \mathbf{c}_i - центры базисных радиальных функций. Это формула нейронной сети на основе радиальной базисной функции. Расстояние $\|\mathbf{x} - \mathbf{c}\|$ определяется как расстояние в евклидовом пространстве:

$$\|\mathbf{x} - \mathbf{c}\| = AB = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + \dots} \quad (6.4)$$

Функция *newrbe* формирует радиальную базисную сеть с нулевой ошибкой. Сеть с радиальными базисными функциями представляет собой, как правило, сеть с тремя слоями: обычным входным слоем, скрытым радиальным базисным слоем и выходным линейным слоем.

Функция *newrb* формирует радиальную базисную сеть с ненулевой ошибкой в отличие от *newrbe*. Функция *newrb* имеет следующий синтаксис:

net = newrb (P, T, goal, spread).

Радиальные базисные сети используют для аппроксимации функций. Функция *newrb* конструирует скрытый (первый) слой из радиальных базисных нейронов и использует значение средней квадратичной ошибки (*goal*). Функция *newrb* (*P, T, goal, spread*) имеет следующие аргументы: *P* – матрица *Q* входных векторов размерности *R* на *Q*; *T* – матрица *Q* векторов целевых классов *S* на *Q*; *goal* – средняя квадратичная ошибка, по умолчанию 0,0; *spread* – разброс радиальной базисной функции, по умолчанию 1,0. Функция создает и возвращает в качестве объекта радиальную базисную сеть. Большое значение разброса приводит к большей гладкости аппроксимации. Слишком большой разброс требует много нейронов, для того чтобы подстроиться под быстро изменяющуюся функцию, слишком малый – для достижения гладкости аппроксимации. Подобрать значение разброса можно с помощью многократных вызовов функции *newrb*. Создадим в среде MatLab радиальную базисную сеть:

```
net = newrbe(P, T, spread).
```

Функция *newrbe* проектирует радиальную базисную сеть с нулевой ошибкой для заданных векторов. Функция *newrbe(P, T, spread)* имеет три параметра: *P* – матрица *Q* входных векторов размерности *R* на *Q*, *T* – матрица *Q* целевых векторов – описателей класса размерности *S* на *Q*, *spread* – разброс радиальной базисной функции, по умолчанию 1,0. Функция создает радиальную базисную сеть.

Нейронные сети регрессии GRNN (Generalized Regression Neural Network) предназначены для решения задач обобщенной регрессии, анализа временных рядов и аппроксимации функций. Характерной особенностью этих сетей является очень высокая скорость их обучения.

Архитектура сети GRNN показана на рисунке 6.4. Она аналогична архитектуре радиальной базисной сети, но отличается структурой второго слоя, в котором используется блок *normprod* для вычисления нормированного скалярного произведения строки массива весов LW21 и вектора входа a1 в соответствии со следующим соотношением:

$$n^2 = \frac{LW^{21}a^1}{\text{sum}(a^1)} \quad . \quad (6.5)$$

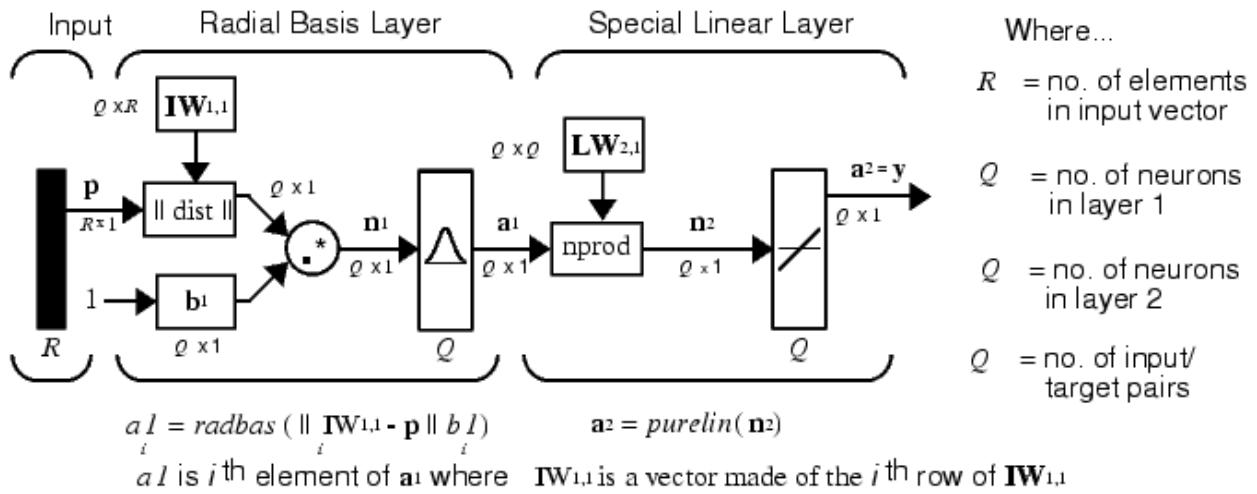


Рисунок 6.4. Схема архитектуры сети GRNN

Первый слой – это радиальный базисный слой с числом нейронов, равным числу элементов Q обучающего множества. В качестве начального приближения для матрицы весов выбирается массив p ; смещение b_1 устанавливается равным вектор-столбцу с элементами $0,8326/\text{SPREAD}$.

Функция $dist$ вычисляет расстояние между вектором входа и вектором веса нейрона; вход функции активации n^1 равен поэлементному произведению взвешенного входа сети на вектор смещения; выход каждого нейрона первого слоя a_1 является результатом преобразования вектора n^1 радиальной базисной функцией $radbas$.

Если вектор веса нейрона равен транспонированному вектору входа p , то взвешенный вход равен 0, а выход функции активации – 1. Если расстояние между вектором входа и вектором веса нейрона равно $spread$, то выход функции активации будет равен 0,5.

Второй слой – это линейный слой с числом нейронов, также равным R , причем в качестве начального приближения для матрицы весов $LW\{2,1\}$ выбирается массив T .

Предположим, что имеем вектор входа r_i , близкий к одному из векторов входа r из обучающего множества. Этот вход r_i генерирует значение выхода слоя a_i , близкое к 1. Это приводит к тому, что выход слоя 2 будет близок к t_i .

Если параметр влияния *SPREAD* мал, то радиальная базисная функция характеризуется резким спадом и диапазон входных значений, на который реагируют нейроны скрытого слоя, оказывается весьма малым. С увеличением параметра *SPREAD* наклон радиальной базисной функции становится более гладким, и в этом случае уже несколько нейронов реагируют на значения вектора входа. Тогда на выходе сети формируется вектор, соответствующий среднему нескольких целевых векторов, соответствующих входным векторам обучающего множества, близких к данному вектору входа.

Чем больше значение параметра *SPREAD*, тем большее число нейронов участвует в формировании среднего значения, и в итоге функция, генерируемая сетью, становится более гладкой.

Для создания нейронной сети GRNN предназначена функция *newgrnn*:

```
net = newgrnn(P, T, spread).
```

Функция *newgrnn(P, T, spread)* имеет следующие входы: P – матрица Q входных векторов размерности Яна Q , T – матрица Q целевых векторов классов размерности S на Q , $spread$ – разброс радиальных базисных функций, по умолчанию 1,0. Функция возвращает НС регрессии. Чем больше разброс, тем более гладкой будет функция аппроксимации. Для того чтобы настроить функцию аппроксимации на исходные данные, используют разброс меньший, чем типичное расстояние между входными векторами.

Функция *newgrnn* создает двухслойную сеть. Первый слой содержит радиальные базисные нейроны, вычисляющие расстояние между входами и весами с помощью *netprod*. Второй слой имеет нейроны с функцией активации *purelin*. Только у первого слоя существует *bias*.

Недостаток сети GRNN заключается в том, что погрешности ненулевые в отличие от радиальных сетей с нулевой ошибкой. Достоинство же в том, что

скорость обучения очень высокая, так как обучение сводится только к присвоению значений весов.

Нейронные сети PNN (Probabilistic Neural Network) предназначены для решения вероятностных задач и, в частности, задач классификации.

Архитектура сети PNN базируется на архитектуре радиальной базисной сети, но в качестве второго слоя использует так называемый конкурирующий слой, который подсчитывает вероятность принадлежности входного вектора к тому или иному классу и в конечном счете сопоставляет вектор с тем классом, вероятность принадлежности к которому выше. Структура сети PNN представлена на рисунке 6.5.

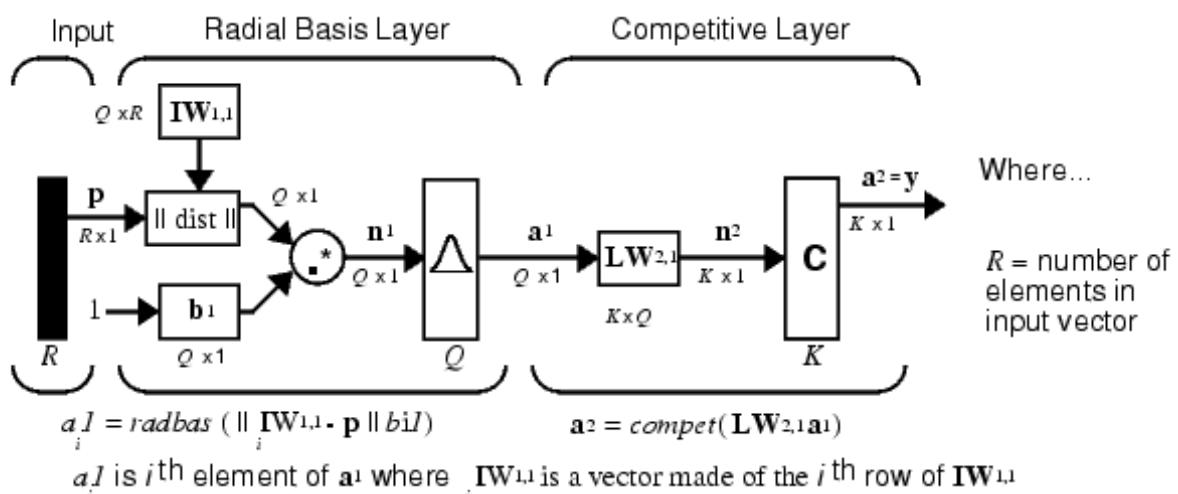


Рисунок 6.5. Схема архитектуры сети PNN

Предполагается, что задано обучающее множество, состоящее из Q пар векторов вход/цель. Каждый вектор цели имеет K элементов, указывающих класс принадлежности, и, таким образом, каждый вектор входа ставится в соответствие одному из K классов. В результате может быть образована матрица связности T размера $K \times Q$, состоящая из нулей и единиц, строки которой соответствуют классам принадлежности, а столбцы – векторам входа. Таким образом, если элемент $T(i,j)$ матрицы связности равен 1, то это означает, что j -й входной вектор принадлежит к классу i .

Весовая матрица первого слоя IW^{11} ($net.IW\{1,1\}$) формируется с использованием векторов входа из обучающего множества в виде матрицы p . Когда по-

дается новый вход, блок $\|dist\|$ вычисляет близость нового вектора к векторам обучающего множества; затем вычисленные расстояния умножаются на смещения и подаются на вход функции активации *radbas*. Вектор обучающего множества, наиболее близкий к вектору входа, будет представлен в векторе выхода a^1 числом близким к 1.

Весовая матрица второго слоя LW^{21} ($net.LW\{2,1\}$) соответствует матрице связности T , построенной для данной обучающей последовательности. Эта операция может быть выполнена с помощью функции *ind2vec*, которая преобразует вектор целей в матрицу связности T . Произведение T^*a^1 определяет элементы вектора a^1 , соответствующие каждому из K классов. В результате конкурирующая функция активации второго слоя *compet* формирует на выходе значение, равное 1 для самого большего по величине элемента вектора n^2 и 0 в остальных случаях. Таким образом, сеть PNN выполняет классификацию векторов входа по K классам.

Вероятностную нейронную сеть (ВНС) проектируют с помощью функции $net = newpnn(P, T, spread)$.

Функция $net = newpnn(P, T, spread)$ имеет такие же параметры, как и вышеописанная функция *newgrnn*. Если разброс близок к нулю, вероятностная нейронная сеть действует как классификатор на основе принципа выбора «ближайшего соседа», в противном случае сеть принимает в расчет несколько близких векторов.

2. Создание и обучение нейронной сети регрессии

Рассмотрим проектирование НС регрессии. Определим:

$P = [1 \ 2 \ 3]$; % входы НС регрессии

$T = [3.0 \ 5.1 \ 4.8]$; % выходы НС регрессии

Воспользуемся функцией *newgrnn* для создания НС регрессии. Зададим разброс радиальных базисных функций (переменная *spread*) меньше, чем 1, для того чтобы получить хорошую аппроксимацию данных и более гладкую функцию.

```

spread = 0.8; % установка разброса радиальных базисных функций
net = newgrnn(P, T, spread); % создание НС регрессии
A = sim (net, P); % имитация работы НС регрессии
plot(P, T, '.', 'markersize', 30); % изображение аппроксимируемой функ-
ции

```

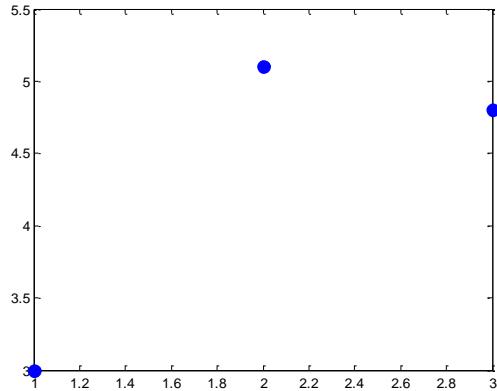


Рисунок 6.6. График аппроксимируемой функции

```

plot(P, A, '.', 'markersize', 30, 'color', [1 0 0]); % изображение работы не-
обученной НС регрессии

```

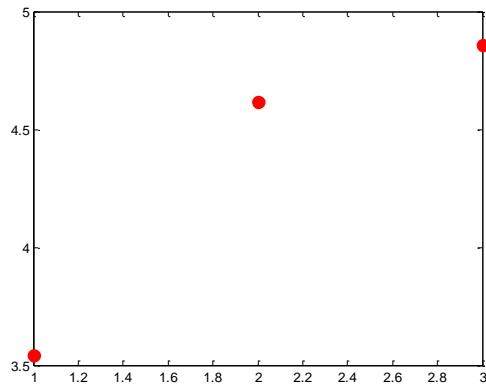


Рисунок 6.7. Выходы необученной нейронной сети

```

cla reset; % очистка координатных осей
p = 4.5; % установка нового входа НС регрессии
a = sim(net, p); % получение отклика НС регрессии
plot(P, T, 'markersize', 30); % изображение аппроксимируемой функ-
ции

```

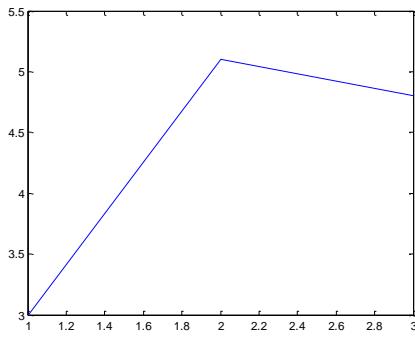


Рисунок 6.8 – Изображение аппроксимируемой функции

```

axis([0 9 3 7]); % установка диапазонов осей X и Y
hold on; % включение режима добавления графиков
plot(p, a, '.', 'markersize', 30, 'color', [1 0 0]); % изображение отклика НС
регрессии на вход p
title( 'Новый входной вектор '); % написать заголовок графика
xlabel('P и p'); % пометить ось X
ylabel('T и a'); % пометить ось Y
cla reset; % очистить координатную сетку
P2 = 0:2:9; % определить последовательность входов P2
Эмулируем отклик сети для различных значений, чтобы увидеть результат
аппроксимации (Рисунок 6.9).
A2 = sim(net, P2); % получить отклик НС регрессии на последователь-
ность входов P2
plot(P2, A2, 'linewidth', 4, 'color', [1 0 0]); % отклик НС регрессии

```

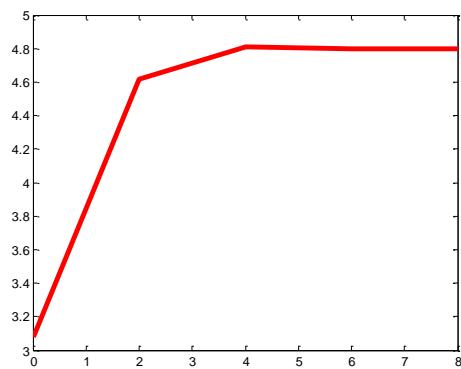


Рисунок 6.9. Отклик нейронной сети регрессии

```

hold on; % включить режим добавления графиков
plot(P, T, '.', 'markersize', 30); % изобразить аппроксимируемую функцию
axis([0 9 3 5.5]); % установить диапазон осей
title('Аппроксимируемая функция'); % озаглавить график
xlabel('P и P2'); % пометить ось X
ylabel('T и A2'); % пометить ось Y

```

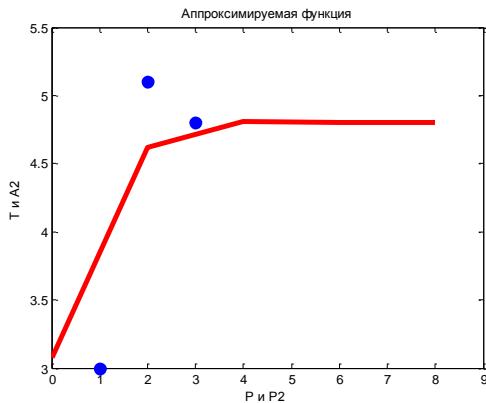


Рисунок 6.10. Аппроксимация точек с помощью нейронной сети регрессии

3. Использование радиальной базисной сети для аппроксимации функций

Рассмотрим аппроксимацию функций на основе радиальной базисной сети.

Пусть

```

p = -4:2:4; % определение диапазона значений радиальной базисной
функции
a = radbas(p); % вычисление радиальной базисной функции на диапазоне
p
plot(p, a); % изображение РБФ
title('Радиальная базисная функция'); % озаглавить график
xlabel('Вход p'); % пометить ось X
ylabel('Выход a'); % пометить ось Y

```

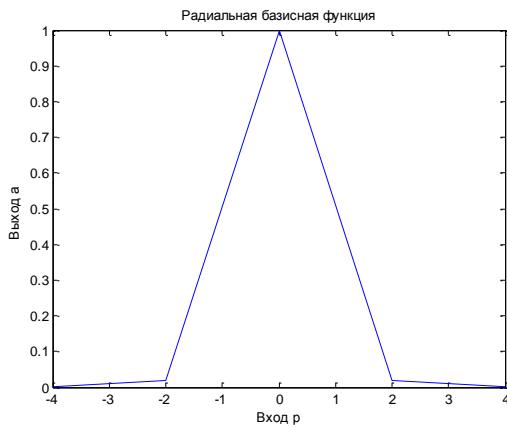


Рисунок 6.11. Радиальная базисная функция

На рисунке 6.11 изображена радиальная базисная функция. Функция *newrbe* создаст необходимую сеть. Зададим аппроксимируемую функцию как набор точек:

P = -1:0.1:1; % определение последовательности аргументов аппроксимируемой функции P

T = [-.6662 -3766 -.1129 .2111 .6565 .3301 .3649 .2006 -.1913 -.3994 -.5022 -.4531 -.1133 .0866 .3333 .4955 .3488 .2833 -.1112 -.6685 -.3255]; % определение значений аппроксимируемой функции, соответствующих P

Изобразим график функции (Рисунок 6.12).

```
plot(P, T, '*'); % изображение аппроксимируемой функции
title('Обучающая выборка'); % озаглавить график
xlabel('Входной вектор P'); % пометить ось X
ylabel('Вектор T'); % пометить ось Y
```

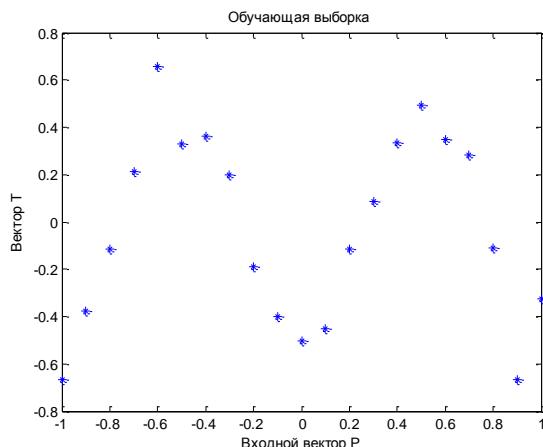


Рисунок 6.12. Обучающая выборка

Далее необходимо найти функцию, которая хорошо описывает заданную 21 точку.

Функция *newrb* создает радиально базисную нейронную сеть для аппроксимации:

e = 0,02; % целевой среднеквадратичной ошибки

sp = 1; % величина разброса радиальной базисной нейронной сети

net = newrb(P, T, e, sp); % создание радиальной базисной сети

X = -1 : .01 : 1; % определение вектора входов

Эмулируем работу сети.

