МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)
Кафедра информационных систем и технологий

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
« 16 » 04 2019 г.

Нейронные сети и нечеткие системы

Методические указания по выполнению лабораторных работ для направления подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем»

Курск 2019

УДК 004

Составитель С.Ю. Сазонов, Е.А. Кулешова

Рецензент
Кандидат технических наук, доцент Ю.А. Халин

Нейронные сети и нечеткие системы: методические указания по выполнению лабораторных работ для направления подготовки «Математическое обеспечение и администрирование информационных систем» / Юго-Зап. гос. ун-т; сост. С.Ю.Сазонов, Е.А. Кулешова. Курск, 2019. 62 с.: Библиогр.: с. 62.

Методические рекомендации предназначены для студентов, обучающихся по направлению подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем»

Текст печатается в авторской редакции.

Подписано в печать 16.09.19. Формат 60x84 1/16.
Усл.печ. л. 16,1. Уч.-изд. л. 14,6. Тираж 100 экз. Заказ. 334 Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94

Лабораторная работа №1

GUI-ИНТЕРФЕЙС ДЛЯ ПАКЕТА NEURAL NETWORKS TOOLBOX

Цель работы: изучение основных свойств и основ работы с GUI – интерфейсом пакета Neural Networks Toolbox в программной среде MatLab 7.

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].

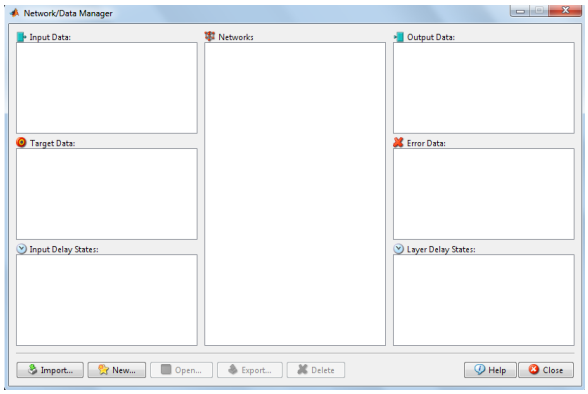

Рисунок 1.1. Окно управления сетью/данными

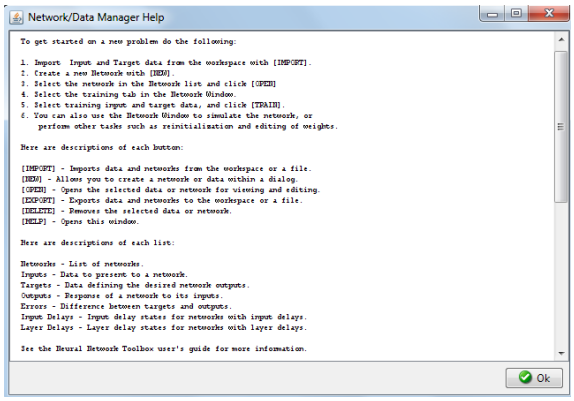

Рисунок 1.2. Окно подсказки

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 Data… – кнопка вызова окна формирования данных (рис. 1.3);

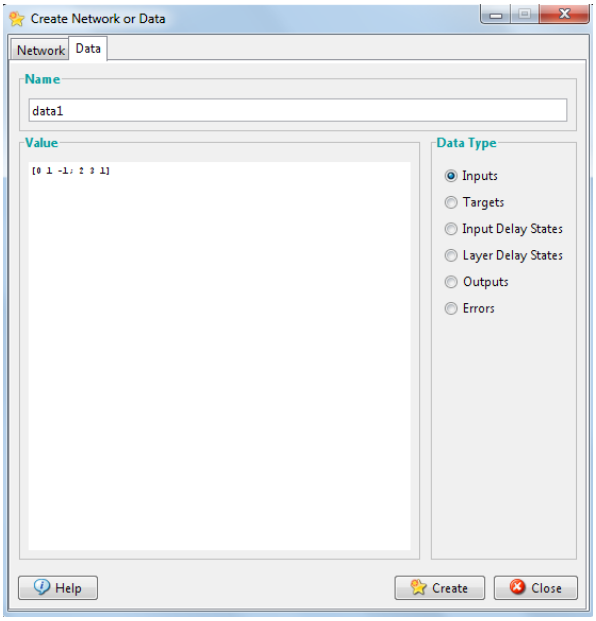

Рисунок 1.3. Окно форматирования данных

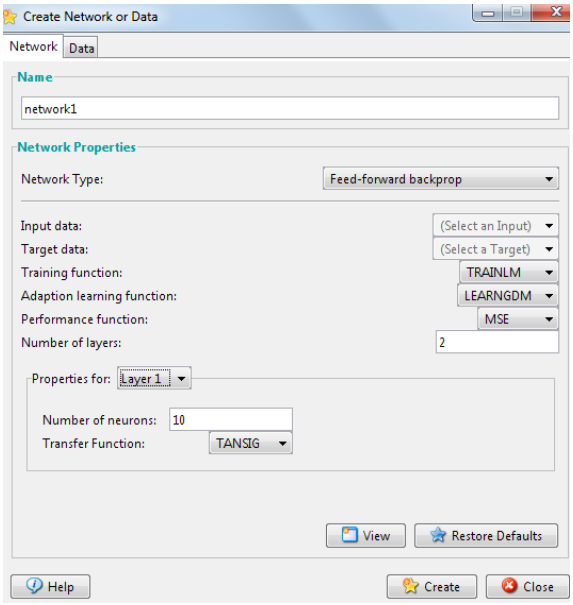

Рисунок 1.4. Окно создания новой нейронной сети

New Network… – кнопка вызова окна 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 систем 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 лини задержки на входе;

Layer Delay States (Состояния ЛЗ слоя) – начальные условия лини 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.

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

Properties for (Свойства) – список слоев.

Namber 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 114аспространениием 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.

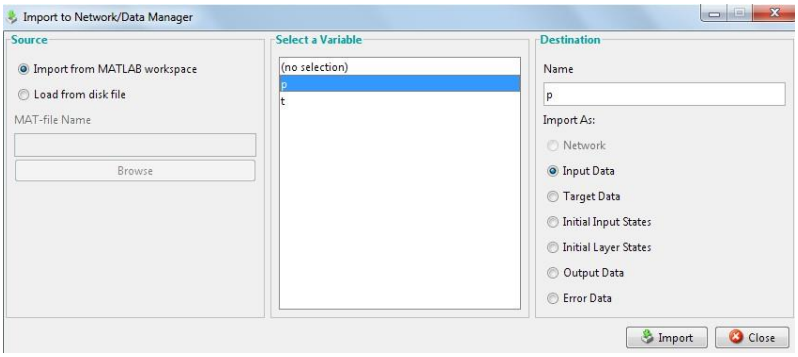

Рисунок 1.5. Окно для импорта и загрузки данных

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.

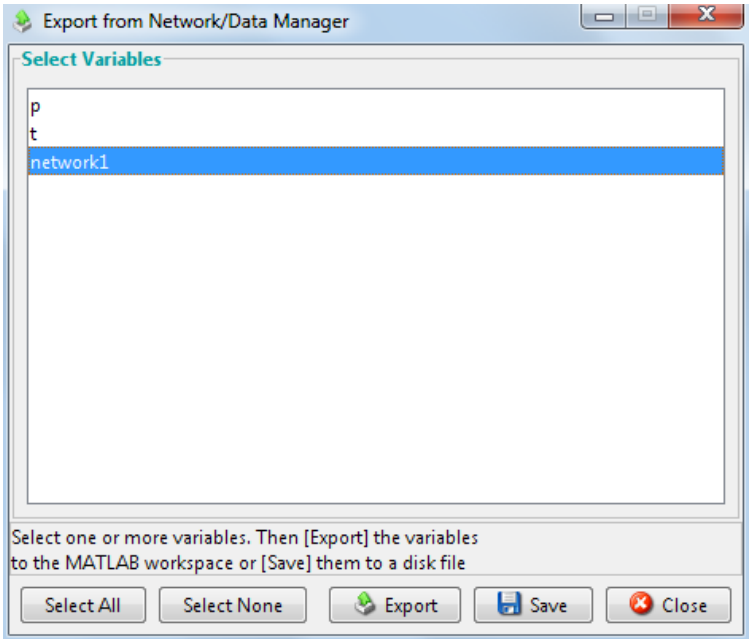

Рисунок 1.6. Окно для экспорта или записи данных в файл

Диалоговая панель *Network* показана на рисунке 1.7.

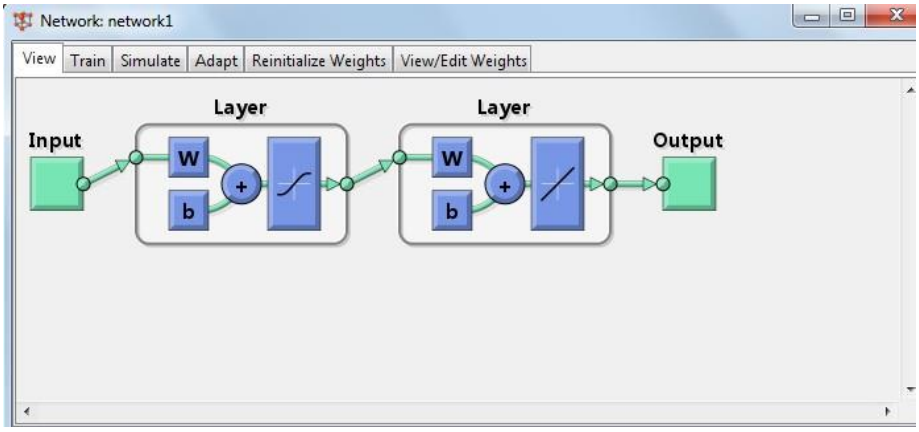

Рисунок 1.7. Диалоговая панель Network

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 Data* 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.

