

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению практических работ

по дисциплине

«ПРИМЕНЕНИЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ И
ГЛУБОКОГО ОБУЧЕНИЯ В СТЕГАНОАНАЛИЗЕ»

для студентов направления подготовки 02.04.01 Математика и
компьютерные науки

профиль «Искусственный интеллект в кибербезопасности»

Оглавление

Введение	4
Практическое занятие 1 Основы работы с программной средой компьютерной математики	5
Практическое занятие 2 Пакет моделирования нейронных сетей neural network tollbox	28
Практическое занятие 3 Пакет моделирования нейронных сетей Neural Fitting	39
Практическое занятие 4 Изучение свойств линейного нейрона и линейной нейронной сети	50
Практическое занятие 5 Изучение многослойного нелинейного персептрона и алгоритма обратного распространения ошибки	71
Практическое занятие 6 Применение нейронных сетей для распознавания образов	96
Практическое занятие 7 Разработка нейросетевого алгоритма классификации цветов ириса	101
Практическое занятие 8 Распознавание букв	109
Практическое занятие 9 Использование нейронной сети для приближения функций	119
Практическое занятие 10 Разработка нейросетевого алгоритма аппроксимации функции	128
Практическое занятие 11 Прогнозирование энергопотребления здания с использованием нейронной сети	134
Практическое занятие 12 Разработка нейросетевого алгоритма управления химической реакцией	137
Практическое занятие 13 Разработка нейросетевого алгоритма оценки	

стоимости дома	140
Практическое занятие 14 Изучение радиальных базисных, сетей регрессии, вероятностных нейронных сетей	143
Практическое занятие 15 Исследование сети Кохонена и алгоритма обучения без учителя	160
Практическое занятие 16 Решение оптимизационных задач с использованием сети Хопфилда	180
Практическое занятие 17 Нейросетевые системы управления	185
Практическое занятие 18 Применение нейронных сетей для предсказания временного процесса	206
Учебно-методическое и информационное обеспечение дисциплины	210

Введение

1. Цель и задачи освоения дисциплины (модуля)

Цель данного курса является формирование и развитие набора общепрофессиональных и профессиональных компетенций по направлению подготовки 02.04.01 Математика и компьютерные науки профиль «Искусственный интеллект в кибербезопасности».

Основные задачами дисциплины:

- формирование знаний по основам теории нейронных сетей (НС).
- обучение методам и алгоритмам представления и обработки данных в нейронных сетях.
- привитие практических навыков по использованию инструментальных средств компьютерного моделирования создания, обучения и применения НС.

Перечень осваиваемых компетенций:

Код	Формулировка
ОПК-1	Способен оценивать роль информации, информационных технологий и информационной безопасности в современном обществе, их значение для обеспечения объективных потребностей личности, общества и государства
ПК-8	Способен создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники

Перечень планируемых результатов обучения по дисциплине, соотнесённых с планируемыми результатами освоения образовательной программы высшего образования

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине, характеризующие этапы формирования компетенций, индикаторов
ОПК-1 Способен оценивать роль информации, информационных	ИД-1 понимает роль информационной безопасности в системе национальной безопасности РФ и формы	ориентируется в основных понятиях, связанных с обеспечением информационно

технологий и информационной безопасности в современном обществе, их значение для обеспечения объективных потребностей личности, общества и государства	информационной борьбы в современном мире	психологической безопасности личности, общества и государства, понятиях информационного противоборства, информационной войны и форм их проявлений в современном мире
ОПК-1 Способен оценивать роль информации, информационных технологий и информационной безопасности в современном обществе, их значение для обеспечения объективных потребностей личности, общества и государства	ИД-2 выявляет и классифицирует угрозы информационной безопасности для современных автоматизированных систем	анализирует источники и даёт классификацию и оценку угрозам информационной безопасности
ПК-8 Способен создавать и исследовать новые математические модели ИД-1ПК-8 С помощью современных информационных технологий, исследует вновь создаваемые Использует знания о математические модели основных нейросетевых структур, возможные 2 в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники	ИД-1 С помощью современных информационных технологий, исследует вновь создаваемые математические модели в естественных науках промышленности	знает как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники
ПК-8 Способен создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий,	ИД-2 Проводит Параллельные компьютерные и сетевые технологии в рамках профессиональной деятельности	умеет создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и

программирования и компьютерной техники		компьютерной техники
ПК-8 Способен создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники	ИД-3 Использует современные языки программирования и пакеты прикладных программ для реализации конкретных алгоритмов и математических моделей	владеет как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники

2. Место дисциплины (модуля) в структуре образовательной программы

Дисциплина «Нейросетевые технологии обработки информации» входит в вариативную часть дисциплин первого блока. Ее освоение осуществляется в 7 семестре.

3. Связь с предшествующими дисциплинами (модулями)

- Основы компьютерных наук: базы данных
- Основы компьютерных наук: операционные системы
- Теория чисел
- Нечеткая логика и ее приложения

4. Связь с последующими дисциплинами (модулями)

- Практика по получению профессиональных умений и опыта профессиональной деятельности
- Преддипломная практика
- Подготовка дипломной работы
- Государственный экзамен

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 1

Основы работы с программной средой компьютерной математики

Цель занятия:

Освоить основные приемы работы с программной средой компьютерной математики MATLAB.

Порядок проведения занятия

1. Запуск программы.
2. Интерфейс пользователя системы MATLAB.
3. Простейшие математические вычисления.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

1. Запуск программы

Для запуска программы из меню ПРОГРАММЫ необходимо выбрать MATLAB (рисунок 1).

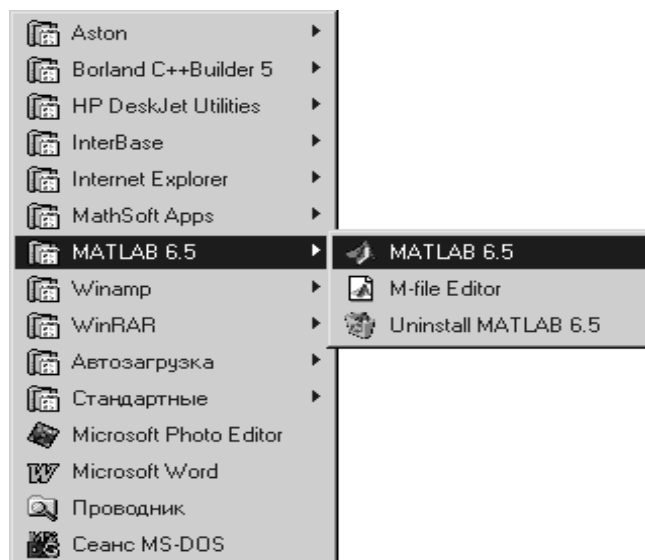


Рисунок 1 – Подготовка к запуску программы MATLAB

После инициализации блоков программы появляется основное окно программы, показанное на рисунке 2. Обычно это окно не раскрыто полностью, а занимает лишь часть экрана.

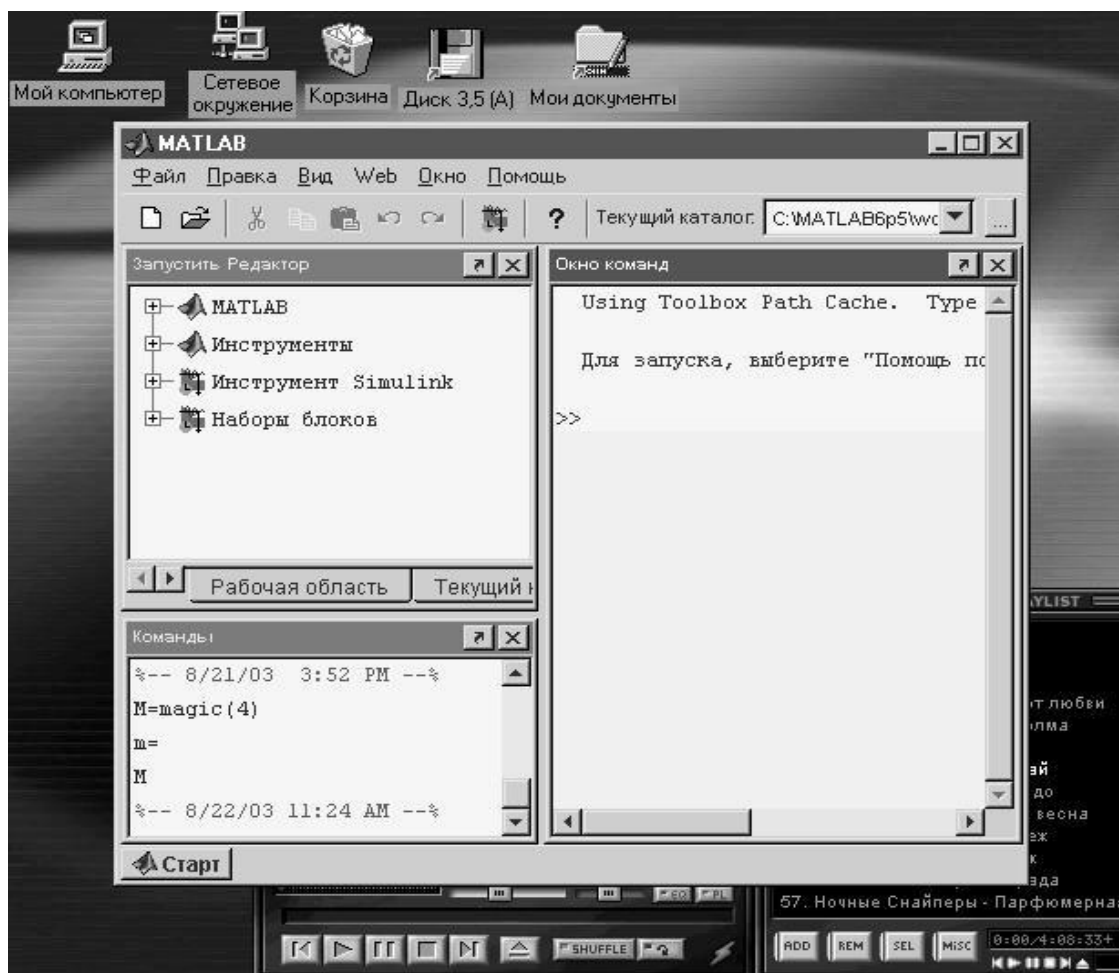
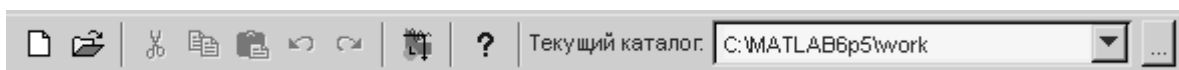


Рисунок 2 – Окно программы MATLAB после инициализации

2. Интерфейс пользователя системы MATLAB

Панель инструментов (рисунок 3) дает наиболее простой и удобный (особенно для начинающих пользователей) способ работы с системой MATLAB. При этом основные команды вводятся указанием курсором мыши на нужную кнопку с нажатием левой клавиши мыши. Кнопки имеют изображение, явно подсказывающее их назначение.



Рис

унок 3 – Меню панели инструментов

Прежде всего перечислим назначение всех кнопок панели инструментов:

- New M-file (Новый m-файл) – выводит пустое окно редактора m-файлов;
- Open file (Открыть файл) – открывает окно для загрузки m-файла;
- Cut (Вырезать) – вырезает выделенный фрагмент и помещает его в буфер;
- Copy (Копировать) – копирует выделенный фрагмент в буфер;
- Paste (Вставить) – переносит фрагмент из буфера в текущую строку ввода;

- Undo (Отменить) – отменяет предшествующую операцию;

- Redo (Повторить) – восстанавливает последнюю отмененную операцию;
- Simulink – открывает окно браузера библиотек Simulink;
- Help (Помощь) – открывает окно справки.

Набор кнопок панели инструментов обеспечивает выполнение наиболее часто необходимых команд и вполне достаточен для повседневной работы с системой. О назначении кнопок говорят и всплывающие подсказки, появляющиеся, когда курсор мыши устанавливается на соответствующую кнопку. Они имеют вид желтого прямоугольника с текстом короткой справки. Пример такой подсказки показан на рисунке 3, у кнопки Help панели инструментов. Любопытно отметить, что доступ к браузерам рабочей области и файловой системы из панели инструментов убран.

Вызов окна открытия нового файла

Кнопка New M-file открывает окно редактора/отладчика m-файлов. Это окно показано на рисунке 4.

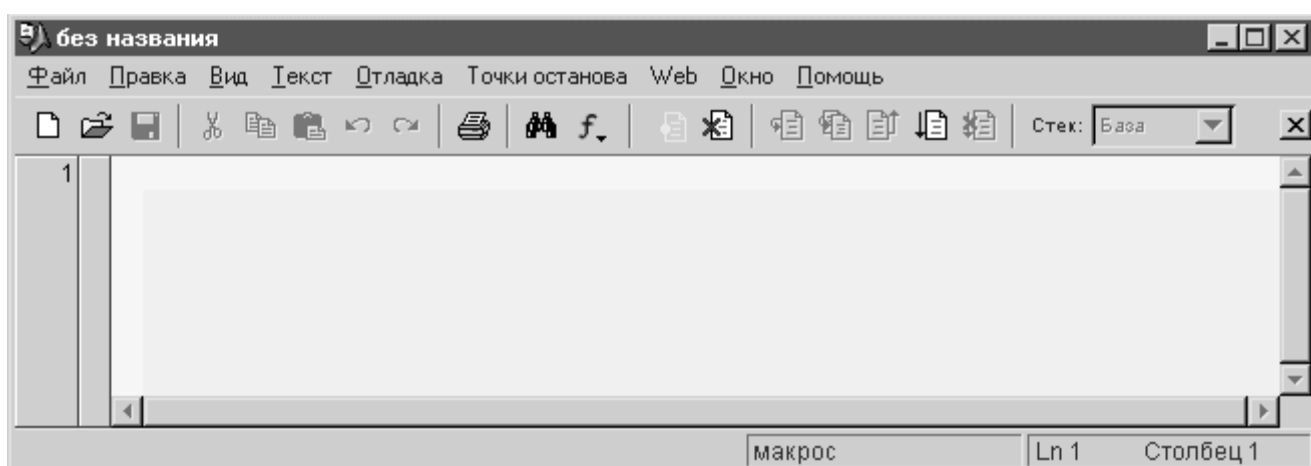


Рисунок 4 – Окно редактора/отладчика m-файлов

По умолчанию файлу дается имя без названия, которое впоследствии (при записи файла) можно изменить на другое, отражающее тему задачи. Это имя отображается в титульной строке окна редактирования m-файла, которое размещается в окне редактора/отладчика и видно на рисунке 4. В редакторе/отладчике можно редактировать несколько m-файлов, и каждый из них будет находиться в своем окне редактирования, хотя активным может быть только одно окно, расположенное поверх других окон.

Вызов окна загрузки имеющегося файла

Кнопка Open file (Открыть файл) служит для загрузки в редактор/отладчик ранее созданных m-файлов, например входящих в пакет расширения (Toolbox) системы или разработанных пользователем. Она открывает окно, которое является типичным элементом интерфейса Windows-приложений и показано на рисунке 5 внутри окна редактора/отладчика m-файлов.

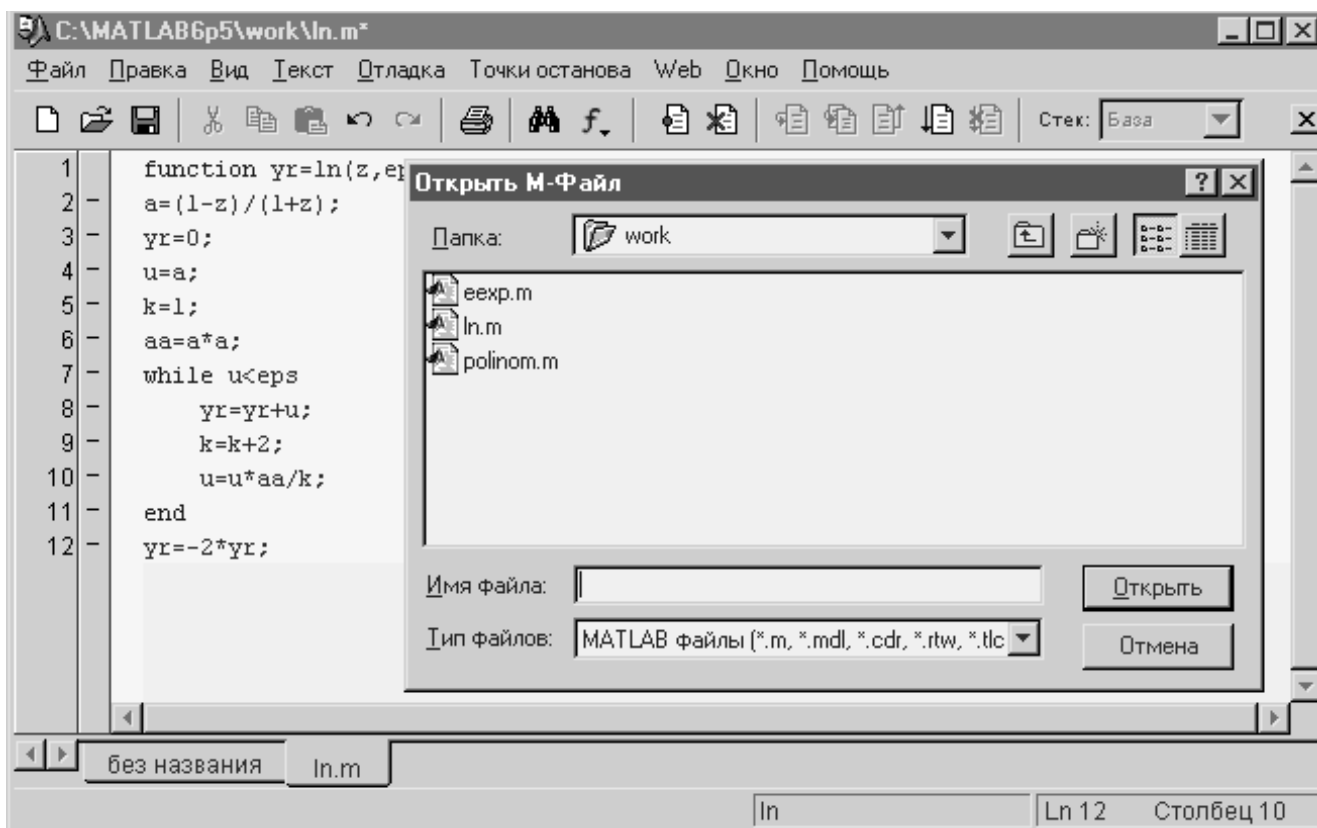


Рисунок 5 – Окно загрузки файла в окне редактора/отладчика

В окне загрузки файлов с помощью раскрывающегося списка Папка и вертикальной полосы прокрутки можно «пройтись» по всем дискам, папкам и файлам. Чтобы выбрать нужный файл для загрузки, его требуется выделить мышью. Выбор завершается двойным щелчком на имени файла, нажатием клавиши Enter или щелчком на кнопке Открыть. Любое из этих действий приводит к загрузке документа в текущее окно системы. Кнопка Отмена или клавиша Esc позволяет отказаться от загрузки.

Для вызова одного из ранее использовавшихся документов достаточно щелкнуть мышью на его имени в списке, находящемся над командой Выход в меню Файл. После загрузки файла с документом его текст появляется в текущем. Его можно начинать редактировать или запускать на исполнение.

Операции с буфером обмена

Кнопки Вырезать, Копировать и Вставить реализуют наиболее характерные команды работы с буфером обмена (Clipboard). Первые две операции относятся к выделенным фрагментам сессии или текста m-файлов (если они выполняются в окне редактора/отладчика). Для выделения объектов можно использовать мышь, перемещая курсор по тексту при нажатой левой кнопке, или клавиши со стрелками в комбинации с клавишей Shift.

На рисунке 6 показан пример выделения содержимого матрицы M в окне документа MATLAB. Эта матрица формируется функцией magic(n) и называется магической, поскольку сумма элементов любого столбца, любой строки и даже любой диагонали равна одному и тому же числу – 34 для

матрицы при $n=4$.

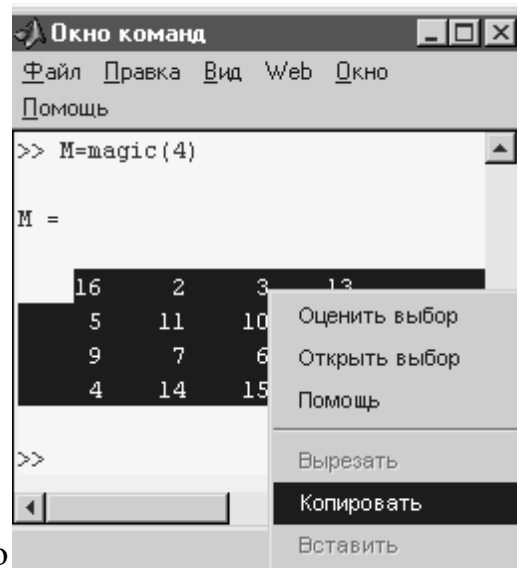


Рисунок 6 – Окно

жимым матрицы М

Команда Cut (Вырезать) осуществляет вырезание выделенного фрагмента и размещение его в буфере. При этом вырезанный фрагмент удаляется из текста документа. Команда Копировать просто копирует выделенный фрагмент в буфер, сохраняя его в тексте. Команда Вставить вызывает объект из буфера (сохраняя объект в буфере) и помещает копию объекта на место в документе, указанное текстовым курсором. Эти операции реализуются как соответствующими кнопками, так и командами меню Редактировать.

В MATLAB можно использовать контекстное меню, появляющееся при нажатии правой кнопки мыши. Например, установив курсор мыши на выделенный фрагмент матрицы М и нажав правую кнопку, можно увидеть меню, показанное на рисунке 6. В нем, кстати, дублируется позиция с командой Сору (Копировать). Есть и ряд других доступных в данный момент команд. Обратите внимание, что в момент подготовки магической матрицы М ее имя появилось в окне браузера рабочей области — в правой части экрана. При этом матрица представляется изображением таблицы.