Y = sim (net, X); % формирование отклика Y

hold on; % включение режима добавления графика

plot(X, Y); % изображение результатов аппроксимации

hold off; % отключение режима добавления графика

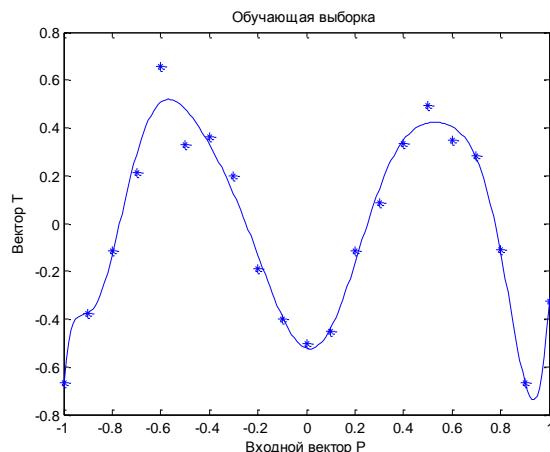


Рисунок 6.13. Результат аппроксимации с РБНС

4. Пример решения аппроксимации функции двух переменных

Пусть функция $z = \exp(x^2) \cdot \cos(-y^3)$ задана на промежутках $x \in [-1; 1]$, $y \in [-1.5; 1.5]$; количество точек разбиений по x есть nx , а по y - ny . Тогда, используя следующий алгоритм построения радиальной базисной сети, можно построить график функции $z = f(x,y)$:

x1 = -1.0; x2 = +1.0; y1 = -1.5; y2 = +1.5;

nx = 30; ny =30 ;

```

step_x = (x2-x1)/(nx-1); step_y = (y2-y1)/(ny-1);
step_min = min(step_x,step_y);
[x,y] = meshgrid([x1:step_x:x2], [y1:step_y:y2]);
z = exp(x.^2).*cos(-y.^3);
surf(x,y,z)
pause;
xx = reshape(x,1,nx*ny);
yy = reshape(y,1,nx*ny);
zz = exp(xx.^2).*cos(-yy.^3);
p = [xx; yy];
t = zz;
goal = 0.05;
spread = 1.0*step_min;
net = newrb(p,t,goal,spread);
net.layers{1}.size
b = sim(net,p);
[zz' b']
c = reshape(b,ny,nx);
surf(x,y,c)

```

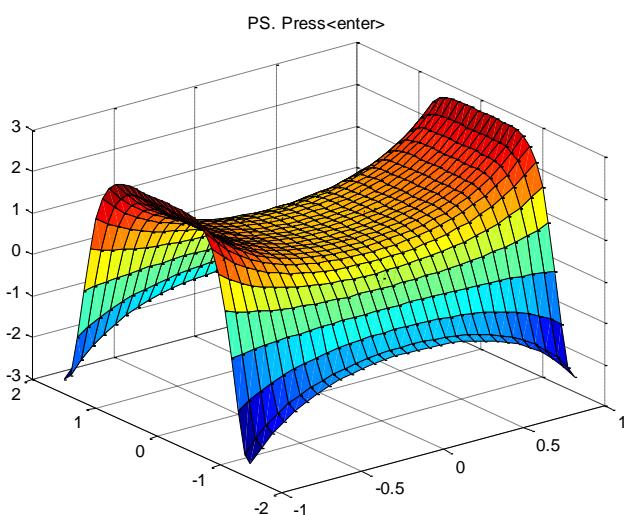


Рисунок 6.14. График исходной функции двух переменных $z = \exp(x^2) \cdot \cos(-y^3)$

На рисунке 6.15 показана характеристика точности обучения радиальной базисной сети и допустимая среднеквадратичная ошибка сети $Goal = 0.05$.

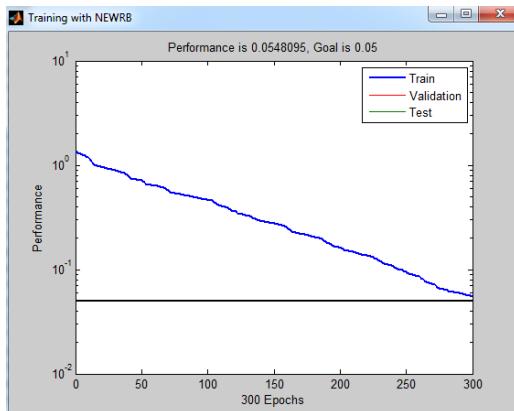


Рисунок 6.15. Характеристика точности обучения в зависимости от количества эпох обучения

На рисунке 6.16 отображён результат аппроксимации нелинейной зависимости, построенный с помощью радиальной базисной функции.

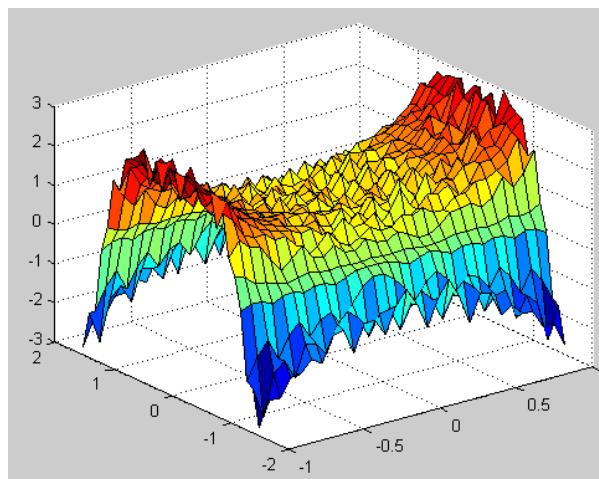


Рисунок 6.16. Результат моделирования исходной функции

Сопоставляя рисунок 6.14 и рисунок 6.16, можно сделать вывод об удовлетворительности полученных результатов. Лучших результатов можно добиться, варьируя параметры $goal$ и $spread$.

5. Использование вероятностной НС для классификации векторов

Рассмотрим задачу классификации с набором входов P и множеством классов, обозначенным V .

```

P = [1 2 3 4 5 6 7]; % определение входов
V = [1 1 3 3 2 1 1]; % определение желаемых выходов
T = ind2vec (V); % конвертирование индексов в векторы, содержащие 1 в
индексных позициях
net = newpnn(P, T); % создание вероятностной НС
Y = sim (net, P); % имитация работы вероятностной НС
W = vec2ind (Y); % конвертирование номеров классов в векторы
W =
1    1    3    3    2    1    1

```

Программа работы и методические указания

1. Создание и исследование моделей сетей GRNN: согласно варианта задания (табл. 6.1) подготовить массивы входных векторов Р и целей Т для нейронной сети GRNN, создать и обучить сеть GRNN для аппроксимации функции.

Таблица 6.1. Варианты задания для создания и исследования сетей GRNN

№ варианта	Значения вектора входа	Значение целевого вектора
1.	[-2 -1 1 0 -1]	[3 1 2 0 -1]
2.	[-2 -1 0 1 2]	[-1 1 2 1 2]
3.	[-4 1 0 1 -2]	[5 1 0 1 2]
4.	[-3 -1 0 1 2]	[-2 1 0 1 -3]
5.	[-2 -1 0 1 2]	[3 -1 0 3 2]
6.	[2 -1 1 -1 2]	[-1 1 1 1 3]
7.	[1 1 -2 2 1]	[2 1 -1 5 3]
8.	[4 1 -1 0 3]	[-3 -1 2 1 5]
9.	[-1 -2 2 2 1]	[3 3 2 -2 4]
10.	[-1 3 1 -2 2]	[3 -2 3 4 -1]
11.	[-1 1 1 1 3]	[-3 -1 0 1 3]
12.	[2 1 -1 5 3]	[-2 -1 0 1 2]
13.	[-3 -1 2 1 5]	[-4 -2 0 2 4]
14.	[3 3 2 -2 4]	[2 -1 1 -1 2]
15.	[-2 -1 0 1 2]	[1 1 -2 2 1]
16.	[-3 -1 0 1 3]	[4 1 -1 0 3]
17.	[3 -1 0 3 2]	[-1 -2 2 2 1]
18.	[-1 1 1 1 3]	[-1 3 1 -2 2]
19.	[2 1 -1 5 3]	[-1 1 1 1 3]
20.	[-3 -1 2 1 5]	[2 1 -1 5 3]

2. Построить модель радиальной базисной сети, аппроксимирующую функцию, приведенную в табл. 6.2.

Таблица 6.2. Варианты аппроксимируемых функций

№ варианта	$z = f(x,y)$
1.	$z = \cos(x)\cos(y), x, y \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
2.	$z = \frac{\sin(x)\sin(y)}{x}, x, y \in [-1,1]$
3.	$z = x^2 y^2, x, y \in [-1.5, 1.5]$
4.	$z = x^3 \sin(y) + 1, x \in [-1,1], y \in [-1.5, 1.5]$
5.	$z = x^2 \cos(y) - 1, x \in [-1,1], y \in [-1.5, 1.5]$
6.	$z = e^x + 3y, x \in [0,1], y \in [-2,1]$
7.	$z = \sqrt{y^2 + x^3}, x \in [-1,1], y \in [-2,2]$
8.	$z = \sqrt{y^3 + x^2}, x \in [-1,1], y \in [-2,2]$
9.	$z = \frac{1}{\sqrt{y^2 + x^3}}, x, y \in [-1,1]$
10.	$z = \sqrt{y^4 + \cos(x^2)}, x, y \in [-1,1]$
11.	$z = x^2 \cos(y) + 1, x \in [-1,1], y \in [-\pi/4, \pi/4]$
12.	$z = x^3 \cos(y) + 1, x \in [-0.5, 0.5], y \in [-\pi/4, \pi/4]$
13.	$z = \sqrt{y^2 + x^4}, x, y \in [-1,1]$
14.	$z = \sin\left(\frac{y}{(1+x^2)}\right) + 1, x \in [-1,1], y \in [-\pi, \pi]$
15.	$z = \sin(x^2) + y^2, x \in [-1.77, 1.77], y \in [-0.3, 0.3]$
16.	$z = \cos(x^2) + xy, x \in [-0.886, 0.886], y \in [-0.3, 0.3]$
17.	$z = (x-1)^2 \cdot \exp(y^2), x \in [0.5, 2], y \in [-1,1]$
18.	$z = \frac{\cos(x)}{x^2 + 1}, x \in [-5, 5], y \in [-\pi/2, \pi/2]$
19.	$z = \sin(x) \cdot \sin(y), x, y \in [-\pi, 0]$
20.	$z = \ln(x) \cdot \ln(y), x, y \in [0.5, 2]$

Контрольные вопросы

1. Какую функцию называют радиальной базисной функцией?
2. Из каких слоев состоит радиально–базисная НС, НС регрессии, вероятностная НС?
3. Какие виды НС предназначены для решения задачи аппроксимации функций, а какие – для классификации объектов?

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Лабораторная работа №7

ИССЛЕДОВАНИЕ СЕТИ КОХОНЕНА И АЛГОРИТМА ОБУЧЕНИЯ БЕЗ УЧИТЕЛЯ

Цель работы: изучение архитектуры самоорганизующихся нейронных слоев Кохонена, а также приобретение навыков построения самоорганизующихся слоев для исследования топологической структуры данных, их объединением в кластеры (группы) и распределением по классам.

1. Краткие теоретические сведения

Сети Кохонена, или самоорганизующиеся карты (*Kohonen maps*), предназначены для решения задач автоматической классификации, когда обучающая последовательность образов отсутствует (обучение без учителя). Сеть Кохонена является двухслойной. Она содержит слой входных нейронов и собственно слой Кохонена. Слой Кохонена может быть одномерным, двумерным и трехмерным. В первом случае нейроны расположены в цепочку; во втором – они образуют двумерную сетку (обычно в форме квадрата или прямоугольника), а в третьем – трехмерную систему. Определение весов нейронов слоя Кохонена

основано на использовании алгоритмов автоматической классификации (кластеризации или самообучения).

На вход сети подаются последовательно значения векторов $\mathbf{x}_l = (x_1, x_2, \dots, x_n)_l$, представляющих отдельные последовательные наборы данных для поиска кластеров, то есть различных классов образов, причем число этих кластеров заранее неизвестно. На стадии обучения (точнее самообучения) сети входной вектор \mathbf{x} попарно сравнивается со всеми векторами \mathbf{w}_j всех нейронов сети Кохонена. Вводится некоторая функция близости d (например, в виде евклидова расстояния). Активный нейрон с номером c слоя Кохонена, для которого значение функции близости $d(\mathbf{x}, \mathbf{w}_c)$ между входным вектором \mathbf{x} , характеризующим некоторый образ, к векторам \mathbf{w}_c максимально, объявляется «победителем». При этом образ, характеризующийся вектором \mathbf{x} , будет отнесен к классу, который представляется нейроном-«победителем».

Рассмотрим алгоритм самообучения сетей Кохонена. Обозначим функцию близости $z = \|\mathbf{x} - \mathbf{w}\|$. Выигрывает нейрон с

$$\|\mathbf{x} - \mathbf{w}_c\| = \min \|\mathbf{x} - \mathbf{w}_j\|. \quad (7.1)$$

Близость \mathbf{x}' и \mathbf{w}' можно переделить, пользуясь скалярным произведением, как

$$z_j = 1 - (\mathbf{x}', \mathbf{w}'_j) = 1 - \sum_{i=1}^n x'_i w'_{ij}. \quad (7.2)$$

На стадии самообучения сети Кохонена осуществляется коррекция весового вектора не только нейрона-«победителя», но и весовых векторов остальных активных нейронов слоя Кохонена, однако, естественно, в значительно меньшей степени – в зависимости от удаления от нейрона-«победителя». При этом форма и величина окрестности вокруг нейрона-«победителя», весовые коэффициенты нейронов которой также корректируются, в процессе обучения изменяются. Сначала начинают с очень большой области – она, в частности, может включать все нейроны слоя Кохонена. Изменение весовых векторов осуществляется по правилу

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t) d_{cj}(t) [\mathbf{x}(t) - \mathbf{w}_j(t)], \quad j = 1, 2, \dots, n, \quad (7.3)$$

где $\mathbf{w}_j(t)$ - значение весового вектора на шаге t самообучения сети, $d_{cj}(t)$ - функция близости между нейронами слоя Кохонена и $\eta(t)$ - изменяемый во времени коэффициент шага коррекции. В качестве $\eta(t)$ обычно выбирается монотонно убывающая функция ($0 < \eta(t) < 1$), то есть алгоритм самообучения начинается сравнительно большими шагами адаптации и заканчивается относительно малыми изменениями.

В результате n -мерное входное пространство R_n отобразится на m -мерную сетку (слой Кохонена). Это отображение реализуется в результате рекуррентной (итеративной) процедуры самообучения (*unsupervised learning*). Отличительная особенность этого отображения – формирование кластеров (clusters) или классов. По завершении процесса самообучения на стадии реального использования сети Кохонена неизвестные входные образы относятся к одному из выявленных кластеров (классов) по близости к некоторому весу, принадлежащему определенному кластеру, выявленному на стадии самообучения.

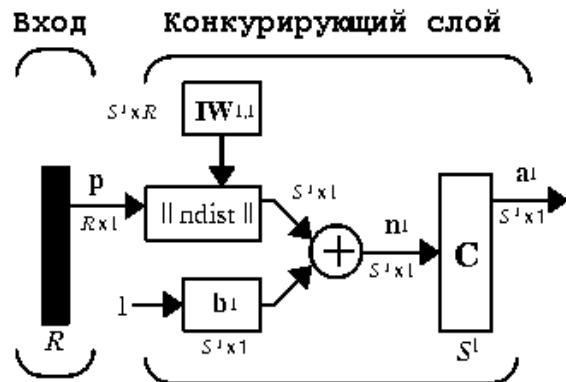


Рисунок 7.1. Архитектура слоя Кохонена

Самоорганизующийся слой Кохонена – это однослойная нейронная сеть с конкурирующей передаточной функцией *compet*, которая анализирует выходные значения нейронов слоя и выдаёт в качестве результата наибольшее из этих значений (значение нейрона-победителя). Функция *newc* создает слой конкурирующих нейронов (слой Кохонена):

`net = newc(PR, S, KLR, CLR),`

где PR – матрица минимальных и максимальных значений для R входных элементов; S – количество нейронов; KLR – уровень обученности Кохонена, по умолчанию 0,01; CLR – рекомендуемый уровень обученности, по умолчанию 0,001. Соперничающий слой представляет собой слой с функцией вычисления расстояний *negdist*, сетевой функцией суммирования *netsum* и функцией активации *compet*. Слой имеет веса на входах и *bias*.

Эта функция формирует однослойную сеть с R нейронами и R входами. Веса входов устанавливаются равными половине диапазона соответствующего вектора входа для всех нейронов. Также для всех нейронов устанавливается одно и то же смещение, равное e^*s . Выходы нейронов поступают на конкурирующую передаточную функцию *compet*, которая определяет победителя. Номер активного нейрона-победителя I^* определяет ту группу (кластер), к которой наиболее близок входной вектор.

Для того чтобы сформированная таким образом сеть решала задачу кластеризации данных, необходимо предварительно настроить ее веса и смещения по обучающей последовательностью векторов с помощью функций настройки *learnk* и *learncon* соответственно, используя процедуру адаптации *adapt* или процедуру обучения *train*.

Функция *learnk* рассчитывает приращение весов dW в зависимости от вектора входа p , выхода a , весов w и параметра скорости настройки lr в соответствии с правилом Кохонена:

$$dw = \begin{cases} lr \cdot (p - w), & a_j \neq 0; \\ 0, & a_j = 0. \end{cases} \quad (7.4)$$

Таким образом, вектор веса, наиболее близкий к вектору входа, модифицируется так, чтобы расстояние между ними стало ещё меньше. Результат такого обучения заключается в том, что победивший нейрон, вероятно, выиграет конкуренцию и в том случае, когда будет представлен новый входной вектор, близкий к предыдущему, и его победа менее вероятна, когда будет представлен вектор, существенно отличающийся от предыдущего. Когда на вход сети по-

ступает всё большее и большее число векторов, нейрон, являющийся ближайшим, снова корректирует свой весовой вектор w . В конечном счёте, если в слое имеется достаточное количество нейронов, то каждая группа близких векторов окажется связанной с одним из нейронов слоя. В этом и заключается свойство самоорганизации слоя Кохонена.

Одно из ограничений всякого конкурирующего слоя состоит в том, что некоторые нейроны оказываются нездействованными, или “мертвыми”. Это происходит оттого, что нейроны, имеющие начальные весовые векторы, значительно удаленные от векторов входа, никогда не выигрывают конкуренции, независимо от продолжительности обучения. Для ликвидации нечувствительности таких нейронов используют положительные смещения, которые добавляются к отрицательным расстояниям удаленных нейронов. Функция *learncon* производит такую корректировку смещений.

В отличие от слоя Кохонена карта Кохонена после обучения поддерживает такое топологическое свойство, когда близким входным векторам соответствуют близко расположенные активные нейроны.

Первоначальная топология размещения нейронов в карте Кохонена формируется при создание карты с помощью функции *newsom*, одним из параметров которого является имя топологической функции *gridtop*, *nextop* или *randtop*, что соответствует размещению нейронов в узлах либо прямоугольной, либо гексагональной сетки, либо в узлах сетки со случайной топологией.

Функция *newsom* создает самоорганизующуюся сеть – карту (*SOM*):

```
net = newsom(PR, [D1, D2, ...], TFCN, DFCN, OLR, OSTEPS, TLR, TND),
```

где PR – матрица минимальных и максимальных значений для R входных элементов; Di – размер i -го слоя; $TFCN$ – функция топологии; $DFCN$ – функция расстояния; OLR – уровень обученности фазы упорядочения, по умолчанию 0,9; $OSTEPS$ – шаги фазы упорядочения, по умолчанию 1000; TLR – уровень обученности фазы настройки, по умолчанию 0,02; TND – расстояние соседства фазы настройки, по умолчанию 1.

Настройка карты Кохонена производится по каждому входному вектору независимо от того, применяется метод адаптации или метод обучения. В любом случае функция *learnsom* выполняет настройку карты нейронов.

Прежде всего определяется нейрон-победитель и корректируется его вектор весов и векторы соседних нейронов согласно соотношению

$$dw = lr \cdot A2 \cdot (p' - w), \quad (7.5)$$

где lr – параметр скорости обучения, равный olr для этапа упорядочения нейронов и tlr для этапа подстройки; $A2$ – массив соседства для нейронов, расположенных в окрестности нейрона-победителя i :

$$A2(i, q) = \begin{cases} 1, & a(i, q) = 1; \\ 0.5, & a(i, q) = 1 \& D(i, j) \leq nd; \\ 0, & \text{в остальных случаях.} \end{cases} \quad (7.6)$$

Здесь $a(i, q)$ – элемент выхода нейронной сети; $D(i, j)$ – расстояние между нейронами i и j ; nd – размер окрестности нейрона-победителя.

Таким образом, вес нейрона-победителя изменяется пропорционально половинному параметру скорости обучения, а веса соседних нейронов – пропорционально половинному значению этого параметра.

Весь процесс обучения карты Кохонена делится на два этапа:

- 1) этап упорядоченности векторов весовых коэффициентов в пространстве признаков;
- 2) этап подстройки весов нейронов по отношению к набору векторов входа.

На этапе упорядочения используется фиксированное количество шагов. Начальный размер окрестности назначается равным максимальному расстоянию между нейронами для выбранной топологии и затем уменьшается до величины, используемой на следующем этапе, и вычисляется по следующей формуле:

$$nd = 1.00001 + (\max(d) - 1)(1 - s / S), \quad (7.7)$$

где $\max(d)$ – максимальное расстояние между нейронами; s – номер текущего шага, а S – количество циклов на этапе упорядочения.