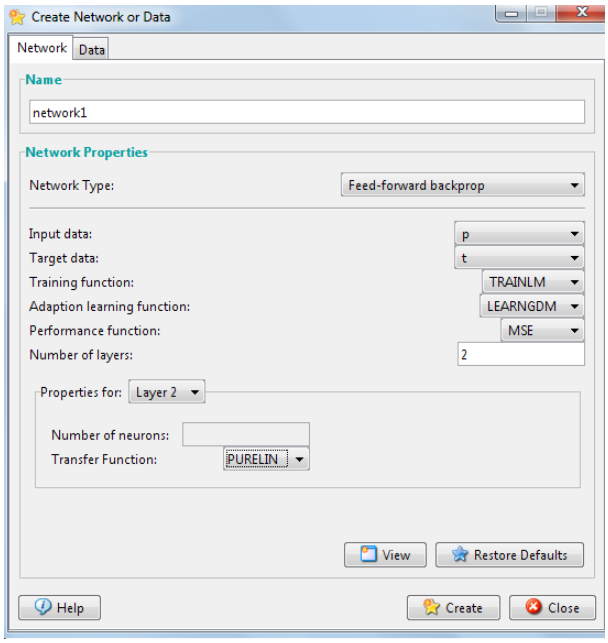

Рисунок 1.8 – Окно создания нейронной сети

Создадим новую нейронную сеть. Для этого в окне *Network/Data Manager* нажмем кнопку *New Network*. В открывшемся окне *Create New Network* 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*), диапазон входов определим (в окне *Create New Network*) с помощью опции *Get from input*, 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.

Для ввода установленных диапазонов и инициализации весов воспользуемся кнопками *Set Ranges* (Установить диапазоны), *Initialize Weights* (Инициализировать веса). Если требуется вернуться к прежним диапазонам, то следует выбрать кнопки *Revert Ranges* (Вернуть диапазоны) и *Revert Weights* (Вернуть веса), но в условиях примера это не нужно.

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.

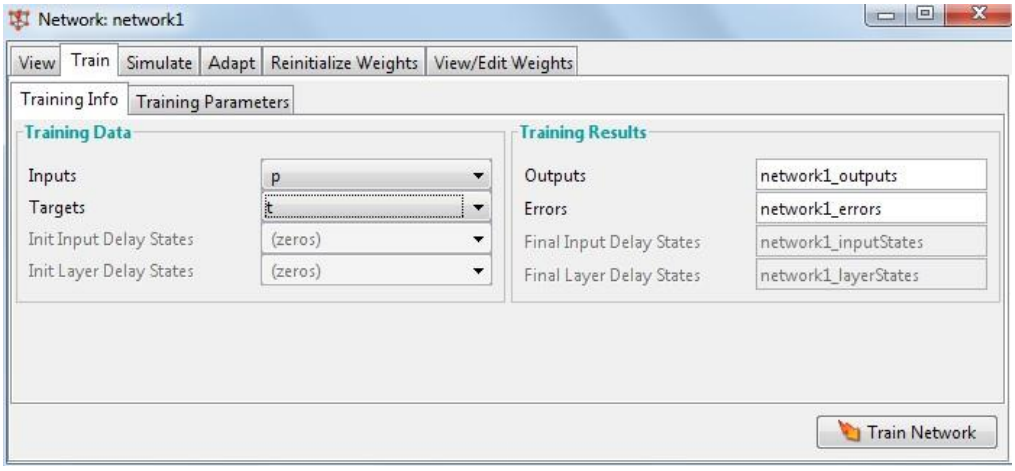

Рисунок 1.9. Окно информации об обучающих последовательностях

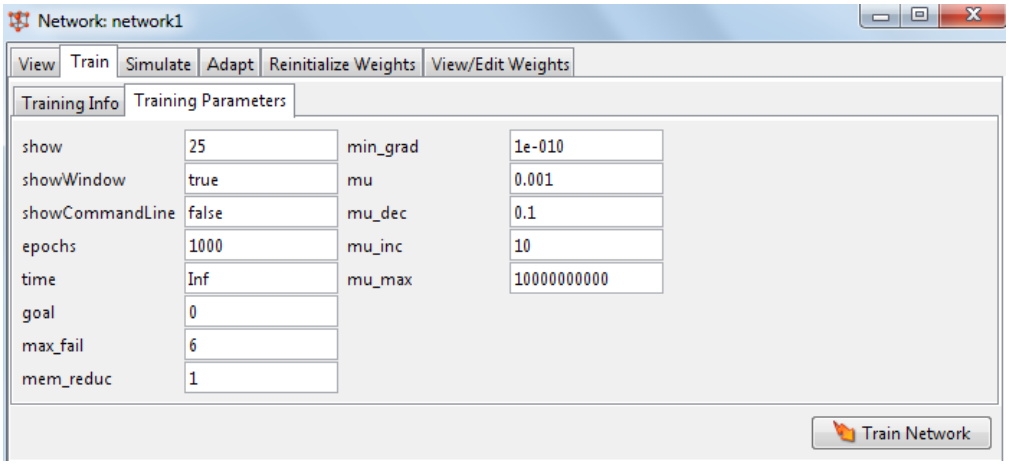

Рисунок 1.10. Окно с информацией о параметрах обучения

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).

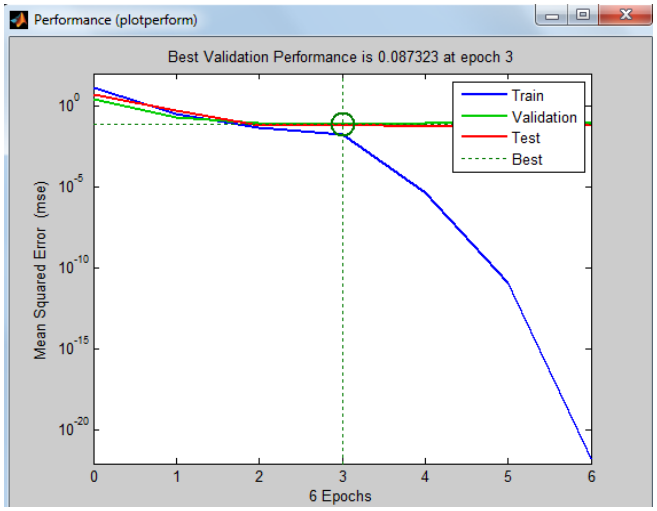

Рисунок 1.11. Изменение ошибки сети в процессе обучения

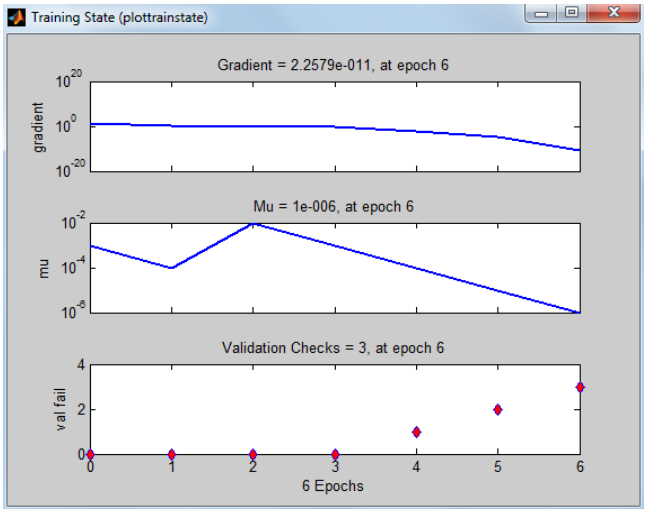

Рисунок 1.12. Окно состояния обучения

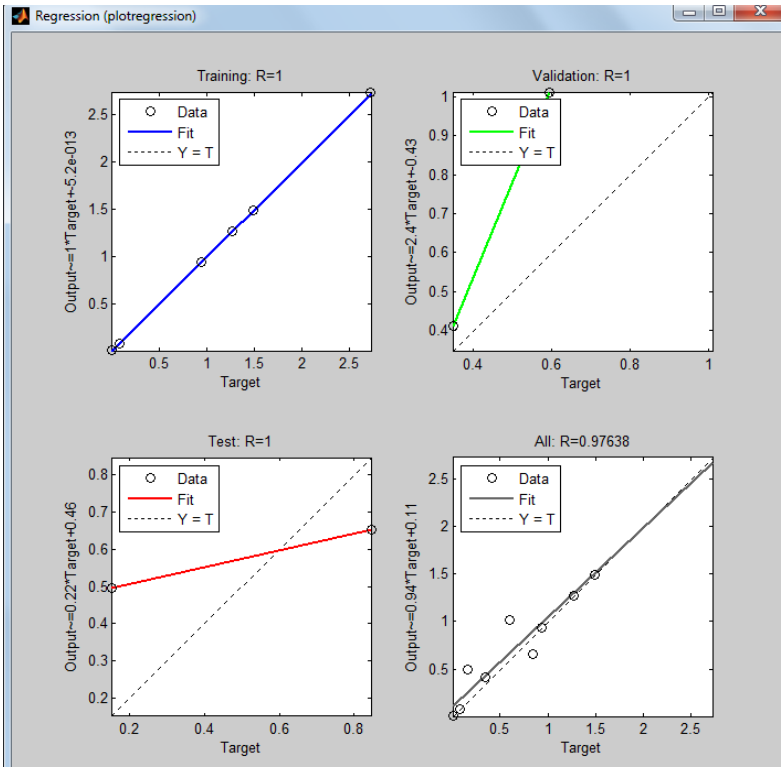

Рисунок 1.13. Окно линейной регрессии между выходом НС и целями

Результаты обучения можно просмотреть в окне *Network/Data Manager*, выбрав кнопку *Manager*. Появится окно (рисунок 1.14), в котором, активизируя имена последовательностей выходов *network1_outputs* или ошибок *network1_errors*, можно просмотреть результаты, используя кнопку *View* (рисунок 1.15).