Содержимое буфера можно перенести в строку ввода, в окно редактора-отладчика m-файлов или даже в другое приложение. Допустим, мы хотим создать матрицу М1 с содержимым, которое размещено в буфере. Для этого достаточно набрать М1=[и нажав правую клавишу мыши, выбрав из появляющегося меню команду Вставить (рисунок 7).

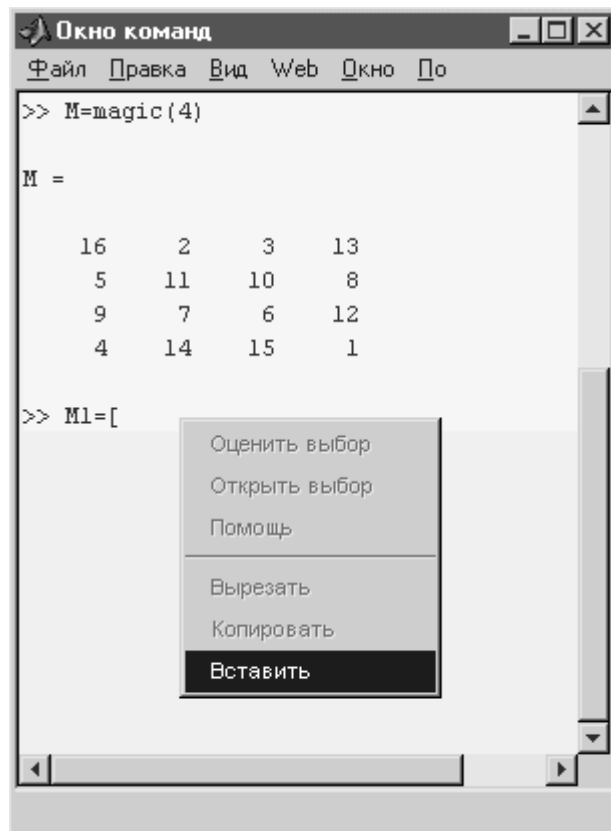


Рисунок 7

рицы из буфера

Исполнив команду Paste, можно увидеть, что данные хранящейся в буфере матрицы появятся после открывающей квадратной скобки. Для создания матрицы M1 остается ввести закрывающую квадратную скобку] и нажать клавишу Enter. На рисунок 8 показано, как создается матрица M1, по содержимому аналогичная матрице M.

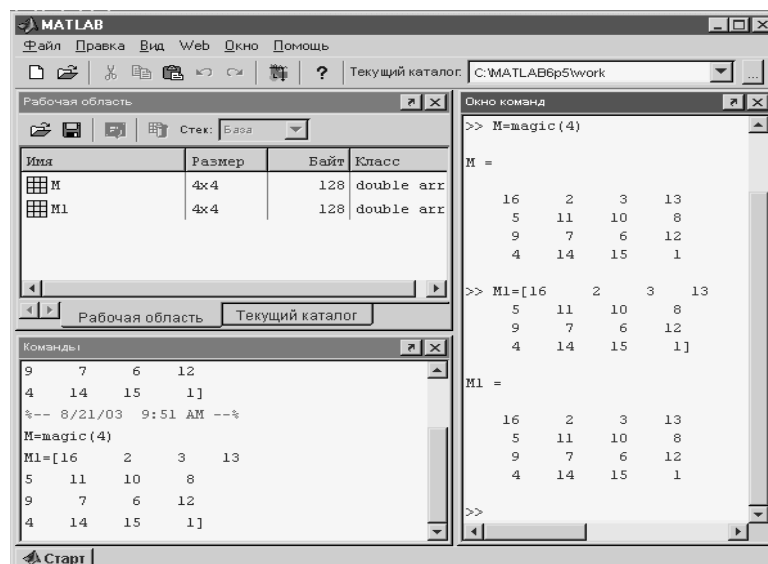


Рисунок 8 – Пример создания матрицы M1 с содержимым, взятым из буфера

Разумеется, этот пример является чисто учебным. Не обращаясь к помощи буфера, можно было бы просто записать M1=M. Все примеры в

тексте методической разработки получены переносом выделенных фрагментов соответствующей сессии в окно текстового редактора Microsoft Word. Возможен и обратный перенос – записанных в документах редактора Microsoft Word примеров в командную строку MATLAB для исполнения примеров.

Отмена результата предшествующей операции

Часто, выполнив какую-то операцию, мы отмечаем, что она оказалась ошибочной. При работе в MATLAB такой ситуации пугаться не стоит, — нажатие кнопки Undo (Отменить) панели инструментов приведет к отмене последнего действия, выполненного в текущей строке. Операции в предыдущих строках документа этой командой не отменяются. Если оказалось, что вы зря произвели отмену последней операции, то ее легко восстановить, введя с панели инструментов операцию Redo (Восстановить).

Вызов справки MATLAB

Последняя кнопка панели инструментов Help (Помощь) открывает окно с перечнем разделов справочной системы. Это окно было показано на рисунке 9.

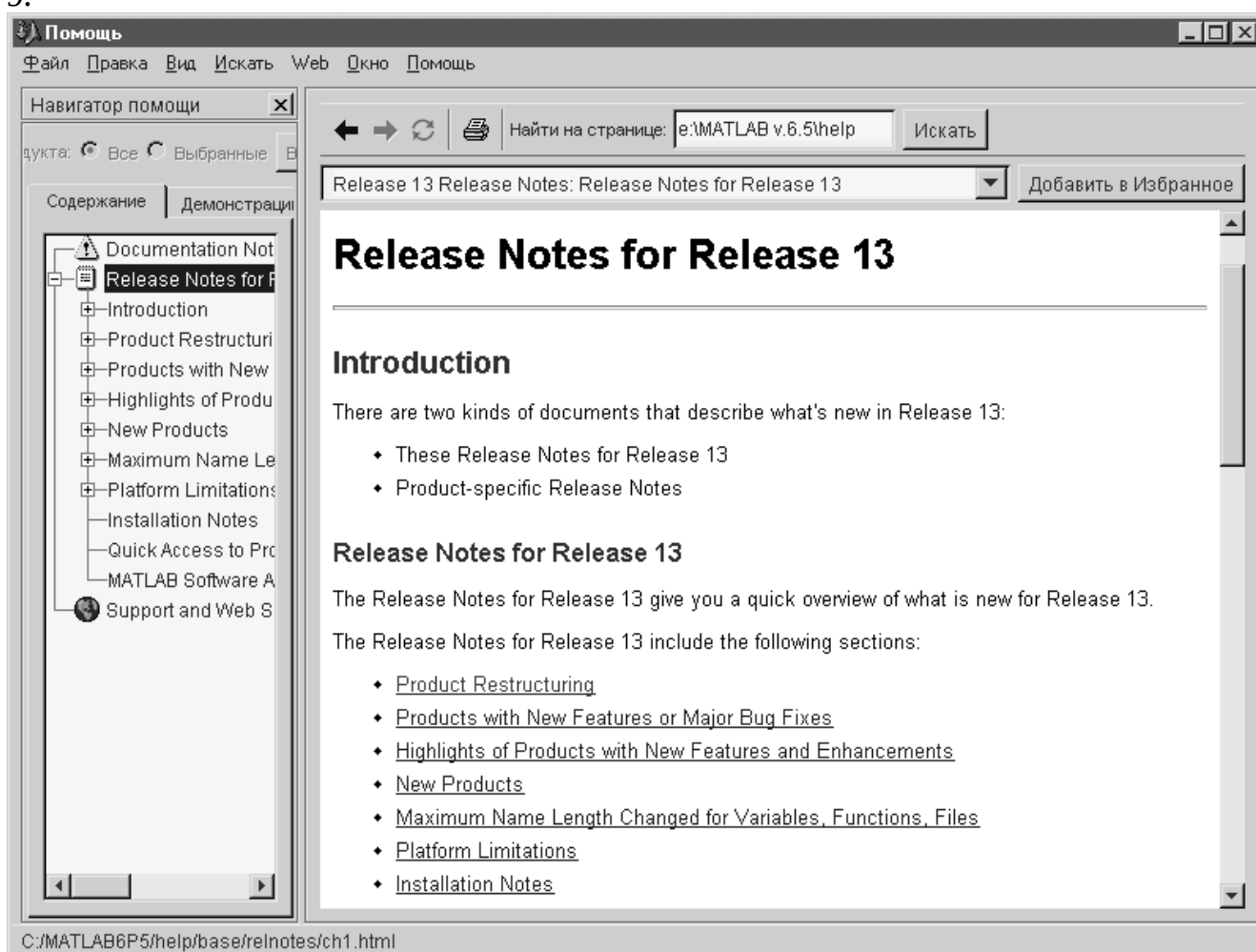


Рисунок 9 – Помощь по MATLAB

Средства контроля рабочей области и файловой системы

Браузер рабочей области

Векторы и матрицы могут занимать большой объем памяти. Конечно, речь не идет о векторах или матрицах, содержащих несколько элементов или даже несколько десятков элементов. Хотя и в этом случае оценка их размеров полезна при разработке алгоритмов матричных вычислений и оценке их эффективности в части использования памяти.

Как отмечалось, в левой части окна системы MATLAB имеется окно специального браузера рабочей области — Workspace Browser. Он служит для просмотра ресурсов рабочей области памяти. Браузер дает наглядную визуализацию содержимого рабочей области. Окно браузера рабочей области выполняет и другие важные функции — позволяет просматривать существующие в памяти объекты, редактировать их содержимое и удалять объекты из памяти. Для вывода содержимого объекта достаточно выделить его имя с помощью мыши и щелкнуть на кнопке

Open (Открыть). Объект можно открыть и двойным щелчком на его имени в списке. Откроется окно редактирования массива Array Editor, показанное на рисунке 10, применительно к матрице M.

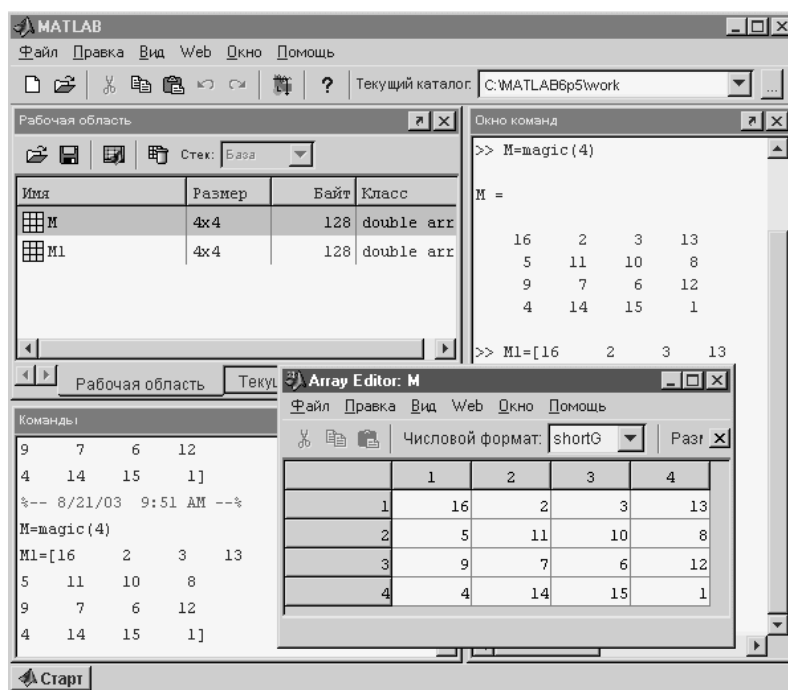


Рисунок 10 – Пример просмотра содержимого матрицы

Окно редактирования матрицы дает удобный доступ для редактирования любого элемента матрицы по правилам, принятым при работе с электронными таблицами. Основное из них — быстрый доступ к любому элементу матрицы. Можно также менять тип значений элементов, выбирая его из списка, предоставляемого меню Numeric format (Формат чисел). В окне также выводятся данные о числе строк и столбцов матрицы.

Браузер файловой структуры

Для просмотра файловой структуры MATLAB служит специальный браузер файловой системы (Path Browser), который запускается при обычной

загрузке системы. Если был установлен упрощенный интерфейс, то для запуска браузера файловой системы используется окно Current Directory (Текущая папка). На рисунке 11 в левой части показано окно этого браузера.

Нетрудно заметить, что браузер файловой системы построен по типу общеизвестного Проводника (Explorer) операционных систем Windows. Окно дает возможность просмотра файловой системы ПК и выбора любого файла. Для примера на рисунке 11 показан выбор m-файла функции натурального логарифма.

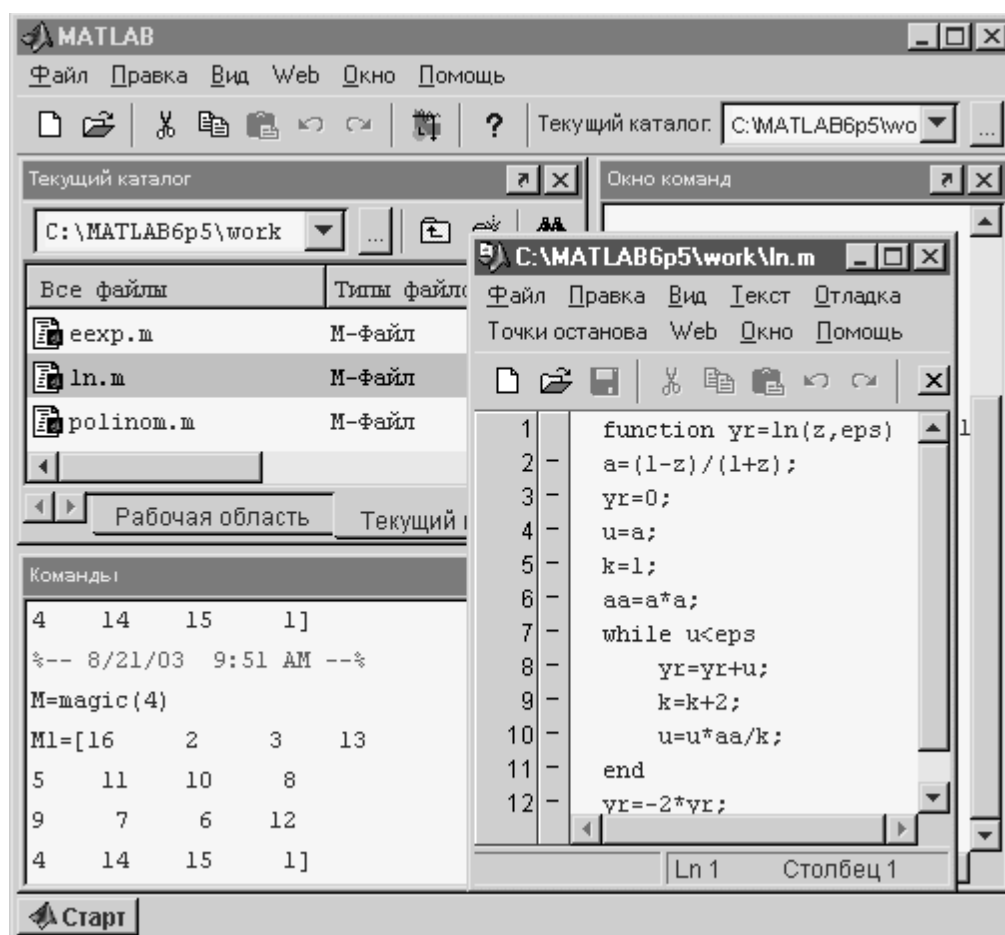


Рисунок 11 – Пример окна браузера

Исполнив команду Open (Открыть) из контекстного меню правой клавиши мыши или дважды щелкнув по выделенной строке с именем файла, можно ввести этот файл в окно редактора/отладчика m-файлов. При этом редактор запустится автоматически и его окно с готовым для редактирования выбранным файлом появится на экране – оно показано в центре экрана на рисунке 9.

Таким образом, браузер просмотра файловой структуры позволяет детально ознакомиться с файловой системой MATLAB и вывести любой из m-файлов или файлов демонстрационных примеров для просмотра, редактирования и модификации.

Работа с меню

Команды, операции и параметры

Открытая позиция строки меню содержит различные операции и команды. Выделенная команда или операция выполняется при нажатии клавиши Enter (Ввод). Выполнение команды можно также осуществить щелчком мыши или нажатием на клавиатуре клавиши, соответствующей выделенному символу в названии команды,

Между командами и операциями нет особых отличий, и в литературе по информатике их часто путают. Мы будем считать командой действие, которое выполняется немедленно. А операцией — действие, которое требует определенной подготовки, например открытие окна для установки определенных параметров.

Параметр (option)— это значение определенной величины, действующее во время текущей сессии. Параметрами обычно являются указания на применяемые наборы шрифтов, размеры окна, цвет фона и т. д.

Меню системы

Перейдем к описанию основного меню системы MATLAB. Это меню выглядит довольно скромно и содержит всего шесть пунктов:

- File — работа с файлами;
- Edit — редактирование сессии;
- View — вывод и скрытие панели инструментов;
- Web — доступ к Интернет-ресурсам;
- Windows — установка Windows-свойств окна;
- Help — доступ к справочным подсистемам.

Подменю File

Подменю File содержит ряд операций и команд для работы с файлами. Оно показано на рисунке 12. Число операций и команд значительно сокращено по сравнению с тем же меню у предшествующей версии системы MATLAB.

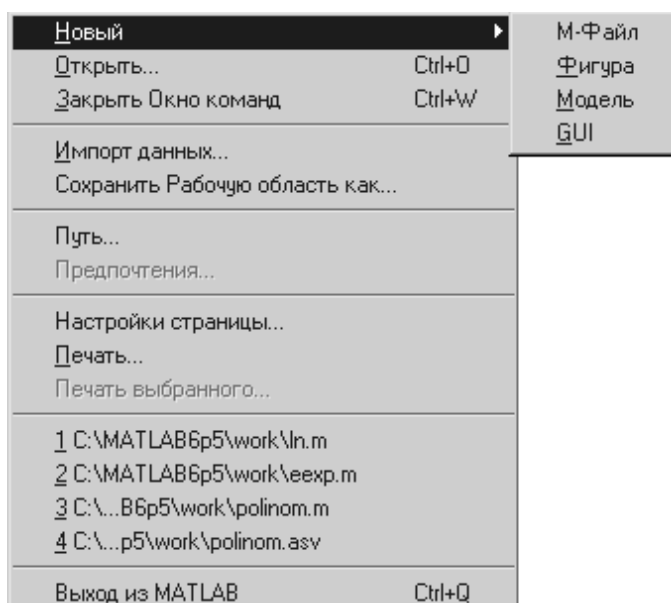


Рисунок 12 – Меню Файл

Теперь меню File содержит следующие операции:

- New - открывает подменю с позициями:
- M-file — открытие окна редактора/отладчика m-файлов;
- Figure — открытие пустого окна графики;
- Model — открытие пустого окна для создания Simulink-модели;
- GUI — открытие окна разработки элементов графического интерфейса пользователя.
- Open — открывает окно загрузки файла.
- Close Command Windows — закрывает окно командного режима работы (оно при этом исчезает с экрана).
- Import data — открывает окно импорта файлов данных.
- Save Workspace As... — открывает окно записи рабочей области в виде файла с заданным именем.
- Set Path — открывает окно установки путей доступа файловой системы.
- Preferences... — открывает окно настройки элементов интерфейса.
- Print... — открывает окно печати всего текущего документа.
- Print Selection... — открывает окно печати выделенной части документа.
- Exit — завершает работу с системой.

Открытие окон для подготовки новых файлов

Команда New, как отмечено, открывает окна для подготовки новых файлов. Для трех типов файлов (m-файлы, графические файлы и файлы Simulink-моделей) окна их редакторов уже описывались. Новой является позиция GUI подменю File. Она открывает окно редактора элементов пользовательского интерфейса, показанное на рисунке 13.

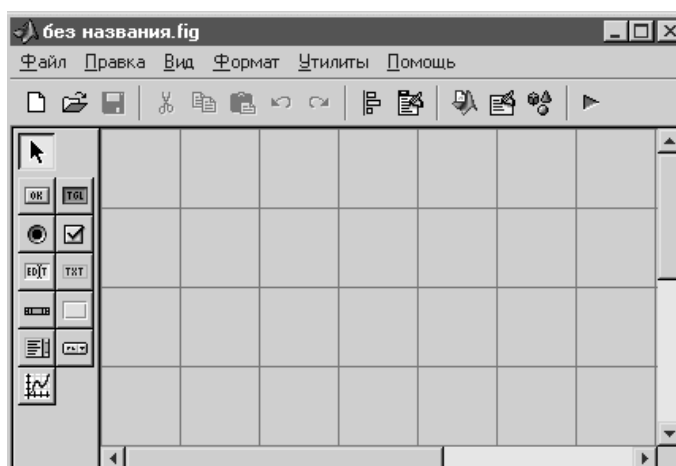


Рисунок 13 – Окно редактора элементов GUI

К подробному описанию этого окна мы еще вернемся. А пока отметим, что работа с редактором довольно очевидна.