Параметр скорости обучения изменяется по правилу

$$lr = trl + (orl - trl)(1 - s / S). \quad (7.8)$$

На этапе подстройки, который продолжается в течение оставшейся части процедуры обучения, размер окрестности остается постоянным и равным

$$nd = tnd + 0.00001, \quad (7.9)$$

а параметр скорости обучения изменяется по следующему правилу

$$lr = tlr \cdot S / s. \quad (7.10)$$

Параметр скорости обучения продолжает уменьшаться, но очень медленно. Малое значение окрестности и медленное уменьшение параметра скорости обучения хорошо настраивают сеть при сохранении размещения, найденного на предыдущем этапе. Число шагов на этапе подстройки должно значительно превышать число шагов на этапе размещения. На этом этапе происходит тонкая настройка весов нейронов по отношению к набору векторов входов.

Нейроны карты Кохонена будут упорядочиваться так, чтобы при равномерной плотности векторов входа нейроны также были распределены равномерно. Если векторы входа распределены неравномерно, то и нейроны будут иметь тенденцию распределяться в соответствии с плотностью размещения векторов входа.

Таким образом, при обучении карты Кохонена решается не только задачи кластеризации входных векторов, но и выполняется частичная классификация.

2. Применение нейронной сети Кохонена для кластеризации

Сформируем координаты случайных точек и построим их расположение на плоскости. Для этого в m-файле необходимо создать следующую программу

c = 8; %Число кластеров

n = 6; %Число векторов в кластере

d = .5; %Среднее отклонение от центра кластера

```

x = [-10 10; -5 5];
[r,q] = size(x);
minv = min(x)';
maxv = max(x)';
v = rand(r,c).*((maxv-minv)*ones(1,c)+minv*ones(1,c));
t = c*n; % число точек
v = [v v v v v v];
v = v+randn(r,t)*d; % координаты точек
P = v;

```

При помощи средства NNTools Manager импортируем данные.

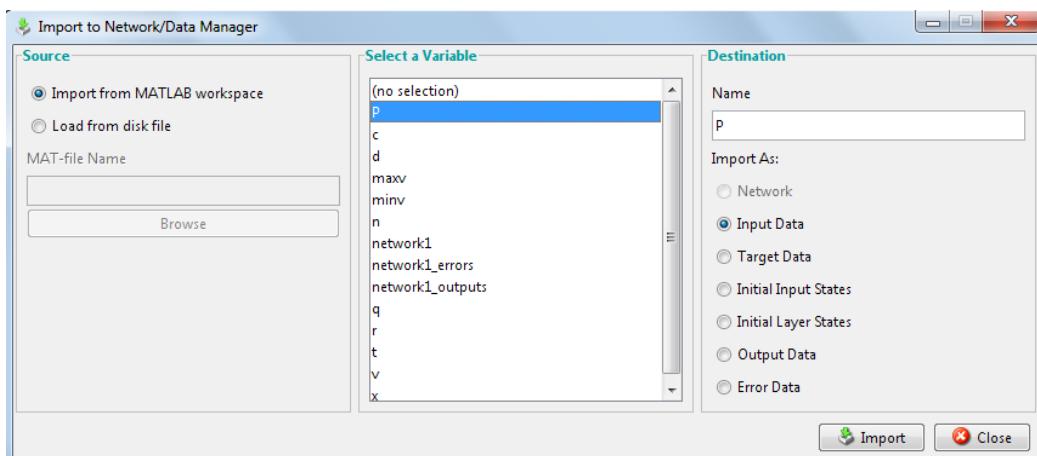


Рисунок 7.2. Окно для импорта и загрузки данных

Для распределения этих данных по классам создадим однослойную нейронную сеть Кохонена.

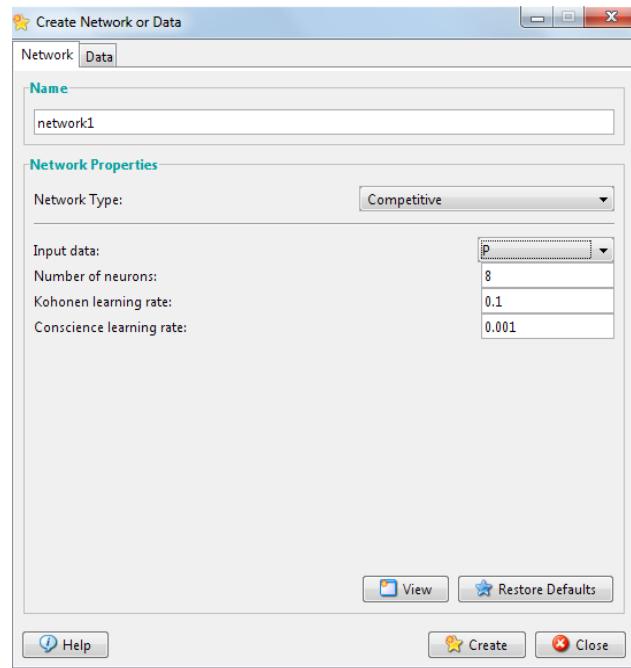


Рисунок 7.3. Окно создания нейронной сети

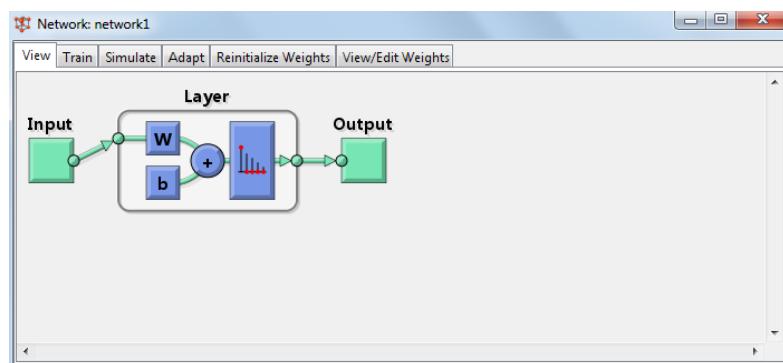


Рисунок 7.4. Диалоговая панель Network

Проведем инициализацию весов сети, установим диапазон изменения входных данных и проведем обучение сети.

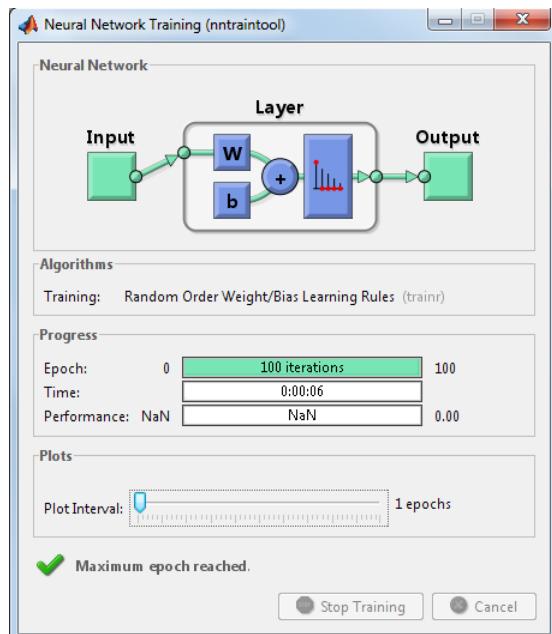


Рисунок 7.5. Окно обучения нейронной сети

Для иллюстрации обучения построим расположение входных данных.

plot (P(1, :), P(2, :), '+r') % рисунок 7.6

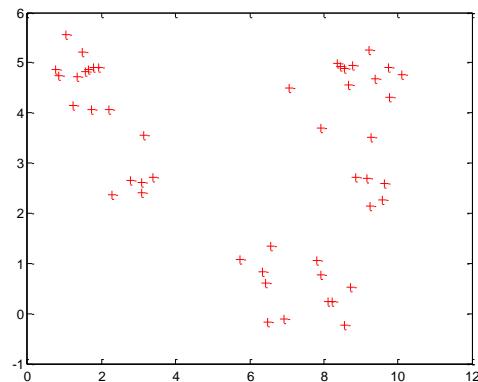


Рисунок 7.6. Распределение входных данных

Построим центры классифицируемых областей. Для чего необходимо экспортировать нейронную сеть в рабочую область системы MatLab.

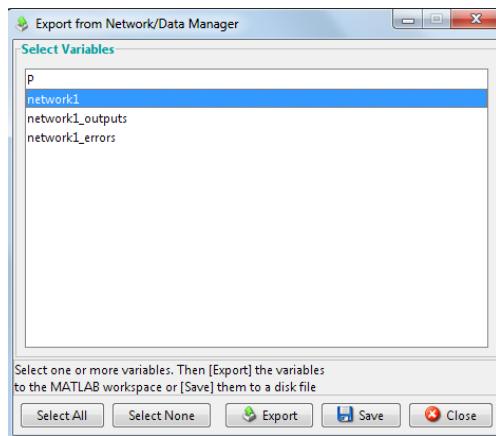


Рисунок 7.7. Окно для импорта и загрузки данных

```
hold on; plot (network1.IW{1}(:, 1), network1.IW{1}(:, 2) , 'ob')
```

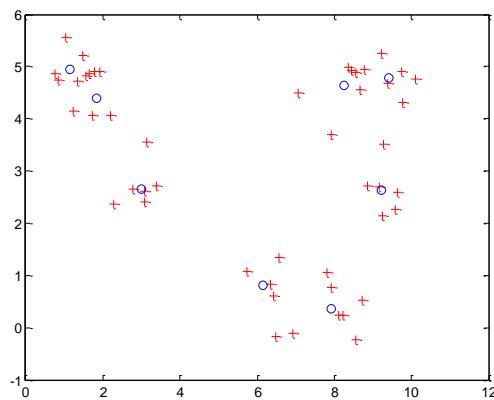


Рисунок 7.8. Распределение входных данных (крестики) и положение центров кластеризации (символ «о»)

Для отнесения двумерного вектора к тому или иному классу необходимо задать его в системе MatLab и симулировать им сеть.

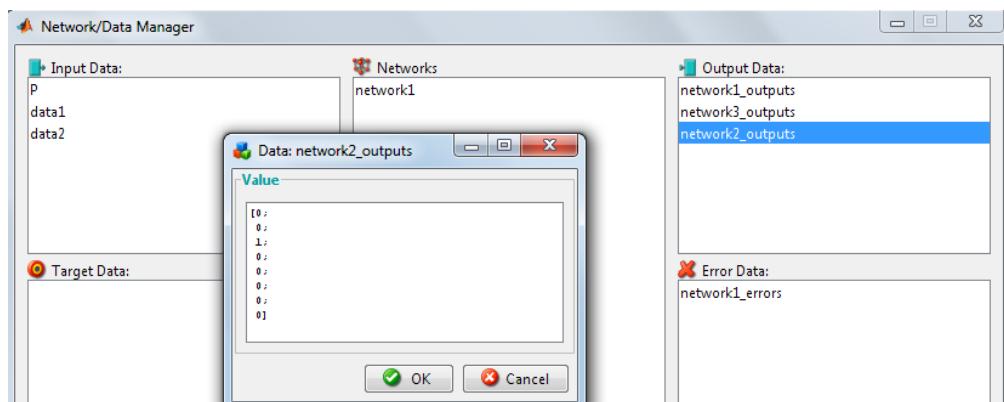


Рисунок 7.9. Выходные значения нейронов, соответствующие номеру класса

data2 = [1.7; 8.7];

```
a = sim(network1, data1)
```

```
a =
```

```
(3,1)    1
```

Возбужденный нейрон показывает номер класса. В нашем случае входной вектор data2 отнесен к третьему классу.

3. Создание и использование соперничающего слоя в командном окне

3.1 Зададим множество, состоящее из четырех двухэлементных векторов P:

```
P = [.2 .5 .1 .8; .3 .7 .2 .6]; % определение входного вектора
```

Соперничающий слой должен разделить входы на два класса:

```
net = newc ([0 1; 0 1], 2); % создание соперничающего слоя
```

```
net = train (net, P); % обучение соперничающего слоя на векторе P
```

Результирующая сеть выдает выходной вектор, конвертируемый в номер класса:

```
Y = sim (net, P) % имитация работы слоя
```

```
Y =
```

```
(2,1)    1
```

```
(1,2)    1
```

```
(2,3)    1
```

```
(1,4)    1
```

$Y_c = \text{vec2ind}(Y)$ % конвертирование выхода сети в номер класса; Y состоит из векторов, содержащих только по одной единице. Функция *vec2ind* показывает индексы единичных позиций.

```
Yc =
```

```
2 1 2 1
```

3.2 Пример решения задачи кластеризации векторов

Пусть заданы 30 случайных векторов, изображённых на графике снежинками (рисунок 7.10). Оцифровав данный график, можно получить массив входных данных $P(1,:)$, $P(2,:)$ (табл. 7.1).

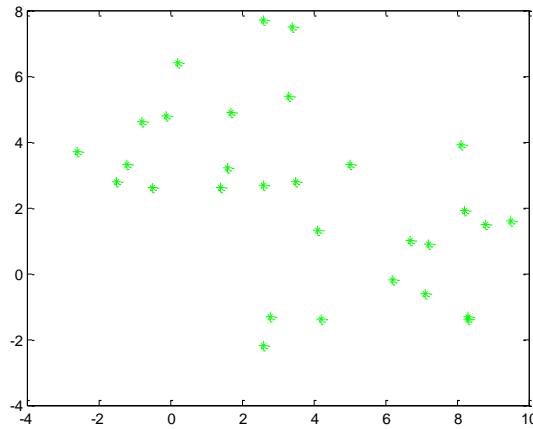


Рисунок 7.10. Распределение входных векторов

Таблица 7.1. Массив входных данных

P(1,:)	-1.2	-0.5	1.4	2.6	9.5	2.8	7.1	2.6	0.2	2.6
P(2,:)	3.3	2.6	2.6	2.7	1.6	-1.3	-0.6	-2.2	6.4	7.7
P(1,:)	-1.5	8.1	6.2	7.2	3.5	-0.8	3.4	1.6	8.3	5.0
P(2,:)	2.8	3.9	-0.2	0.9	2.8	4.6	7.5	3.2	-1.4	3.3
P(1,:)	6.7	1.7	-0.1	3.3	4.1	8.8	-2.6	8.3	4.2	8.2
P(2,:)	1.0	4.9	4.8	5.4	1.3	1.5	3.7	-1.3	-1.4	1.9

Следующий алгоритм демонстрирует процедуру обучения самоорганизующейся нейронной сети Кохонена.

```

plot(P(1,:), P(2,:), '*g');
title('Input vectors');
xlabel('P(1,:);');
ylabel('P(2,:);');
hold on;
nclusters = 6;
a1 = -3;
a2 = 10;
b1 = -3;
b2 = 9;
net = newc([a1 a2; b1 b2], nclusters, 0.1, 0.0005);
wo = net.IW{1};

```

```

bo = net.b{1};

net.trainParam.epochs=100;
net.trainParam.show=20;
net = train(net,P);
w = net.IW{1};
bn = net.b{1};
plot(w(:,1),w(:,2),'kp');

```

На рисунке 7.11 представлены исходные данные (снежинки) и полученные центры кластеризации (звезды).

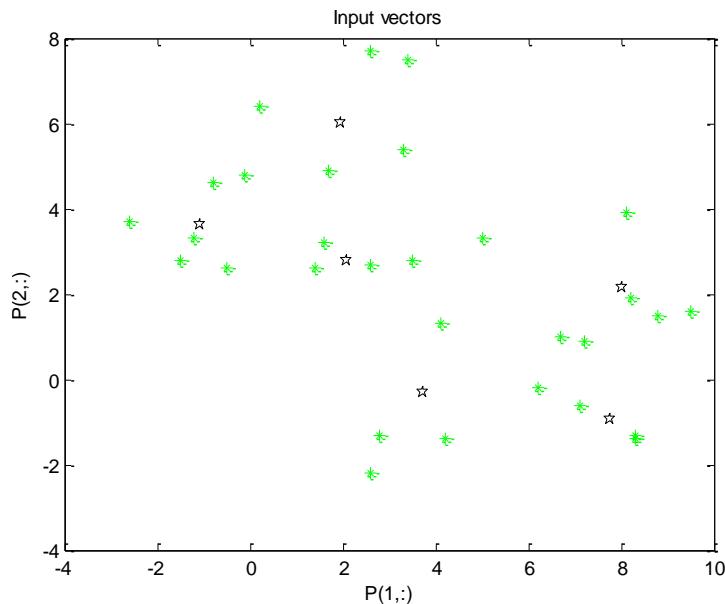


Рисунок 7.11. Распределение входных данных (крестики) и положение центров кластеризации (звездочки)

4. Применение SOM для кластеризации векторов

Рассмотрим работу SOM на примере задачи кластеризации 1000 двухэлементных векторов, расположенных в ограниченной прямоугольной области (Рисунок 7.12).

```

P = rands(2, 1000); % определение множества векторов для кластеризации
plot(P(1,:), P(2,:), '*g') % изображение множества векторов

```

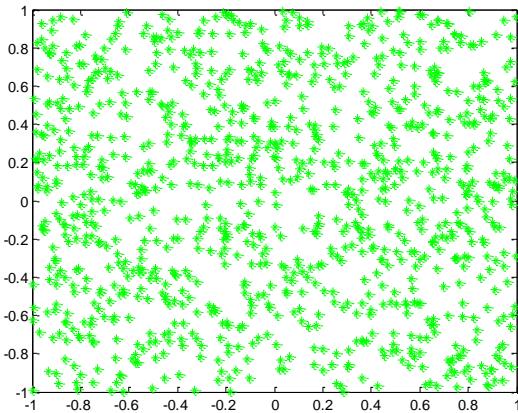


Рисунок 7.12. Множество исходных векторов для кластеризации

```
net = newsom ([0 1; 0 1], [5 6]); % создание SOM размерностью 5x6.
```

Используем сеть размерностью 5x6, чтобы классифицировать вышеопределенные векторы. Создадим сеть с 30 центрами, к которым сеть будет относить каждый вектор на основе ближайшего соседства.

```
plotsom(net.iw{1,1}, net.layers{1}.distances); % визуализация SOM
```

Сначала центр прямоугольника совпадает с центрами классов. Обучим сеть на 100 эпохах и изобразим результат:

```
net.trainParam.epochs = 100; % установка количества эпох
```

```
net = train(net, P) % обучение сети
```

Во время обучения появляется панель инструментов (рисунок 7.13).

Есть несколько полезных визуализаций, доступ к которым можно получить из этого окна. При нажатии *SOM Weight Positions*, появляется фигура (рисунок 7.14), которая показывающая расположение точек данных и весовых векторов. Как показано на рисунке, только после 100 итераций алгоритма, веса синапсов настроились таким образом, чтобы узлы решетки «располагались» в местах локальных сгущений данных, то есть описывали кластерную структуру облака данных.

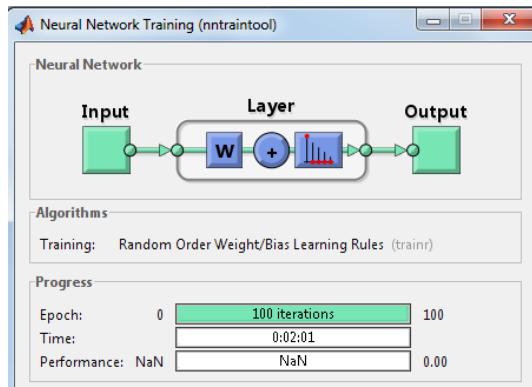


Рисунок 7.13. Окно обучения нейронной сети

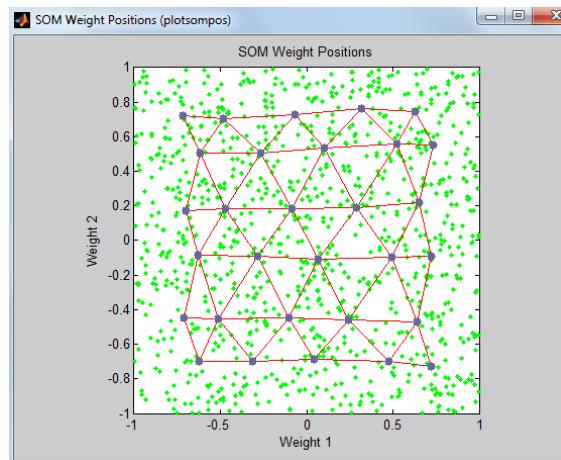


Рисунок 7.14. Расположение точек данных и весовых векторов

При высокой размерности входа, нельзя одновременно визуализировать все веса. В этом случае, можно нажать *SOM Neighbor Distances* (рисунок 7.15).

Здесь используется следующее кодирование цвета: синие шестиугольники представляют собой нейроны. Красные линии соединяют соседние нейроны. Цвета в областях, содержащих красные линии указывают расстояния между нейронами. Более темные цвета представляют большие расстояния. Чем светлее цвета, тем меньше расстояния. Сравнивая рисунки 7.15 и 7.15, убеждаемся в их идентичности.

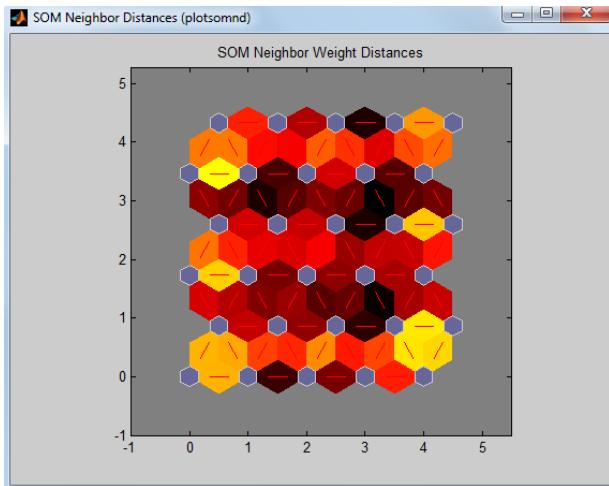


Рисунок 7.15. Сила взаимодействия (веса нейронов) соседних нейронов в карте Кохонена

Нажав *SOM Sample Hits* можно увидеть количество точек данных, связанных с каждым нейроном (рисунок 7.16). Лучше всего, если данные достаточно равномерно распределены между нейронами. В этом примере, данные менее связаны с центральными нейронами, но в целом распределение удовлетворительное.