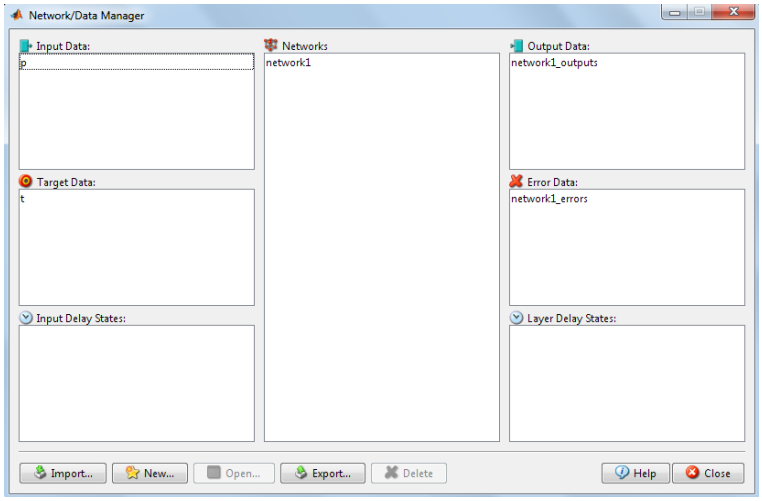

Рисунок 1.14. Окно Network/Data Manager

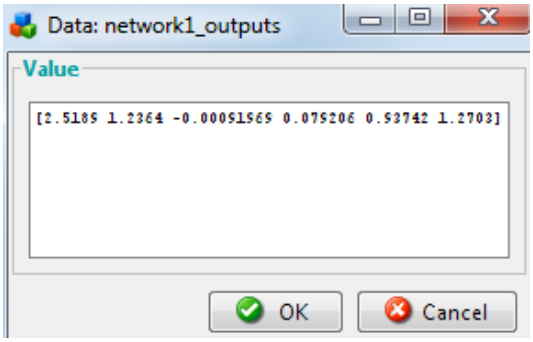

Рисунок 1.15. Значения выходов сети

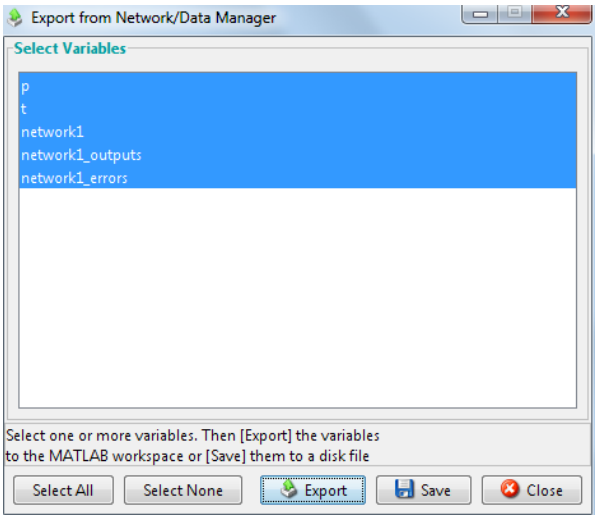

Рисунок 1.16. Окно для экспорта или записи данных в файл

При необходимости можно экспортировать 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.

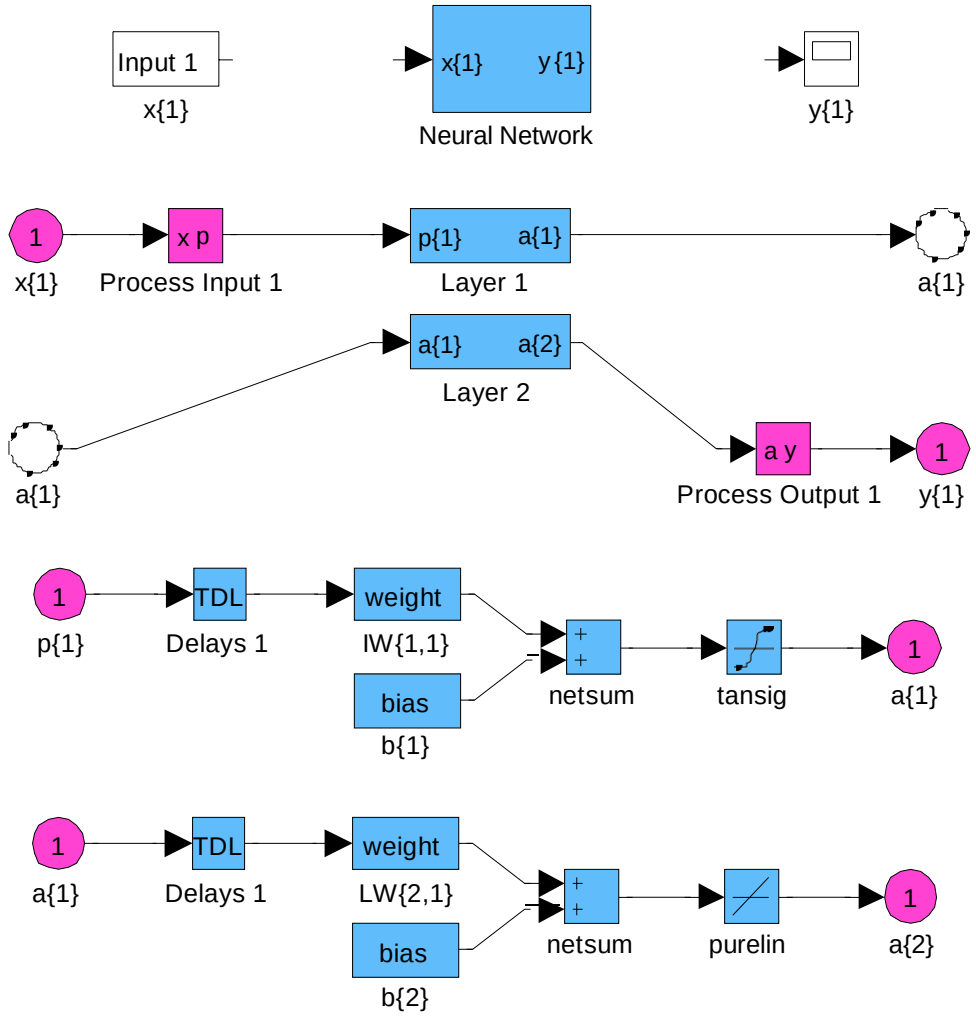

Рисунок 1.17. Структурные схемы, созданной нейронной сети в среде Simulink

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.	$y = 2\sin^2(x_1) \cdot \cos(x_2),\ x_1, x_2 \in [-1;1]$
2.	$y = 4\frac{\cos(x_1)}{x_2},\ x \in [1;\pi]$
3.	$y = x_1 \cdot \cos(x_2),\ x_1, x_2 \in [-1;1]$,
4.	$y_1 = \frac{x_1 - x_2}{x_1 + x_2},\ x_1, x_2 \in [1;5]$
5.	$y = \cos(x_2) \cdot \sin(x_1),\ x_1, x_2 \in [-1;1]$
6.	$y = x_1 \cdot \sin(x_2),\ x_1, x_2 \in [-1;1]$
7.	$y_1 = x_1 \cdot x_2 + \sin x_2,\ x_1, x_1 \in [-\pi;1]$
8.	$y_1 = 1{,}5 \cdot x_1 + x_1^3,\ x_1, x_2 \in [-1;1]$
9.	$y_1 = \sqrt{x_1^2 + x_2^2},\ x_1, x_2 \in [-1;1]$
10.	$y_1 = 2{,}3 \cdot x_1 \cdot x_2 - 0{,}5 \cdot x_1^2 + 1{,}8 \cdot x_2^2,$ $x_1, x_2 \in [1;10]$
11.	$y = 6 \cdot x_1 + 5 \cdot x_1 \cdot x_2 + x_2^2,\ x_1, x_2 \in [-5;5]$
12.	$y = \sin(x_1) \cdot \cos(x_2),\ x_1, x_2, \in [-\pi;\pi]$
13.	$y = 2 \cdot x_1 \cdot \cos x_2,\ x_1 \in [0;1], x_2 \in [-\pi;\pi]$
14.	$y = 2 \cdot x_1^2 + \cos(x_2),\ x_1, x_2 \in [-1;1]$
15.	$y = 2 \cdot x_2 \cdot \sin^2(x_1),\ x_1, x_2 \in [-1;1]$
16.	$y = 4\frac{\sin(x_2)^2}{\cos(x_1)},\ x_1, x_2 \in [-\pi;\pi]$
17.	$y = arctg\frac{x_2}{x_1},\ x_1, x_2 \in [-1;1]$
18.	$y = (3 \cdot x_1^2 - x_2^2)^3,\ x_1, x_2 \in [-1;1]$
19.	$y = 2 \cdot x_2 \cdot \sin^2(x_1),\ x_1, x_2 \in [-1;1]$
20.	$y = 2 \cdot tg(x_2)^2 \cdot \cos(x_1),\ x_1, x_2 \in [-1;1]$

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 персептрона, имеет ступенчатую функцию активации hardlim с жесткими ограничениями (рисунок 2.1).

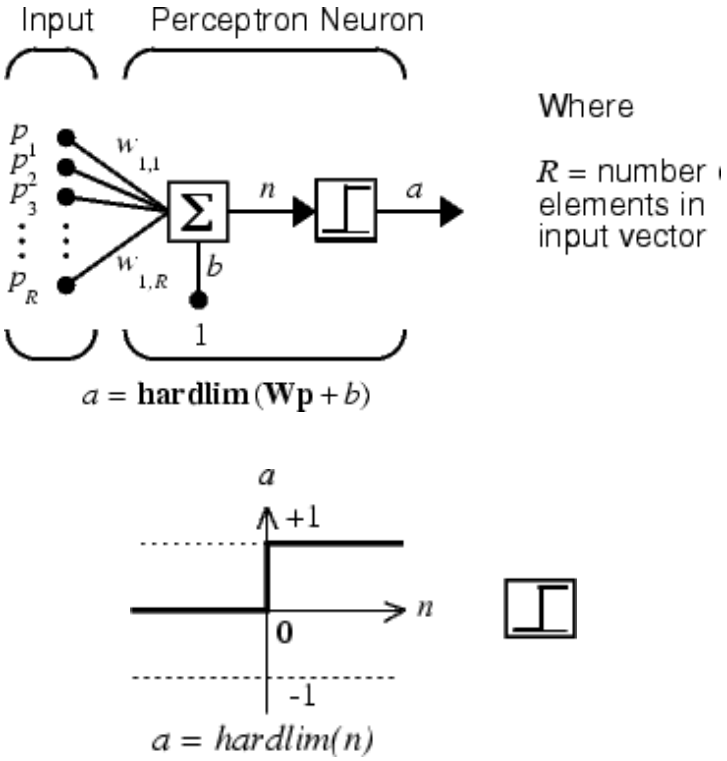

Рисунок 2.1. Структура порогового нейрона

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

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

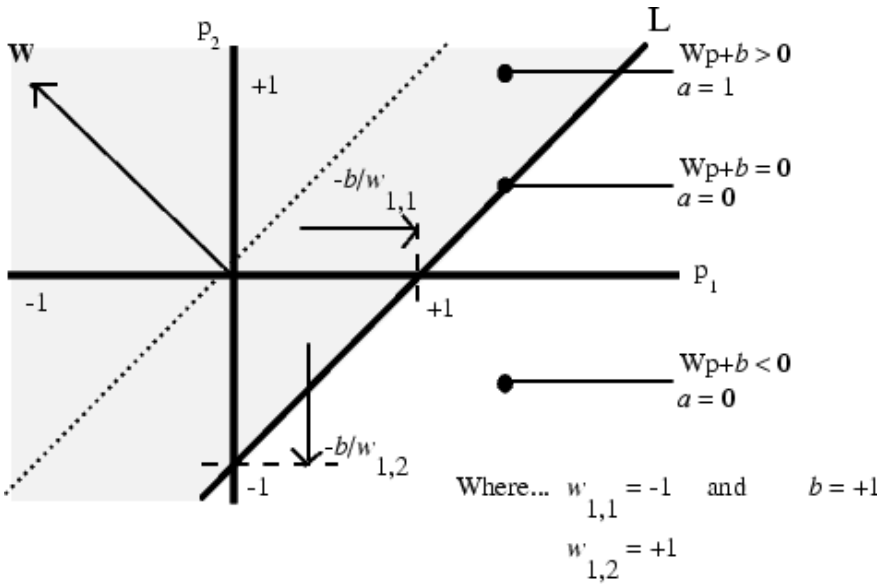

Рисунок 2.2. Разделение нейроном пространства входа на две области

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 персептрона с двумя входами и смещением.