Загрузка и сохранение файлов

Команды Open... (Открыть) и Import data... (Импортировать данные)

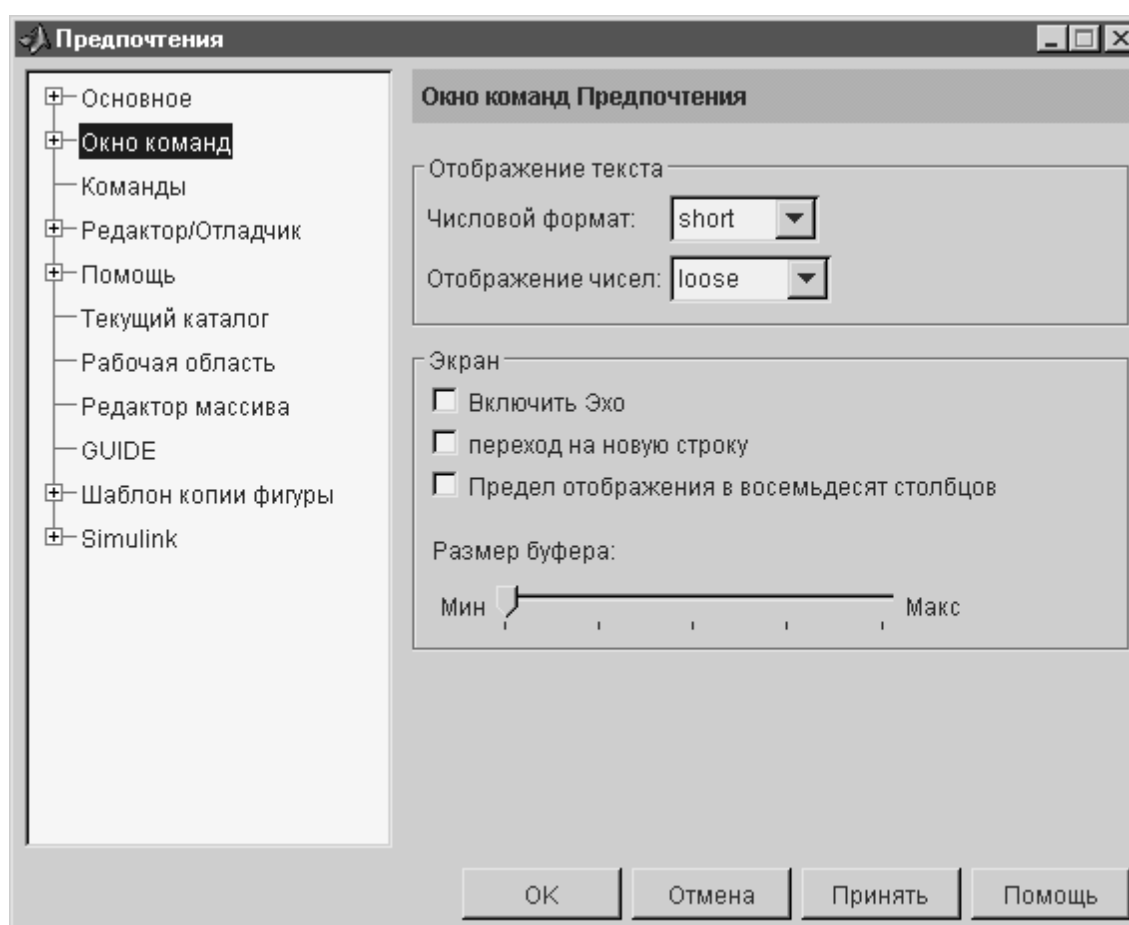
выводят стандартные окна для загрузки m-файлов и файлов данных. Команда Save Workspace As... (Сохраните рабочую область как...) открывает стандартное окно записи файлов с расширением .mat. Они хранят определения переменных, массивов, функций пользователя и иных объектов, созданных в ходе текущей сессии работы. Эти команды в силу общеизвестности в более подробных комментариях не нуждаются.

Установка путей доступа файловой системы

Команда Set Path... (Установить путь) открывает окно редактора путей доступа файловой системы. Окно дает список папок с файлами системы MATLAB. Имеется возможность! переноса папок вверх или вниз по списку, уничтожения их и переименования. По умолчанию задается правильная установка путей доступа, так что данными возможностями стоит пользоваться только в особых обстоятельствах, например при! случайном переносе папок в другое место или при их переименовании.

Настройка элементов интерфейса

Окно настройки элементов интерфейса представлено на рисунке 14.



Интерфейс MATLAB стал настолько удобным и даже изысканным, что мало вероятно, что кому-либо захочется менять его в мере, большей, чем это предусмотрено командами подменю View. Однако такая возможность есть — команда Preferences... (Предпочтения) выводит окно детальной настройки

элементов интерфейса. В левой части этого окна имеется древообразный список элементов интерфейса системы, а в правой части—поле задания параметров для выбранного типа элементов. Поскольку изменение параметров производится обычно очень редко, мы не будем детально рассматривать это окно. Заинтересованный читатель наверняка разберется с нужными ему параметрами самостоятельно.

Меню Edit — средства редактирования документов

Меню Edit (рисунок 15) содержит операции и команды редактирования, типичные для большинства приложений Windows:

- Undo (Отменить) — отмена результата предшествующей операции;
- Redo (Повторить) — отмена действия последней операции Undo;
- Cut (Вырезать) — вырезание выделенного фрагмента и перенос его в буфер;
- Copy (Копировать) — копирование выделенного фрагмента в буфер;
- Paste (Вставить) — вставка фрагмента из буфера в текущую позицию курсора;
- Clear (Очистить) — операция очистки выделенной области;
- Select All (Выделить) — выделение всей сессии;
- Delete (Стереть) — уничтожение выделенного объекта;
- Clear Command Windows (Очистить командное окно) — очистка текста сессии (с сохранением созданных объектов);
- Clear Command History (Очистить окно истории команд) — очистка окна истории;
- Clear Workspace — очистка окна браузера рабочей области.

Правка	Вид	Web	Окно	Помощь
Отмена				Ctrl+Z
Повтор				
Вырезать				Ctrl+X
Копировать				Ctrl+C
Вставить				Ctrl+V
Специальная вставка...				
Выбрать все				
Удалить				Delete
Найти...				
Очистить Окно команд				
Очистить Историю команд				
Очистить Рабочую область				

Рисунок 15 – Меню Edit

Назначение указанных команд и операций уже обсуждалось. Отметим лишь, что команда Clear Command Window очищает окно командного режима

работы и помещает курсор в верхний левый угол окна. Однако все определения, сделанные в течение стертых таким образом сессий,

сохраняются в памяти компьютера. Для очистки экрана используется также команда `clc`, вводимая в командном режиме.

Меню View и Window

В MATLAB что уже описывалось, набор команд меню View существенно расширен, и теперь с помощью этого меню можно существенно менять вид пользовательского интерфейса.

Меню Window активно только в случае, если в систему загружены файлы. При этом оно имеет единственную команду Close All (закрыть все окна) и открывающийся список всех загруженных файлов. Он позволяет выбрать окно указанного пользователем файла и сделать его открытым.

Основы редактирования и отладки m-файлов

Интерфейс редактора/отладчика m-файлов

Для подготовки, редактирования и отладки m-файлов служит специальный многооконный редактор. Он выполнен как типичное приложение Windows. Редактор можно вызвать командой `edit` из командной строки или командой `New→M-file` меню File. После этого в окне редактора можно создавать свой файл, пользоваться средствами его отладки и запуска. Перед запуском файла его необходимо записать на диск, используя команду `File→Save as` в меню редактора.

На рисунке 16 показано окно редактора/отладчика с текстом простого файла в окне редактирования и отладки.

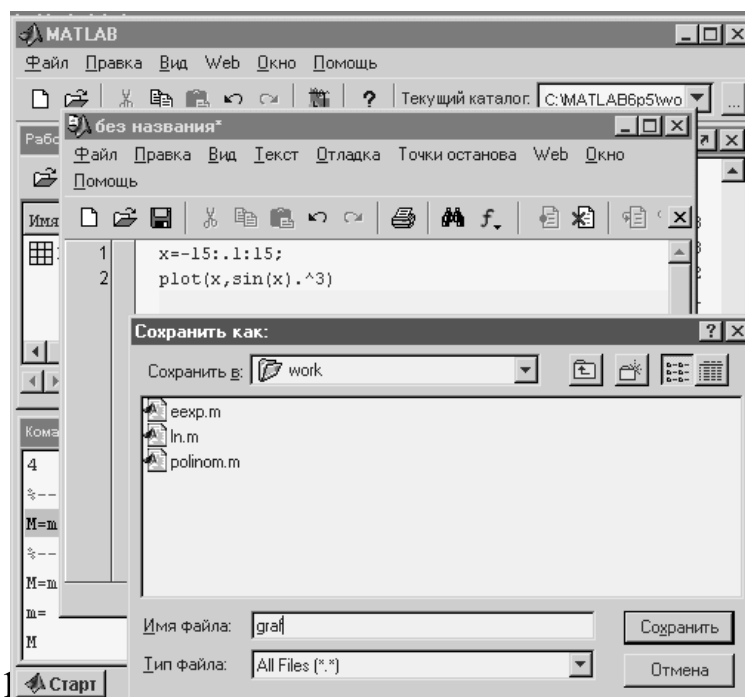


Рисунок 16 Окно редактора/отладчика с текстом простого файла на диск

Подготовленный текст файла (это простейшая и наша первая программа на языке программирования MATLAB) надо записать на диск. Для

этого используется команда Save As, окно которой видно на рис. 16 внутри окна системы редактора/ отладчика. Работа с окном команды Save As

уже описывалась.

После записи файла на диск можно заметить, что команда Run в меню Tools (Инструменты) редактора становится активной (до записи файла на диск она пассивна) и позволяет произвести запуск файла. Запустив команду Run, можно наблюдать исполнение m-файла.

Для удобства работы с редактором/отладчиком строки программы в нем нумеруются в последовательном порядке. Редактор является многооконным. Окно каждой программы оформляется как вкладка. Цветовые выделения и синтаксический контроль.

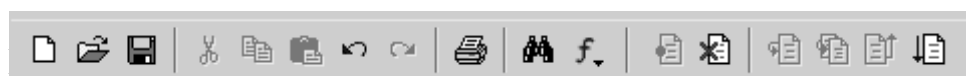
Редактор/отладчик m-файлов выполняет синтаксический контроль программного кода по мере ввода текста. При этом используются следующие цветовые выделения:

- ключевые слова языка программирования — синий цвет;
- операторы, константы и переменные — черный цвет;
- комментарии после знака % — зеленый цвет;
- символьные переменные (в апострофах) — зеленый цвет;
- синтаксические ошибки — красный цвет.

Благодаря цветовым выделениям вероятность синтаксических ошибок снижается,

Панель инструментов редактора и отладчика

Редактор имеет свое меню и свою инструментальную панель. Внешний вид инструментальной панели показан на рисунке 17. По стилю данная панель похожа на панель инструментов окна командного режима работы, но имеет несколько иной набор кнопок.



Назначение кнопок панели инструментов редактора/отладчика следующее:

- New — создание нового t-файла;
- Open — вывод окна загрузки файла;
- Save — запись файла на диск;
- Print — печать содержимого текущего окна редактора;
- Cut — вырезание выделенного фрагмента и перенос его в буфер;
- Copy — копирование выделенного объекта в буфер;
- Paste — размещение фрагмента из буфера в позиции текстового курсора;
- Undo — отмена предшествующей операции;
- Redo — повтор отмененной операции;

- Find text — нахождение указанного текста;
- Show function — показ функции;
- Set/Clear Breakpoint — установка/сброс точки прерывания;

- Clear All Breakpoints — сброс всех точек прерывания;
- Step — выполнение шага трассировки;
- Step In — пошаговая трассировка с заходом в вызываемые т-файлы;
- Step Out — пошаговая трассировка без захода в вызываемые т-файлы;
- Save and Run — запись и сохранение;
- Exit Debug Mode — выход из режима отладки.

С назначением ряда из этих кнопок вы уже знакомы, поскольку оно аналогично описанному ранее для основного окна MATLAB. А вот о назначении других кнопок надо поговорить.

Работа с точками прерывания

Основным приемом отладки m-файлов является установка в их тексте точек прерывания (Breakpoints). Они устанавливаются (и сбрасываются) с помощью кнопок Set/Clear Breakpoint. Сброс всех точек прерывания обеспечивается кнопкой Clear All Breakpoints.

Вы можете задать выполнение программы без остановки при заходе, но с остановкой при выходе (кнопка Step Out), и с остановкой при заходе в вызываемые m-файлы (кнопка Step In). Кнопка Exit Debug Mode (Выход из режима отладки) прекращает операции отладки.

3. Простейшие математические вычисления

MATLAB — это математический пакет прикладных программ, основанный на использовании матриц. Пакет содержит обширную библиотеку программ по численным методам, использует двух- и трехмерную графику, а также форматы языков высокого уровня. Подходящий формат позволяет с помощью MATLAB быстро выполнять и модифицировать программы.

Арифметические операции

+	Сложение
—	Вычитание
*	Умножение
/	Деление
^	Возведение в степень
pi, e, i	Постоянные
Пример	<code>>>(2+3*pi)/2</code> <code>ans = 5.7124</code>

Встроенные функции

Ниже кратко перечислены некоторые функции, имеющиеся в MATLAB. Следующий пример иллюстрирует, как можно комбинировать функции и арифметические операции. Описание других функций можно найти, используя легко доступную справку в диалоговом режиме: `abs(#)` `cos(#)` `exp(#)` `log(#)` `log10(#)` `cosh(#)` `sin(#)` `tan(#)` `sqrt(#)` `floor(#)` `acos(#)` `tanh(#)`.

Пример `>>3*cos(sqrt(4.7))`
 `ans = -1.6869`

По умолчанию выводится приблизительно пять десятичных значащих

цифр. Команда `format long` позволяет вывести приблизительно 15 десятичных значащих цифр.

Пример `>>format long`
 `>>3*cos(sqrt(4.7))`
 `ans = -1.68686892236893`

Операторы присваивания

Для присваивания выражениям различных имен используется знак равенства.

Пример `>>a=3-floor(exp(2.9))`
 `a = -15`

Точка с запятой, расположенная в конце выражения, предполагает вывод символа на экран компьютера (выход).

Пример `>>b=sin(a);`
 `>>2*b^2`
 `ans=0.8457`

Определение функций

В MATLAB пользователь может определить функцию, построив М-файл (в конце файла ставится `.m`) в М-файле Editor/Debugger. Однажды определенная пользователем функция вызывается таким же способом, как и встроенные функции.

Пример Определим функцию $\text{fun}(x)=1+x-x^2/4$ в m-файле `fun.m`.

В Editor/Debugger она записывается следующим образом.

```
function y=fun(x)
y=1+x-x.^2/4;
```

Для обозначения переменных можно употреблять различные буквы и для названия функций — различные имена, но следует использовать один и тот же формат. Функцию, однажды записанную как М-файл под именем `fun.m`, можно вызывать в MATLAB Command Window так же, как любую другую функцию.

Пример `>>cos(fun(3))`
 `ans= -0.1782`

Полезным и эффективным способом вычисления функций является использование команды `feval`. Она требует, чтобы функция вызывалась, как строка.

Пример `>>feval('fun',4)`
 `ans=1`

Матрицы

Все переменные в MATLAB интерпретируются как матрицы или массивы. Матрицы можно вводить непосредственно:

Пример >>A=[1 2 3; 4 5 6; 7 8 9]

A= 1 2 3

4 5 6

7 8 9

Точка с запятой используется для разделения строк матрицы. Отметим,

что элементы матрицы следует разделять одиночным пробелом. Альтернативный способ ввода матрицы — строка за строкой.

Пример >>A=[1 2 3

4 5 6

7 8 9]

A= 1 2 3

4 5 6

7 8 9

Матрицы можно формировать, используя встроенные функции.

Пример >>Z=zeros(3,5); Создание нулевой матрицы размера 3x5.

>>X=ones(3,5); Создание матрицы размера 3x5, состоящей из единиц.

>>Y=0:0.5:2 Вывод на экран матрицы размера 1x5.

Y=0 0.5000 1.0000 1.5000 2.0000

>>cos(Y) Создание матрицы размера 1x5, каждый элемент которой является cos от Y

ans=1.0000 0.8776 0.5403 0.0707 -0.4161

Элементами матрицы можно манипулировать несколькими способами.

Пример >>A(2,3) Выделение одного элемента матрицы A

ans = 6

>>A(1:2, 2:3) Выделение подматрицы A

ans = 2 3

5 6

>>A([1 3],[1 3]) Другой способ выделения подматрицы

A ans= 1 3

7 9

>>A(2,2)=tan(7.8);

Присвоение нового значения элементу матрицы A

Дополнительные команды для матриц можно найти, используя справку в диалоговом режиме или документацию к пакету прикладных программ.

Операции с матрицами

+ Сложение

– Вычитание

* Умножение

^ Возведение в степень

' Сопряжение и транспонирование

Пример >>B=[1 2; 3 4];
 >>C=B' C равна транспонированной матрице B
 C= 1 3
 2 4
 >>3*(B*C)^3
 ans= 13080 29568
 29568 66840

Операции с массивами

Одним из наиболее полезных свойств пакета MATLAB является количество операций, которые можно выполнять над отдельными элементами матрицы. Такая операция была продемонстрирована выше, когда элементы матрицы размера 1x5 записывались как cos от элементов вводимой матрицы. Операции сложения, вычитания и умножения матрицы на скаляр всегда осуществляются поэлементно, но это не относится к операциям умножения, деления и возведения матрицы в степень. Данные три операции можно производить поэлементно, как и предыдущие, но с указателем операции: `.*`, `./` и `.^`. Важно понять, как и когда именно их можно использовать. Операции над массивами являются решающими для эффективного построения и выполнения пакетом MATLAB программ и графиков.

```
Пример    >>A=[1 2;3 4];
           >>A^2 Произведение матриц AA
           ans=
           7 10 15 22
           >>A.^2 Квадрат каждого элемента A
           ans= 1 4
                9 16
           >>cos(A./2)
           Деление каждого элемента A на 2 и вычисление cos от каждого
           элемента
           ans=
           0.8776 0.5403 0.0707 -0.4161
```

Графики

Пакет MATLAB позволяет строить двух- и трехмерные графики кривых и поверхностей. Дополнительные возможности и описание графиков в пакете MATLAB можно найти, используя справку в диалоговом режиме или документацию к пакету прикладных программ.

Команду `plot` используют для построения графика двухмерных функций. Следующий пример показывает, как создать программу построения графика функций $y=\cos(x)$ и $y=\cos^2(x)$ на интервале $[0; \pi]$.

```
Пример    >>x=0:0.1:pi;
           >>y=cos(x);
           >>z=cos(x).^2;
           >>plot(x, y, x, z, 'o')
```

В первой строке программы задается область с шагом 0,1. В следующих двух строках задаются две функции. Заметим, что первые три строки записываются точкой с запятой. Точка с запятой необходима, чтобы запретить вывод матриц `x`, `y` и `z` на экран. В четвертой строке содержится команда `plot`, которая и строит график. Первые параметры команды `plot`, `x` и `y`, строят график

функции $y=\cos(x)$. Третий и четвертый параметры, x и z , строят график функции $y=\cos^2(x)$. Последний параметр – 'o'; в нем

содержится результат построения графика в каждой точке (x_k, z_k) , где $z = \cos^2(x_k)$.

В третьей строке необходимо использовать оператор `".^"`, так как сначала вычисляется значение `cos` каждого элемента матрицы `x`. Затем каждый элемент матрицы `cos(x)` с помощью команды `.^` возводится в квадрат.

Команда построения графика `fplot` — это полезная альтернатива команде `plot`. Форма ее записи — `fplot('name', [a, b], n)`. Она строит график функции `name.m` по множеству `n` точек на интервале `[a; b]`. По умолчанию число `n` равно 25.

Пример `>>fplot('tanh', [-2, 2])` График $y = \tanh(x)$ на `[-2; 2]`

Команды `plot` и `plot3` соответственно используются для построения параметрических кривых в двух- и трехмерном пространстве. Они особенно полезны для создания зрительных образов решений дифференциальных уравнений в двух и трех измерениях.

Пример Построим эллипс $c(t) = (2\cos(t), 3\sin(t))$, где $0 < t < 2\pi$, используя следующие команды

```
>>t=0:0.2:2*pi;
```

```
>>plot (2*cos(t),3*sin(t))
```

Пример График кривой $c(t) = (2\cos(t), t^2, 1/t)$, $0.1 < t < 4\pi$, получаем с помощью следующих команд

```
>>t=0.1:0.1:4*pi;
```

```
>>plot3(2*cos(t),t.^2,1./t);
```

Графики трехмерных поверхностей получаем, задав подмножество прямоугольной области определения функции с помощью команды `meshgrid`, и затем используем команду `mesh` или `surf` для построения графика. Эти графики полезны для создания зрительных образов решений дифференциальных уравнений в частных производных.

Пример `>>x=-pi:0.1:pi; >y=x;`

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

```
>>z=sin(cos(x+y));
```

```
>>mesh(z)
```

Циклы и условные операторы

Операторы отношений

`==` Равно

`~=` Не равно

`<` Меньше

`>` Больше

`<=` Меньше или равно

`>=` Больше или равно

Логические операторы

~ Not (дополнение)

& And (справедливо, если справедливы оба операнда)

| Or (справедливо, если один из двух или оба операнда справедливы)

Булевы величины

1 True

0 False

Операторы for, if и while установлены в MATLAB аналогично их прототипам в других языках программирования. Они имеют следующую основную форму.

for (переменная цикла = выражение цикла)
--

Выполняемый оператор

end

if (логическое выражение)

Выполняемые операторы

else (логическое выражение)

Выполняемые операторы

end

while (выражение условия)

Выполняемые операторы

end

В следующем примере показано, как использовать вложенные циклы для построения матрицы. Запишем файл как М-файл под именем nest.m. Введя nest в MATLAB Command Window, можно создать матрицу A. Заметим, что если смотреть из верхнего левого угла, то элементы матрицы A – это элементы треугольной матрицы Паскаля.

Пример

```
for i=1:5
A(i, 1)=1;
A(1, i)=1;
end
for i=2:5
for j=2:5
A(i, j)=A(i, j-1)+A(i-1, j);
end
end
A
```

Команда break используется для выхода из цикла.

Пример

```
for k=1:100
x=sqrt(k);
if ((k>10)&(x-floor(x)==0))
break
end
end
k
```

Команду `disp` можно использовать для вывода на дисплей текста или матрицы.

```
Пример      n=10;
             k=0;
             while k<=n
               x=k/3;
               disp([x x^2 x^3])
               k=k+1;
             end
```

Программы

Эффективным способом создания программ является использование определенных программистом функций. Их записывают в виде М-файлов. Эти программы разрешают пользователю задавать ввод и вывод параметров. Они легко вызываются, как подпрограммы в других программах. Приведенный ниже пример позволяет увидеть результат построения треугольника Паскаля по первому числу. Введем следующую функцию в MATLAB Editor/Debugger и запишем ее как М-файл под именем `pasc.m`.

```
Пример      function P = pasc(n, m)
             % Вход - n -- количество строк
             %      - m -- первое число
             % Выход - P -- треугольник Паскаля
             for j=1:n
               P(j, 1)=1;
               P(1, j)=1;
             End
             for k=2: n
               for j=2: n
                 P(k,j)=rem(P(k,j-D,m)+rem(P(k-l,j) ,m);
               end
             end
```

Теперь в MATLAB Command Window введем `P=pasc(5,3)`, чтобы увидеть первые пять строк треугольника Паскаля по модулю 3, или попытаемся построить `P=pasc(175,3)`; (обратите внимание на точку с запятой). Введем `spr(P)` (строится разреженная матрица для больших значений n).