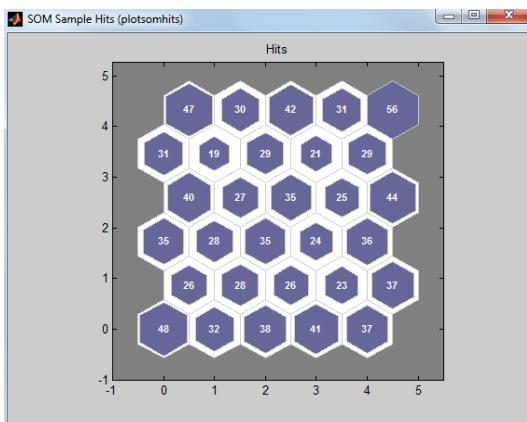


Рисунок 7.16. Количество точек данных, связанных с каждым нейроном

Визуализацию весов, соединяющих каждый вход с каждым из нейронов, можно увидеть нажав *SOM Weight Planes* (рисунок 7.17). Здесь темные цвета представляют большие веса. Если схемы соединения из двух входов очень похожи, можно предположить, что входы высоко коррелированы. В нашем случае вход 1 имеет связи, которые очень отличаются от входа 2.

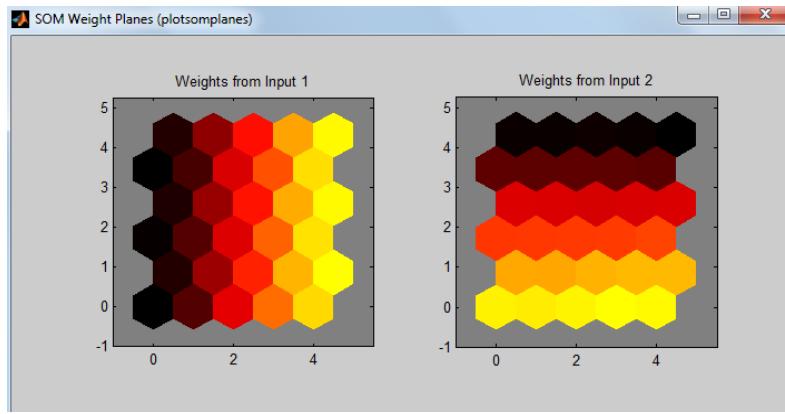


Рисунок 7.17. Визуализация весов, соединяющих каждый вход с каждым из нейронов

```
plotsom(net.iw{ 1,1}, net.layers{1}.distances) % визуализация SOM
```

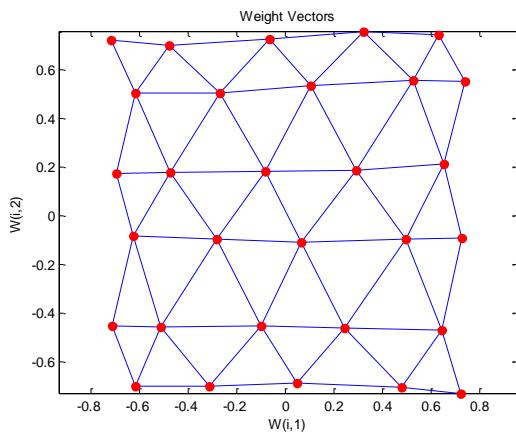


Рисунок 7.18. Самоорганизующаяся карта кластеризации

```
p = [0.3; 0.5]; % определение нового вектора
a = 0; % начальная установка переменной a
a = sim(net, p) % получение отклика SOM на вход p
a =
(8,1)    1
```

Возбужденный нейрон показывает номер кластера. В нашем случае входной вектор p отнесен к восьмому кластеру.

Программа работы и методические указания

- Создать слой Кохонена для двух векторов входа, проанализировать его структурную схему и параметры вычислительной модели, произвести обучение сети и моделирование, выполнив следующие команды:

```

P = [.1 .8 .1 .9; .2 .9 .1 .8]; % для обучения слоя;
net = newc([01;01],2); % создание слоя;
gensim (net); % структура слоя;
net = train(net,P); % обучение слоя;
w = net.Iw{1,1}; % веса после обучения;
b = net.b{1}; % смещение после обучения;
plot(P(1,:),P(2,:),'+k')
title ('Векторы входа'), xlabel('P(1,:)'), ylabel('P(2,:)')
hold on
plot (w, 'or')
P1= [0.2:0.1:0.7; 0.2:0.1:0.7];
y = sim(net,P1)
yc = vec2ind(Y)

```

2. Создать одномерную карту Кохонена из 10 нейронов, обучить её на последовательности из 100 двухэлементных векторов единичной длины, распределенных равномерно в пределах от 0 до 90° , построить график распределения векторов по кластерам и выполнить моделирование сети для одного вектора входа, выполнив следующие команды:

```

angels=0 : 0.5 +pi/99 : 0.5*pi;
p=[sin(angels); cos(angels)];
plot(P(1, 1:10:end), P(2, 1:10:end), '*8')
hold on
net=newsom([0 1 ;0 1], [10]);
net.trainparam.epochs=2000;
net.trainparam.show=100;
[net,tr]=train(net,P);
plotsom(net.iw{1,1}, net.layers{1}.distances)
figure(2)
a=sim(net,P) % – моделирование на обучающем

```

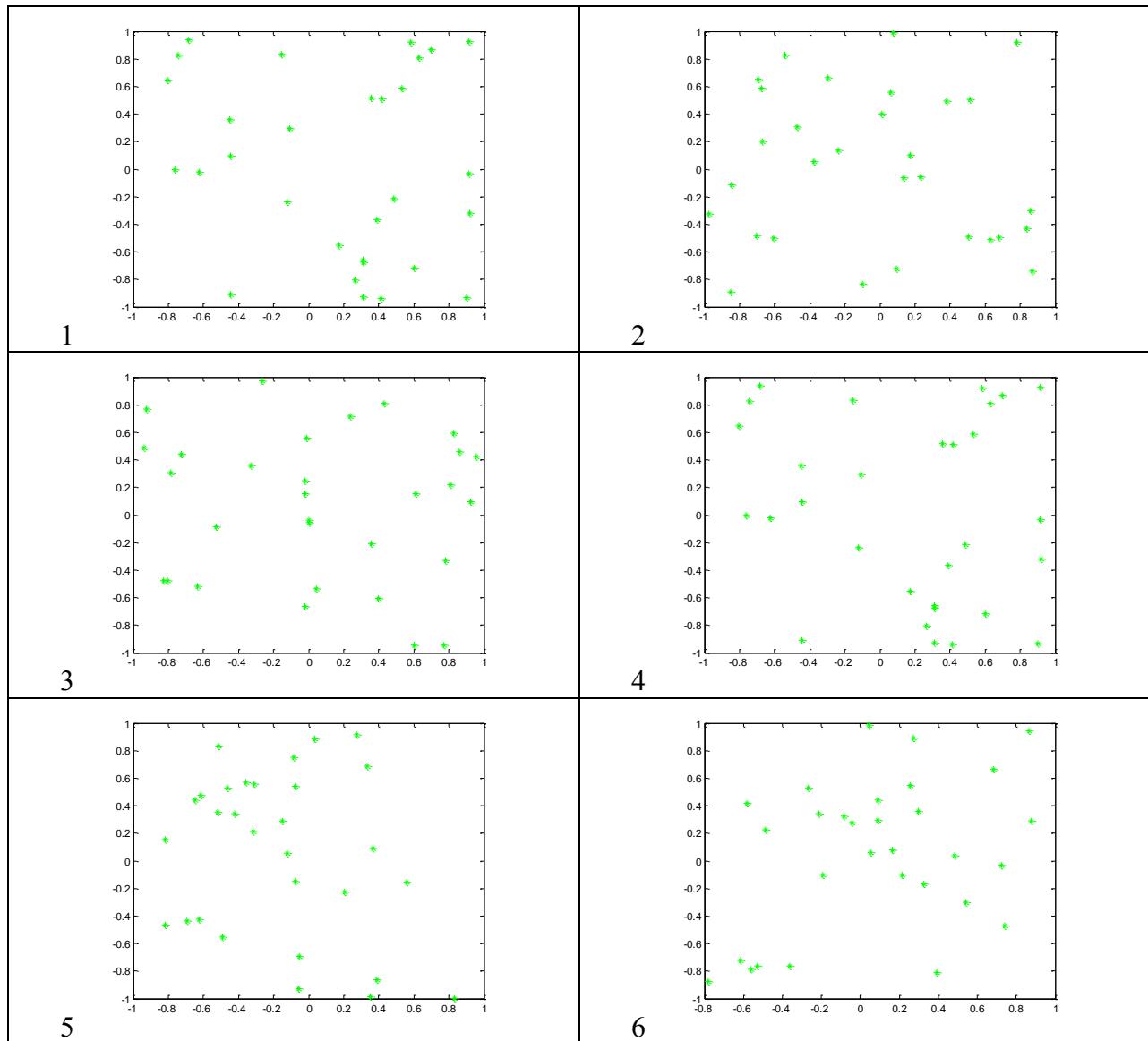
% множество и построение

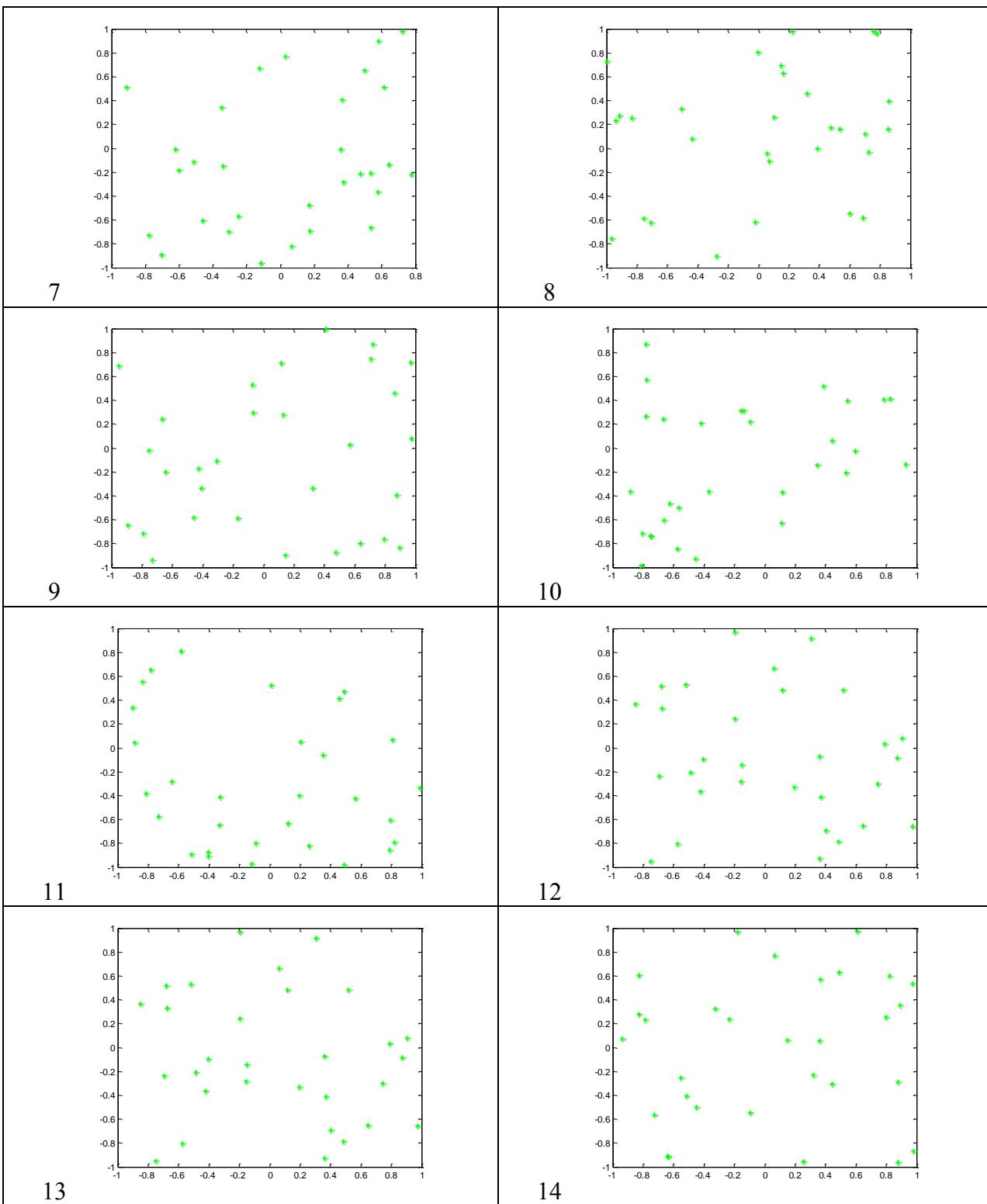
% столбцовой диаграммы.

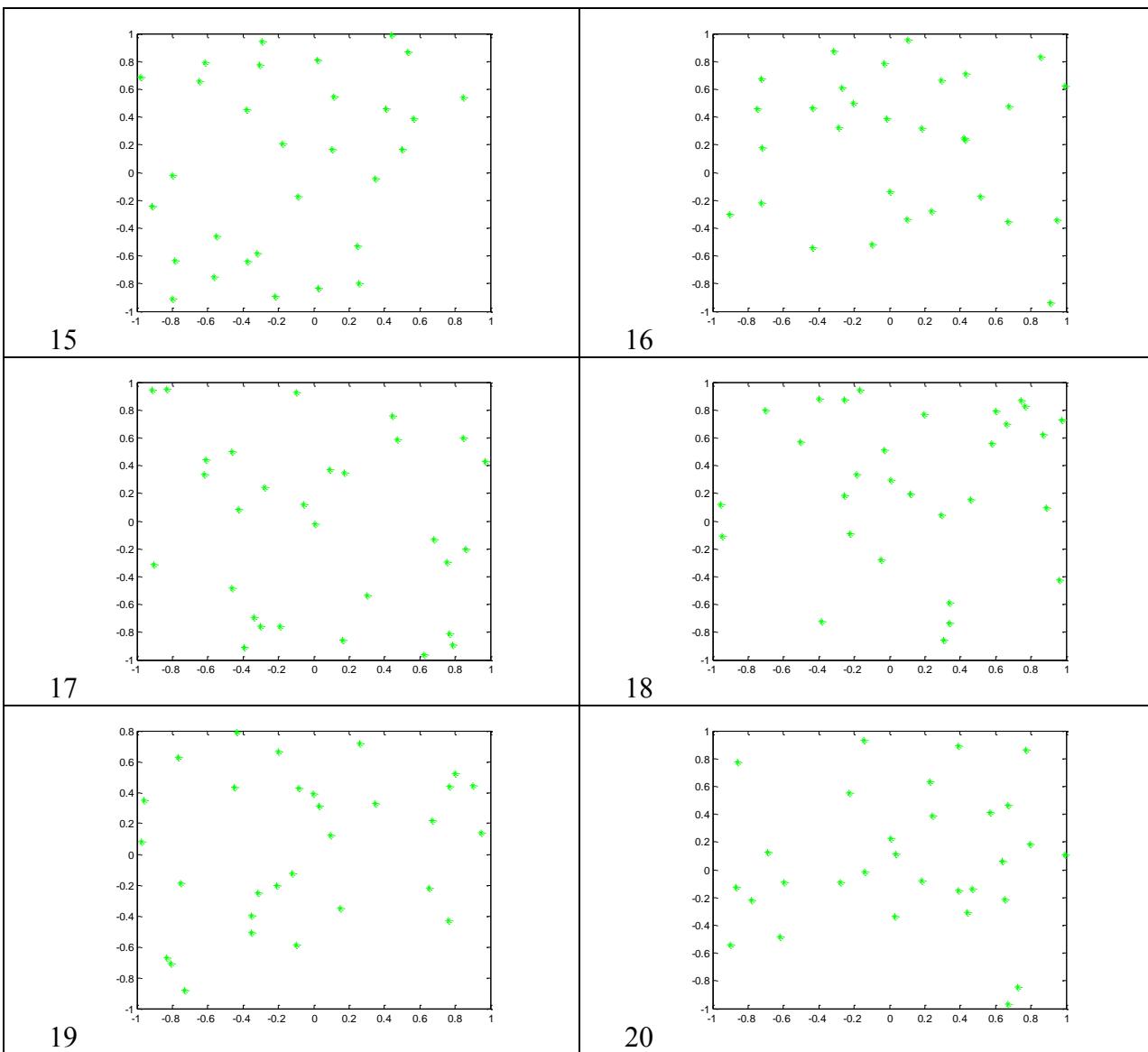
a=sim(net,[1;0]) % – отнесен к 10-му кластеру

3. Оцифровать и провести кластеризацию данных (табл. 2), используя слой Кохонена и самоорганизующуюся карту SOM. Вариант соответствует номеру по журналу.

Таблица 2. Данные для проведения кластерного анализа







Контрольные вопросы

1. В чем заключается задача кластеризации?
2. Какую структуру имеет НС Кохонена?
3. Каким алгоритмом обучается НС Кохонена?

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Лабораторная работа №8

ИССЛЕДОВАНИЕ СЕТИ ХОПФИЛДА

Цель работы: исследовать устойчивость сети Хопфилда и её сходимость к аттракторам, применение сети Хопфилда для распознавания образов.

1. Краткие теоретические сведения

Сети с обратными связями имеют пути, передающие сигналы от выходов к входам, то отклик таких сетей является динамическим, т. е. после приложения нового входа вычисляется выход и, передаваясь по сети обратной связи, модифицирует вход. Затем выход повторно вычисляется, и процесс повторяется снова и снова. Для устойчивой сети последовательные итерации приводят к все меньшим изменениям выхода, пока в конце концов выход не становится постоянным. Для многих сетей процесс никогда не заканчивается, такие сети называют неустойчивыми. Неустойчивые сети обладают интересными свойствами и изучались в качестве примера хаотических систем. Однако такой большой предмет, как хаос, находится за пределами этой книги. Вместо этого мы сконцентрируем внимание на устойчивых сетях, т. е. на тех, которые в конце концов дают постоянный выход.

Проблема устойчивости ставила в тупик первых исследователей. Никто не был в состоянии предсказать, какие из сетей будут устойчивыми, а какие будут находиться в постоянном изменении. Более того, проблема представлялась столь трудной, что многие исследователи были настроены пессимистически относительно возможности без решения.

Дж. Хопфилд сделал важный вклад как в теорию, так и в применение систем с обратными связями. Поэтому некоторые из конфигураций известны как сети Хопфилда. Названные в его честь сети Хопфилда являются рекуррентными или сетями с обратными связями и предназначены для распознавания образов. Обобщенная структура этой сети представляется, как правило, в виде системы с обратной связью выхода с входом. В сети Хопфилда входные сигналы нейронов являются одновременно и выходными сигналами сети: $x_i(k) = y_i(k-1)$,

при этом возбуждающий вектор особо не выделяется. В классической системе Хопфилда отсутствует связь нейрона с собственным выходом, что соответствует $w_{ii} = 0$, а вся матрица весов является симметричной: $w_{ij} = w_{ji}$

$$\hat{W} = \hat{W}^T. \quad (8.1)$$

Симметричность матрицы весов гарантирует сходимость процесса обучения. Процесс обучения сети формирует зоны притяжения некоторых точек равновесия, соответствующих обучающим данным. При использовании ассоциативной памяти мы имеем дело с обучающим вектором $\mathbf{x} = (x_1, x_2, \dots, x_n)$, либо с множеством этих векторов, которые в результате проводимого обучения определяют расположение конкретных точек притяжения (аттракторов).

Каждый нейрон имеет функцию активации сигнум со значениями ± 1 :

$$sign(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}. \quad (8.2)$$

Это означает, что выходной сигнал i -го нейрона определяется функцией:

$$y_i = sign(\sum_{j=1}^N w_{ij} x_j + b_i), \quad (8.3)$$

где N обозначает количество нейронов, $N = n$. Часто постоянная составляющая b_i , определяющая порог срабатывания отдельных нейронов, равна 0. Тогда циклическое прохождение сигнала в сети Хопфилда можно представить соотношением:

$$y_i(k) = sign(\sum_{j=1}^N w_{ij} y_j(k-1)) \quad (8.4)$$

с начальным условием $y_j(0) = x_i$.

В процессе функционирования сети Хопфилда можно выделить два режима: обучения и классификации. В режиме обучения на основе известных обучающих выборок \mathbf{x}_i подбираются весовые коэффициенты w_{ij} . В режиме классификации при зафиксированных значениях весов и вводе конкретного начального состояния нейронов $\mathbf{y}(0) = \mathbf{x}$ возникает переходный процесс, протекающий в соответствии с выражением (8.2) и заканчивающийся в одном из ло-

кальных устойчивых положений, задаваемом биполярным вектором со значениями $y_j = \pm 1$, для которого $\mathbf{y}(k) = \mathbf{y}(k - 1)$.