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

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

Эта линия перпендикулярна к вектору весов w и смещена на величину b. Векторы входа выше линии L соответствуют положительному потенциалу нейрона, и, следовательно, выход персептрона для этих векторов будет равен 1; векторы входа ниже линии L соответствуют выходу персептрона, 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 learnp 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:

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

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

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] = train(NET, P, T, P_i, A_i)

[net, tr] = train(NET, P, T, P_i, A_i W, TV).

Функция 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] = 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**
hardlim (N)	Возвращает 1, если N положительное и 0 в противном случае
tansig (N)	Вычисляет гиперболический тангенс от входа
purelin	Вычисляет выход слоя от сетевого входа

Таблица 2.2. Функции графического интерфейса и вспомогательные функции

Функция	**Назначение**
axis([Xmin Xmax Ymin Ymax])	Устанавливает диапазоны координатных осей
title {'строка')	Выводит в графическое окно рисунков заголовок графика
rand(M, N)	Возвращает матрицу размерности M на N со случайными значениями
xlabel('строка') ylabel('строка')	Подписывают наименование координатных осей
cla reset	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 B размерности M, ES – матрица ошибки размерности Л/на N, V– угол зрения по умолчанию [-37,5, 30]
plotsom(POS)	Изображает позицию нейрона красной точкой, связывая синей линией нейроны, 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 P1.

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

Y = sim(net, P1); % имитация работы нейрона net на последовательности входов P1 желаемых выходов – T, 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 P1

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 P и выходы T.

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 P2

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

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

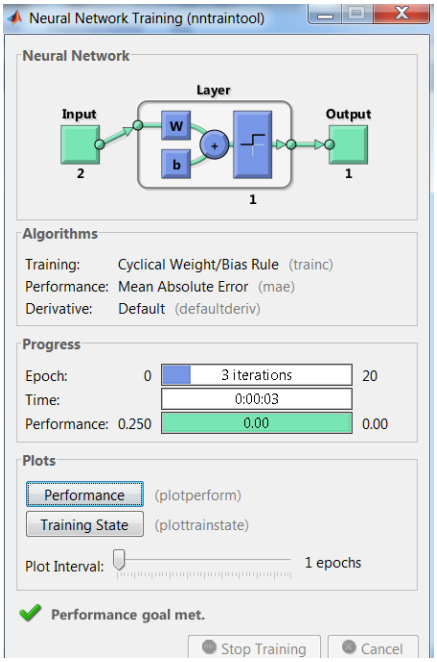

Рисунок 2.3. Окно обучения нейронной сети

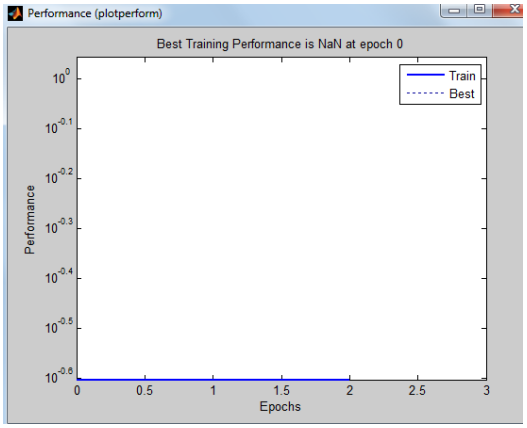

Рисунок 2.4. Изменение ошибки сети в процессе обучения

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 hard-lim.

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)

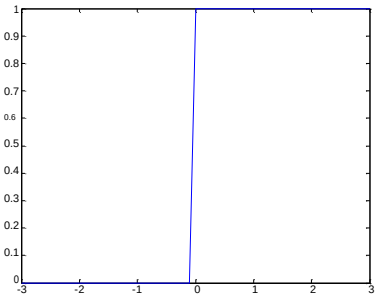

Рисунок 2.5. График функции активации hardlim

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 *newp* 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 двухэлементных 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)

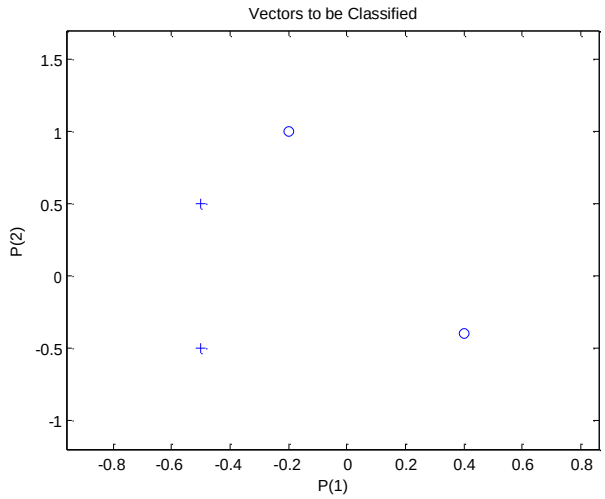

Рисунок 2.6. Изображение входных векторов на плоскости

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

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 T

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 разделяющей прямой нейрона после адаптации

drawnow; % очистка окна графиков

end; % конец цикла while

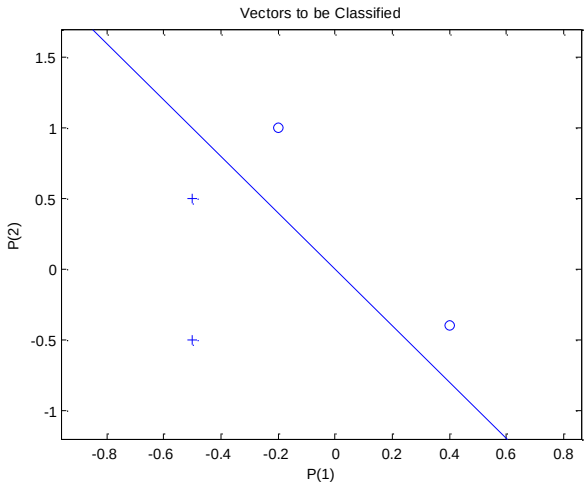

Рисунок 2.7. Изображение разделения пороговым нейроном входных векторов

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

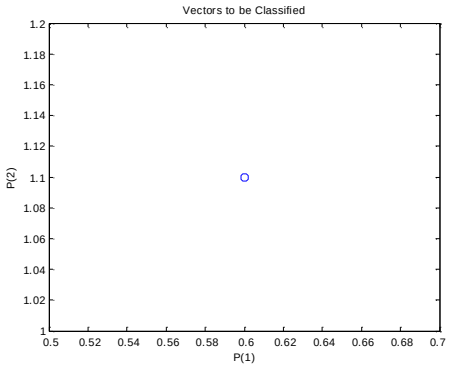

Рисунок 2.8. Изображение входа P, отнесенного нейроном к категории a

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 T1, который известным образом зависит от входов P1):

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 P1

mse(e) % измерение ошибки

ans =

1.1000

Получим довольно большую ошибку. Вновь адаптируем сеть на 10 шагах последовательности, используя предыдущее значение Pf 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 несепарабельные задачи

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

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

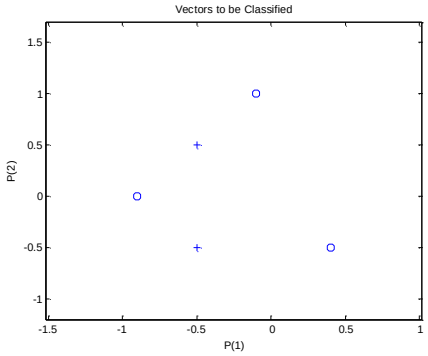

Рисунок 2.9. Изображение входных векторов на плоскости

T = [1 1 0 0 0]; % выходы – реакции нейрона на входы P

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 классифицикации 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 управляющей структуры изображения для новой разделяющей прямой

drawnow; % изображение разделяющей прямой

end; % конец цикла

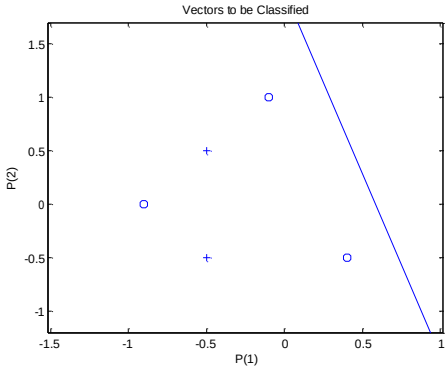

Рисунок 2.10. Изображение невозможности разделения пороговым нейроном линейно неразделимых векторов