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

На m -языке Matlab написать функцию, которая:

1. Возвращает матрицу элементы которой $a_{ij}=i+j$.

2. Возвращает матрицу элементы которой $a_{ij}=i-j$.
3. Возвращает матрицу элементы которой $a_{ij}=i*j$.
4. Решает квадратное уравнение.

5. Вычисляет значение полинома, заданного коэффициентами.
6. Выводит на экран название Вашей специальности.
7. Возвращает матрицу элементы которой $a_{ij}=2^{i+j}$.
8. Вычисляет функцию логической неравнозначности.
9. Вычисляет логическую функцию стрелка Пирса.
10. Вычисляет логическую функцию штрих Шеффера.
11. Сравнивает два числа.
12. Вычисляет кубический корень.
13. Разворачивает матрицу в вектор.
14. Выводит на экран Вашу фамилию, имя и отчество.
15. Возвращает матрицу элементы которой пронумерованы по спирали, начиная с центра.
16. Подсчитывает количество нулевых элементов матрицы.
17. Возвращает сумму элементов матрицы.
18. Подсчитывает количество отрицательных элементов матрицы.
19. Транспонирует матрицу.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая версия программы Matlab установлена на Вашем компьютере?
2. В чем отличие версии Matlab установленной на компьютер, от описанной?
3. Какие пакеты программы Matlab установлена на Вашем компьютере?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 2

Пакет моделирования нейронных сетей `neural network toolbox`

Цель занятия:

Изучить интерфейс пакета моделирования нейронных сетей и получить практические навыки в создании компьютерных моделей искусственных нейронных сетей.

Подготовка к занятию:

Изучить по рекомендованной литературе и конспекту лекций основные понятия искусственных нейронных сетей, иметь практические навыки в использовании пакета компьютерной математики для реализации простейших математических вычислений.

Порядок проведения занятия

1. Графический интерфейс пакета моделирования нейронных сетей.
2. Создание программных моделей нейронных сетей.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

1. Графический интерфейс пакета моделирования нейронных сетей

Отличительной особенностью 12-го выпуска программных продуктов фирмы Math-Works является включение в их состав различных инструментальных средств организации диалога с пользователем. Как правило, это GUI-интерфейсы не являются исключением и пакет по нейронным сетям ППП NNT, 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. Конечно, 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 освоения новых объектов.

Вызов GUI-интерфейса NNTool возможен либо командой `nntool` из командной строки, либо из окна запуска приложений Launch Pad с помощью опции NNTool из раздела Neural Network Toolbox. После вызова на экране терминала появляется окно Network/Data Manager (Управление сетью/данными) (рисунок 1).

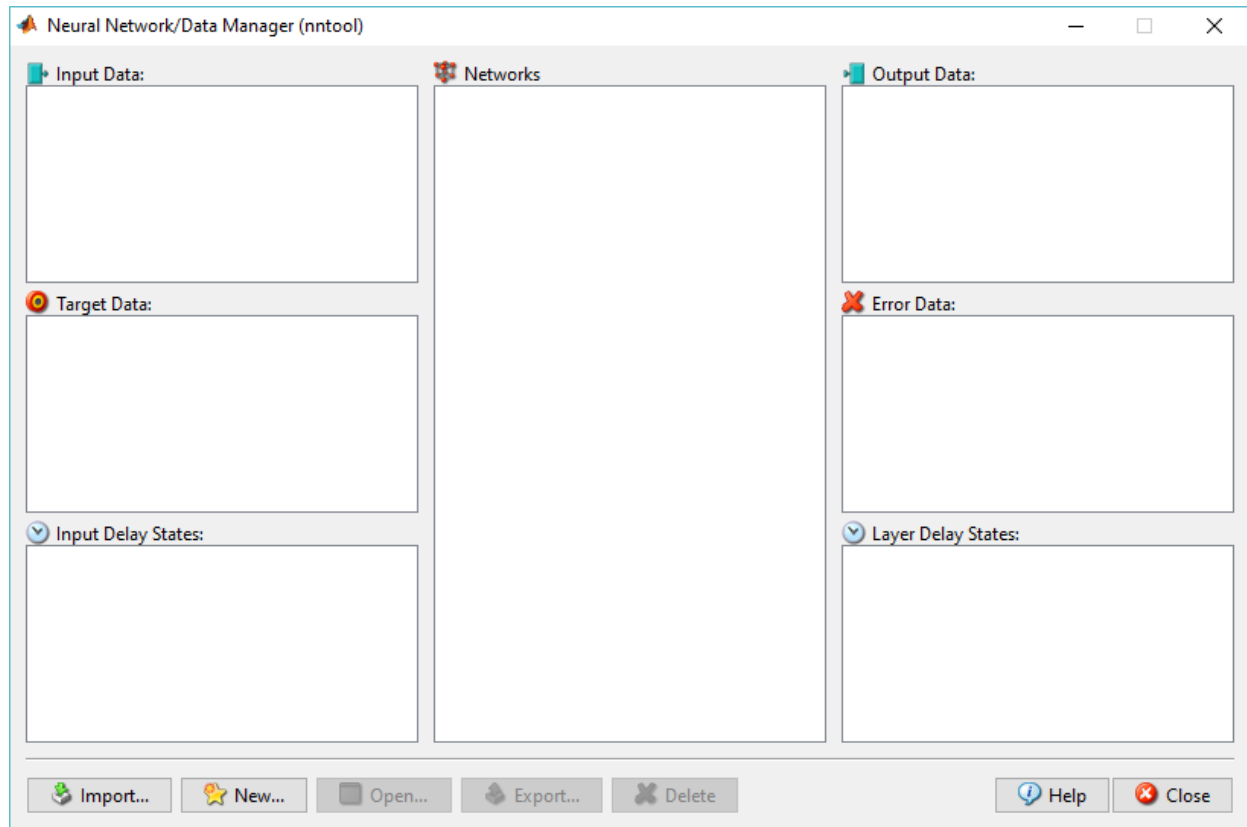


Рисунок 1 - Общий вид окна Network/Data Manager

Здесь:

Help - кнопка вызова окна подсказки Network/Data Manager Help (рисунок 2),

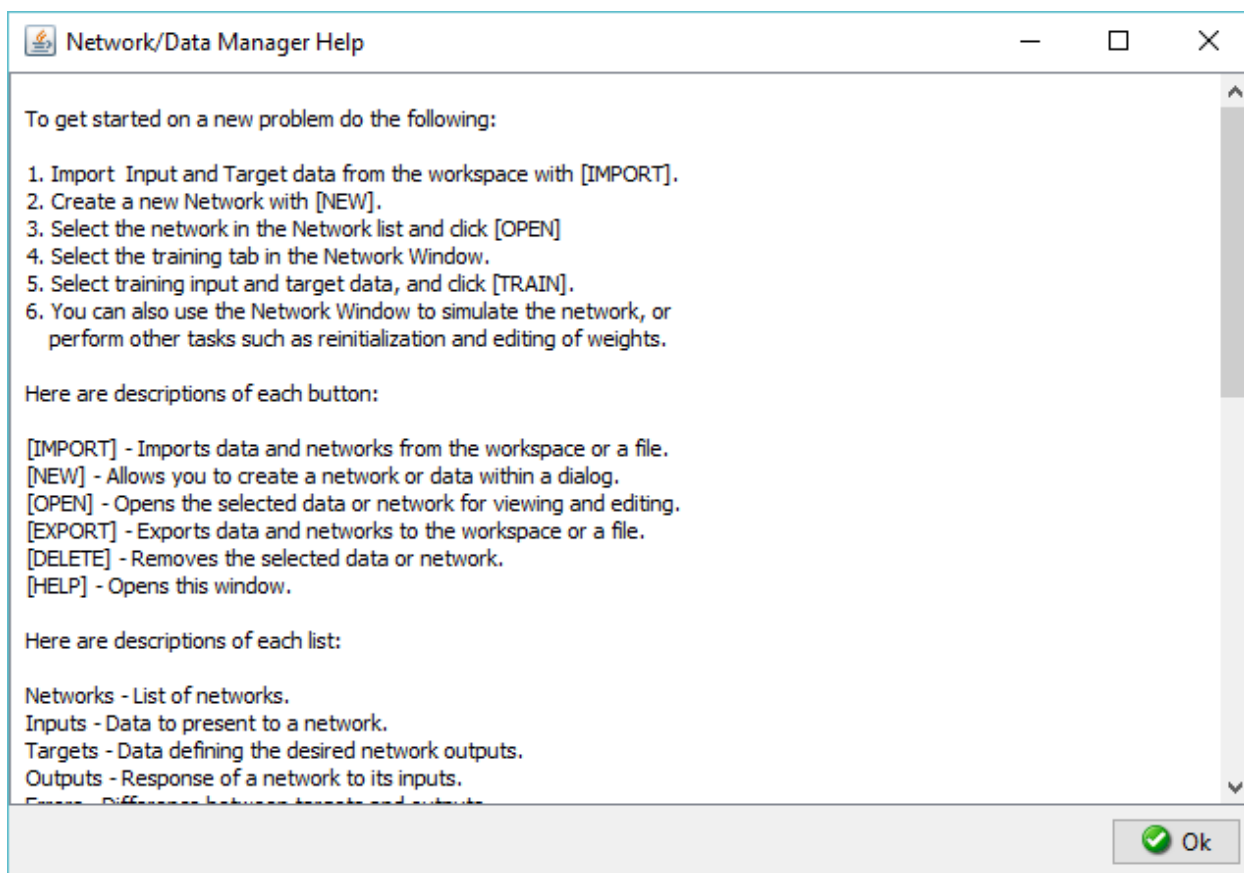


Рисунок 2 -Окно подсказки Network/Data Manager Help

New Data... - кнопка вызова окна формирования данных Create New Data (рисунок 3),

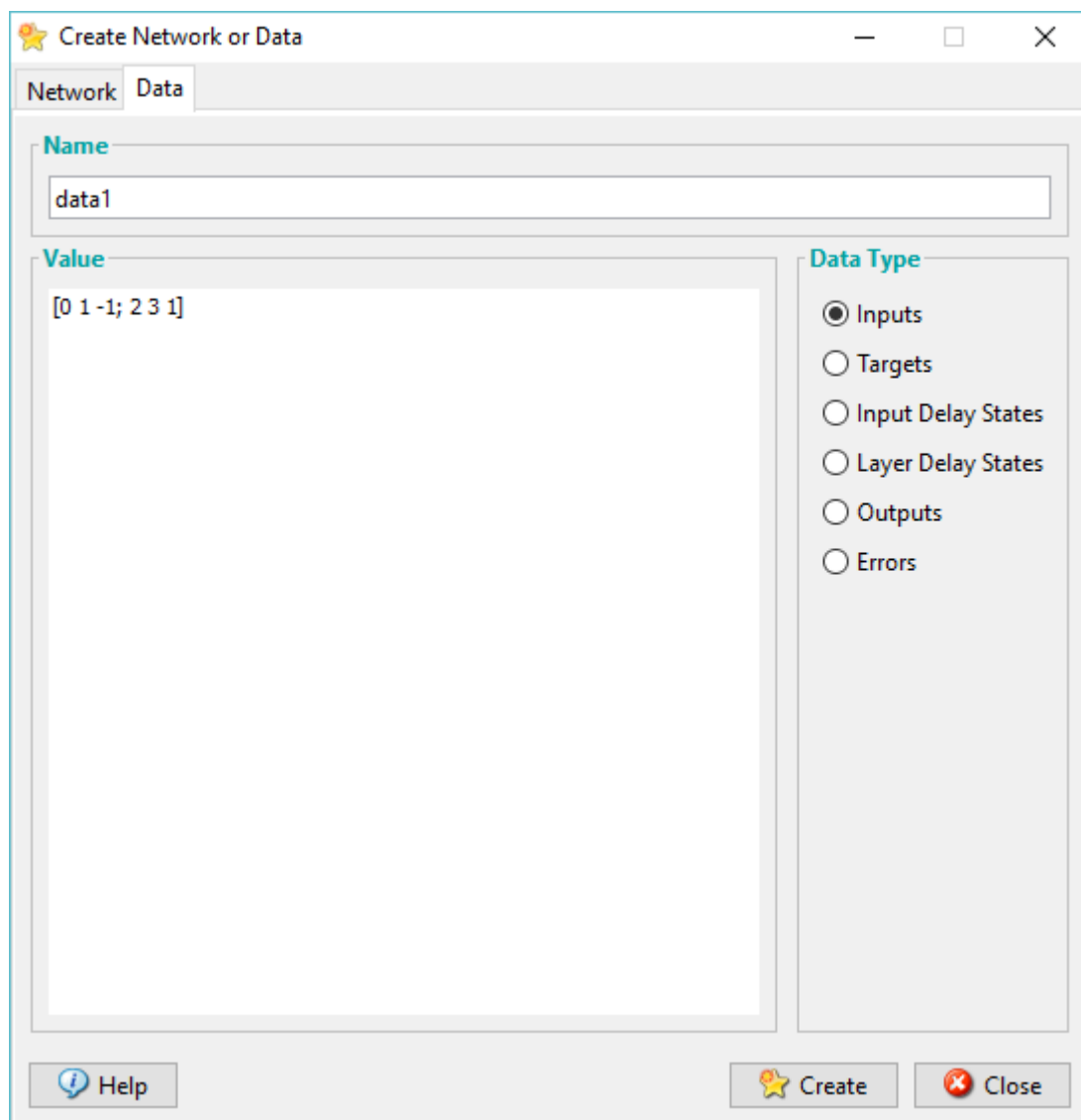


Рисунок 3 - Окно формирования данных

New Network... - кнопка вызова окна создания новой нейронной сети Create New Network (рисунок 4),

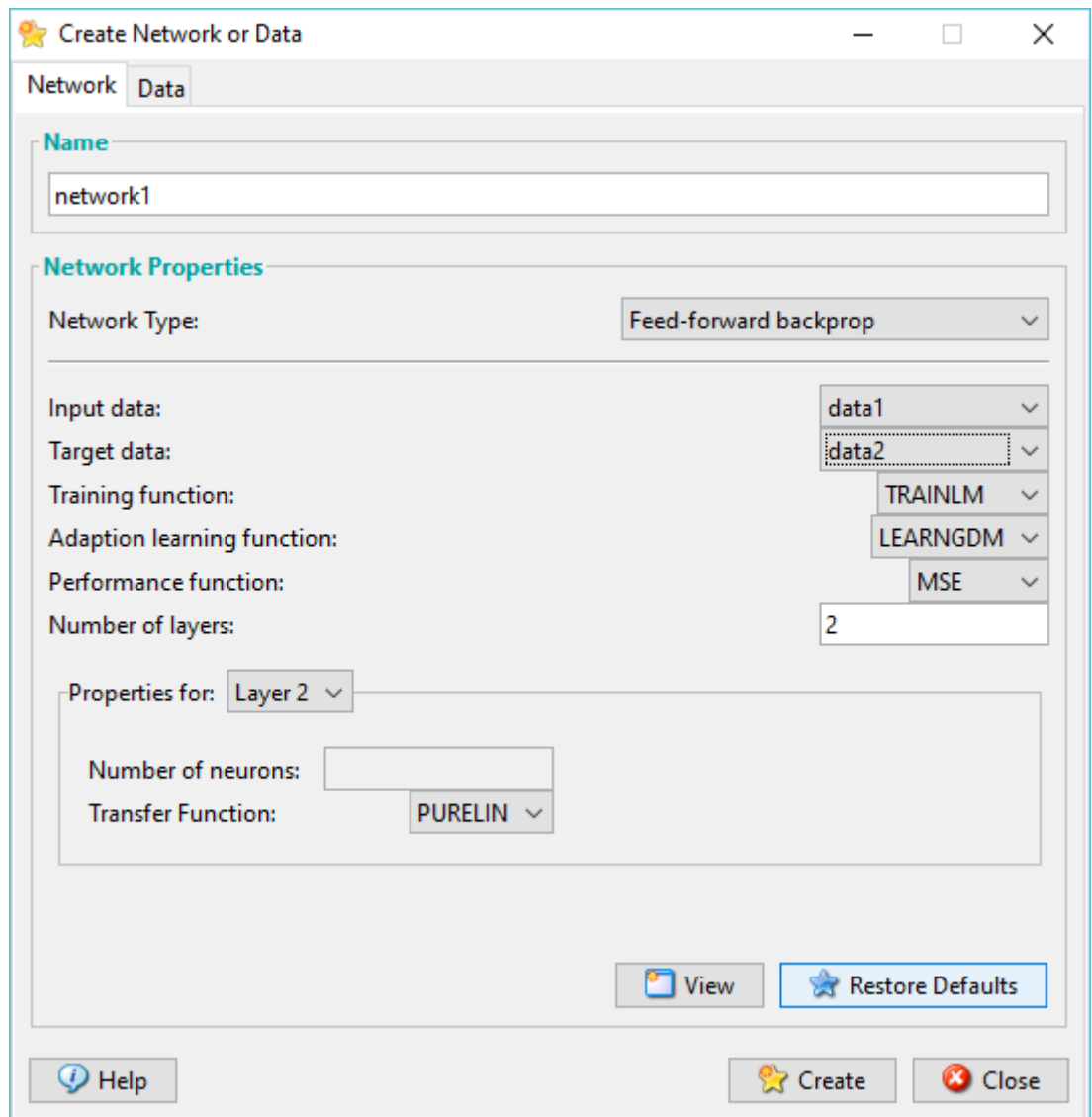


Рисунок 4 - Окно создания новой нейронной сети

Import... - кнопка вызова окна для импорта или загрузки данных Import or Load to Network/Data Manager (рисунок 5),

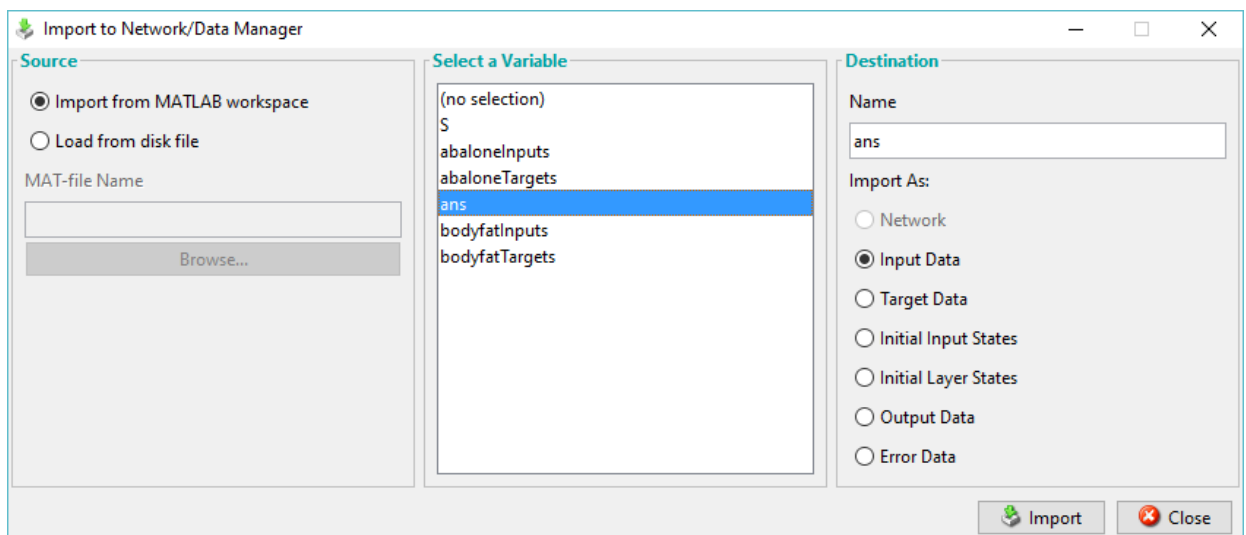


Рисунок 5 - Окно для импорта или загрузки данных

Export... - кнопка вызова окна для экспорта или записи данных в файл Export or Save from Network/Data Manager (рисунок 6).

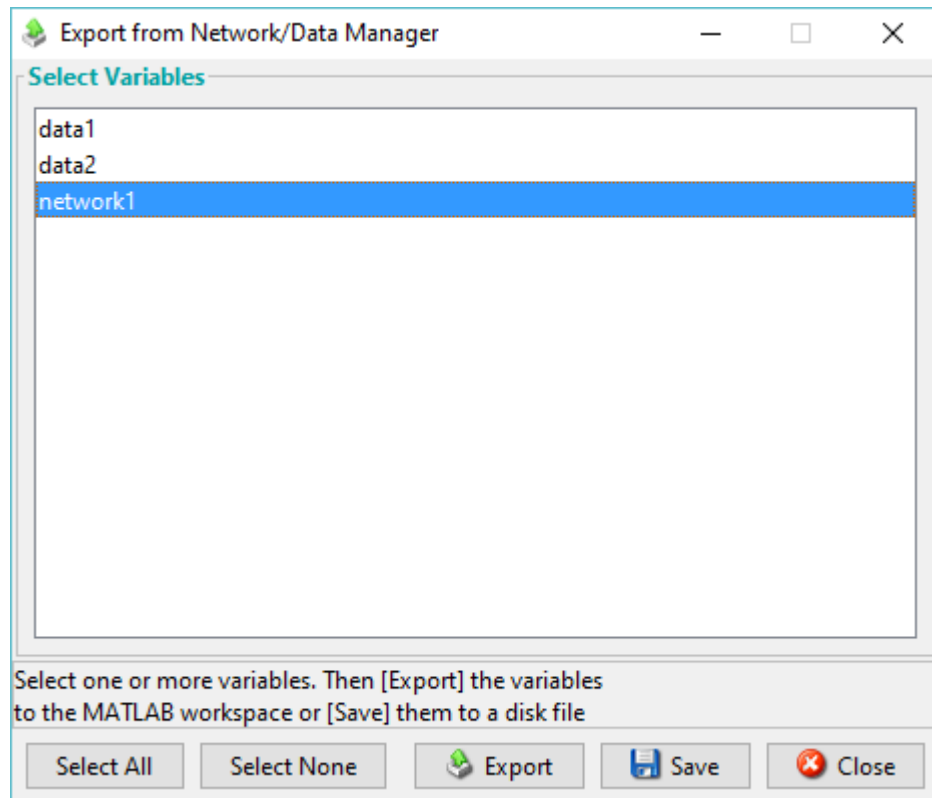


Рисунок 6 - Окно для экспорта или записи данных в файл Кнопки

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 активизации самой нейронной сети. Они позволяют просмотреть, удалить, инициализировать, промоделировать, обучить или адаптировать нейронную сеть и будут описаны ниже.