Обучение не носит рекуррентного характера. Достаточно ввести значения (правило Хебба) весов, выразив их через проекции вектора точки притяжения эталонного образа:

$$w_{ij} = \frac{1}{N} x_i x_j, \quad (8.5)$$

В соответствии с этим правилом сеть дает правильный результат при входном примере, совпадающим с эталонным образцом, поскольку:

$$\sum_{j=1}^N w_{ij} x_j = \frac{1}{N} \sum_{j=1}^N x_i (x_j x_j) = x_i \sum_{j=1}^N 1 \cdot \frac{1}{N} = x_i, \quad (8.6)$$

так как вследствие биполярности значений элементов векторах x всегда $x_j^2 = 1$.

При вводе большого количества обучающих выборок $\mathbf{x}^{(k)}$ для $k = 1, 2, \dots, p$ веса w_{ij} подбираются согласно обобщенному правилу Хебба в соответствии с которым:

$$w_{ij} = \frac{1}{N} \sum_{k=1}^l x_i^{(k)} x_j^{(k)}. \quad (8.7)$$

Благодаря такому режиму обучения веса принимают значения, определяемые усреднением множества обучаемых выборок. В случае множества обучаемых выборок актуальным становится вопрос о стабильности ассоциативной памяти.

Дискретная сеть Хопфилда имеет следующие характеристики: она содержит один слой элементов; каждый элемент связывается со всеми другими элементами, но не связан с самим собой; за один шаг работы обновляется только один элемент сети; элементы обновляются в случайном порядке; выход элемента ограничен значениями 0 или 1.

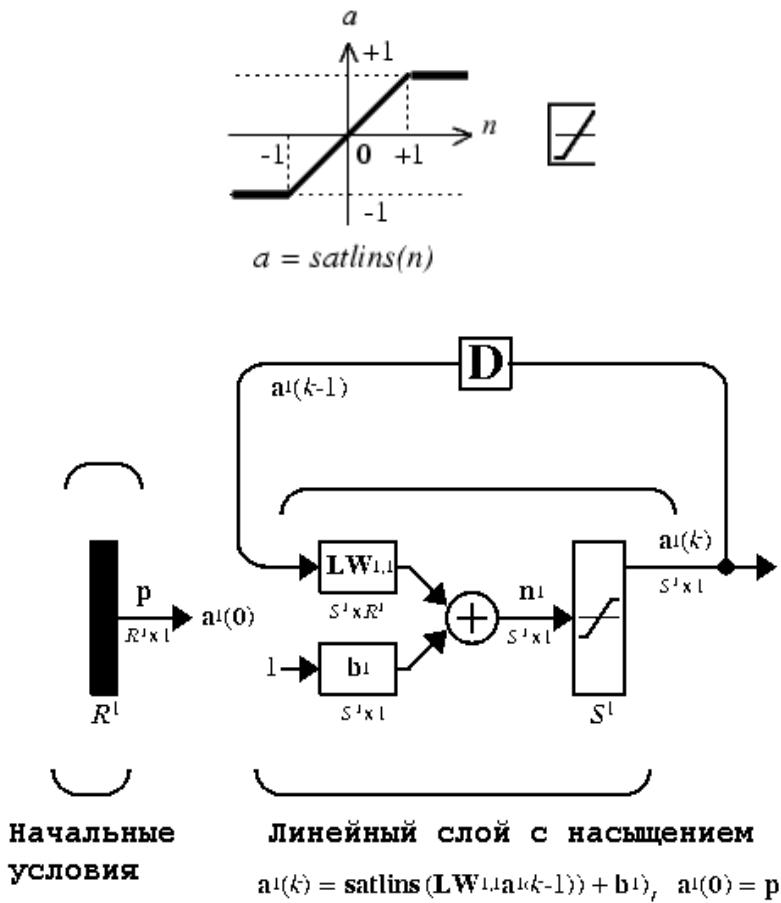


Рисунок 8.1. Архитектура модифицированной сети Хопфилда

Входами \mathbf{p} в этой сети просто представляют начальные условия. Сеть Хопфилда использует насыщенные линейные satlins передаточной функции. Для входа меньше, чем -1 выход функции satlins равен -1 . Для входа в диапазоне от -1 до $+1$ он просто возвращает входное значение. Для входа выше $+1$ значение функции равно $+1$.

Состояние сети – это просто множество текущих значений выходных сигналов от всех нейронов. В первоначальной сети Хопфилда состояние каждого нейрона менялось в дискретные случайные моменты времени, в последующей работе состояния нейронов могли меняться одновременно. Так как выходом бинарного нейрона может быть только ноль или единица (промежуточных уровней нет), то текущее состояние сети является двоичным числом, каждый бит которого является выходным сигналом некоторого нейрона.

Функционирование сети легко визуализируется геометрически. На рисунке 8.2 показан случай двух нейронов в выходном слое, причем каждой вершине

квадрата соответствует одно из четырех состояний системы (00, 01, 10, 11). На рисунке 8.3 показана трехнейронная система, представленная кубом (в трехмерном пространстве), имеющим восемь вершин, каждая из которых помечена трехбитовым бинарным числом. В общем случае система с n нейронами имеет 2^n различных состояний и представляется n -мерным гиперкубом.

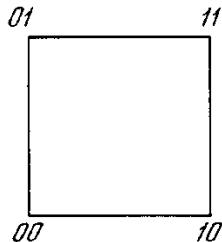


Рисунок 8.2. Два нейрона порождают систему с четырьмя состояниями

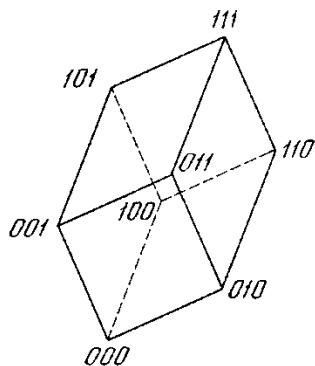


Рисунок 8.3. Три нейрона порождают систему с восемью состояниями

Когда подается новый входной вектор, сеть переходит из вершины в вершину, пока не стабилизируется. Устойчивая вершина определяется сетевыми весами, текущими входами и величиной порога. Если входной вектор частично неправилен или неполон, то сеть стабилизируется в вершине, ближайшей к желаемой.

Функция для создания сети Хопфилда имеет вид:

`net=newhop(T),`

где T – массив размера $R * Q$, объединяющий Q целевых векторов со значениями +1 или -1 для элементов; R – число элементов вектора входа.

После того как начальные условия заданы в виде массива T , определяющего ряд целевых вершин замкнутого гиперкуба, сеть для каждой вершины генерирует выход, который по обратной связи подаётся на вход. Этот

процесс при создании сети повторяется много раз, пока её выход не установится в положение равновесия для каждой из целевых вершин. При подаче затем произвольного входного вектора сеть Хопфилда переходит в результате рекурсивного процесса к одной из точек равновесия, наиболее близкой к входному сигналу.

Динамическая модель рекуррентного слоя одной из модификаций сети Хопфилда описывается следующим образом:

$$\begin{cases} a^1(k) = \text{satlin}(LW^{11}a^1(k-1) + b^1); \\ a^1(0) = p. \end{cases} \quad (8.8)$$

Когда сеть Хопфилда спроектирована, она может быть проверена с одним или большим числом векторов входа. Весьма вероятно, что векторы входа, близкие к целевым точкам равновесия, найдут свои цели. Способность сети Хопфилда быстро обрабатывать наборы векторов входа позволяет проверить сеть за относительно короткое время. Сначала следует проверить, что точки равновесия целевых векторов действительно принадлежат вершинам гиперкуба, а затем можно определить области притяжения этих точек и обнаружить паразитные точки равновесия (ложная память).

2. Создание нейронной сети Хопфилда

2.1 Создадим нейронную сеть Хопфилда с четырьмя нейронами и определим 4 точки равновесия в виде следующего массива целевых векторов

$$T = [1 -1; -1 1; 1 1; -1 -1]';$$

На рисунке 8.4 показаны эти 4 точки равновесия на плоскости состояний сети Хопфилда

```
plot(T(1,:), T(2,:),'*r')
axis([-1.1 1.1 -1.1 1.1])
title('Точки равновесия сети Хопфилда')
xlabel('a(1)'), ylabel('a(2)')
```

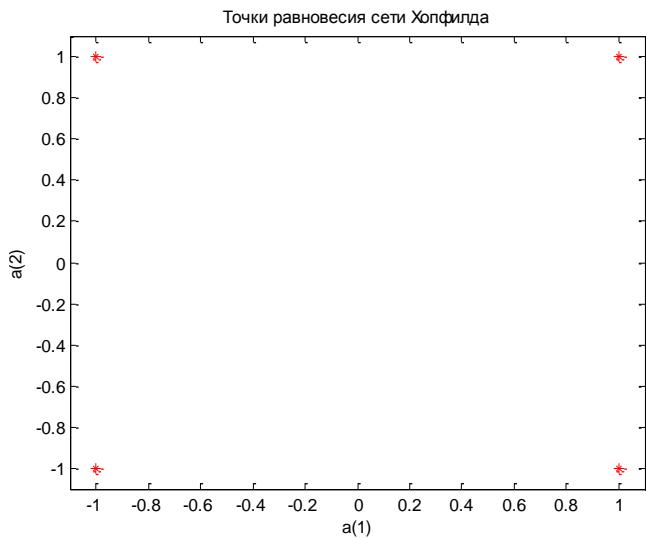


Рисунок 8.4. Точки равновесия на плоскости состояний сети Хопфилда

Рассчитаем веса и смещения модифицированной сети Хопфилда, используя М-функцию *newhop*

net = newhop(T);

W = net.LW{1,1}

b = net.b{1,1}

W =

1.1618 0

0 1.1618

B =

1.0e-016 *

0

-0.1797

Проверим, принадлежат ли вершины квадрата сети Хопфилда:

Ai = T;

[Y,Pf,Af] = sim(net,4,[],Ai)

Y =

1 -1 1 -1

-1 1 1 -1

Pf =

[]

Af =

1 -1 1 -1

-1 1 1 -1

Как и следовало ожидать, выходы сети равны целевым векторам.

Теперь проверим поведение сети при случайных начальных условиях.

```
plot(T(1,:), T(2,:),'*r', 0,0,'rh'), hold on, axis([-1.1 1.1 -1.1 1.1])
xlabel('a(1)'), ylabel('a(2)')
new = newhop(T);
[Y,Pf,Af] = sim(net,4,[],T);
for i = 1:20
    a = {rands(2,1)};
    [Y, Pf, Af] = sim(net,{1,20},[],a);
    record = [cell2mat(a) cell2mat(Y)];
    start = cell2mat(a);
    plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:))
end
```

Результат представлен на рисунке 8.5.

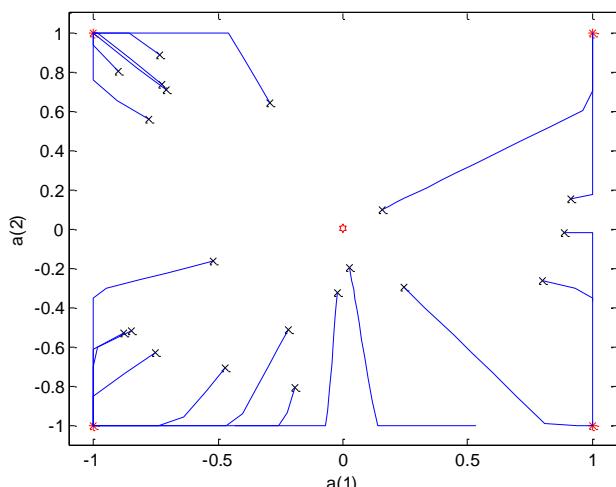


Рисунок 8.5. Переходный процесс заканчивающийся в одном из локальных устойчивых положений

2.2 Создадим сеть Хопфилда с двумя устойчивыми точками в трехмерном пространстве

$$T = [-1 \ -1 \ 1; 1 \ -1 \ 1]';$$

net = newhop(T);

gensim(net)

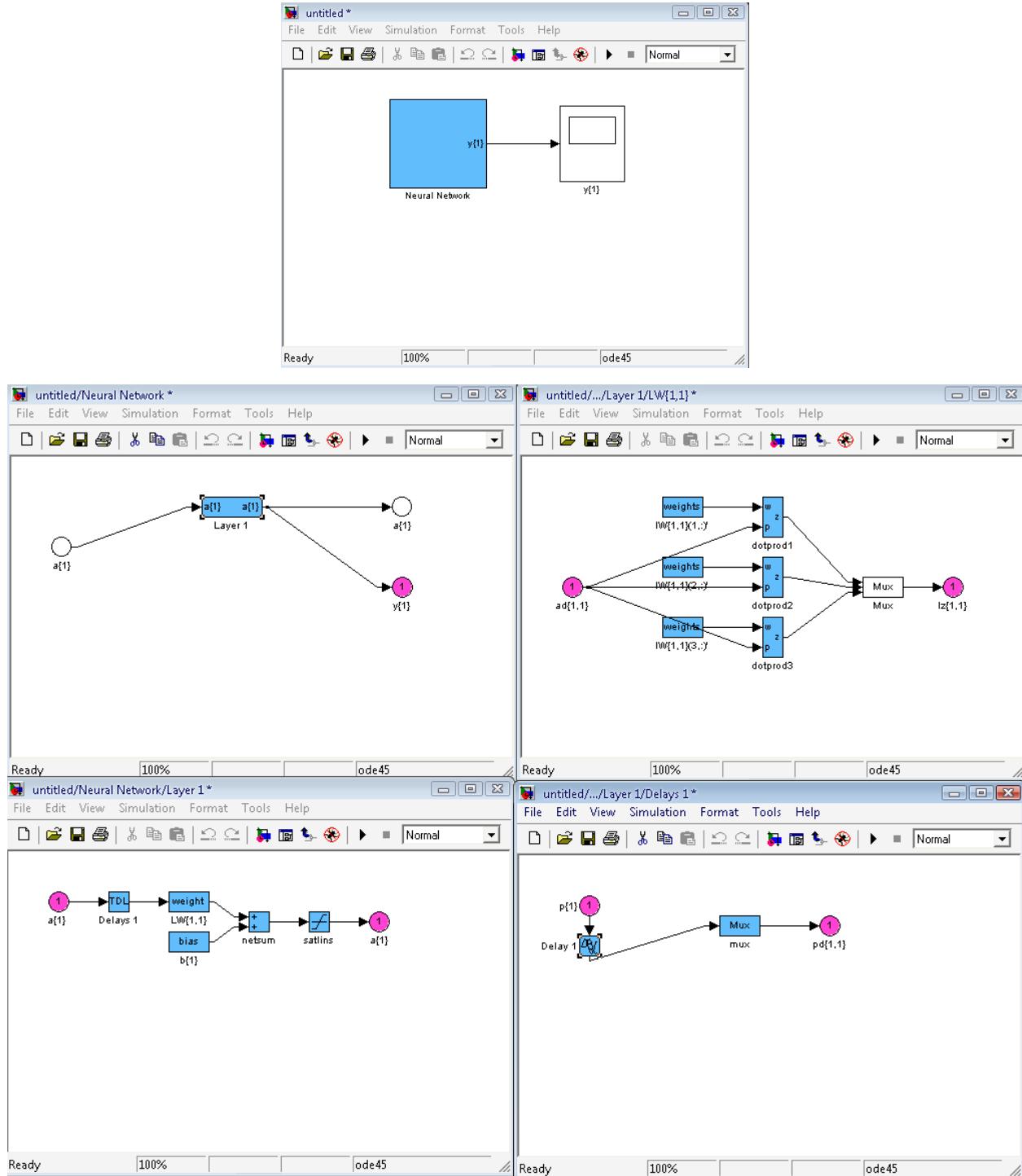


Рисунок 8.6. Структурные схемы, созданной сети Хопфилда в среде Simulink

Проверим, что сеть устойчива в этих точках и используем их как начальные условия для линии задержки. Если сеть устойчива, можно ожидать, что выходы Y будут те же самые

$Ai = T;$

$[Y, Pf, Af] = sim(net, 2, [], Ai);$

Y

$Y =$

-1 1

-1 -1

1 1

Таким образом, вершины гиперкуба являются устойчивыми точками равновесия сети Хопфилда.

Проверим сходимость сети при произвольном входном векторе Ai

$Ai = \{-0.8; -0.7; 0.6\};$

$[Y, Pf, Af] = sim(net, \{1 5\}, \{\}, Ai);$

$Y\{1\}$

$ans =$

-0.9295

-1.0000

0.9884

Сеть Хопфилда обеспечила переход к устойчивому положению равновесия, ближайшему к заданному входу.

Сети Хопфилда разработаны для того, чтобы приводить случайные входные векторы к точкам равновесия, которые соответствуют определенным, заданным пользователем, целям.

Программа работы и методические указания

Создать и обучить сеть Хопфилда распознавать зашумленные символы латинского алфавита, исследовать влияние величины искажений на качество распознавания. Пример для 1-го символа А:

```
[alphabet, targets] = prprob;
```

```
i = 1;
```

```
ti = alphabet(:, i);
```

```
letter{i} = reshape(ti, 5, 7)';
```

```
letter{i}
```

```
ans =
```

```
0 0 1 0 0  
0 1 0 1 0  
0 1 0 1 0  
1 0 0 0 1  
1 1 1 1 1  
1 0 0 0 1  
1 0 0 0 1
```

```
plotchar(reshape(letter{i}', 1, 35)');
```

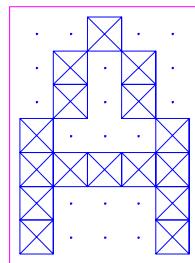


Рисунок 8.7. Эталонный символ А

```
net = newhop(letter{i});
```

```
[Y,Pf,Af] = sim(net,5,[], letter{i});
```

```
Y
```

```
Y =
```

```
-0.0459 -0.0459 1.0000 -0.0459 -0.0459  
-0.0229 1.0000 -0.0229 1.0000 -0.0229  
-0.0229 1.0000 -0.0229 1.0000 -0.0229  
1.0000 -0.0153 -0.0153 -0.0153 1.0000  
1.0000 1.0000 1.0000 1.0000 1.0000  
1.0000 -0.0153 -0.0153 -0.0153 1.0000  
1.0000 -0.0153 -0.0153 -0.0153 1.0000
```

```
noisyA = alphabet(:,1) + randn(35,1) *0.4;
```

```
reshape(noisyA, 5, 7)'
```

```

ans =
-0.2462  0.2992  0.9230  0.3554 -0.3059
-0.5609  0.4310  0.1953  0.9290 -0.0784
0.5677  1.1166  0.0791  1.6351 -0.3218
1.2786  0.3340 -0.0975  0.0863  0.5337
0.5408  1.0419  1.2889  2.0342  0.7332
1.0749 -0.0330 -0.7732 -0.1756  0.2821
1.3362 -0.3552  0.0400 -0.2178  1.1214

```

```
plotchar(noisyA);
```

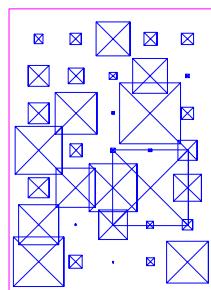


Рисунок 8.8 – Зашумленный символ А

```
[Y,Pf,Af] = sim(net,5,[],reshape(noisyA, 5, 7)');
```

```
Y
```

```

Y =
0.0702  0.0554  1.0000 -0.2144  0.1257
-0.2520  1.0000 -0.0512  1.0000  0.0029
-0.0377  0.9957  0.1371  1.0000 -0.0525
1.0000 -0.1042 -0.0992  0.0253  1.0000
1.0000  1.0000  1.0000  1.0000  1.0000
1.0000 -0.1767  0.0126  0.0507  1.0000
1.0000 -0.0614 -0.1562  0.0088  1.0000

```

```
plotchar(reshape(Y', 1, 35)');
```

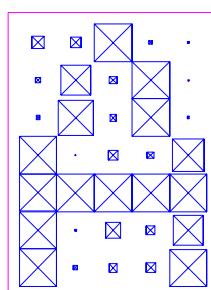


Рисунок 8.9. Распознанный символ А

Контрольные вопросы

1. Какова активационная характеристика нейронов сети Хопфилда?
2. Чему равно начальное состояние нейронов сети Хопфилда?
3. Как рассчитывается матрица синаптических коэффициентов сети Хопфилда? Какими свойствами она обладает?
4. Как приближенно оценивается объем памяти сети Хопфилда?
5. На какие типы можно разделить множество аттракторов сети Хопфилда? Что такое «ложная память»?

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.

Лабораторная работа №9

НЕЙРОСЕТЕВЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

Цель работы: исследование применения нейронных сетей для решения задач управления.

Введение

В настоящее время нейронные сети нашли успешное применение для проектирования систем управления динамическими процессами. Универсальные возможности аппроксимации с помощью многослойного персептрона делают их полезным инструментом для решения задач идентификации, проектирования и моделирования нелинейных регуляторов.

Ниже описаны три архитектуры нейронных сетей, которые реализованы в ППП Neural Network Toolbox в виде следующих контроллеров:

- контроллера с предсказанием (*NN Predictive Controller*);
- контроллера на основе модели авторегрессии со скользящим средним (*NARMA-L2 Controller*);

- контроллера на основе эталонной модели (*Model Reference Controller*).

Применение нейронных сетей для решения задач управления позволяет выделить два этапа проектирования:

- этап идентификации управляемого процесса;
- этап синтеза закона управления.

На этапе идентификации разрабатывается модель управляемого процесса в виде нейронной сети, которая на этапе синтеза используется для синтеза регулятора. Для каждой из трех архитектур используется одна и та же процедура идентификации, однако этапы синтеза существенно различаются.