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

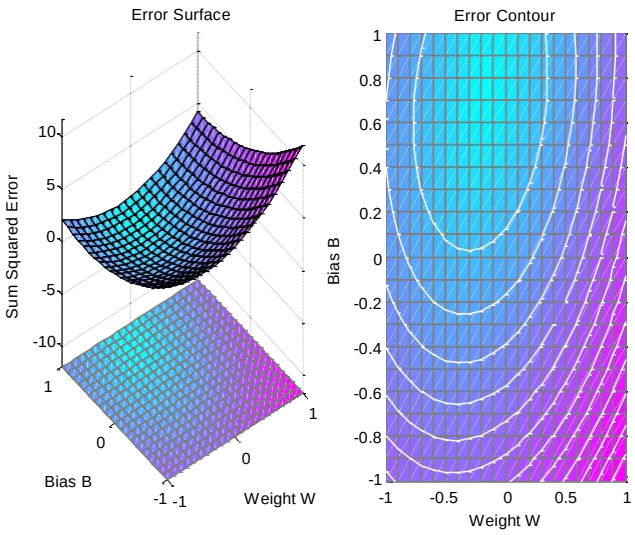

Рисунок 2.11. Изображение поверхности ошибки в координатных осях – весов и пороговых значений

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

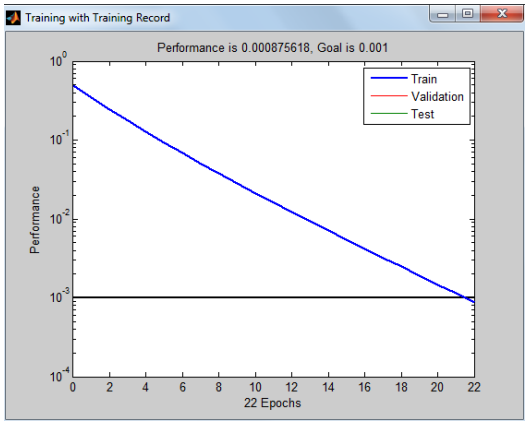

Рисунок 2.12. Изменение среднеквадратической ошибки сети в процессе обучения

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 персептрона 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 облучения 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.

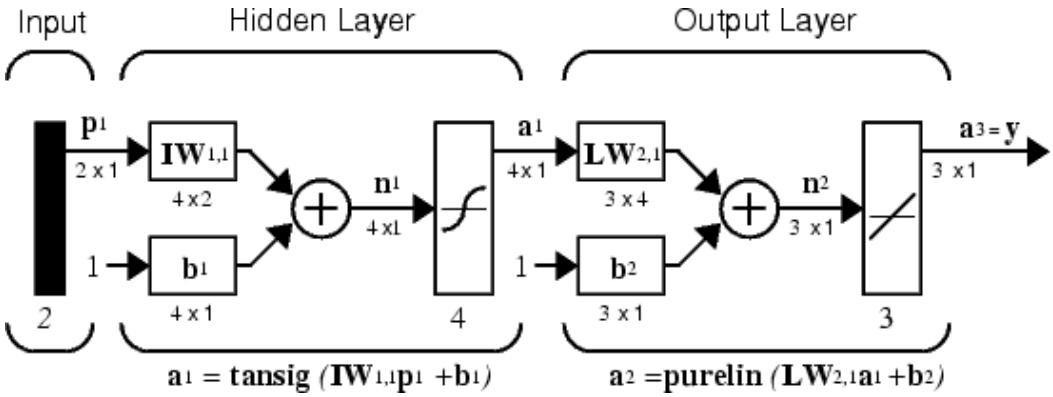

Рисунок 3.1. Схема архитектуры нейронной сети с прямой передачей сигнала

Здесь приняты обозначения: p^1 – 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 SNl], {TF1 TF2 TFNl}, 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-

ственно. 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 P в последовательность целей T:

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

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

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

gensim(net)

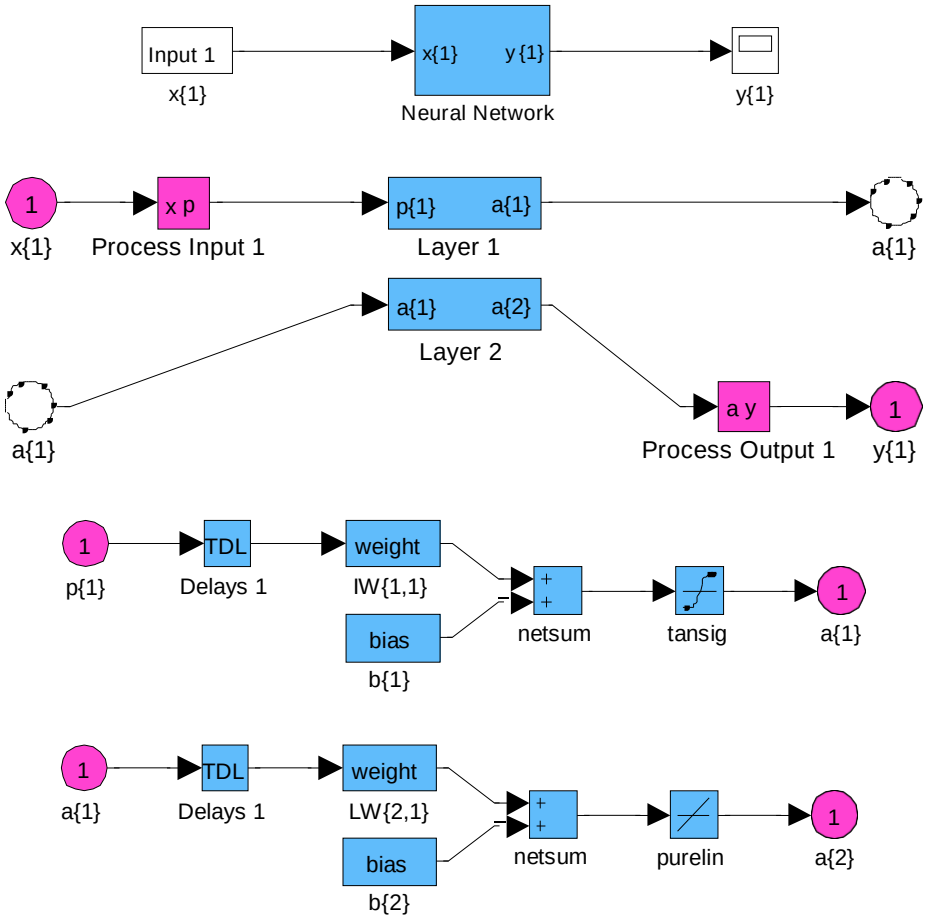

Рисунок 3.2. Структурная схема нейронной сети

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

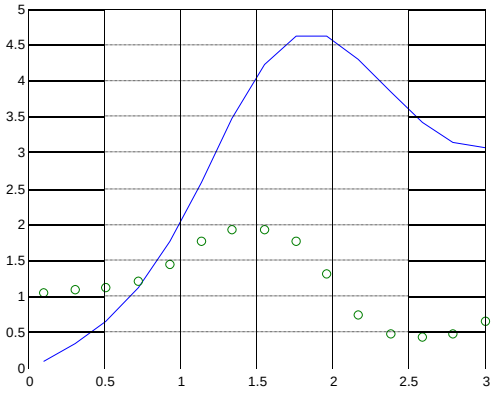

Рисунок 3.3. Графики сигналов выхода и цели

Обучим сеть в течение 50 циклов:

net.trainParam.epochs = 50;

net = train(net, P, T);