Прежде всего, рассмотрим назначение и способы работы с перечисленными выше окнами:

Окно Network/Data Manager Help. Это окно подсказки показано на рисунке 2 и описывает правила работы с диспетчером Network/Data Manager при создании нейронной сети.

Чтобы создать нейронную сеть, необходимо выполнить следующие операции:

- Сформировать последовательности входов и целей (кнопка New Data), либо загрузить их из рабочей области системы MATLAB или из файла (кнопка Import).

- Создать новую нейронную сеть (кнопка New Network), либо загрузить ее из рабочей области системы MATLAB или из файла (кнопка Import).
- Выбрать тип нейронной сети и нажать кнопку Train..., чтобы открыть окно для задания параметров процедуры обучения
- Открыть окно Network для просмотра, инициализации, моделирования, обучения и адаптации сети.

Окно Create New Data. Это окно показано на рисунке 3 и включает 2 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.

Различают следующие типы данных:

- Inputs (Входы) - последовательность значений входов;
- Targets (Цели) - последовательность значений цели;
- Input Delay States (Состояния ЛЗ входа) - начальные условия линии задержки на входе;
- Layer Delay States (Состояния ЛЗ слоя) - начальные условия линии задержки в слое;
- Outputs (Выходы) - последовательность значений выхода сети;
- Errors (Ошибки) - разность значений целей и выходов.

Как правило, пользователь задает только последовательности входа и цели, т. е. типы данных Inputs и Targets. При этом следует помнить, что при адаптации нейронной сети данные должны быть представлены в виде массива ячеек.

Окно Create New Network. Это окно показано на рисунке 4 и включает поля для задания параметров создаваемой сети. В зависимости от типа сети количество полей и их названия изменяются.

Обратимся к описанию 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 нижеследующей таблице. Интерфейс NNTool позволяет создавать нейронные сети только с одним или двумя слоями.

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

Training function (Функция обучения) - список обучающих функций
Adaption learning function (Функции настройки для режима адаптации) –

список функций настроек.

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

Number of layers (Количество слоев) - количество слоев нейронной сети
Properties for (Свойства) - список слоев Layer 1 (Слой 1), Layer 2 (Слой 2)
Number of neurons (Количество нейронов) - количество нейронов в слое
Transfer function (Функция активации) - функция активации слоя.

Окно Import or Load to Network/Data Manager. Это окно показано на рисунке 15 и включает 3 поля:

Source (Источник) - поле для выбора источника данных. Это либо рабочая область системы MATLAB (кнопка выбора Import from MATLAB Workspace), либо файл (кнопка выбора Load from disk file).

Если выбрана первая кнопка, то в поле Select a Variable вы можете видеть все переменные рабочей области и, выбрав одну из них, можете определить ее в поле Destination (Назначение) как последовательность входа Inputs (Входы).

Если выбирается кнопка Load from disk file, то активизируется поле MAT-file Name и кнопка Browse, что позволяет начать поиск и загрузку файла из файловой системы.

Окно Export or Save from Network/Data Manager Это окно показано на рис. 16 и позволяет передать данные из рабочей области GUI интерфейса NNTool в рабочую область системы MATLAB или записать их в виде файла на диске.

В данном случае нами выбрана переменная network1, которая принадлежит к классу network object и описывает нейронную сеть. После того как эта переменная экспортирована в рабочую область, можно, например, построить модель нейронной сети в системе Simulink с помощью оператора gensim.

Диалоговая панель Network показана на рисунке 7.

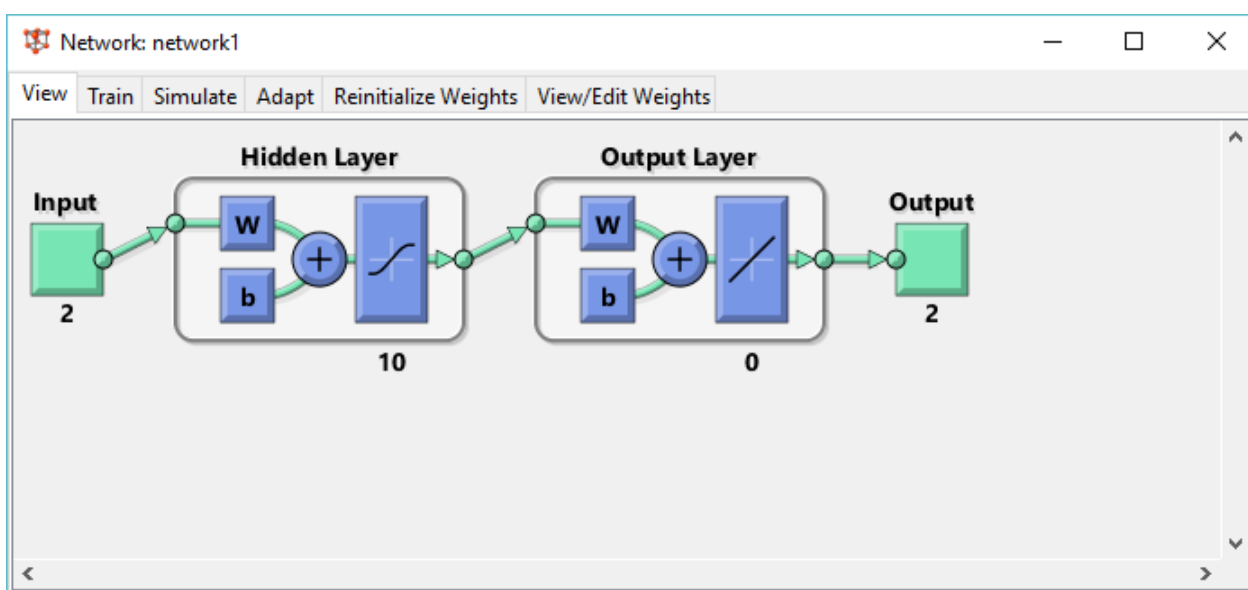


Рисунок 7 - Диалоговая панель Network

Она открывается только в том случае, когда в окне Network/Data

Manager выделена созданная сеть и становятся активными кнопки View, Initialize, Simulate, Train, Adapt.

Панель имеет 6 закладок:

- View (Просмотреть) - структура сети;
- Initialize (Инициализация) - задание начальных весов и смещений;
- Simulate (Моделирование) - моделирование сети,
- Train (Обучение) - обучение сети;
- Adapt (Адаптация) - адаптация и настройка параметров сети;
- Weights (Веса) - просмотр установленных весов и смещений.

Особенности работы с соответствующими окнами будут рассмотрены на приведенных ниже примерах создания конкретных нейронных сетей.

2. Создание программных моделей нейронных сетей

Создать и обучить нейронную сеть выполнению операции

$$y = x_1^2 + x_2,$$

если заданы последовательности входа

$$P = [1 \ 0.5 \ 0 \ 1, -2 \ 0 \ 0.5 \ 1]$$

и цели

$$T = [-1 \ 0.25 \ 0.5 \ 2].$$

Сформируем последовательности входов и целей в рабочей области GUI-интерфейса NNTool, используя окно Create New Data.

Выберем нейронную сеть типа feed-forward backprop с прямой передачей сигнала и с обратным распространением ошибки. Схема этой сети показана на рисунке 7.

Затем выполняется обучение сети, для чего выбирается закладка Train и открывается диалоговая панель, показанная на рисунке 8.

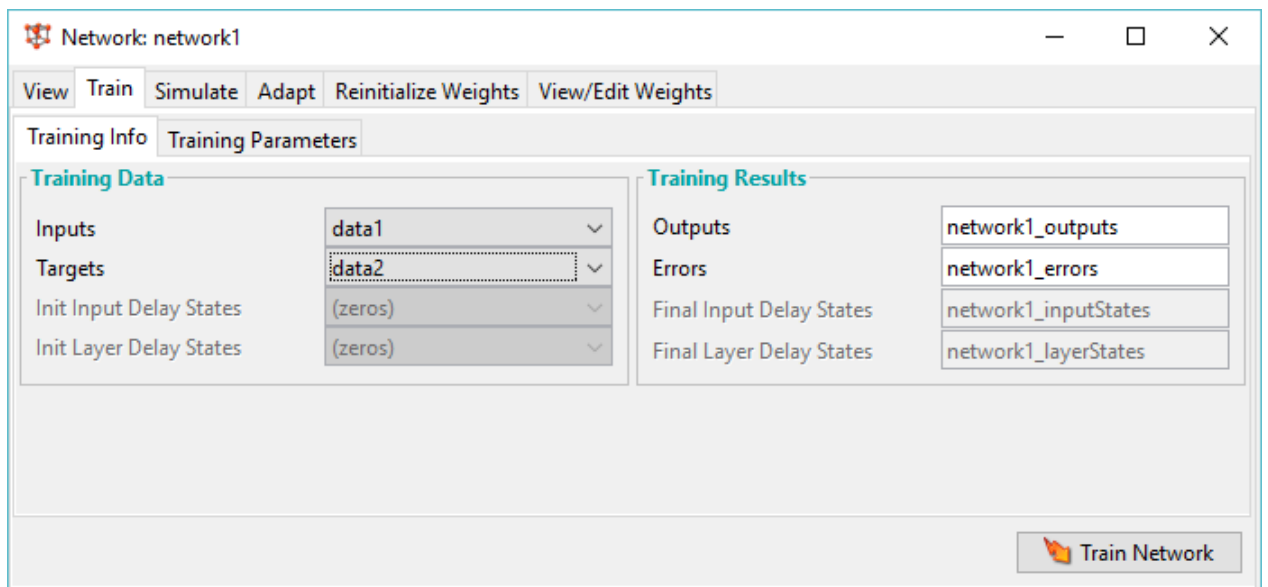
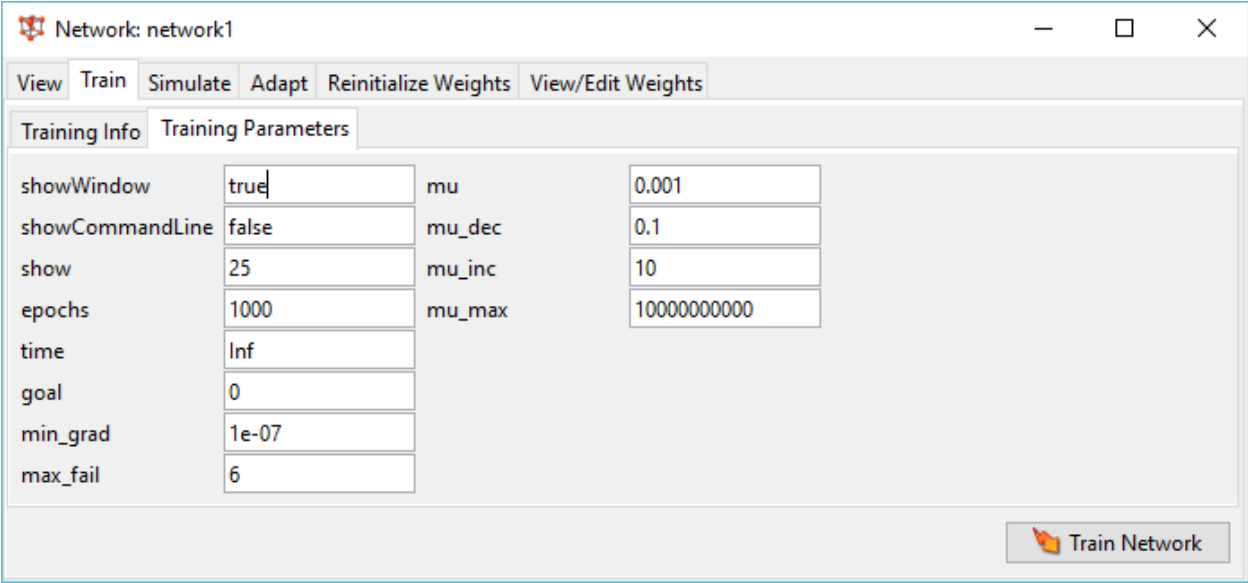


Рисунок 8 - Диалоговая панель обучения нейронной сети

Панель имеет три закладки:

- **Training Info** (Информация об обучающих последовательностях) (рисунок 8);
- **Training Parametrs** (Параметры обучения) (рисунок 9),



showWindow	true	mu	0.001
showCommandLine	false	mu_dec	0.1
show	25	mu_inc	10
epochs	1000	mu_max	10000000000
time	Inf		
goal	0		
min_grad	1e-07		
max_fail	6		

Рисунок 9 - Диалоговая панель параметров обучения нейронной сети

Применяя эти закладки, можно установить имена последовательностей входа и цели, а также параметров процедуры обучения

Теперь можно приступить к обучению сети (кнопка Train Network).

Качество обучения сети с прямой передачей сигнала на выбранной обучающей последовательности поясняется на рисунке 10. Практически нулевая точность достигается за 13 циклов обучения.

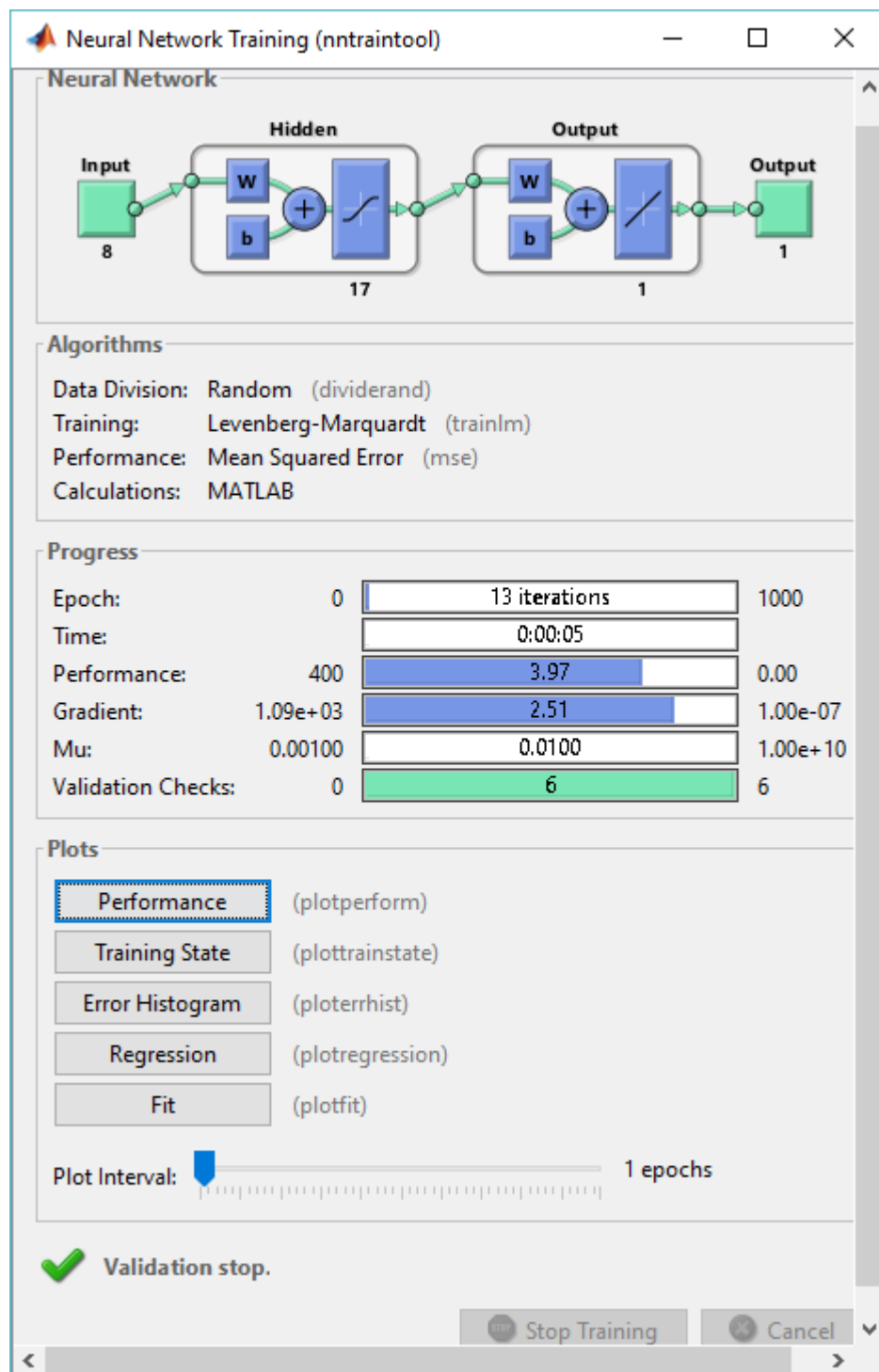


Рисунок 10 - процесс обучения нейронной сети

Соответствующие веса и смещения можно увидеть, если выбрать закладку Weights (рисунок 11).

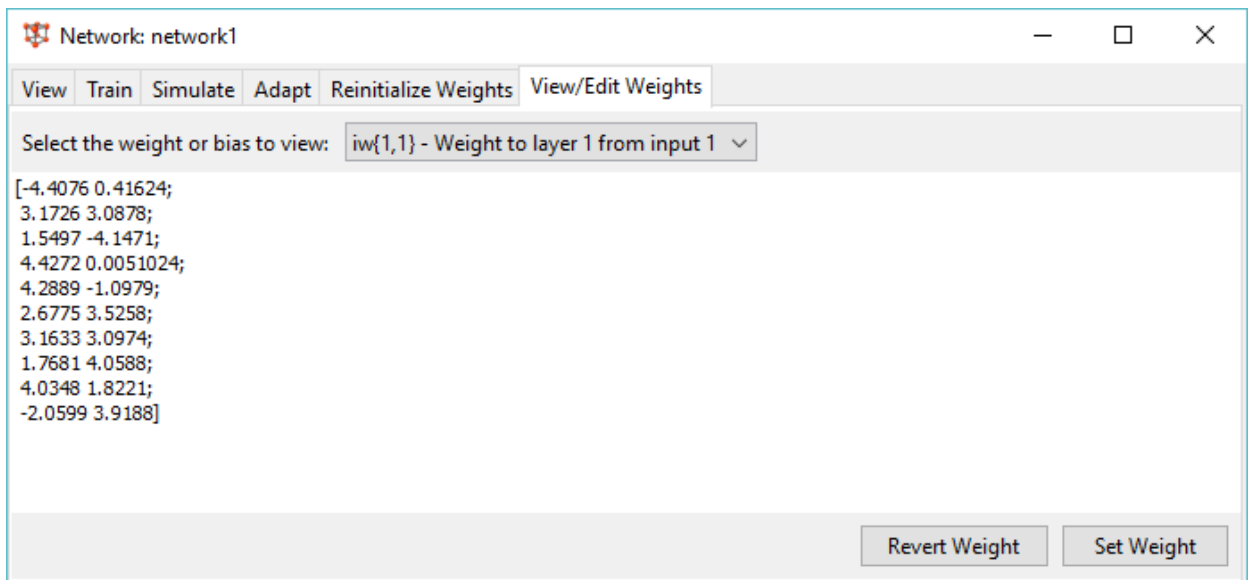


Рисунок 11 - Панель весов и смещений нейронной сети

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

При помощи пакета Neural Network Toolbox создайте модель нейронной сети прямого распространения с обратным распространением ошибки со следующими параметрами:

Количество входов – 2;

Количество выходов – 1;

Количество нейронов в скрытом слое – 5;

Функция активации:

- для скрытого слоя – арктангенс или логистическая;

- для выходного – линейная.

Обучите нейронную сеть на основе обучающего примера, приведенного в описании лабораторной работе. Экспортируйте полученную модель в Simulink. Вычертите ее в отчет. Симулируйте нейронную сеть произвольными данными. Сравните полученный выход сети с посчитанным по формуле. Сделайте вывод.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие компьютерные модели нейронных сетей можно создать при помощи пакета Neural Network Toolbox?
2. В чем отличие инструментов моделирования нейронных сетей пакета Neural Network Toolbox версии Matlab 6 и 7?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 3

Пакет моделирования нейронных сетей Neural Fitting

Цель занятия:

1. Расширить и углубить знания, полученные на лекциях по основам теории нейронных сетей.
2. Приобрести первичные навыки по проведению исследований и экспериментов с использованием компьютерных моделей информационных систем.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Инструментальные средства моделирования искусственных нейронных сетей

Отличительной особенностью 7-й версии программы Matlab является расширенный состав инструментальных средств моделирования искусственных нейронных сетей Neural Network Toolbox. Для запуска одного из инструментальных средств пакета в окне команд необходимо набрать:

```
>> nnstart
```

при этом появиться окно, представленное на рисунке 1.