При управлении с предсказанием модель управляемого процесса используется для того чтобы предсказать его будущее поведение, а алгоритм оптимизации применяется для расчета такого управления, которое минимизирует разность между желаемыми и действительными изменениями выхода модели.

При управлении на основе модели авторегрессии со скользящим средним регулятор представляет собой достаточно простую реконструкцию модели управляемого процесса.

При управлении на основе эталонной модели регулятор – это нейронная сеть, которая обучена управлять процессом так, чтобы он отслеживал поведение эталонного процесса. При этом модель управляемого процесса активно используется при настройке параметров самого регулятора.

В последующих разделах обсуждаются все три структуры систем управления и архитектуры соответствующих нейросетевых контроллеров. Каждый раздел включает краткое изложение принципа управления динамическим процессом и сопровождается описанием сценария функционирования проектируемой системы, который реализован в виде комбинации GUI-интерфейса и динамической модели регулятора в системе *Simulik*.

Динамические модели систем управления с нейросетевыми регуляторами размещены в специальном разделе *Control Systems* набора *Demos Neural Network Toolbox* (рисунок 9.1).

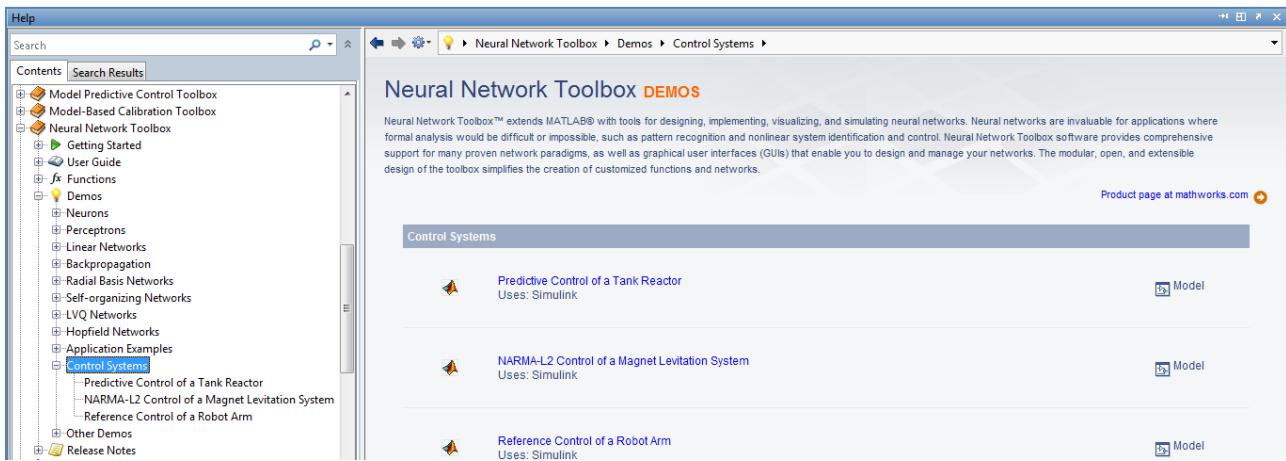


Рисунок 9.1. Размещение динамических моделей систем управления с нейросетевыми регуляторами

Поскольку ни один конкретный регулятор не является универсальным, то описаны функциональные возможности всех трех типов регуляторов, каждый из которых имеет свои преимущества и недостатки.

Регулятор с предсказанием. Этот регулятор использует модель управляемого процесса в виде нейронной сети, для того чтобы, предсказать будущие реакции процесса на случайные сигналы управления. Алгоритм оптимизации вычисляет управляющие сигналы, которые минимизируют разность между желаемыми и действительными изменениями сигнала на выходе модели, и таким образом оптимизируют управляемый процесс. Построение модели управляемого процесса выполняется автономно с использованием нейронной сети, которая обучается в групповом режиме с использованием одного из алгоритмов обучения. Контроллер, реализующий такой регулятор, требует значительного объема вычислений, поскольку для расчета оптимального закона управления оптимизация выполняется на каждом такте управления.

Регулятор NARMA-L2. Изо всех архитектур этот регулятор требует наименьшего объема вычислений. Данный регулятор – это просто некоторая реконструкция нейросетевой модели управляемого процесса, полученной на этапе автономной идентификации. Вычисления в реальном времени связаны только с реализацией нейронной сети. Недостаток метода состоит в том, что модель процесса должна быть задана в канонической форме пространства состояния,

которой соответствует сопровождающая матрица, что может приводить к вычислительным погрешностям.

Регулятор на основе эталонной модели. Требуемый объем вычислений для этого регулятора сравним с предыдущим. Однако архитектура регулятора с эталонной моделью требует обучения нейронной сети управляемого процесса и нейронной сети регулятора. При этом, обучение регулятора оказывается достаточно сложным, поскольку обучение основано на динамическом варианте метода обратного распространения ошибки. Достоинством регуляторов на основе эталонной модели является то, что они применимы к различным классам управляемых процессов.

1. Регулятор с предсказанием

Регулятор с предсказанием, реализованный в пакете Neural Network Toolbox, использует модель нелинейного управляемого процесса в виде нейронной сети для того, чтобы предсказывать его будущее поведение. Кроме того, регулятор вычисляет сигнал управления, который оптимизирует поведение объекта на заданном интервале времени.

Идентификация управляемого процесса. Схема подсистемы идентификации показана на рисунке 9.2. Она включает модель управляемого процесса в виде нейронной сети, которая должна быть обучена в автономном режиме так, чтобы минимизировать ошибку между реакциями процесса и модели $e = y_p - y_m$, на последовательность пробных сигналов u .

Нейронная сеть регулятора управляемого процесса имеет два слоя нейронов и использует линии задержки, чтобы запомнить предшествующие значения входов и выходов процесса с целью предсказать будущие значения выхода (рисунок 9.3).

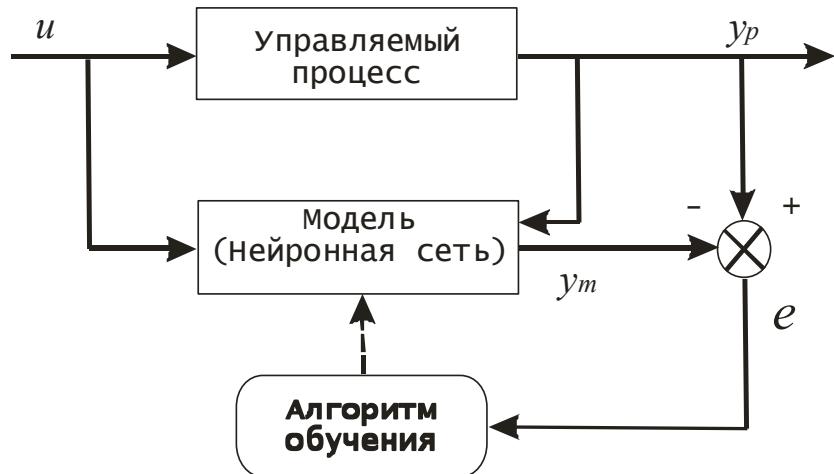


Рисунок 9.2 – Схема подсистемы идентификации

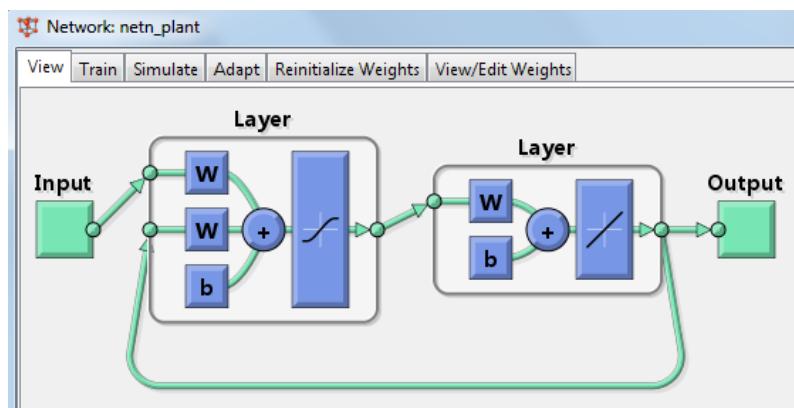


Рисунок 9.3 – Нейронная сеть регулятора управляемого процесса

Настройка параметров этой сети выполняется автономно методом группового обучения, используя данные, полученные при испытаниях реального объекта. Для обучения сети может быть использован любой из обучающих алгоритмов для нейронных сетей.

Принцип управления с предсказанием. Управление с предсказанием использует принцип меняющегося горизонта, когда нейросетевая модель управляемого процесса предсказывает реакцию объекта управления на определенном интервале времени в будущем. Предсказания используются программой численной оптимизации для того, чтобы вычислить управляющий сигнал, который минимизирует следующий критерий качества управления:

$$J = \sum_{j=N_1}^{N_2} (y_r(t+j) - y_m(t+j))^2 + \rho \sum_{j=1}^{N_u} (u'(t+j-1) - u'(t+j-2))^2, \quad (9.1)$$

где константы N_1 , N_2 и N_u задают пределы, внутри которых вычисляются ошибка слежения и мощность управляющего сигнала. Переменная u' описывает пробный управляющий сигнал, y_r – желаемая, а y_m – истинная реакция модели управляемого процесса. Величина ρ определяет вклад, который вносит мощность управления в критерий качества.

Структурная схема (рисунок 9.4) иллюстрирует процесс управления с предсказанием. Регулятор состоит из нейросетевой модели управляемого процесса и блока оптимизации. Блок оптимизации определяет значения u' , которые минимизируют критерий качества управления, а соответствующий управляющий сигнал управляет процессом.

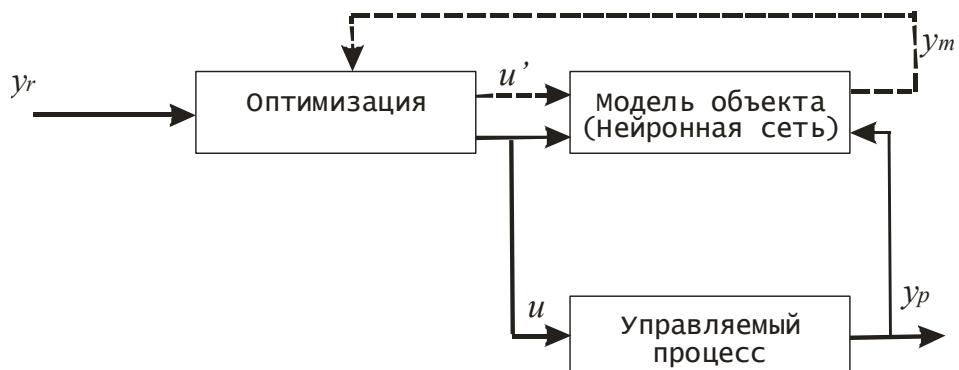


Рисунок 9.4 – Процесс управления с предсказанием

Реализация регулятора с предсказанием. В качестве примера исследуется процесс управления каталитическим реактором с непрерывным перемешиванием (*Continuous Stirred Tank Reactor - CSTR*), схема которого показана на рисунке 9.5.

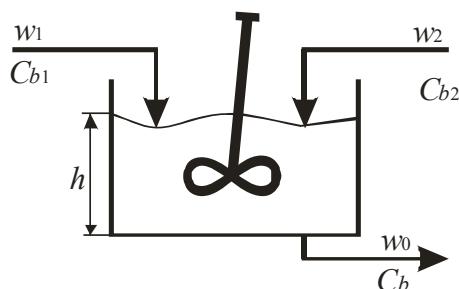


Рисунок 9.5. Схема каталитического реактора с непрерывным смешиванием

Динамическая модель управляемого процесса описывается следующими обыкновенными нелинейными дифференциальными уравнениями:

$$\begin{cases} \frac{dh}{dt} = w_1(t) + w_2(t) - 0,2\sqrt{h}; \\ \frac{dC_b}{dt} = (C_{b1} - C_b) \frac{w_1(t)}{h} + (C_{b2} - C_b) \frac{w_2(t)}{h} - \frac{k_1 C_b}{(1 + k_2 C_b)^2} \end{cases} \quad (9.2)$$

где h – уровень жидкости в резервуаре; $w_1(t)$ - скорость потока продукта с концентрацией C_{b1} ; $w_2(t)$ – скорость потока разбавленного продукта с концентрацией C_{b2} ; C_b – концентрация продукта на выходе объекта.

Исходные концентрации приняты равными $C_{b1} = 29.4$ и $C_{b2} = 0.1$. Константы скорости расхода равны $k_1 = k_2 = 1$.

Цель регулирования состоит в поддержании концентрации продукта путем регулирования скорости потока $w_2(t)$. Для простоты принято $w_1(t) = 0.1$. В этом примере уровень раствора в резервуаре не регулируется.

Нелинейная динамическая модель катализитического реактора, соответствующая уравнениям (9.2), показана на рисунке 9.6.

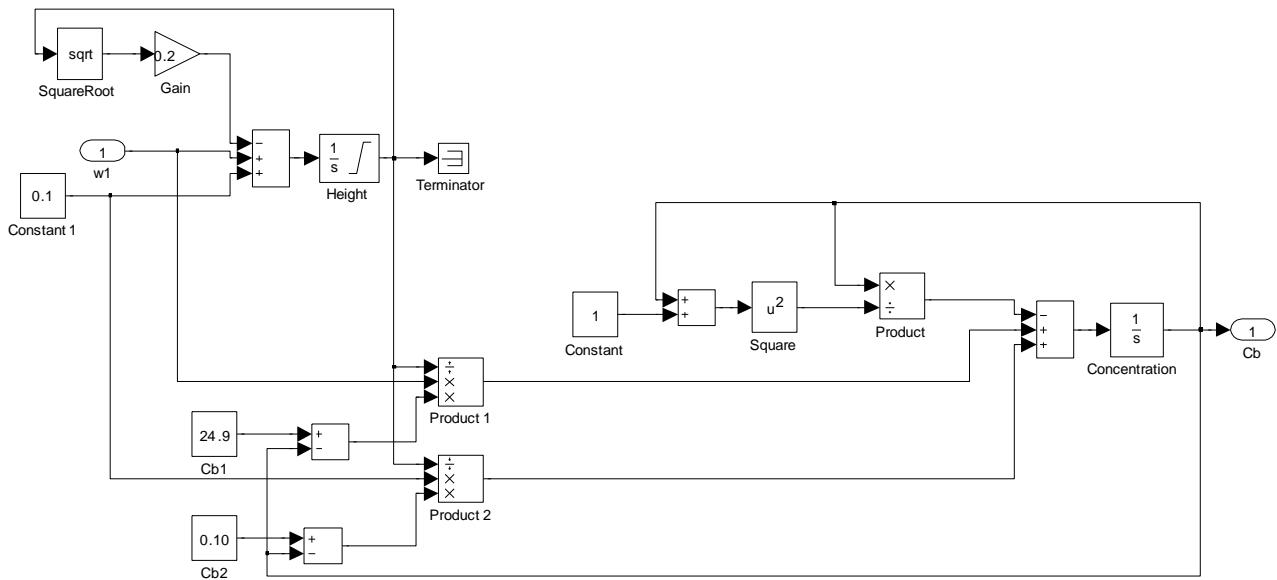


Рисунок 9.6 Нелинейная динамическая модель катализитического реактора

После запуска запуск демонстрационного примера открывается окно системы Simulink со следующей структурой системы управления (рисунок 9.7).

Эта структура включает блок управляемого процесса *Plant* и блок контроллера *NN Predictive Controller*, а также блоки генерации эталонного ступенчатого сигнала со случайной амплитудой *Random Reference*, отсчета времени *Clock*, построения графиков *Graph*. Особенность этой структуры заключается в том, что она выполняет не только функции блок-схемы системы Simulink, но и функции графического интерфейса пользователя GUI.

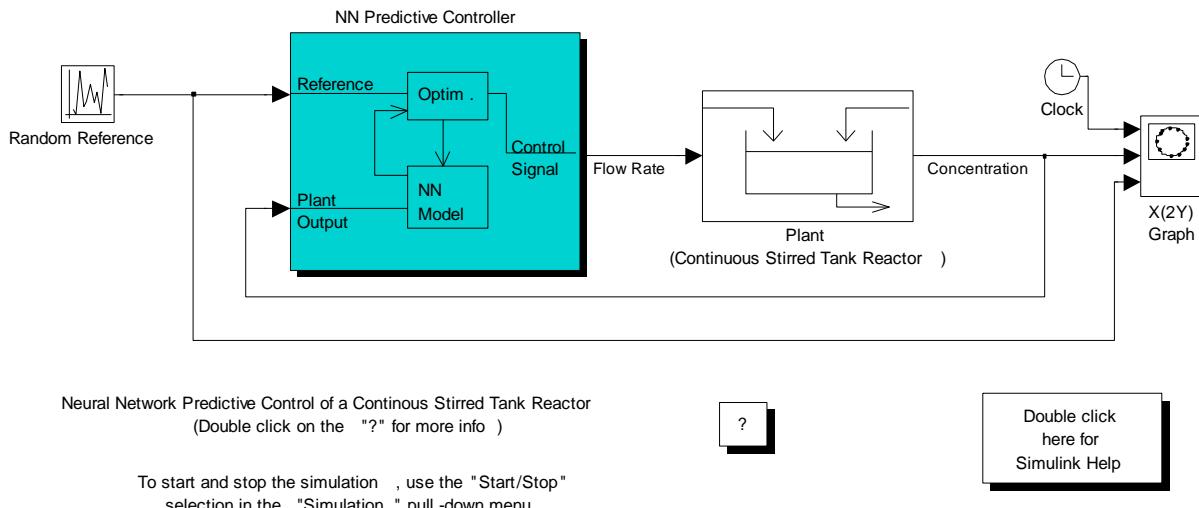


Рисунок 9.7. Структура системы управления в Simulink

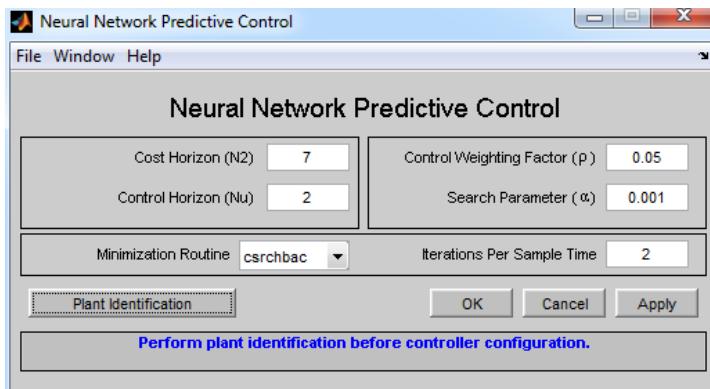


Рисунок 9.8 – Графический интерфейс пользователя нейросетевой системы управления с предсказанием

Для того чтобы начать работу, необходимо активизировать блок *NN Predictive Controller* двойным щелчком левой кнопки мыши. Появится окно, показанное на рисунке 9.8.

Оно выполняет функции графического интерфейса пользователя. Обратите внимание на информацию, которая руководит вашими действиями и указана в области фрейма в виде сообщения. Прежде чем установить параметры контроллера, постройте модель управляемого процесса. Это означает, что прежде всего необходимо выполнить идентификацию управляемого процесса, т. е. построить его нейросетевую модель, воспользовавшись специальной процедурой *Plant Identification*.

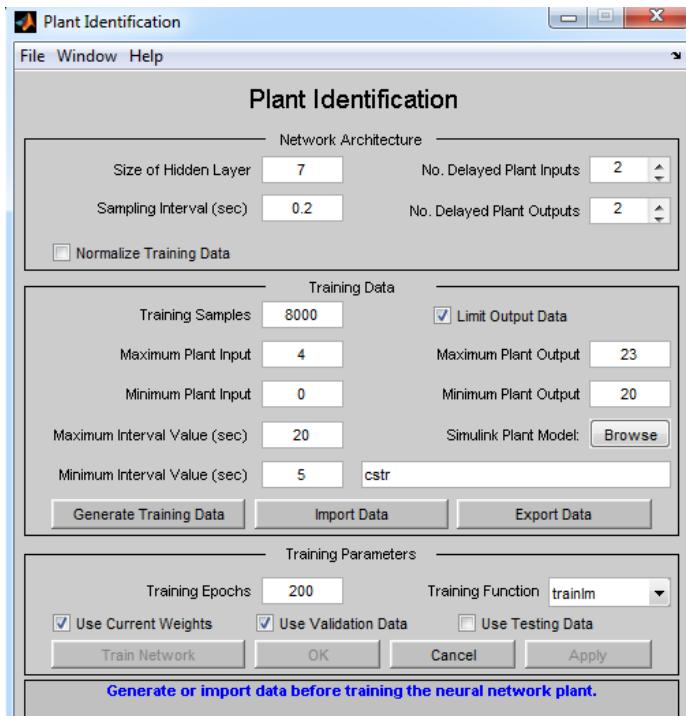


Рисунок 9.9. Окно процедуры идентификации

Вид окна *Plant Identification* приведен на рисунке 9.9. Это окно универсально и может быть использовано для построения нейросетевых моделей для любого динамического объекта, который описан моделью Simulink. В рассматриваемом случае такой моделью является нелинейная динамическая модель катализического реактора CSTR. Процедура идентификации позволяет построить нейронную сеть, которая будет моделировать динамику управляемого процесса. Если модель должна использоваться при настройке контроллера, то ее следует создать прежде, чем начнется расчет контроллера. Кроме того, вам может потребоваться создание новой модели объекта, если спроектированный контроллер будет функционировать неудовлетворительно.