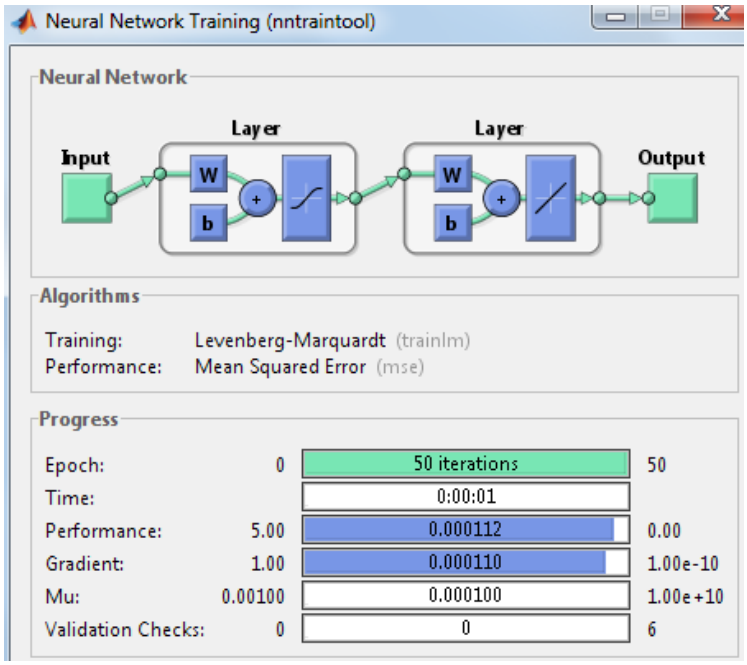

Рисунок 3.4. Окно обучения нейронной сети

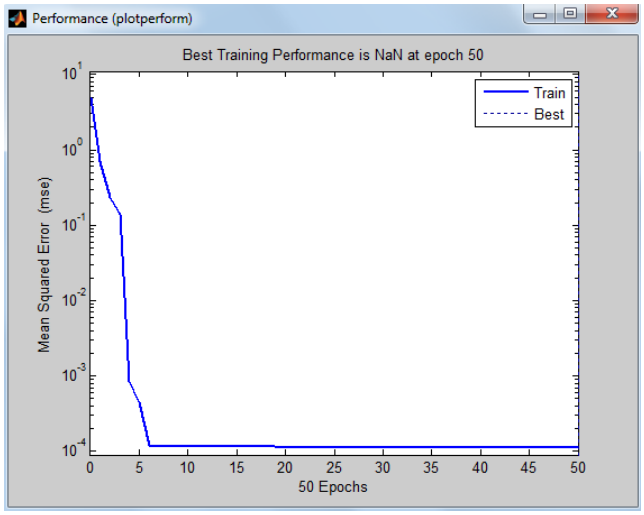

Рисунок 3.5. Изменение ошибки сети в процессе обучения

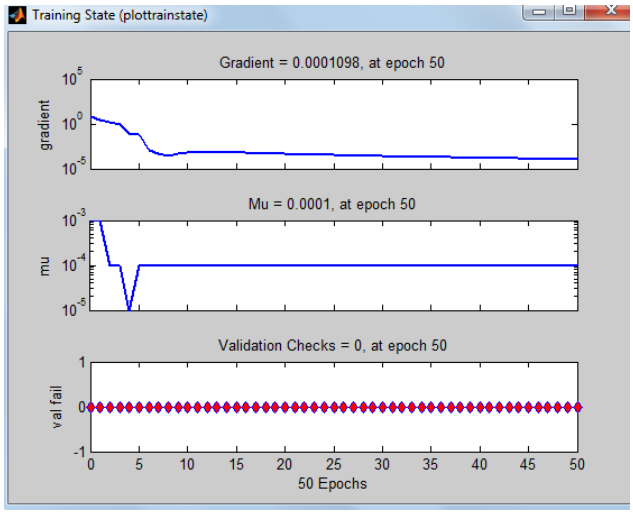

Рисунок 3.6 – Окно состояния обучения

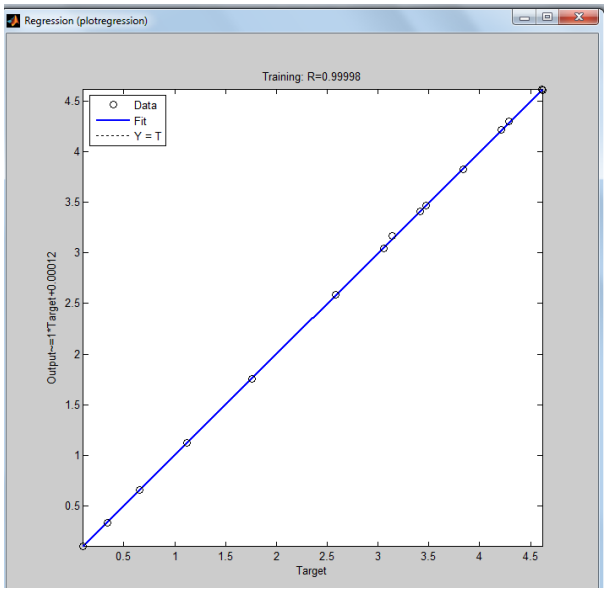

Рисунок 3.7 – Окно линейной регрессии между выходом НС и целями

Выполним моделирование сформированной двухслойной сети, используя обучающую последовательность входа

Y = sim(net, P);

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

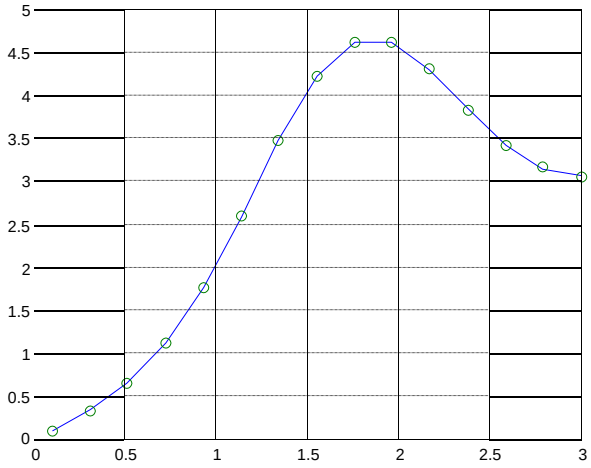

Рисунок 3.8. Результат аппроксимации векторов двухслойным персептроном

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

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 персептрон 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);

Lr – коэффициент обучаемости.

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

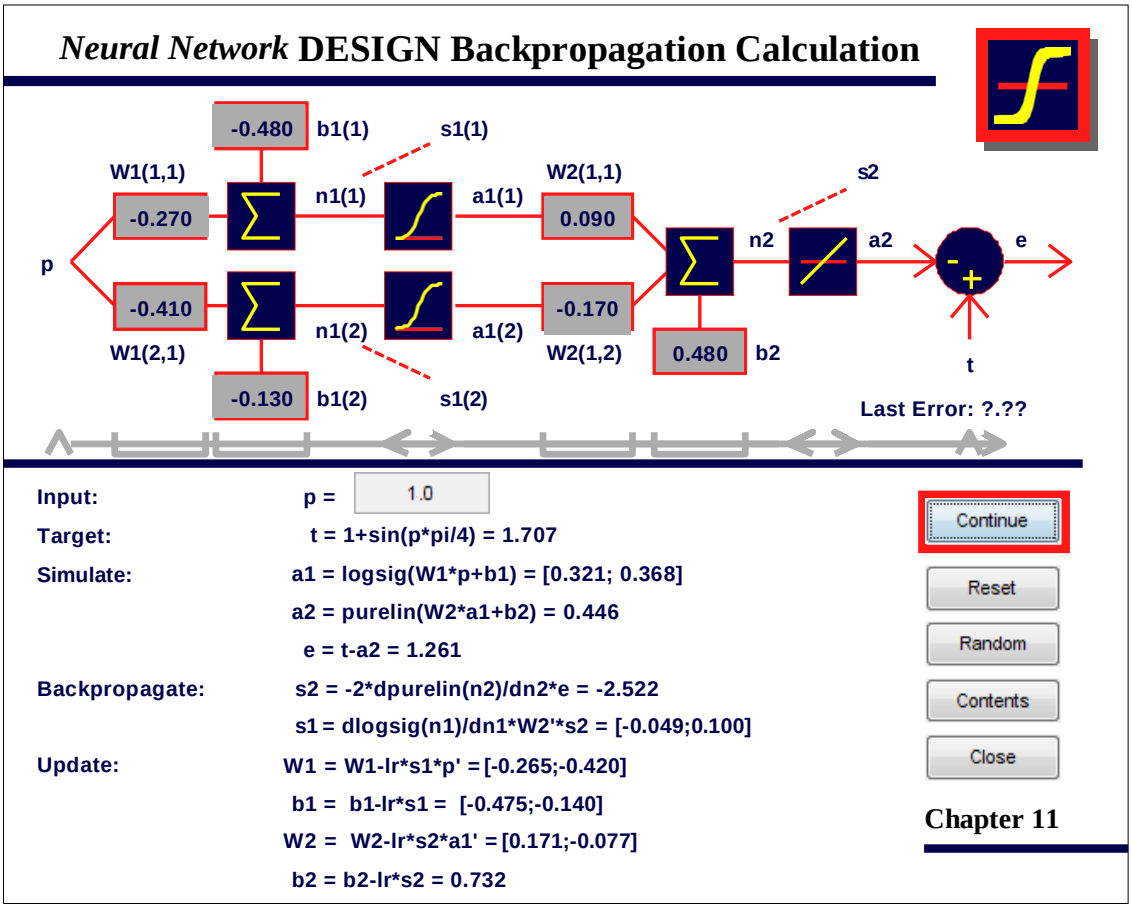

Рисунок 3.9. Demo пример работы алгоритма *Backpropagation*

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.