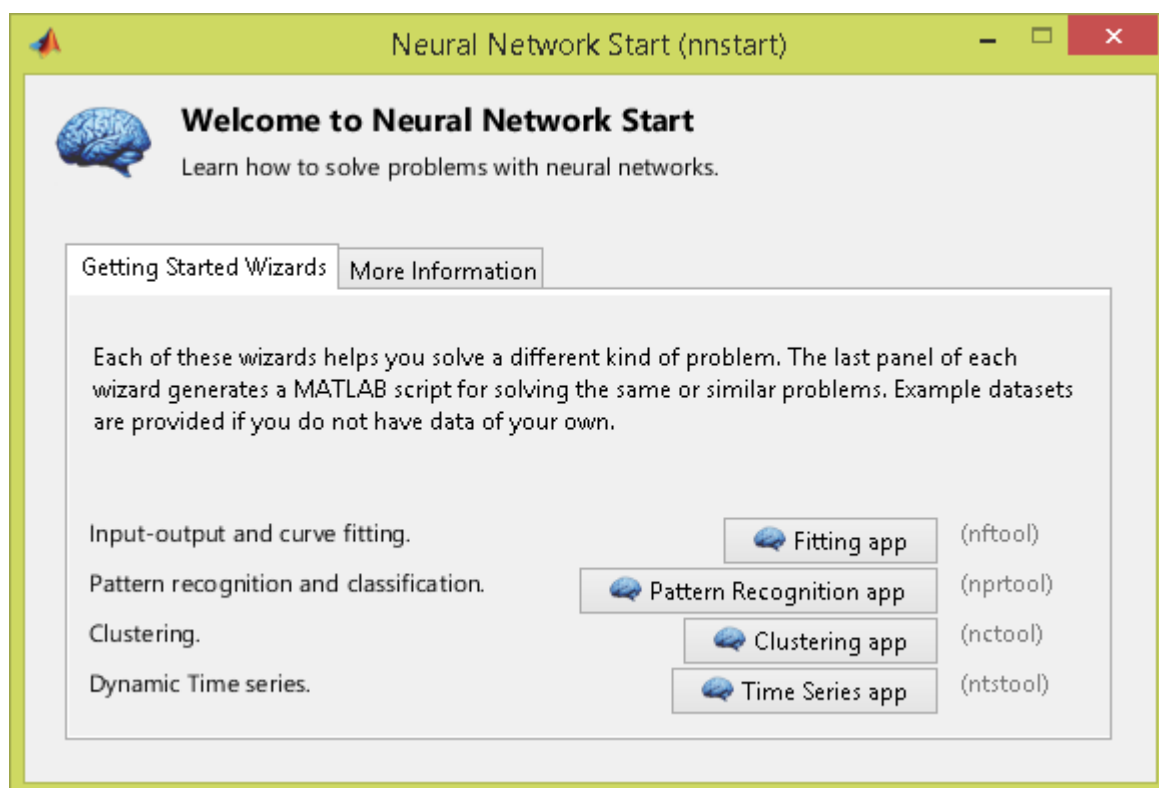


Рисунок 1 – Окно запуска инструментальных средств пакета Neural Network Toolbox

Состав пакета инструментальных средств Neural Network Toolbox приведен в таблице 1.

Таблица 1 – Состав инструментальных средств пакета Neural Network Toolbox

Название	Область применения	Команда
----------	--------------------	---------

		запуска
Fitting app	Аппроксимация функций	nftool
Pattern recognition app	Распознавание образов	nprtool
Clustering app	Классификация и кластеризация данных	nctool
Time series app	Анализ временных рядов	ntstool

Этапы проектирования компьютерных моделей искусственных нейронных сетей

Процесс проектирования компьютерных моделей искусственных нейронных сетей средствами MATLAB при решении задач приближения функций, распознавания образов, кластеризации и анализа временных рядов можно разбить на следующие этапы:

1. Подготовка исходных данных.
2. Задание архитектура используемой парадигмы нейронной сети.
3. Инициализация весов и смещений.
4. Обучение сети.
5. Проверка функционирования сети.
6. Использование сети.

Постановка задачи приближения функций

Задача аппроксимации функции заключается в определении по экспериментальным данным функции, наилучшим образом приближающейся к неизвестной и удовлетворяющей определенным критериям. Эта задача актуальна при моделировании сложных динамических систем.

Интерфейс пакета Network Fitting

Искусственные нейронные сети являются наилучшим аппроксиматором функций. Воспользоваться инструментальными средствами моделирования искусственных нейронных применительно к задаче приближения функций можно двумя способами:

1. С использованием визуального средства моделирования, запустив приложение Network Fitting Tool при помощи кнопки Fitting app, или командой `nftool` из окна команд. При этом запуститься мастер создания компьютерной модели нейронной сети приложения

функции. Окно мастера Neural Fitting представлено на рисунке 2.

2. При помощи интерфейса командной строки, с использованием функций пакета Network Fitting Tool.

На левой стороне приложения мастера Network Fitting приведено описание задачи приближения функции, а на правой – приведено изображение и приведено описание архитектуры используемой нейросетевой парадигмы:

1. Нейронная сеть прямого распространения с обратным

- распространением ошибки.
2. Количество слоев сети – 2.
 3. Функция активации нейронов скрытого слоя – логистическая, выходного – линейная.

Для создания компьютерной модели нейронной сети приближения функции необходимо нажать кнопку далее мастера Network Fitting.

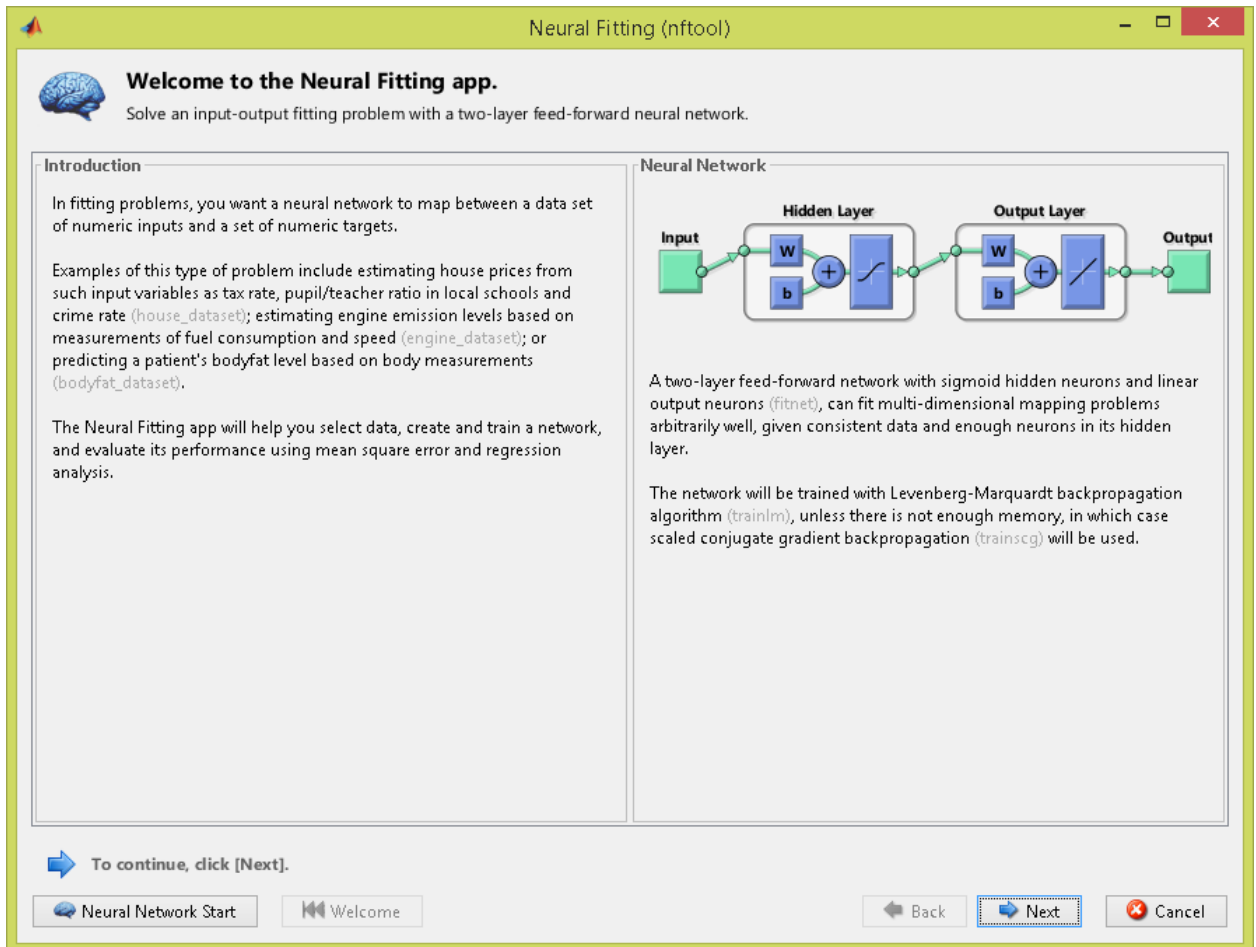


Рисунок 2 – мастер создания нейронной сети приближения функции при этом появится окно второго шага мастера Network Fitting – выбор данных (Select Data), представленное на рисунке 3.

Для примера рассмотрим процесс создания компьютерной модели нейронной сети логической функции И двух аргументов. Для задания исходных данных в рабочей области MATLAB необходимо сформировать матрицу входных данных `inputs`, содержащей все возможные комбинации аргументов и вектор значений логической функции `targets`. В окне команд для этого необходимо набрать:

```
>> inputs = [0 1 0 1; 0 0 1 1];  
>> targets = [0 0 0 1];
```

В рабочей области будут сформированы две матрицы `inputs` и `targets` (рисунок 4).

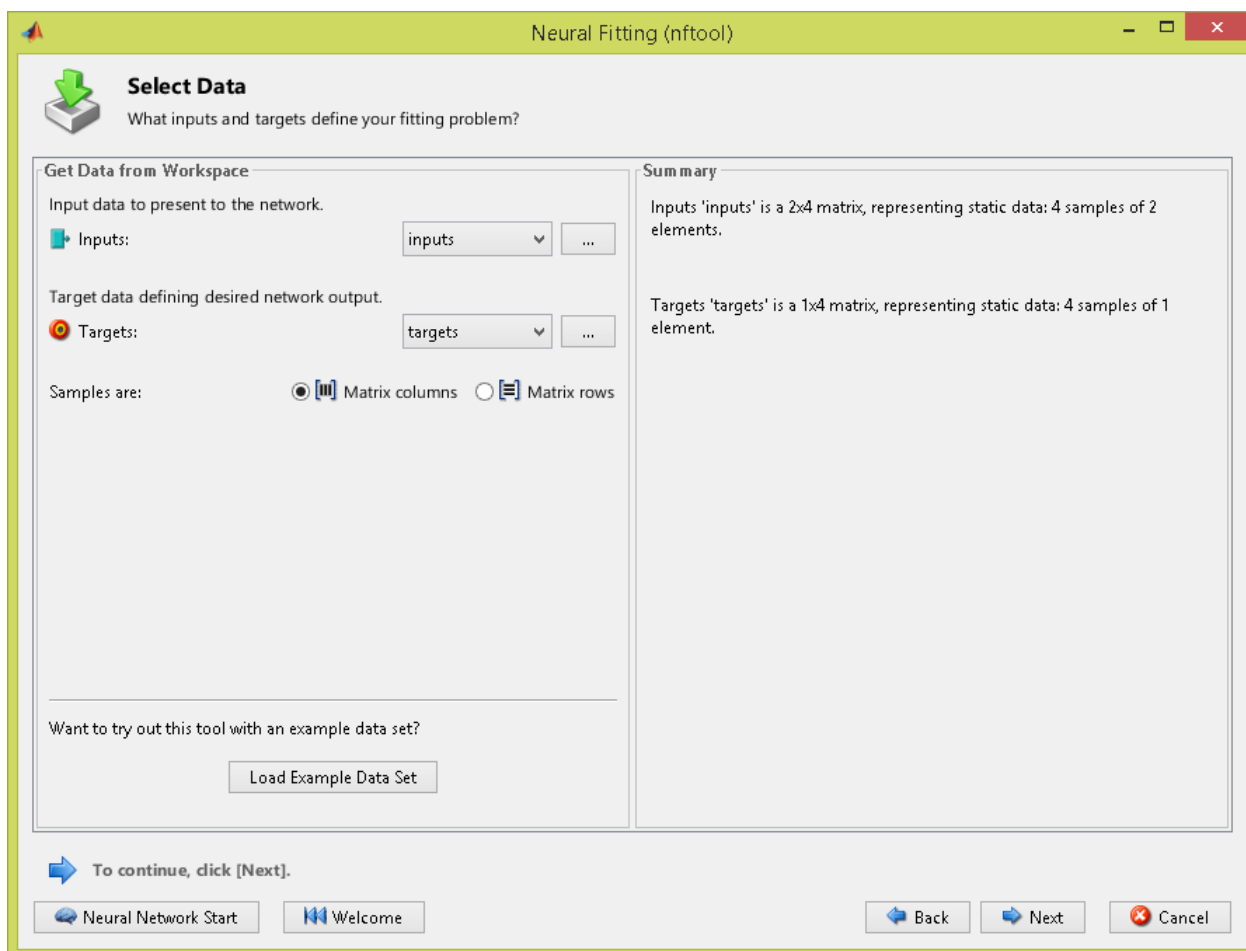


Рисунок 3 – Окно выбора данных мастера создания нейронной сети приближения функции

Workspace	
Name ▲	Value
inputs	[0 1 0 1; 0 0 1 1]
targets	[0 0 0 1]

Рисунок 4 – Рабочая область MATLAB с исходными данными для нейронной сети приближения функции

Для задания исходных данных для обучения нейронной сети в окне выбора данных необходимо выбрать созданные матрицы для входных данных Inputs и выходных данных Targets создаваемой сети. При этом на правой стороне мастера будет сформирован отчет и о проделанной работе. Для продолжения создания нейронной сети необходимо нажать кнопку Next. При этом будет выведено окно третьего этапа мастера создания проверочных и тестовых данных (Validation and Test Data, рисунок 5).

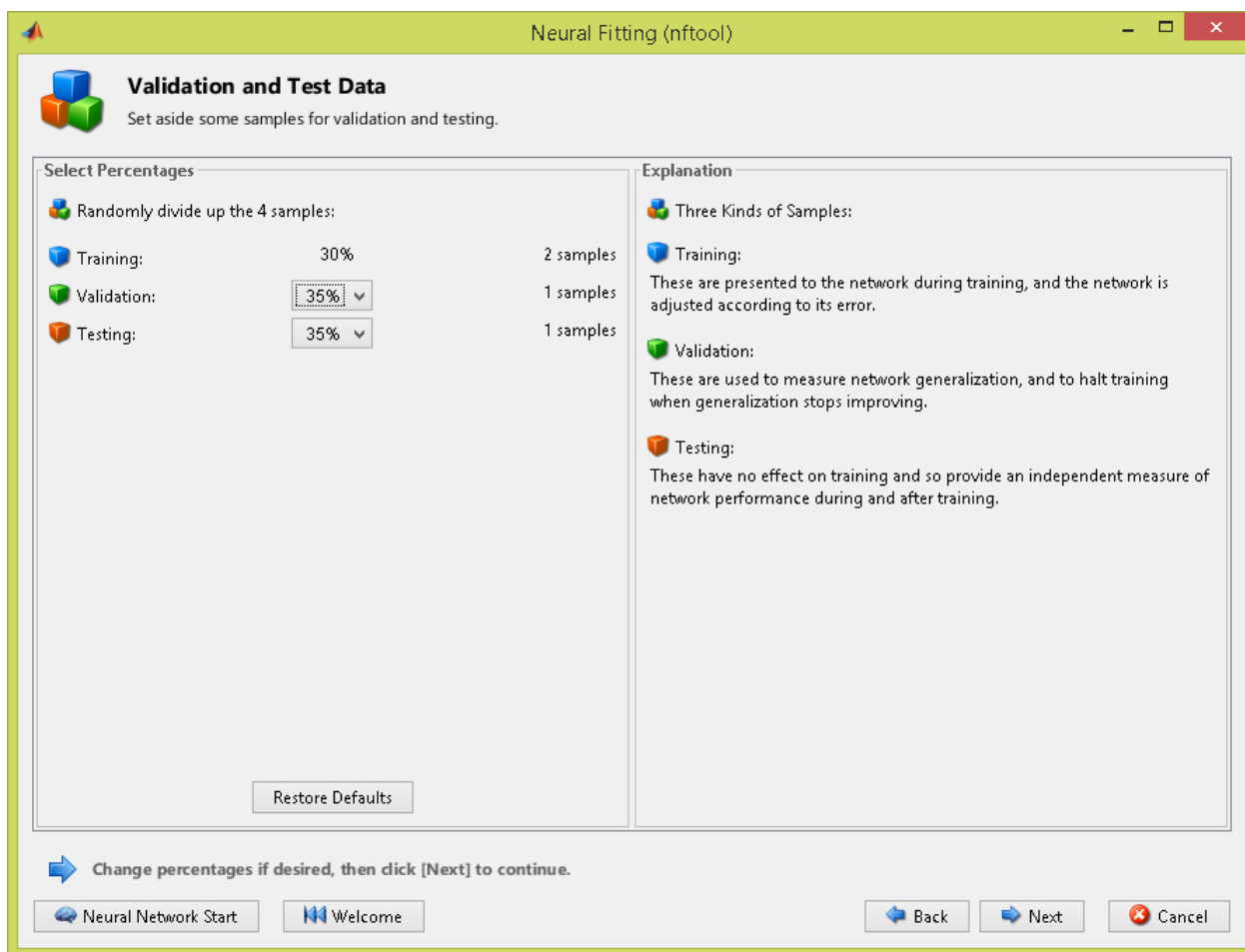
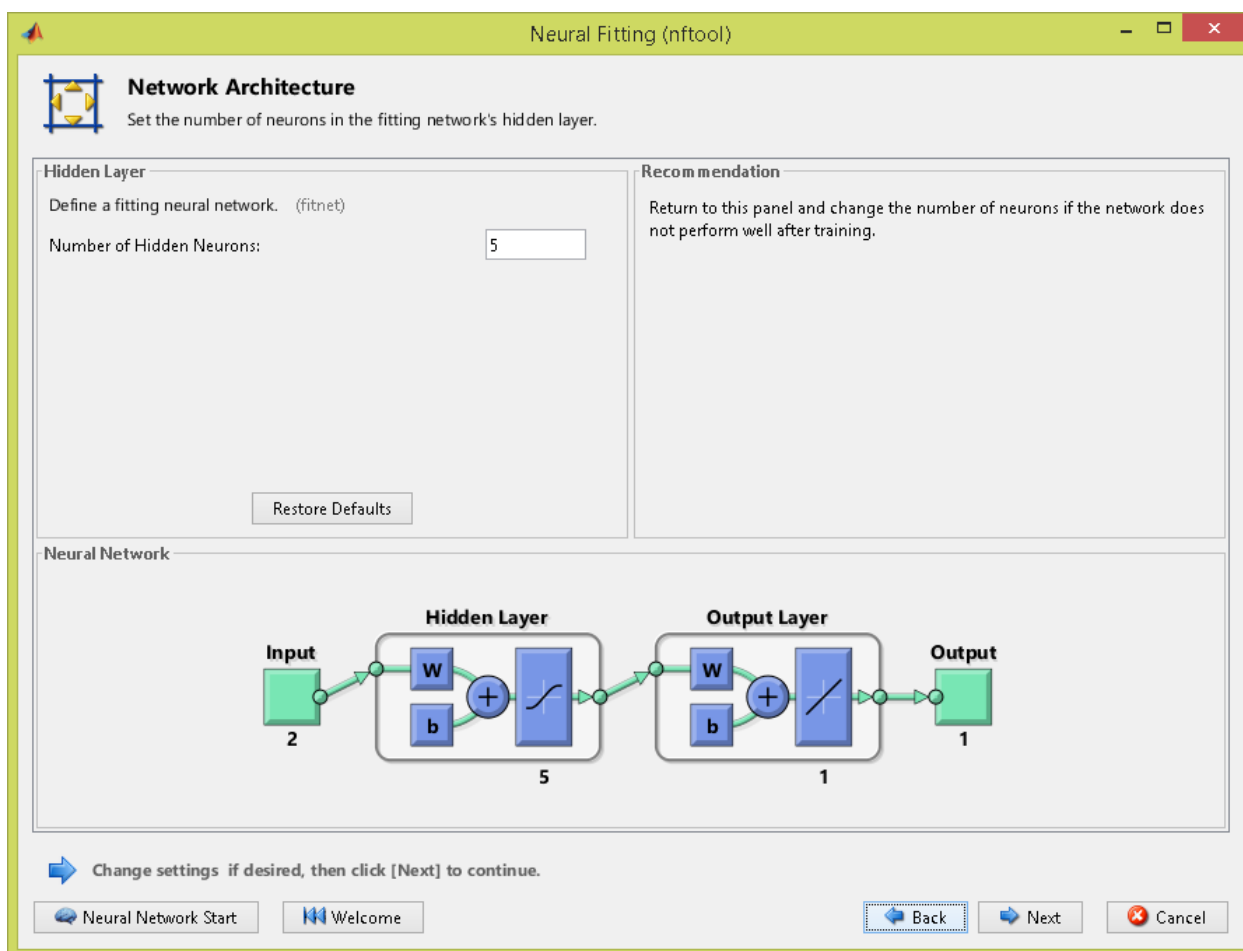


Рисунок 5 – Окно создания проверочных и тестовых данных

На нем представлено распределение исходных данных для обучения нейронной сети на обучающую выборку, тестовые и проверочные данные. Данные этих наборов выбираются случайным образом. При этом имеется возможность увеличить или уменьшить количество наборов для проверочной и тестовой выборки. Так для рассматриваемого примера 30% - обучающая выборка и по 35% - тестовые и проверочные данные.

После задания данных для обучения, тестов и проверки, для завершения этого этапа необходимо нажать кнопку Next. После чего будет выведено окно задания параметров создаваемой нейронной сети Network Architecture (рисунок 6). В поле количество нейронов для скрытого слоя (Number of Hidden Neurons) необходимо ввести требуемое число. По умолчанию мастер предполагает использовать 10 нейронов. Для рассматриваемого примера, при двух входах нейронной сети, достаточно 5 нейронов.



После нажатия кнопки Next будет выведено окно мастера, посвященное обучению нейронной сети (Train Network, рисунок 7). Оно позволяет выбрать алгоритм обучения нейронной сети.

В поле выбора алгоритма обучения нейронной сети (Choose a Trading Algorithm, рисунок 7) выбран градиентный алгоритм обучения. Для начала процесса обучения необходимо нажать кнопку Train в правой части окна мастера. После окончания процесса обучения выводится окно (Neural Network Trading, рисунок 8), содержащее результаты обучения нейронной сети. Для рассматриваемого примера нейронная сеть обучилась за 4 итерации до ошибки $7,04 \cdot 10^{-26}$.