Выбор процедуры Generate Training Data приведет к тому, что будет запущена программа генерации обучающей последовательности на интервале 1600 с для модели каталитического реактора cstr.mdl. Программа генерирует обучающие данные путем воздействия ряда случайных ступенчатых сигналов на модель Simulink управляемого процесса. Графики входного и выходного сигналов объекта управления выводятся на экран (рисунок 9.10).

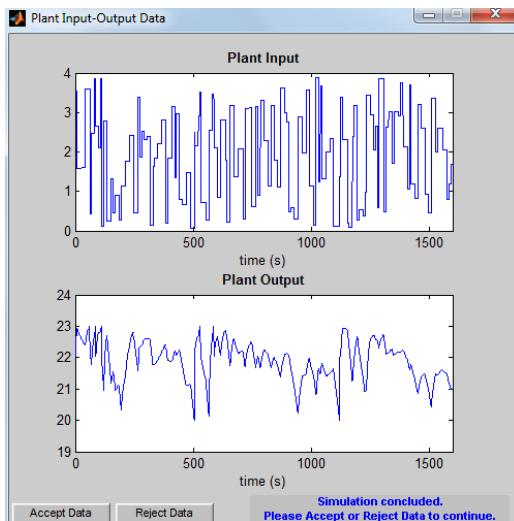


Рисунок 9.10. Графики входного и выходного сигналов объекта управления

По завершении генерации обучающей последовательности пользователю предлагается либо принять сгенерированные данные (*Accept Data*), либо отказаться от них (*Reject Data*).

Если вы принимаете данные, приложение возвращает вас к несколько измененному окну *Plant Identification*. Здесь часть окон недоступны, а кнопка *Generate Training Data* заменена на кнопку *Erase Generated Data*, что позволяет удалить сгенерированные данные.

В окне фрейма содержится сообщение “Обучающая последовательность состоит из 8000 замеров”. Можно начинать обучение нейронной сети. Для этого следует воспользоваться кнопкой *Train Network* (Обучить сеть). Начнется обучение нейросетевой модели (рисунок 9.11). После завершения обучения его результаты отображаются на графиках, где построены соответственно результаты на обучающем и контрольном множестве (рисунок 9.12).

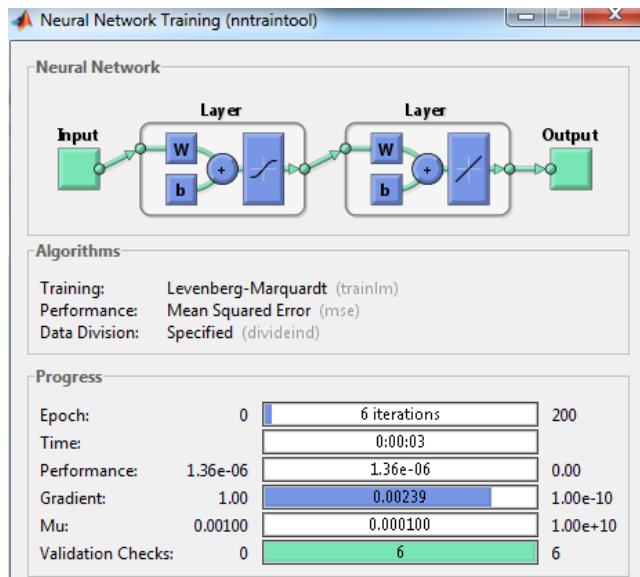


Рисунок 9.11. Окно обучения нейронной сети

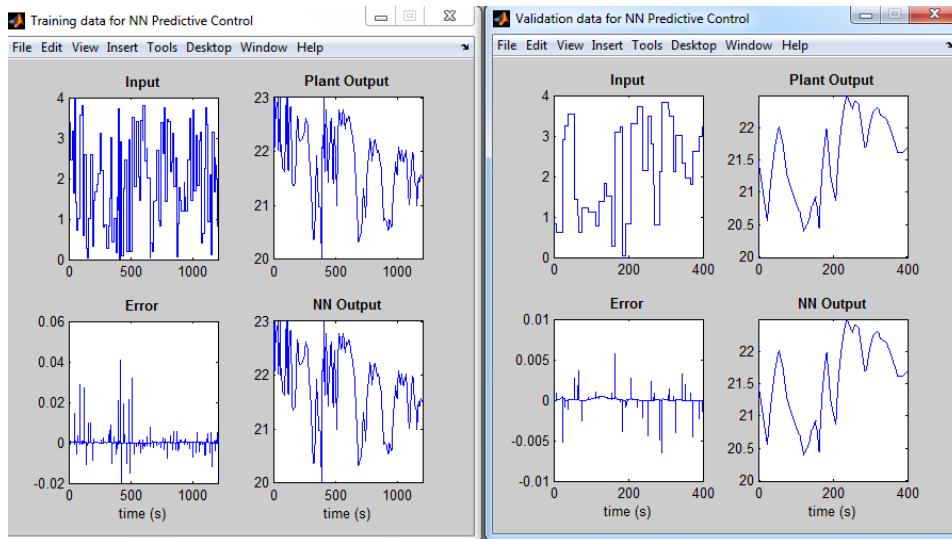


Рисунок 9.12. Результаты обучения и тестирования на обучающем и контрольном множестве

Текущее состояние отмечено в окне *Plant Identification* сообщением «обучение завершено». Вы можете сгенерировать или импортировать новые данные, продолжить обучение или сохранить полученные результаты, выбрав кнопки *OK* или *Apply*. В результат параметры нейросетевой модели управляемого процесса будут введены в блок *NN Predictive Controller* системы Simulink.

После этого, вновь возвращаемся к окну *Neural Network Predictive Controller*, вводим параметры регулятора в блок *NN Predictive Controller* нажатием кнопок *OK* или *Apply* и начинаем моделирование, выбрав опцию *Start* из меню

Simulation. В процессе моделирования выводятся графики входа и выхода управляемого процесса (рисунок 9.13).

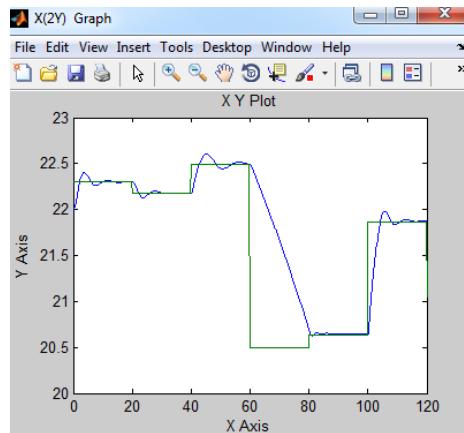


Рисунок 9.13. Процесс моделирования

Из анализа полученных данных следует, что реакция системы на ступенчатые воздействия со случайной амплитудой вполне удовлетворительна, имеет колебательный характер с достаточно быстрым затуханием; на интервале 40 с все воздействия эффективно отрабатываются. Таким образом, регулятор с предсказанием, реализованный в виде нейронной сети, можно использовать для управления каталитическим реактором с непрерывным перемешиванием.

2. Регулятор NARMA-L2

Нейросетевой регулятор, описанный в этом разделе, использует в качестве модели управляемого процесса модель нелинейной авторегрессии со скользящим средним (*Nonlinear Autoregressive-Moving Average - NARMA-L2*).

Схема демонстрационного примера управления магнитной подушкой показана на рисунке 9.14.

Управляемым объектом является магнит, который движется только в вертикальном направлении в электромагнитном поле, как это схематично показано на рисунке 9.15.

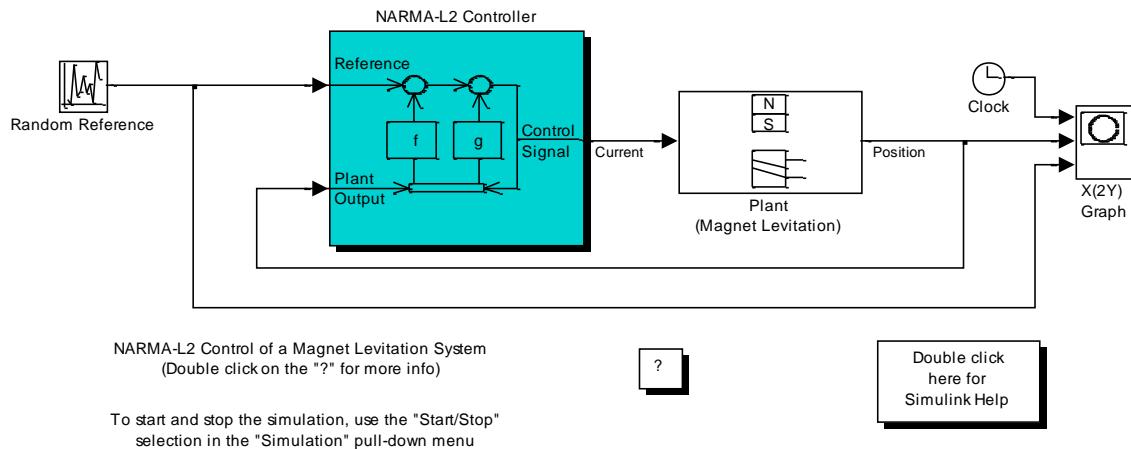


Рисунок 9.14. Схема управления магнитной подушкой

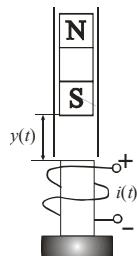


Рисунок 9.15. Схема управления магнитом в электромагнитном поле

Уравнение движения этой системы имеет вид:

$$\frac{d^2y(t)}{dt^2} = -g + \frac{\alpha}{M} \frac{i^2(t)}{y(t)} - \frac{\beta}{M} \frac{dy(t)}{dt} \quad (9.3)$$

где $y(t)$ - расстояние движущегося магнита от электромагнита; g - ускорение силы тяжести; α - постоянная магнитного поля, зависящая от числа витков обмотки и намагниченности электромагнита; $i(t)$ – управляющий ток электромагнита; M – масса магнита; β – коэффициент вязкого трения.

Соответствующая динамическая модель, реализованная в системе Simulink, показана на рисунке 9.14. Точно такую же модель, но с конкретными числовыми данными вы сможете увидеть на экране терминала, если активизируете блок *Plant (Magnet Levitation)* в окне *Model Browser*.

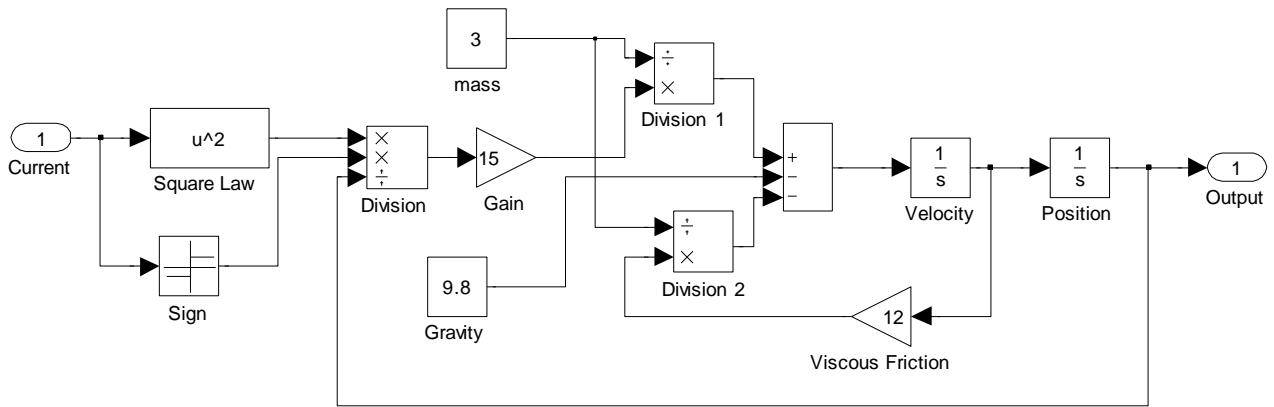


Рисунок 9.16. Модель поведения магнита в электромагнитном поле

Рассматриваемая динамическая система является нелинейной. Построим ее дискретную нелинейную модель, как авторегрессионную модель со скользящим средним, или NARMA-модель, в форме

$$y(k+d) = N[y(k), y(k-1), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-n+1)] \quad (9.4)$$

где $y(k)$ - выход модели; d - число тактов предсказания; $u(k)$ - вход модели. На этапе идентификации необходимо построить нейронную сеть для аппроксимации нелинейной функции N .

Если требуется спроектировать следящую систему, которая обеспечивает движение по заданной траектории

$$y(k+d) = y_r(k+d), \quad (9.5)$$

то это означает, что необходимо сформировать нелинейный регулятор следующего общего вида:

$$u(k) = G[y(k), y(k-1), \dots, y(k-n+1), y_r(k+d), u(k-1), \dots, u(k-m+1)]. \quad (9.6)$$

Хотя такой регулятор с помощью нейронной сети и может быть сформирован, однако в процессе минимизации среднеквадратичной ошибки он требует чрезмерных вычислений, поскольку использует динамический вариант метода обратного распространения ошибки.

Одно из решений, предложенных Narendra and Mukhopadhyay, заключается в использовании приближенных моделей для представления системы. Контроллер, используемый в данном разделе основан на NARMA-L2 приближенной модели:

$$\begin{aligned}\bar{y}(k+d) = & f[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)] + \\ & + g[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-m+1)] \cdot u(k)\end{aligned}\quad (9.7)$$

Эта следящая модель, где вход контроллера $u(k)$ не содержит внутри нелинейности. Преимущество данной модели является то, что можно рассчитать вход управления, который обеспечивает движение по заданной траектории $y(k+d) = y_r(k+d)$. В результате контроллер будет иметь вид

$$u(k) = \frac{y_r(k+d) - f[y(k), \dots, y(k-n+1), u(k-1), \dots, u(k-n+1)]}{g[y(k), \dots, y(k-n+1), u(k-1), \dots, u(k-n+1)]}.\quad (9.8)$$

Использование этого уравнения непосредственно может вызвать проблемы реализации, так как необходимо определить управляющий вход $u(k)$ на основании выходного сигнала в то же время, $y(k)$. Так, вместо того, используют модель

$$\begin{aligned}y(k+d) = & f[y(k), y(k-1), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-m+1)] + \\ & + g[y(k), y(k-1), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-n+1)] \cdot u(k+1),\end{aligned}\quad (9.9)$$

где $d \geq 2$. На рисунке 9.17 показана структура нейронной сети реализации данного выражения.

NARMA-L2 Controller. Уравнение, описывающее функционирование NARMA-L2 контроллера имеет следующий вид:

$$u(k) = \frac{y_r(k+d) - f[y(k), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-n+1)]}{g[y(k), y(k-1), \dots, y(k-n+1), u(k), \dots, u(k-n+1)]}.\quad (9.10)$$

которое реализуется для $d \geq 2$. На рисунке 9.18 показана блок-схема Narma-L2 контроллера.

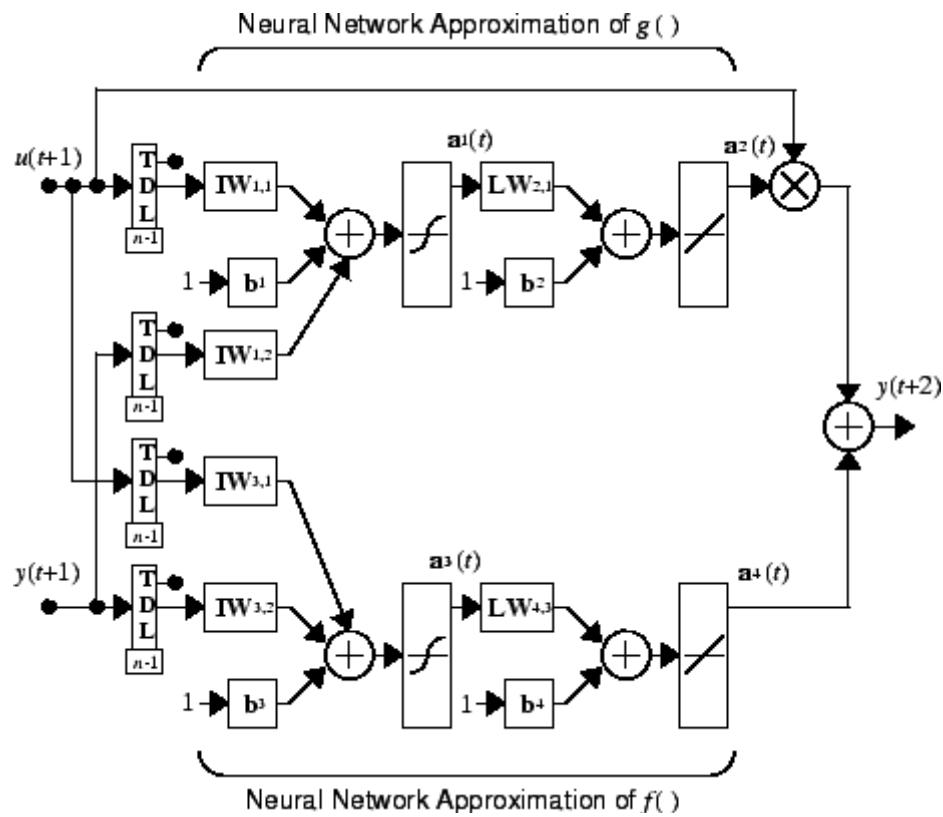


Рисунок 9.17. Нейросетевая реализация следящей системы

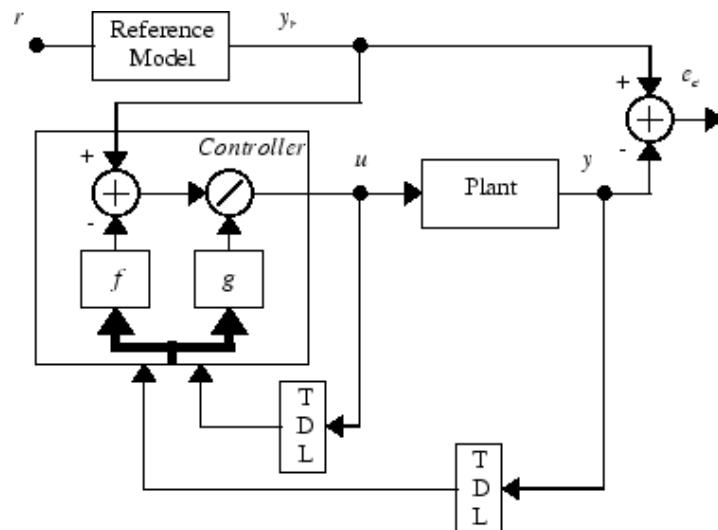


Рисунок 9.18. Блок-схема Narma-L2 контроллера

Этот контроллер может быть реализован с ранее рассмотренной NARMA-L2 моделью, как показано на рисунке 9.19.

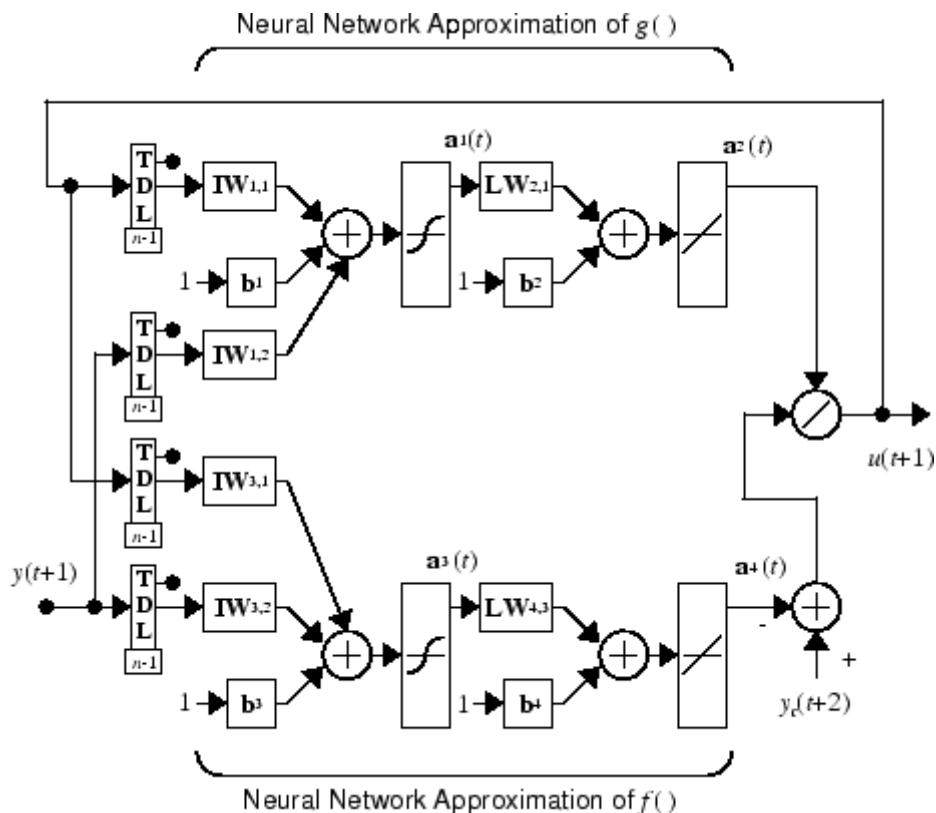


Рисунок 9.19. Нейросетевая реализация контроллера с NARMA-L2 моделью

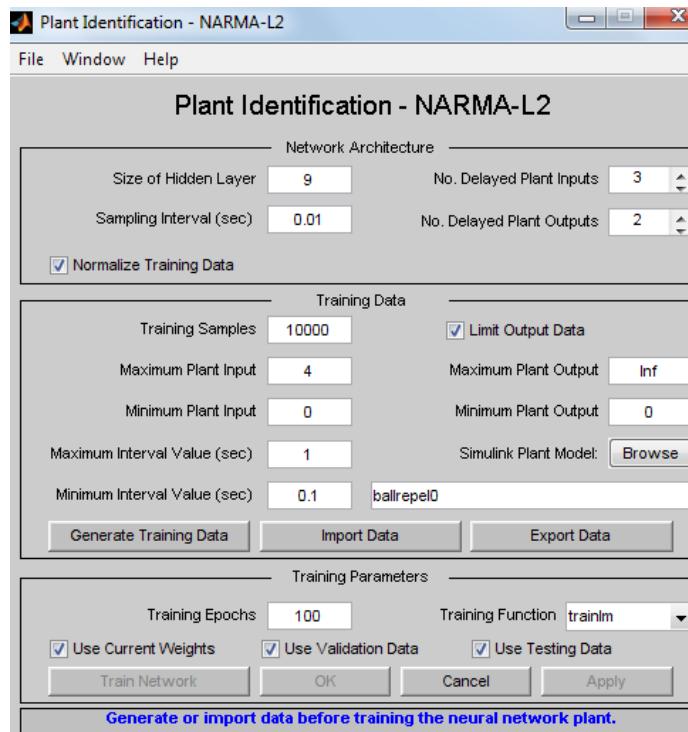


Рисунок 9.20. Окно примера NARMA-L2

Для того чтобы начать работу, необходимо активизировать блок NARMA-L2 Controller двойным щелчком левой кнопки мыши. Появится окно, показанное на рисунке 9.20.