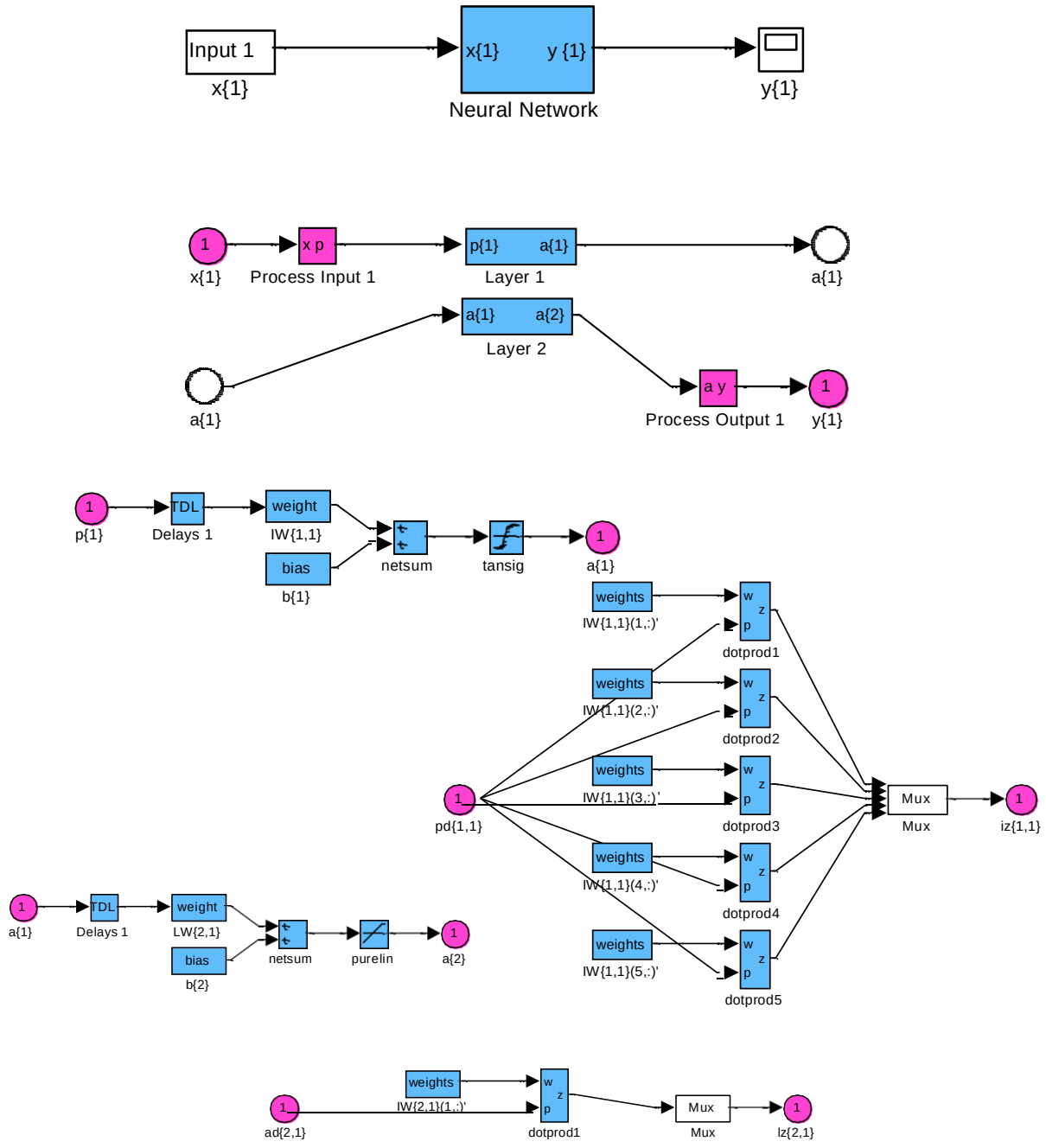

Рисунок 3.10. Структурная схема модели нейронной сети

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 обучении

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

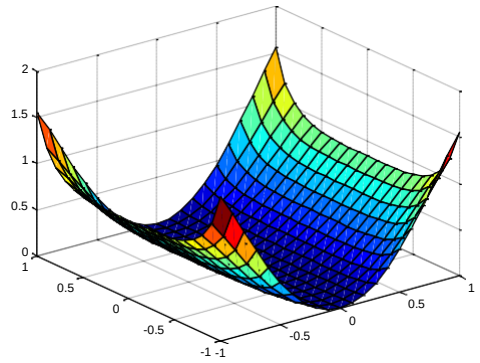

Рисунок 3.11. График поверхности целевой функции

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

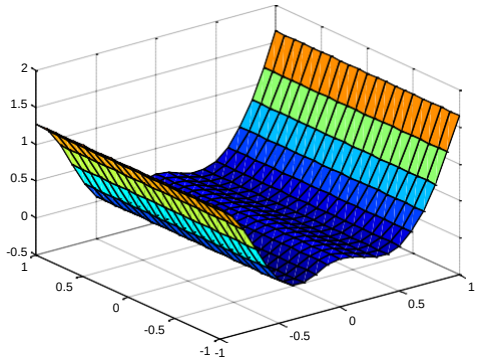

Рисунок 3.12. График поверхности функции аппроксимируемой нейронной сетью

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 линейным 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.

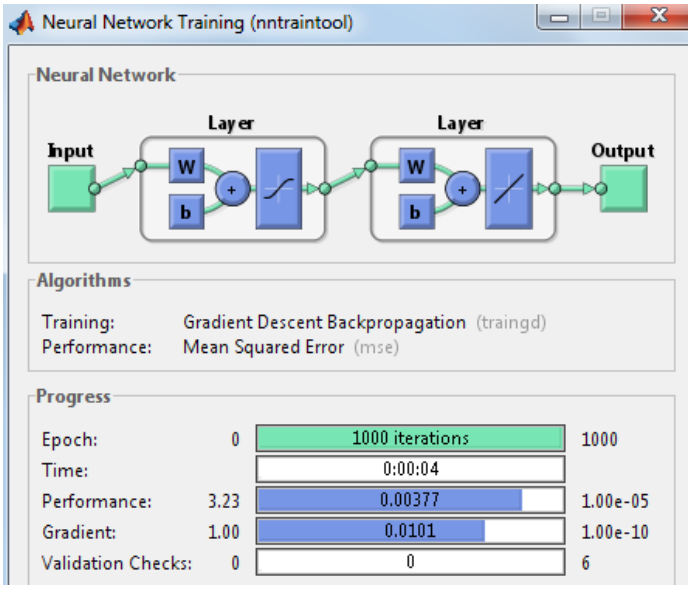

Рисунок 3.13. Окно обучения нейронной сети

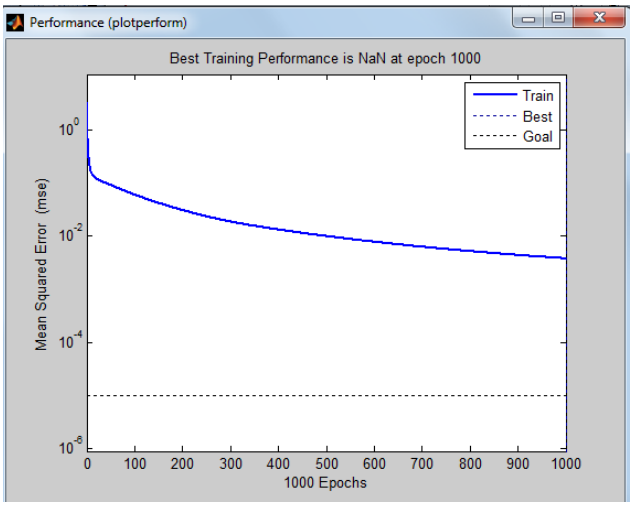

Рисунок 3.14. График изменения ошибки в зависимости от числа выполненных циклов обучения

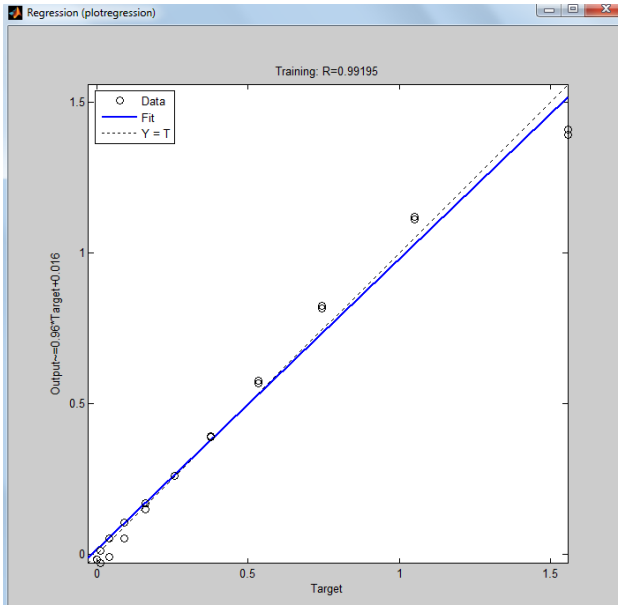

Рисунок 3.15. Окно линейной регрессии между выходом НС и целями

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

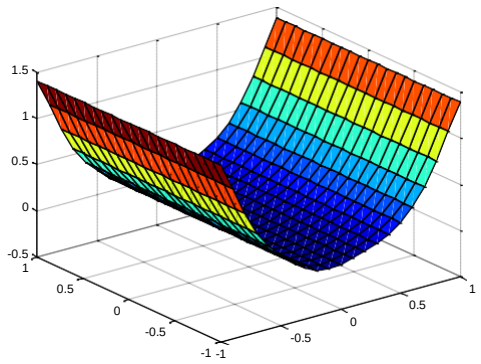

Рисунок 3.16. График поверхности функции аппроксимируемой нейронной сетью

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:

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

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 learngdm

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

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

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

net.layerWeights{2,1}.learnParam.lr=0.2; % 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

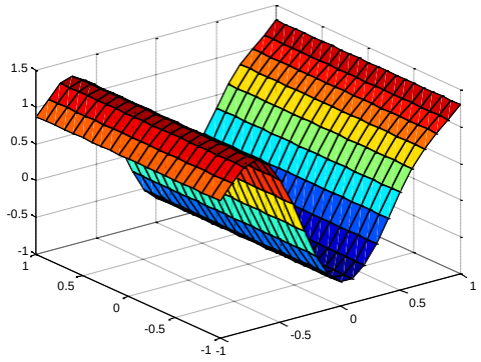

Рисунок 3.17. График поверхности функции аппроксимируемой нейронной сетью

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 learngdm. 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 $J_N \geq 1.04 J_{N-1}$, 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.

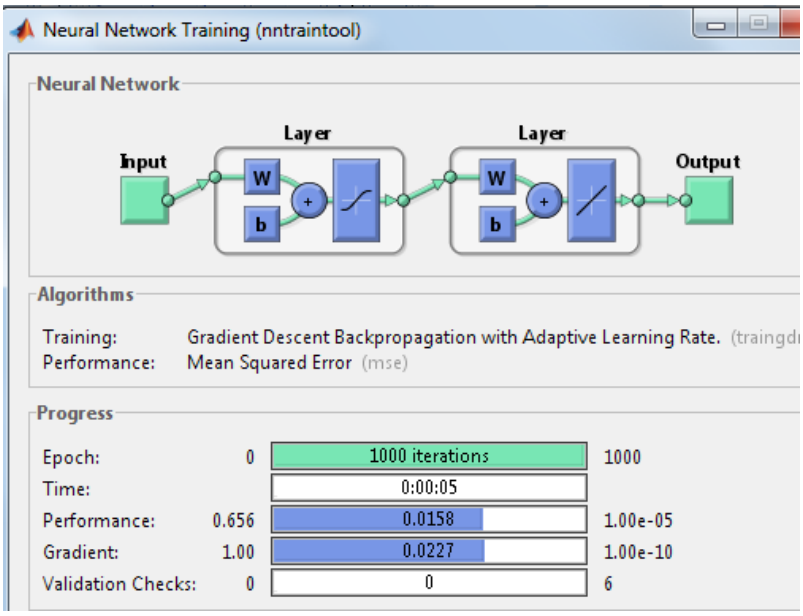

Рисунок 3.18. Окно обучения нейронной сети

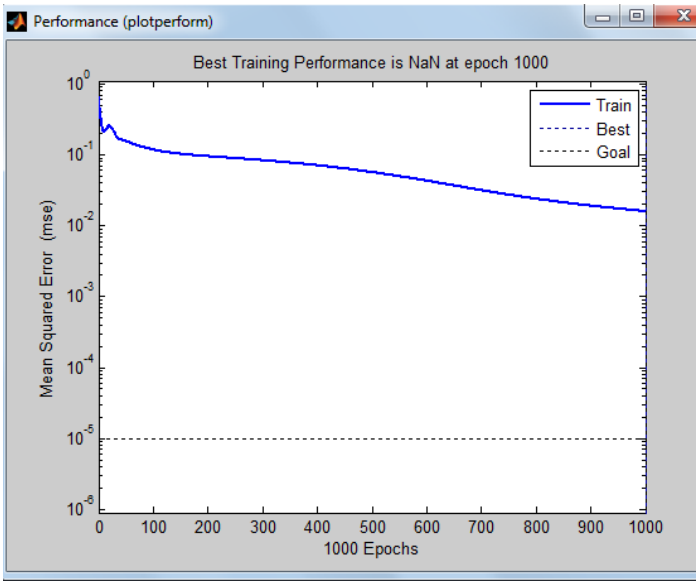

Рисунок 3.19. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

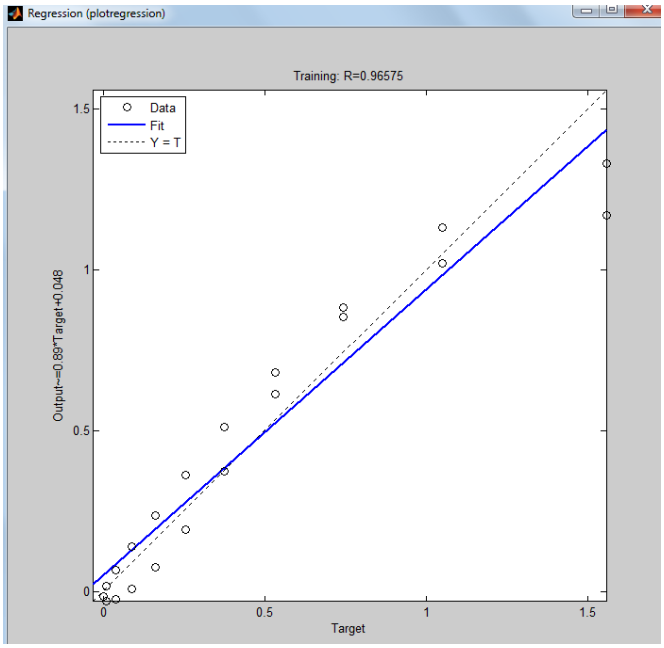

Рисунок 3.20. Окно линейной регрессии между выходом НС и целями

a=sim(net,p);