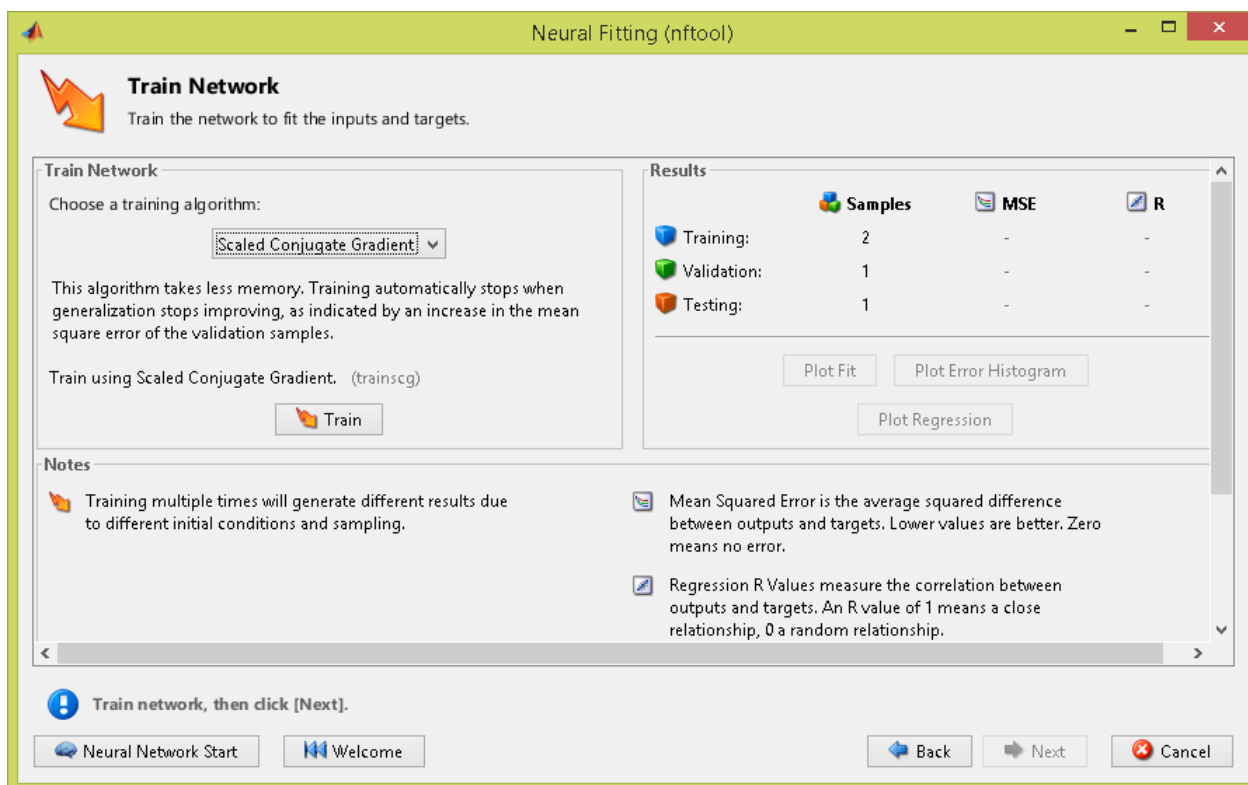


Рисунок 7 – Окно выбора алгоритма обучения нейронной сети

Для построения изменения функции ошибки в процессе обучения необходимо нажать кнопку Performance окна Neural Network Trading. График изменения функции ошибки представлен на рисунке 9.

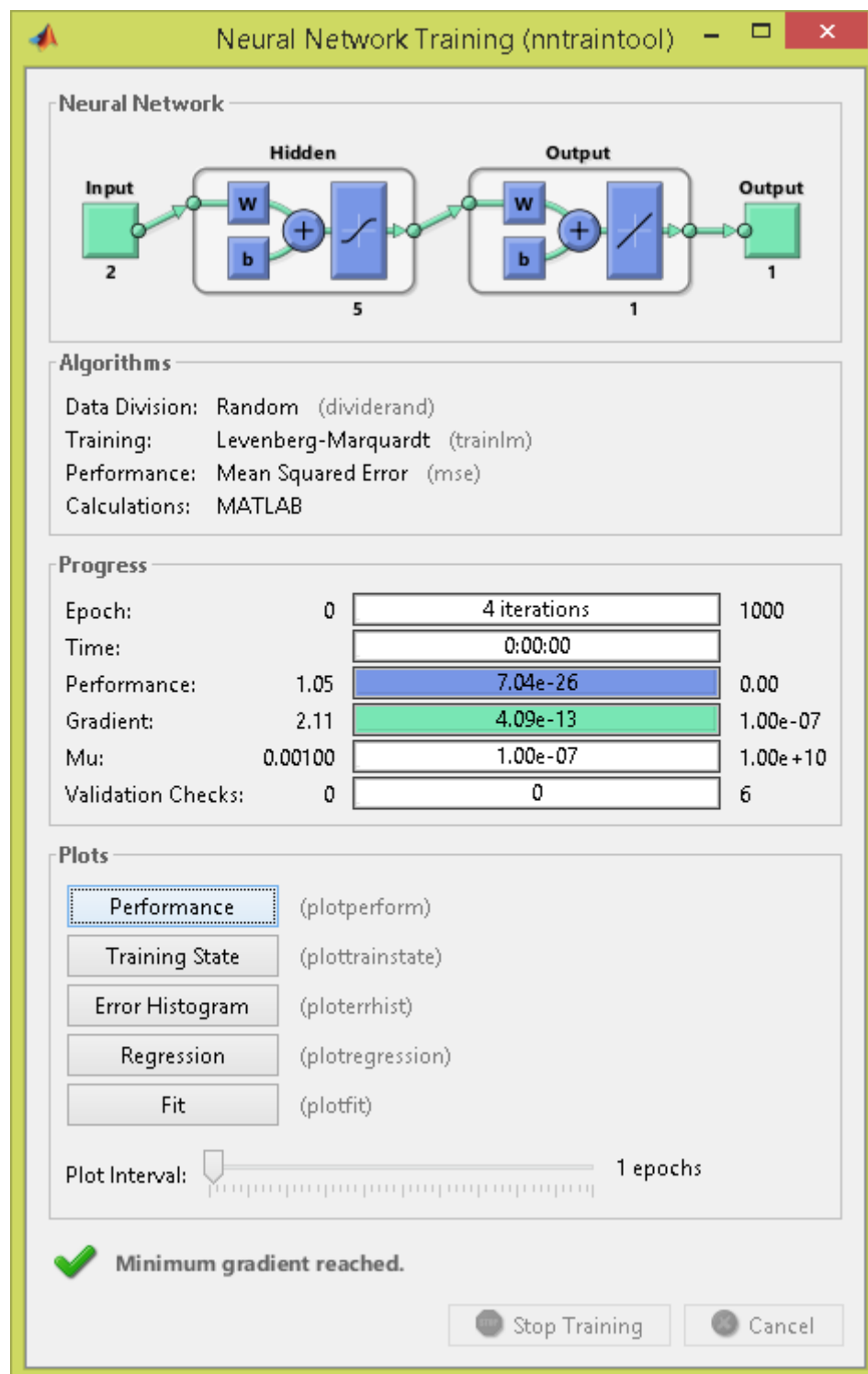


Рисунок 8 – Окно выбора алгоритма обучения нейронной сети

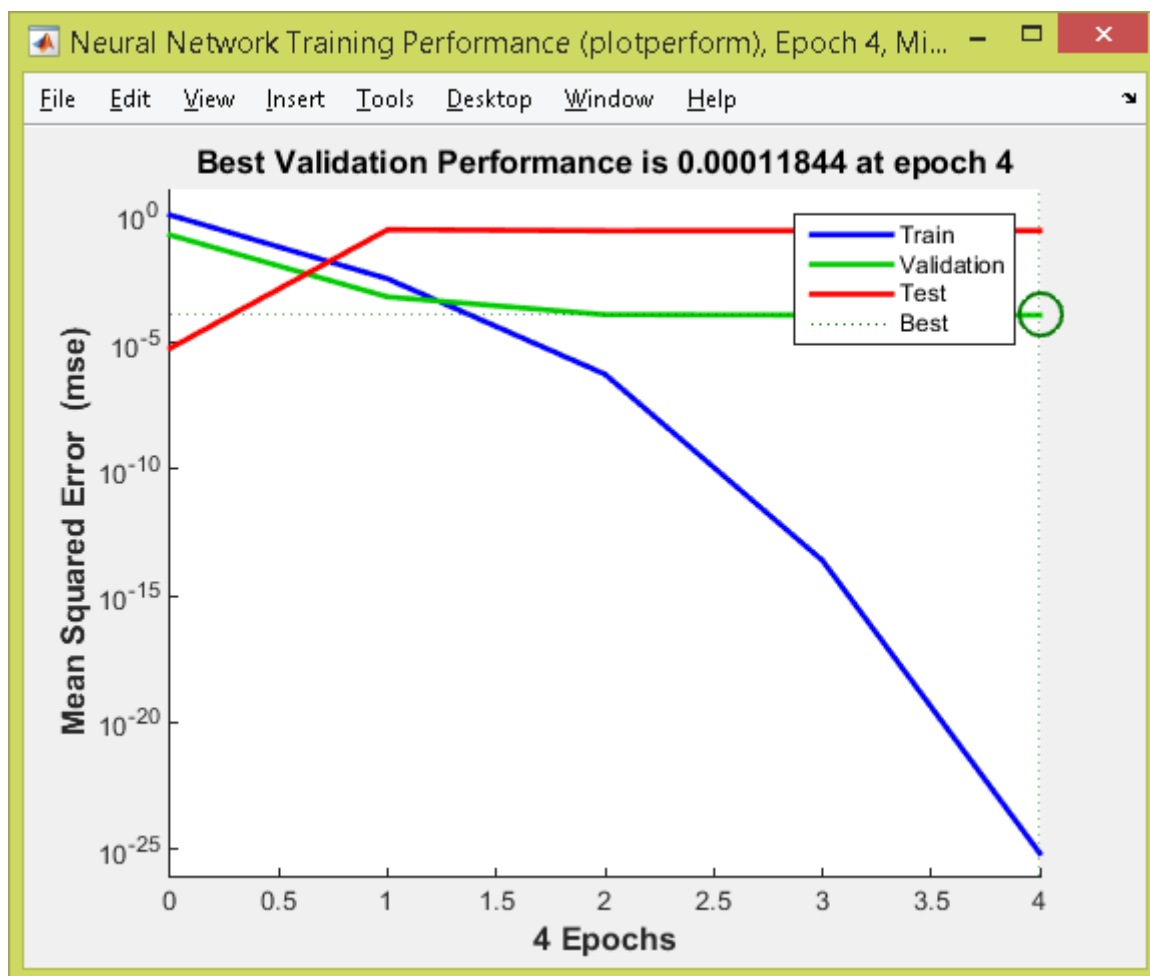


Рисунок 9 – График изменения функции ошибки в процессе обучения нейронной сети

Демонстрационные примеры

Для перехода к демонстрационным примерам пакета Network Fitting на втором шаге работы мастера, при выборе данных Select Data (рисунок 3.2), необходимо нажать кнопку Load Example Data Set. После этого откроется окно Fitting Data Set Chooser (рисунок 3.9), позволяющее загрузить данные для одного из семи демонстрационных примеров:

1. Simple Fitting Problem (пример простейшей аппроксимации),
2. House Pricing (оценка стоимости недвижимости),
3. Abalone Rings (оценка возраста морских моллюсков),
4. Body Fat (оценка лишнего веса для человека),
5. Building Energy (оценка энергетической потребности здания),
6. Chemical (управление химической реакцией),
7. Engine (оценка вращающего момента двигателя).

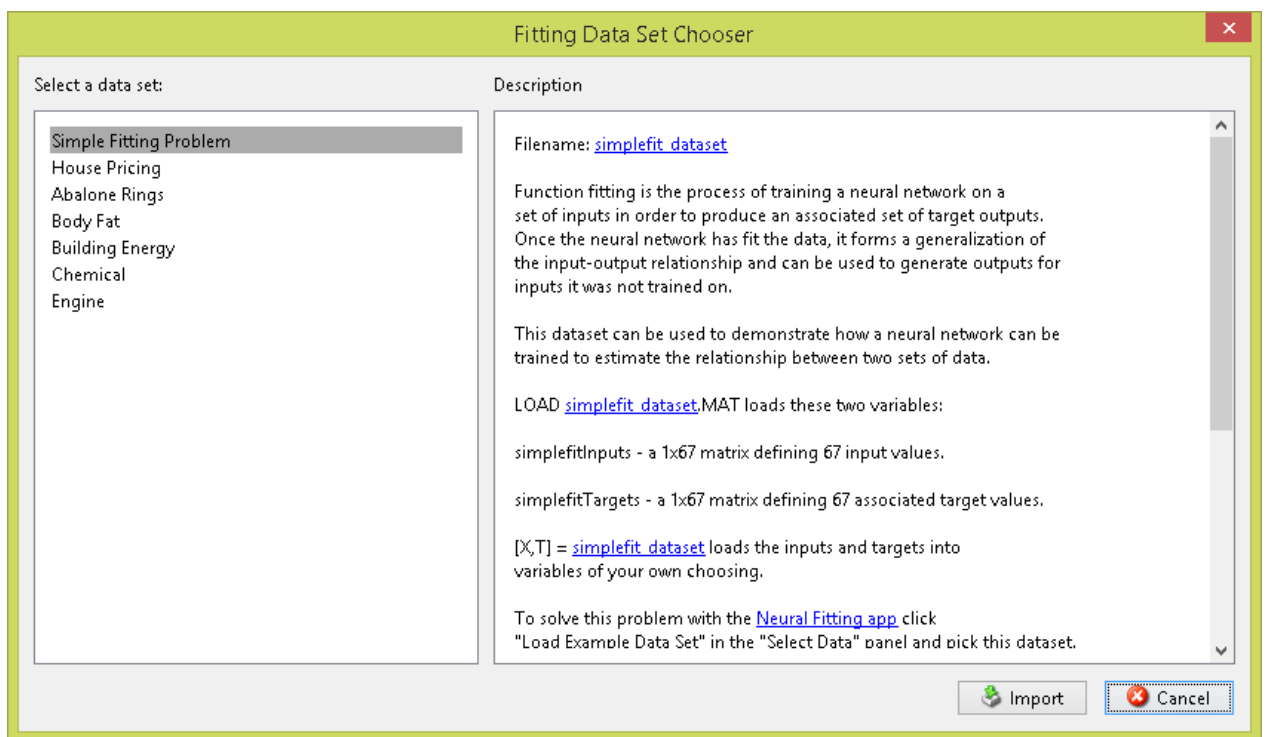


Рисунок 10 – Выбор данных для демонстрационных примеров пакета Network Fitting

1. Что представляет собой искусственный нейрон?
2. Что называют функцией активации?
3. В чем назначение функции активации?
4. Что называется обучением нейронной сети?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 4

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 (рис. 2.1).

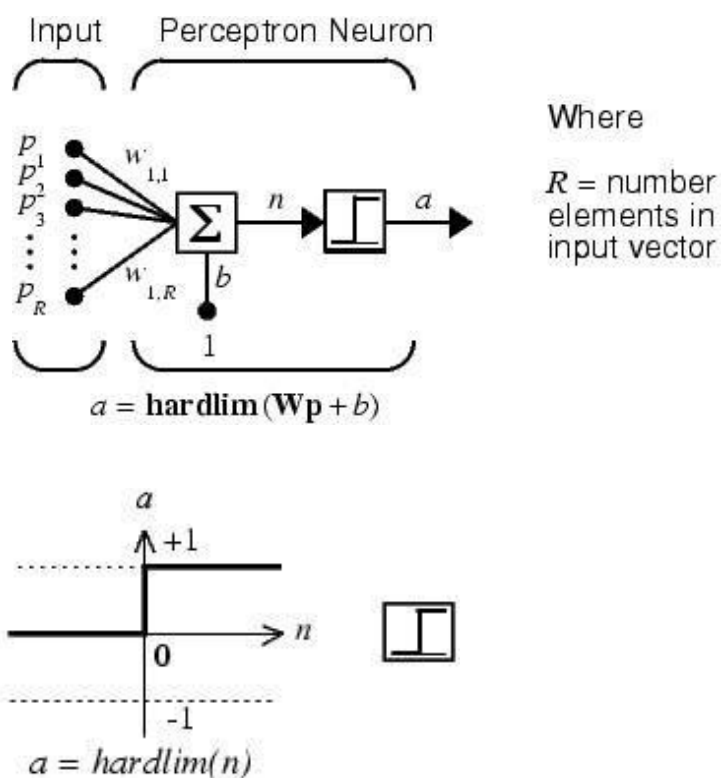


Рисунок 2.1. Структура порогового нейрона

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.

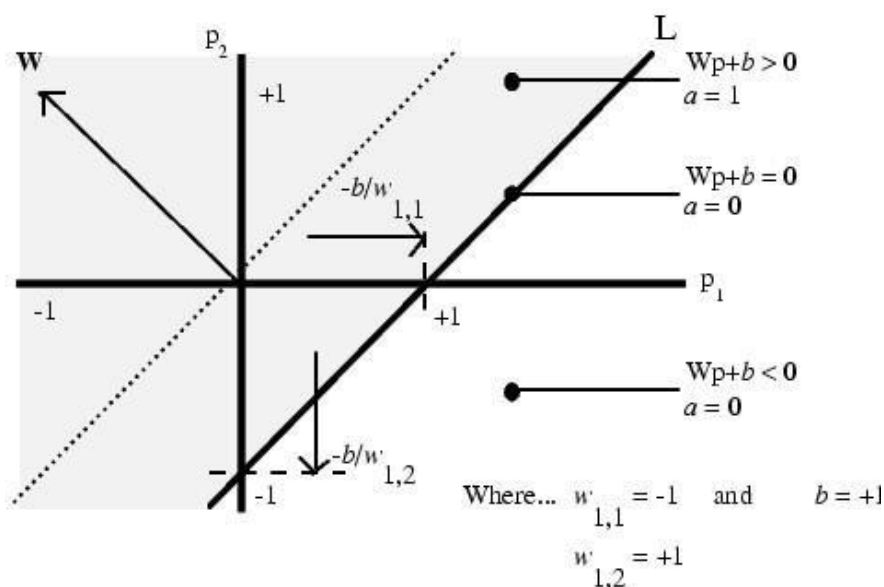


Рисунок 2.2. Разделение нейроном пространства входа на две области

Функция активации с жесткими ограничениями придает персептронную 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

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

Эта линия перпендикулярна к вектору весов w и смещена на величину b . Векторы входа выше линии L соответствуют положительному потенциалу нейрона, и, следовательно, выход 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 изменяет 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 [56, 57]:

$$\text{net} = \text{newp}(\text{PR}, \text{S}, \text{TF}, \text{LF}),$$

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:

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

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:

$$dW = \text{learnp}([], p, [], [], [], [], e, [], [], [], []).$$

Использование пустого списка `[ ]` означает параметр по умолчанию.

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

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

$$\text{т.е. } dw = e \cdot p'$$

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

Функция обучает нормализованные веса:

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

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

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

$$\text{т.е. } dw = e \cdot np'$$

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:

$$[\text{net}, Y, E, P_f, A_f] = \text{adapt}(\text{net}, P, T, P_i, A_i).$$

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, *P_f* – условия задержки входов, *A_f* – условия задержки слоя. Параметры *P_i* и *P_f* необязательные и необходимы только для сетей, имеющих задержки на входах и слое.

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

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

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

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:

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

где *net* – сеть;

P – входы сети;

P_i, – исходные условия задержки входов сети;

A_i, – исходные условия задержки слоя.