Обратите внимание, что это окно дает возможность обучить модель NARMA-L2. Отдельного окна для обучения регулятора нет, так как регулятор NARMA-L2, в отличие от регулятора с предсказанием, определяется непосредственно по модели. Это окно работает так же, как и другие окна идентификации объекта управления, поэтому повторно подробно процесс обучения рассматривать не будем. Для уменьшения диапазона изменения амплитуды входных колебаний установим параметр *Minimum Plant Input = 0*.

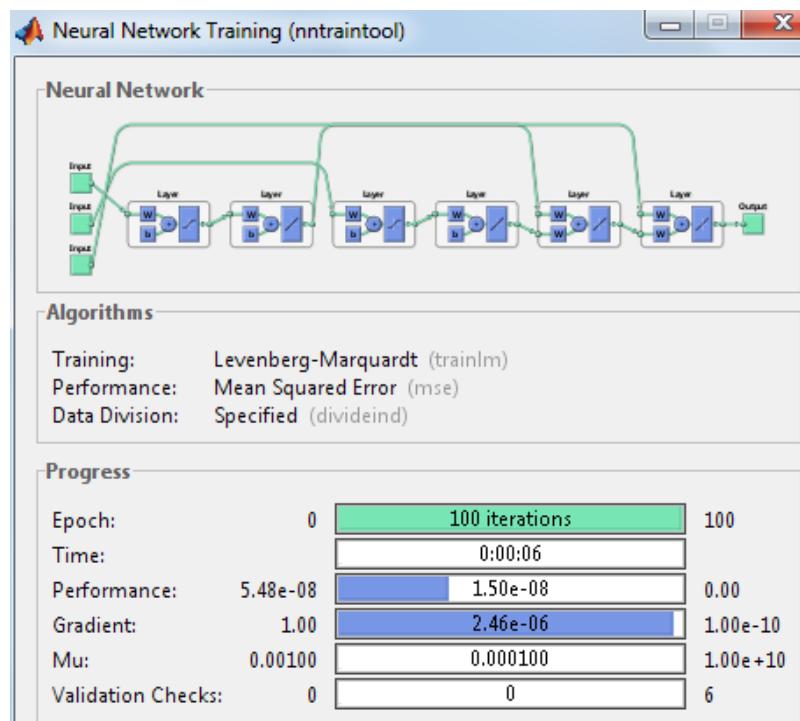


Рисунок 9.21. Окно обучения нейронной сети

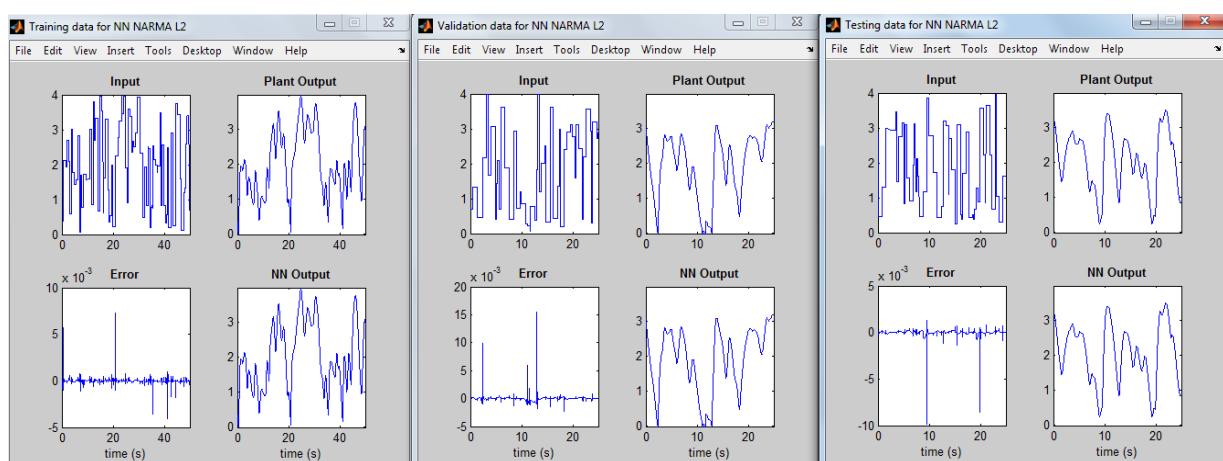


Рисунок 9.22. Результаты обучения и тестирования на обучающем и контрольном множестве

После завершения обучения нейросетевой модели (рисунок 9.21) его результаты отображаются на графиках, где построены соответственно результаты на обучающем, контрольном и тестовом множествах (рисунок 9.22).

После окончания обучения нажать на клавишу OK для ввода данных регулятора в модель Simulink. Возвращаемся к модели Simulink и начинаем моделирование, выбрав опцию *Start* из меню *Simulation*. Графики задающего сигнала и выхода системы приведены на рисунке 9.23.

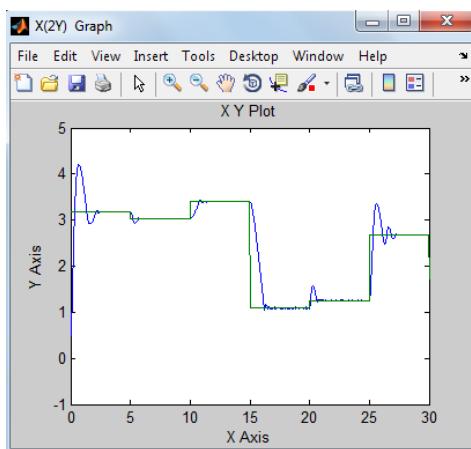


Рисунок 9.23. Задающий и выходной сигналы системы управления

Из анализа полученных данных следует, что реакция системы на ступенчатые воздействия со случайной амплитудой вполне удовлетворительна, имеет колебательный характер с достаточно быстрым затуханием, на интервале 15 с все установки эффективно отрабатываются. Таким образом, регулятор NARMA-L2, реализованный в виде нейронной сети, можно использовать для управления магнитной подушкой

3. Регулятор на основе эталонной модели

В этом разделе описана система управления с эталонной моделью, при реализации которой используются две нейронные сети: для регулятора и для модели объекта управления. Схема демонстрационного примера управления звеном робота показана на рисунке 9.24.

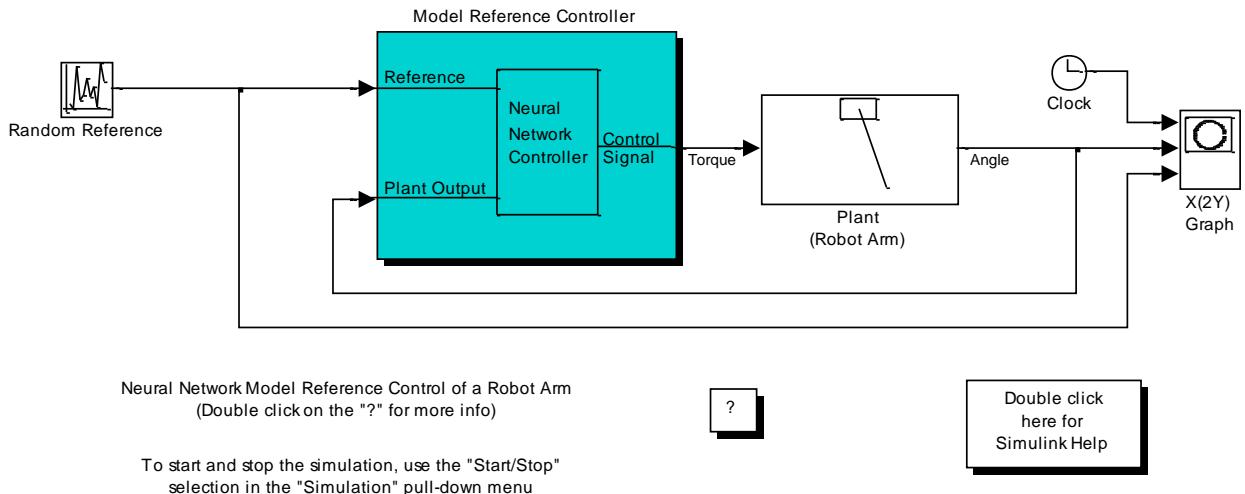


Рисунок 9.19. Схема демонстрационного примера регулятора на основе эталонной модели

В этом демонстрационном примере цель состоит в управлении движением одного звена робота, как это показано на рисунке 9.25.

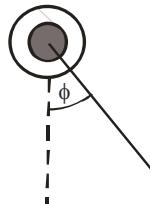


Рисунок 9.25. Схема управления движением звена робота

Уравнения движения звена:

$$\frac{d^2\phi}{dt^2} = -10 \sin \phi - 2 \frac{d\phi}{dt} + u, \quad (9.11)$$

где ϕ – угол поворота звена; u – момент, развиваемый двигателем постоянного тока.

Цель обучения регулятора состоит в том, чтобы движение звена отслеживало выход эталонной модели:

$$\frac{d^2y_r}{dt^2} = -9r - 6 \frac{dy_r}{dt} + u, \quad (9.12)$$

где y_r – выход эталонной модели; r – задающий сигнал на входе модели.

Соответствующая динамическая модель, реализованная в системе Simulink, показана на рисунке 9.26.

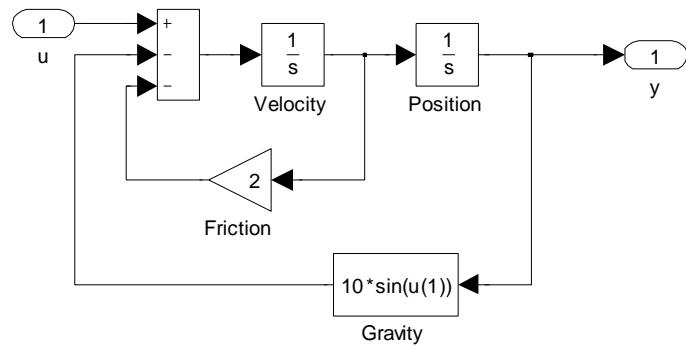


Рисунок 9.26. Динамическая модель звена робота

Структурная схема, поясняющая принцип построения системы управления с эталонной моделью, показана на рисунке 9.27.

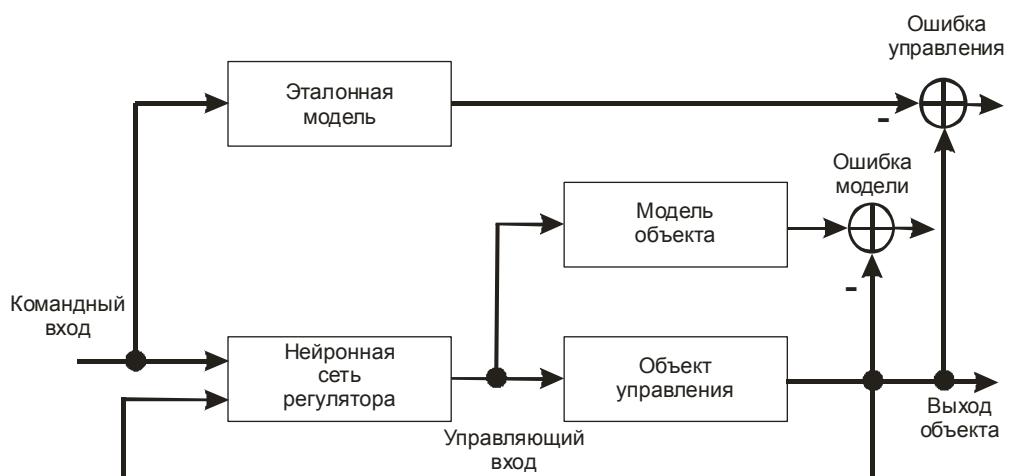


Рисунок 9.27. Схема системы управления с эталонной моделью

В ней следует выделить эталонную модель, которая задает желаемую траекторию движения звена робота, удовлетворяющую дифференциальному уравнению (9.11), а также нейронные сети, реализующие регулятор и модель объекта управления.

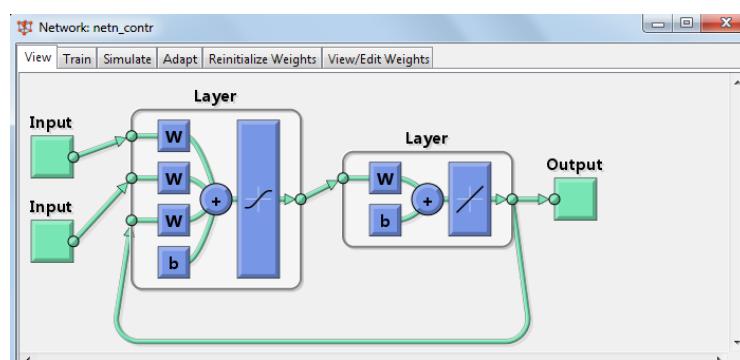


Рисунок 9.28. Схема системы управления с эталонной моделью

Архитектуру нейронной сети регулятора можно описать профилем 2-13-1 (2 входа, 13 нейронов скрытого слоя и 1 выход – рисунок 9.28).

Выполнить запуск демонстрационного примера можно несколькими способами:

- в окне запуска приложений *Launch Pad* выбрать опцию *Demos* для ППП Neural Network Toolbox;
- ввести команды *mrefrobotarm* или *mrefrobotarm2* в командном окне системы MatLab в зависимости от того, используется ли модель с ненормированными или нормированными данными.

Для того чтобы начать работу, необходимо активизировать блок *Model Reference Controller* двойным щелчком левой кнопки мыши. Появится окно, показанное на рисунке 9.29.

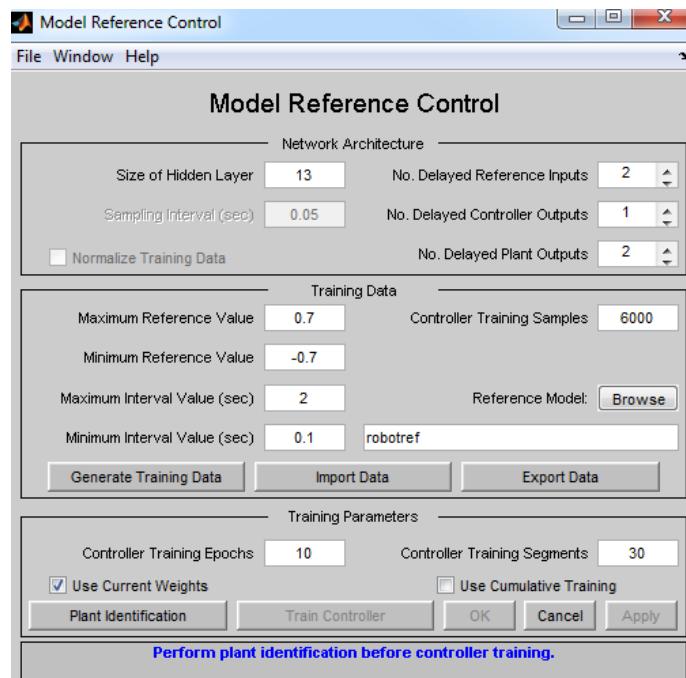


Рисунок 9.29. Окно системы управления на основе эталонной модели

Особенность рассматриваемой системы управления заключается в том, что следует построить две нейронные сети: модели объекта управления и самого регулятора. Разумно начать с построения модели объекта управления и в окне, показанном на рисунке 9.29, выбрать кнопку *Plant Identification*. При этом откроется окно *Plant Identification*, которое показано на рисунке 9.30.

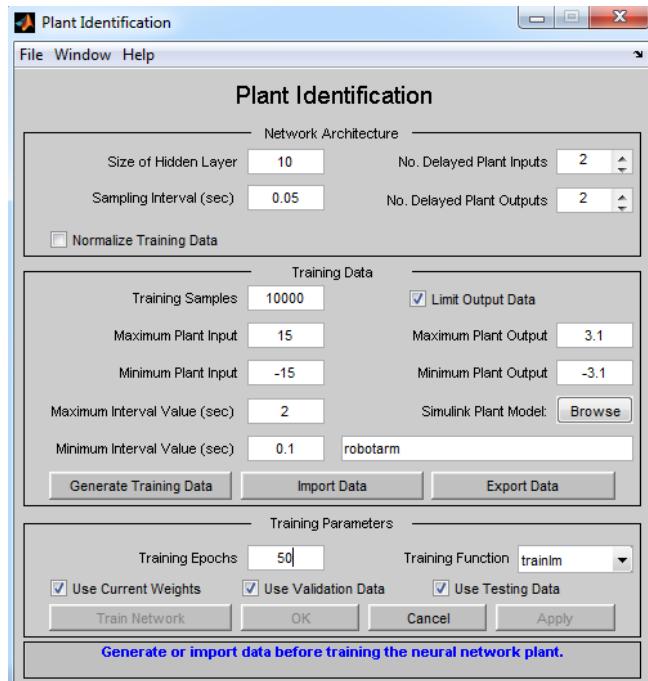


Рисунок 9.30. Окно задания параметров системы управления на основе эталонной модели

Для построения нейросетевой модели объекта управления следует выбрать кнопку *Generate Training Data* и далее следовать рассмотренной ранее процедуре идентификации. Параметру *Training Epochs* целесообразно установить значение, равное 50, чтобы сократить время вычислений. Если результаты идентификации вас удовлетворяют, то следует подтвердить это нажатием кнопки *Accept Data* в окне *Plant Input-Output Data*, в противном случае выбрать кнопку *Reject Data* и затем сформировать новые данные. После этого вы вновь возвращаетесь в окно *Model Reference Control* (рисунок 9.29).

Теперь для обучения регулятора следует выбрать кнопку *Generate Training Data*, чтобы сгенерировать обучающие данные. Эти данные в виде графиков появятся в окне *Input-Output Data for NN Model Reference Control*, и вновь необходимо подтвердить или отвергнуть эти данные. Если данные приемлемы, то в окне *Model Reference Control* следует выбрать кнопку *Train Controller* (Обучить регулятор). После того как обучение окончено, графики выходов эталонной модели и объекта управления выводятся на экран. Обучение регулятора занимает весьма значительное время, поскольку обучение использует динамический вариант метода обратного распространения ошибки.

Если точность слежения за эталонной моделью неудовлетворительна, то можно продолжить обучение регулятора с тем же набором данных, снова воспользовавшись кнопкой *Train Controller*. Если для продолжения обучения необходимо использовать новый набор данных, следует воспользоваться кнопками *Generate Data* или *Import Data*, причем, если вы хотите продолжить обучение с выбранными весами, следует сделать отметку в окне контроля *Use Current Weights*.

По окончании обучения регулятора нажать на клавишу ОК, вернуться к модели Simulink и начать моделирование, выбрав опцию Start из меню Simulation. Из анализа полученных данных (рисунок 9.31) следует, что реакция системы на ступенчатые воздействия со случайной амплитудой носит монотонный характер и отрабатывается в пределах 60 с. Это свидетельствует о хорошем качестве регулятора *Model Reference Controller* для управления звеном роботоманипулятора.

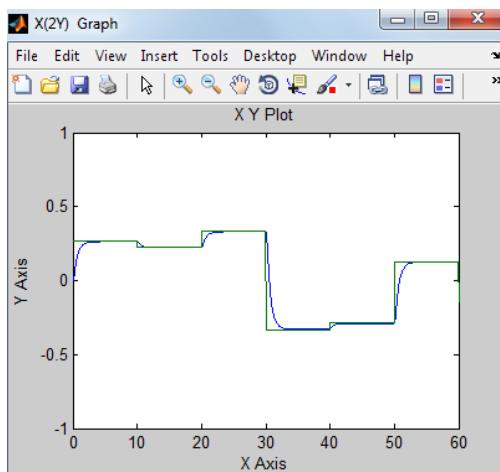


Рисунок 9.31. Задающий и выходной сигналы системы управления на основе эталонной модели

Содержание отчета

- цель работы;
- краткое описание действий по пунктам;
- графики по всем пунктам программы;
- выводы по работе.