e=t-a;

mse(e)

ans =

0.0158

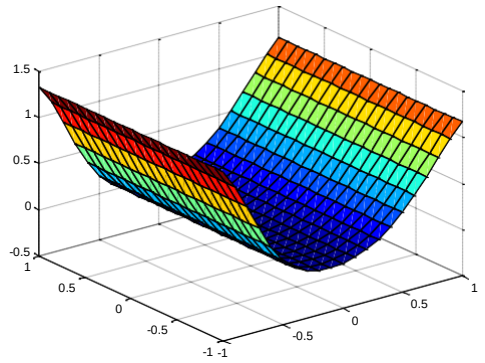

Рисунок 3.21. График поверхности функции аппроксимируемой нейронной сетью

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.

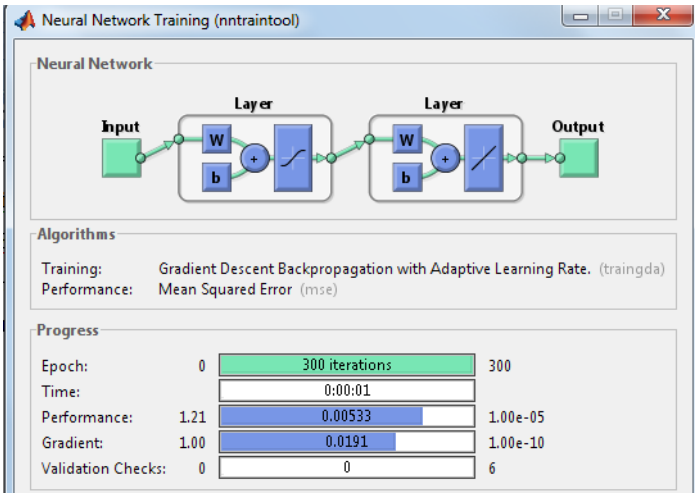

Рисунок 3.22. Окно обучения нейронной сети

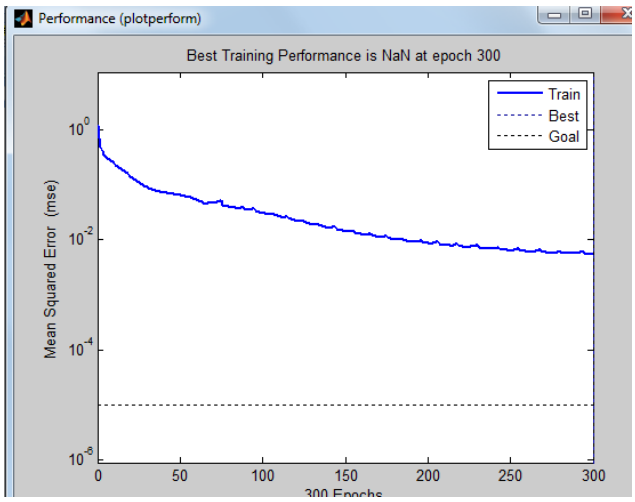

Рисунок 3.23. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

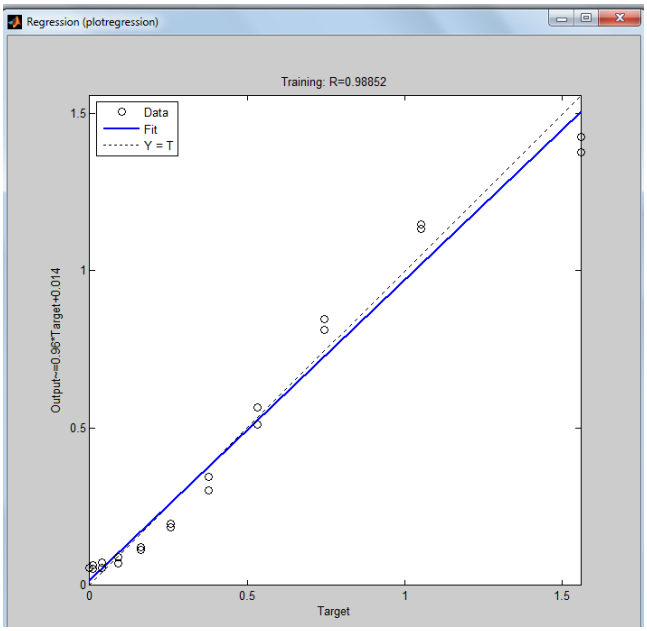

Рисунок 3.24. Окно линейной регрессии между выходом НС и целями

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

epochs: 1000

time: Inf

goal: 0

max_fail: 6

min_grad: 1.0000e-010

delt_inc: 1.2000

delt_dec: 0.5000

delta0: 0.0700

deltamax: 50

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.

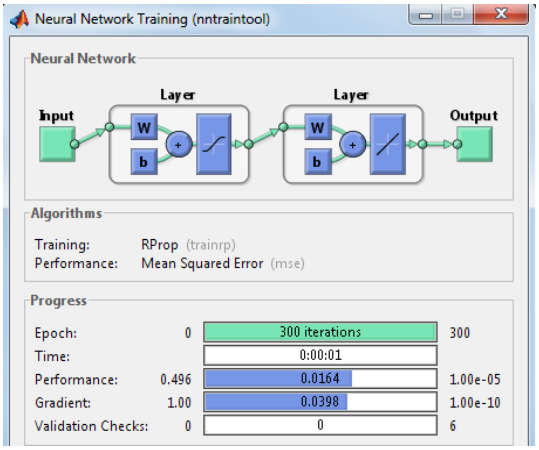

Рисунок 3.26. Окно обучения нейронной сети

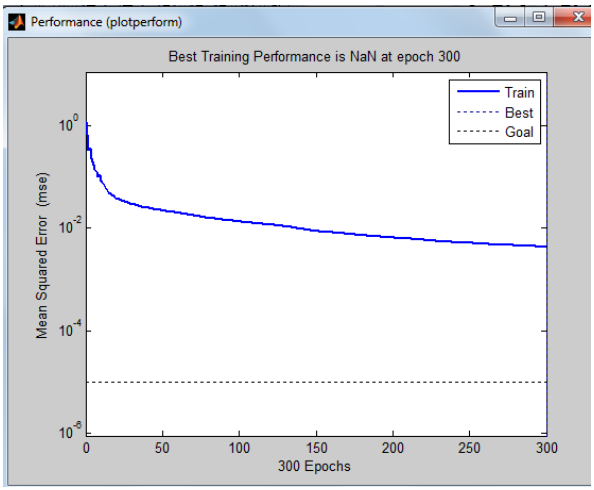

Рисунок 3.26. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

a=sim(net,p);

e=t-a;

mse(e)

0.0043

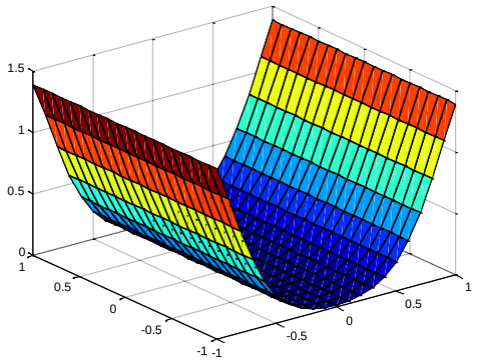

Рисунок 3.27. График поверхности функции аппроксимируемой нейронной сетью

Нетрудно заметить, что количество циклов обучения по сравнению с алго- ритмом GDA сократилось практически еще в 3 раза и составило по отношению к алгоритму GD значение, близкое к9.

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*.

Содержание отчета

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

Контрольные вопросы к лабораторным работам

GUI-интерфейсдляпакета Neural Networks Toolbox

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

Изучение свойств линейного нейрона и линейной нейронной сети

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

Изучение многослойного нелинейного персептрона и алгоритма обратного распространения ошибки

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

Библиографический список

1. Емельянов, С. Г. Интеллектуальные системы на основе нечеткой логики и мягких арифметических операций [Текст] : учебник / С. Г. Емельянов, В. С. Титов, М. В. Бобырь. - Москва : Аргамак-Медиа, 2014. - 338, [7] с. : табл., граф.
2. Галушкин, А. И. Нейронные сети: основы теории [Текст] / А. И. Галушкин. - М. : Горячая линия - Телеком, 2012. - 496 с.
3. Яхъяева, Г. Э. Основы теории нейронных сетей [Электронный ресурс] / Г.Э. Яхъяева. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 200 с.– Режим доступа: biblioclub.ru
4. Барский, А. Б. Логические нейронные сети [Электронный ресурс] : учебное пособие / А. Б. Барский. - М. : Интернет-Университет Информационных Технологий, 2007. - 352 с. – Режим доступа: biblioclub.ru
5. Вейвлеты в нейродинамике и нейрофизиологии [Текст] : монография / А. А. Короновский, В. А. Макаров, А. Н. Павлов и др. - М. : Физматлит, 2013. – 272 с.
6. Лубенцова, Е. В. Системы управления с динамическим выбором структуры, нечеткой логикой и нейросетевыми моделями [Электронный ресурс] : монография / Е. В. Лубенцова ; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь : СКФУ, 2014. - 248 с. – Режим доступа: biblioclub.ru