Функция возвращает

Y – выходы сети;

P_f – окончательные условия задержки входов;

A_f – 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
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 .
Функция	Назначение
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 M 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
<code>net.trainParam.epochs</code> 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 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 P2
net.trainParam.epochs = 20; % установка количества проходов

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

<P2, T2>

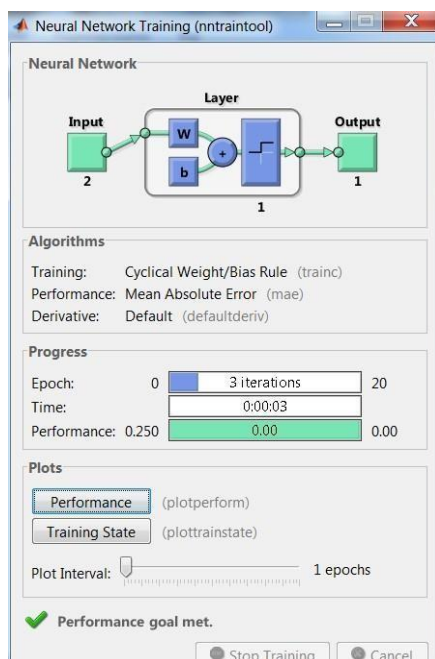


Рисунок 2.3. Окно обучения нейронной сети

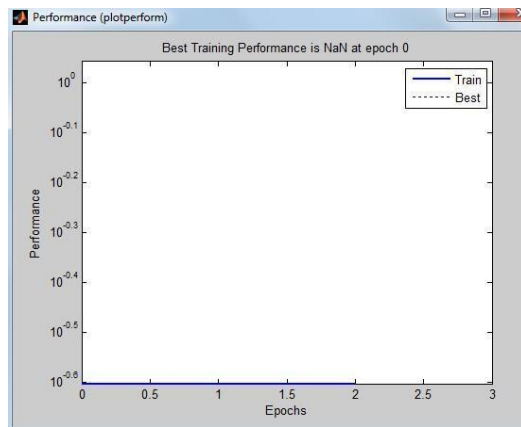


Рисунок 2.4. Изменение ошибки сети в процессе обучения

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 `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)`

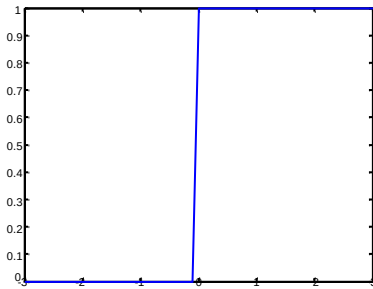


Рисунок 2.5. График функции активации `hardlim`

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 `p1`
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:

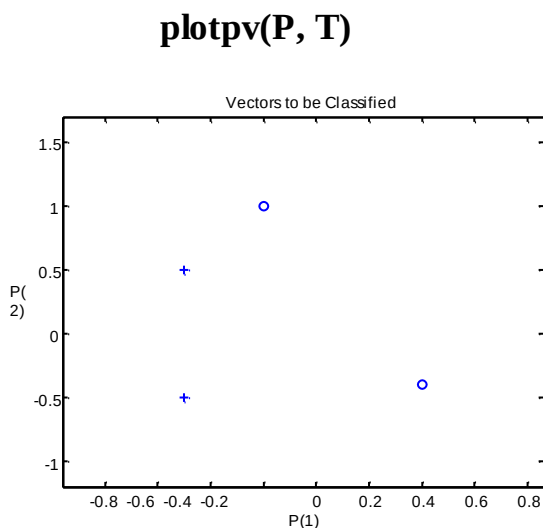


Рисунок 2.6. Изображение входных векторов на плоскости

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

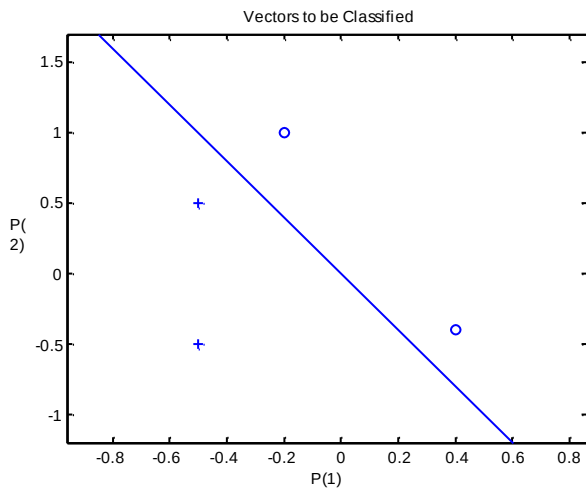


Рисунок 2.7. Изображение разделения пороговым нейроном входных векторов

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
```

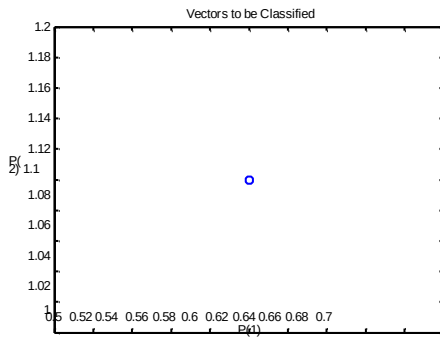


Рисунок 2.8. Изображение входа P , отнесенного нейроном к категории a

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 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

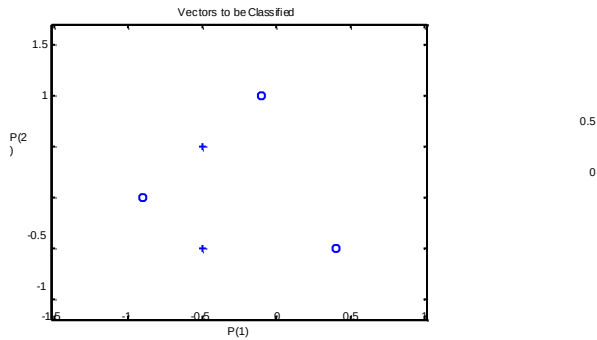


Рисунок 2.9. Изображение входных векторов на плоскости

T = [1 1 0 0 0]; % выходы – реакции нейрона на входы 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 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 управляющей структуры изображения для новой разделяющей прямой

drawnow; % изображение разделяющей прямой

end; % конец цикла

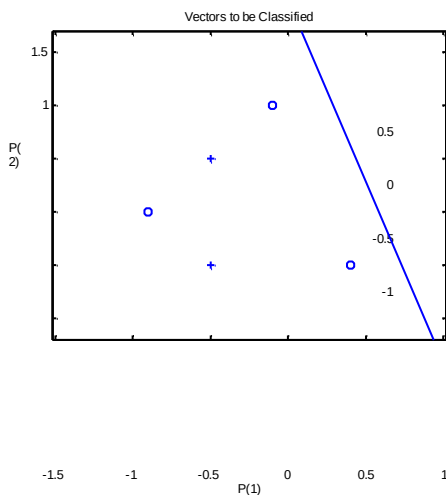


Рисунок 2.10.
Изображение
невозможности
разделения пороговым
нейроном линейно
неразделимых векторов

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

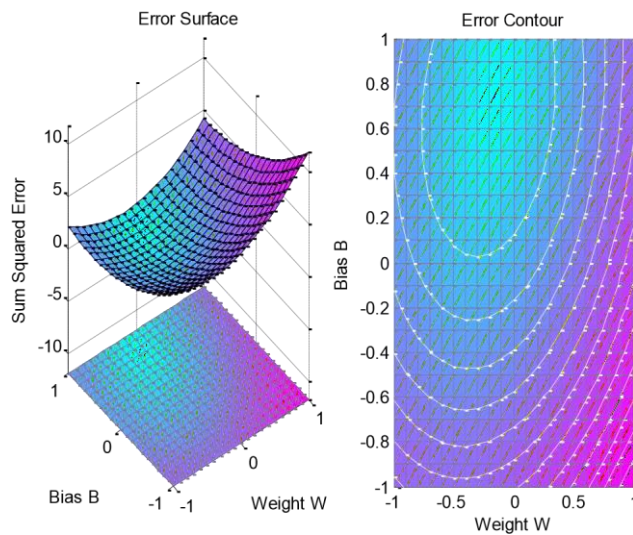


Рисунок 2.11. Изображение поверхности ошибки в координатных осях – весов и пороговых значений

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
```

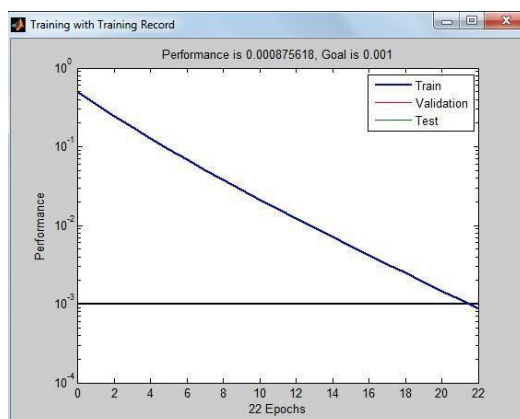


Рисунок 2.12. Изменение среднеквадратической ошибки сети в процессе обучения

```
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.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 5

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 [57].

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.

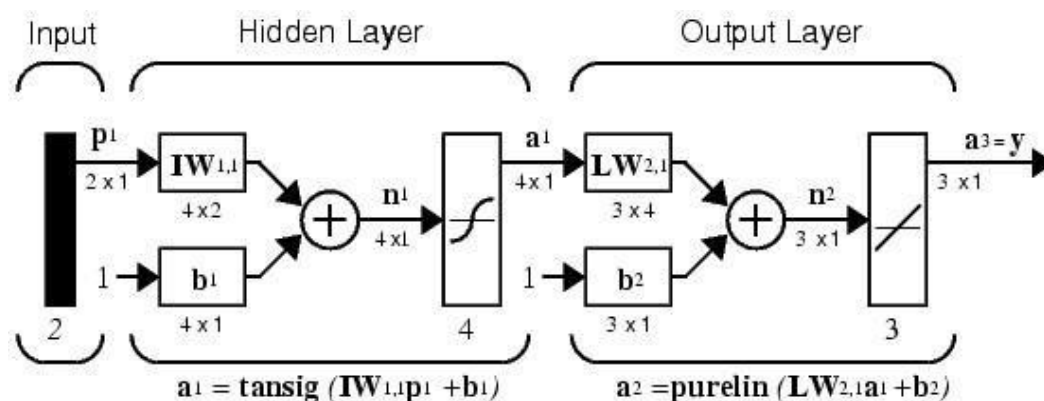


Рисунок 3.1. Схема архитектуры нейронной сети с прямой передачей сигнала

Здесь приняты обозначения: p^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:

$\text{net} = \text{newff}(\text{PR}, [\text{S1 S2 SN}], \{\text{TF1 TF2 TFN}\}, \text{BTF}, \text{BLF}, \text{PF}).$

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 *learnsgdm*; *PF* – функция измерения ошибки, по умолчанию *mse*. Функция *newff* возвращает многослойную нейронную сеть прямого и обратного распространения сигнала и ошибки соответственно. 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)
```

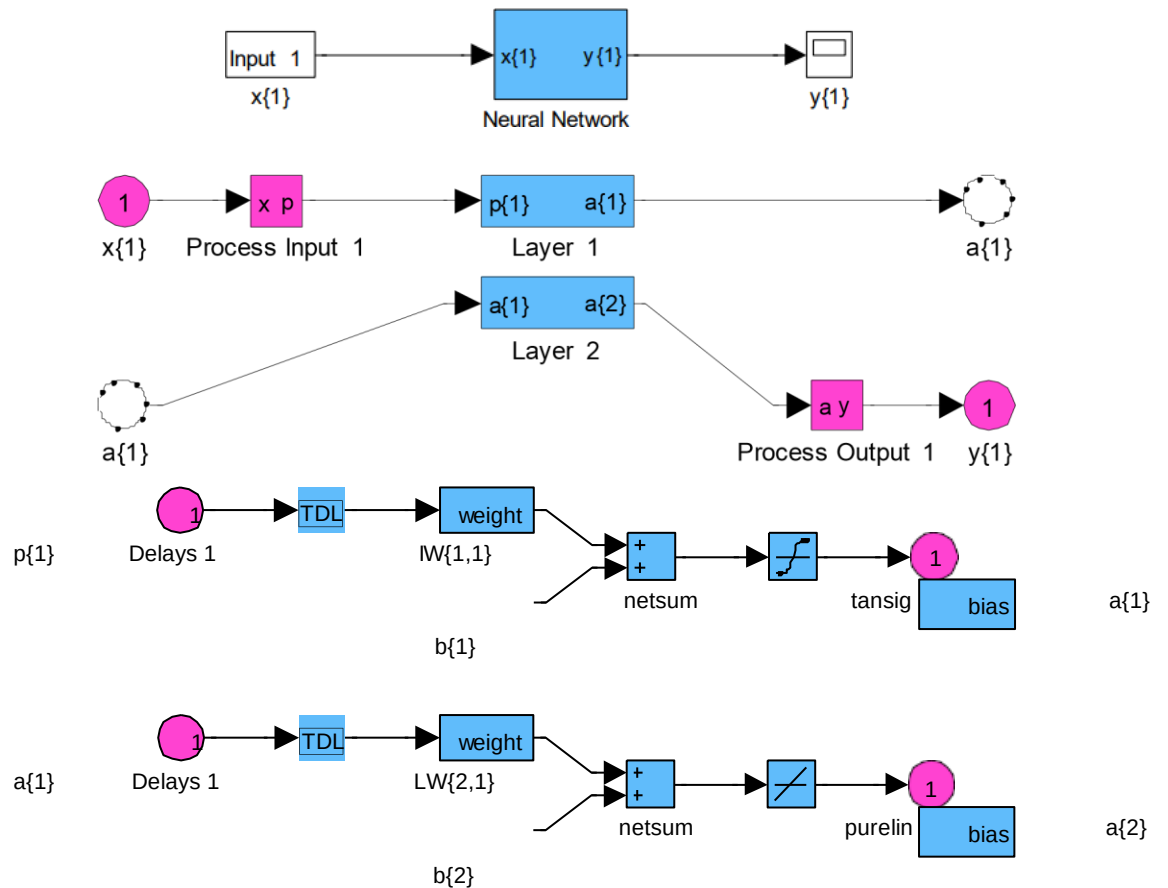


Рисунок 3.2. Структурная схема нейронной сети

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
```

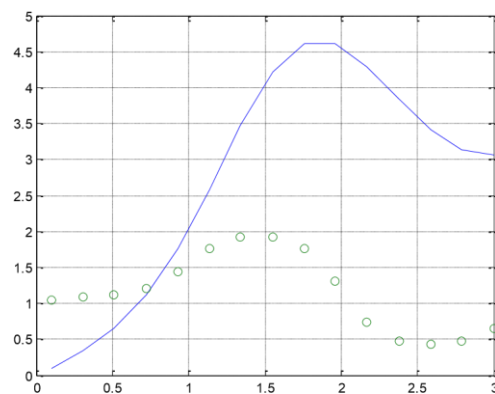


Рисунок 3.3. Графики сигналов выхода и цели

Обучим сеть в течение 50 циклов:

```
net.trainParam.epochs = 50;
```

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

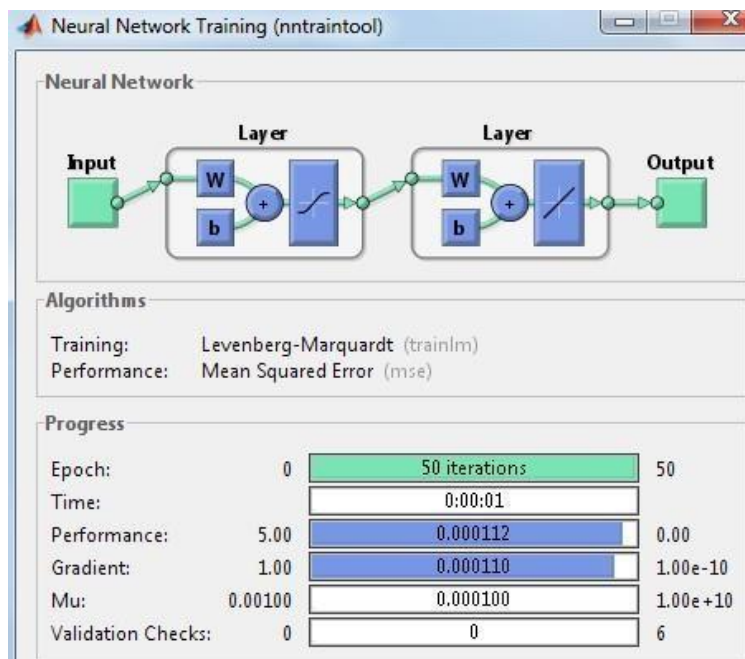


Рисунок 3.4. Окно обучения нейронной сети

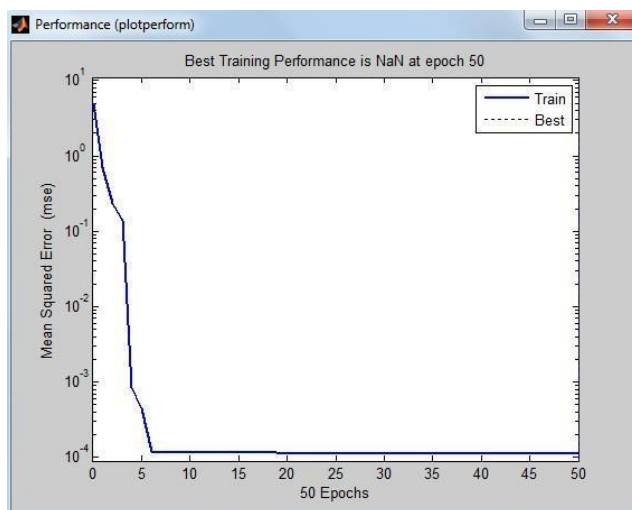


Рисунок 3.5. Изменение ошибки сети в процессе обучения

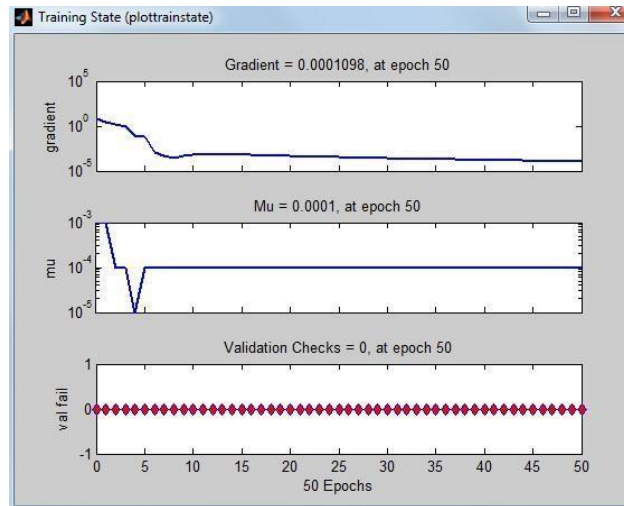


Рисунок 3.6 – Окно состояния обучения

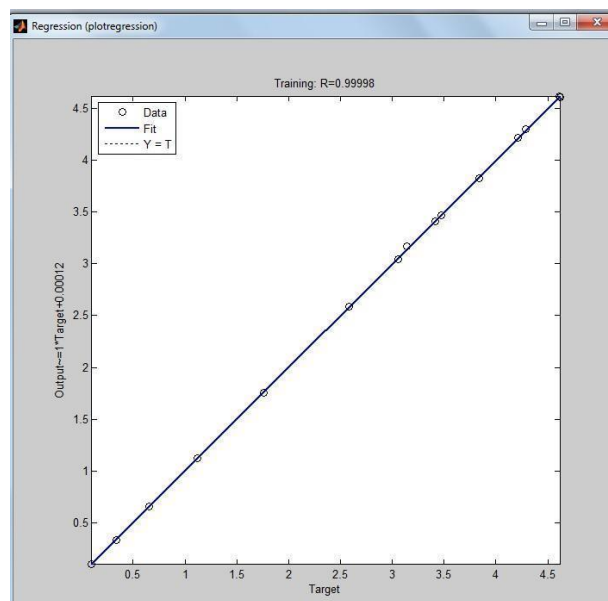


Рисунок 3.7 – Окно линейной регрессии между выходом НС и целями
 Выполним моделирование сформированной двухслойной сети,
 используя обучающую последовательность входа

$Y = \text{sim}(\text{net}, P);$

$\text{plot}(P, T, P, Y, 'o'), \text{grid on}$

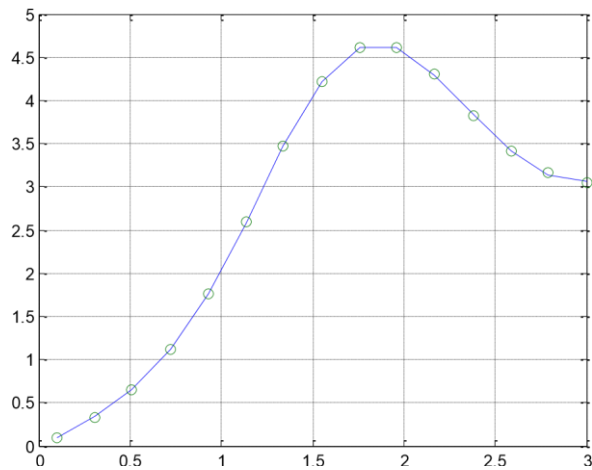


Рисунок 3.8. Результат аппроксимации векторов двухслойным персептроном

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);

L_r – коэффициент обучаемости.

Пусть входной сигнал $P = 1.0$, а желаемый выход $t = 1 + \sin\left(p * \frac{\pi}{4}\right) = 1.707$ (рисунок 3.9):

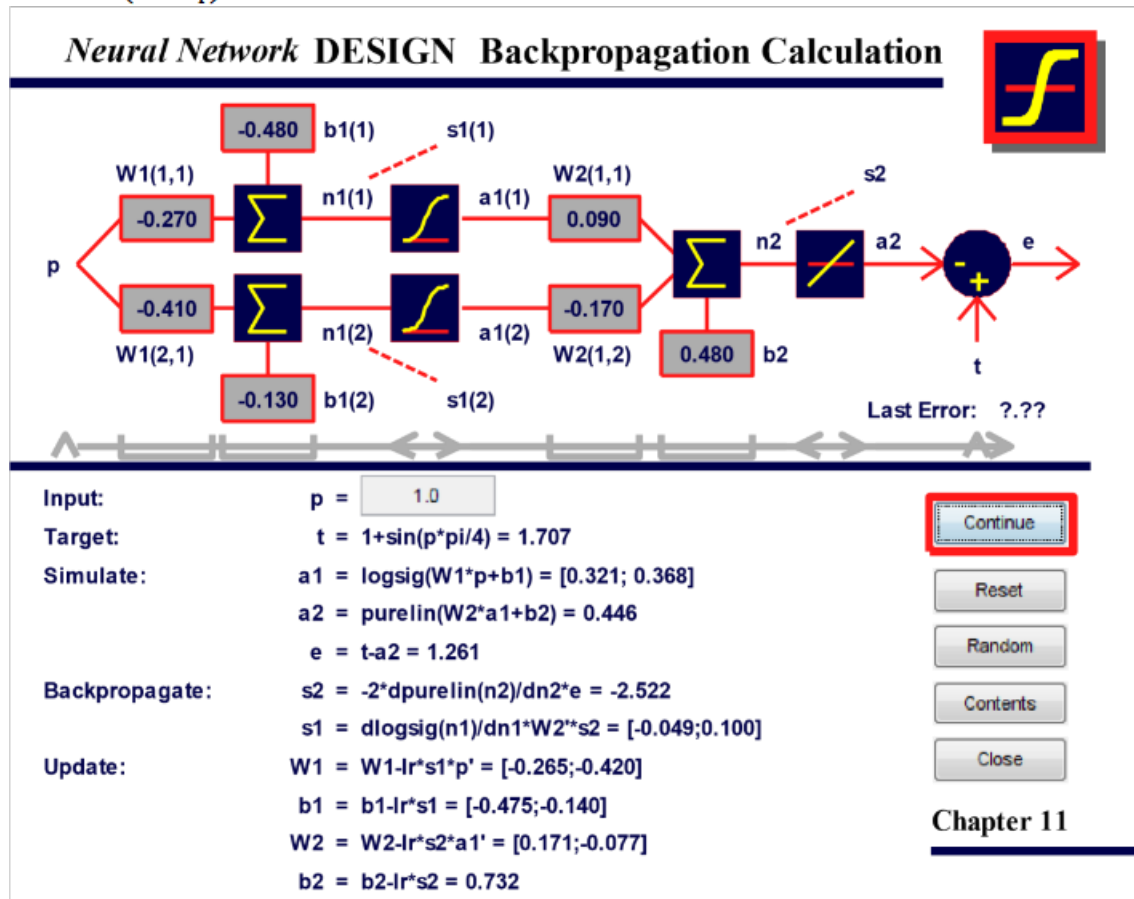


Рисунок 3.9. Демо пример работы алгоритма *Backpropagation*

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 [54].

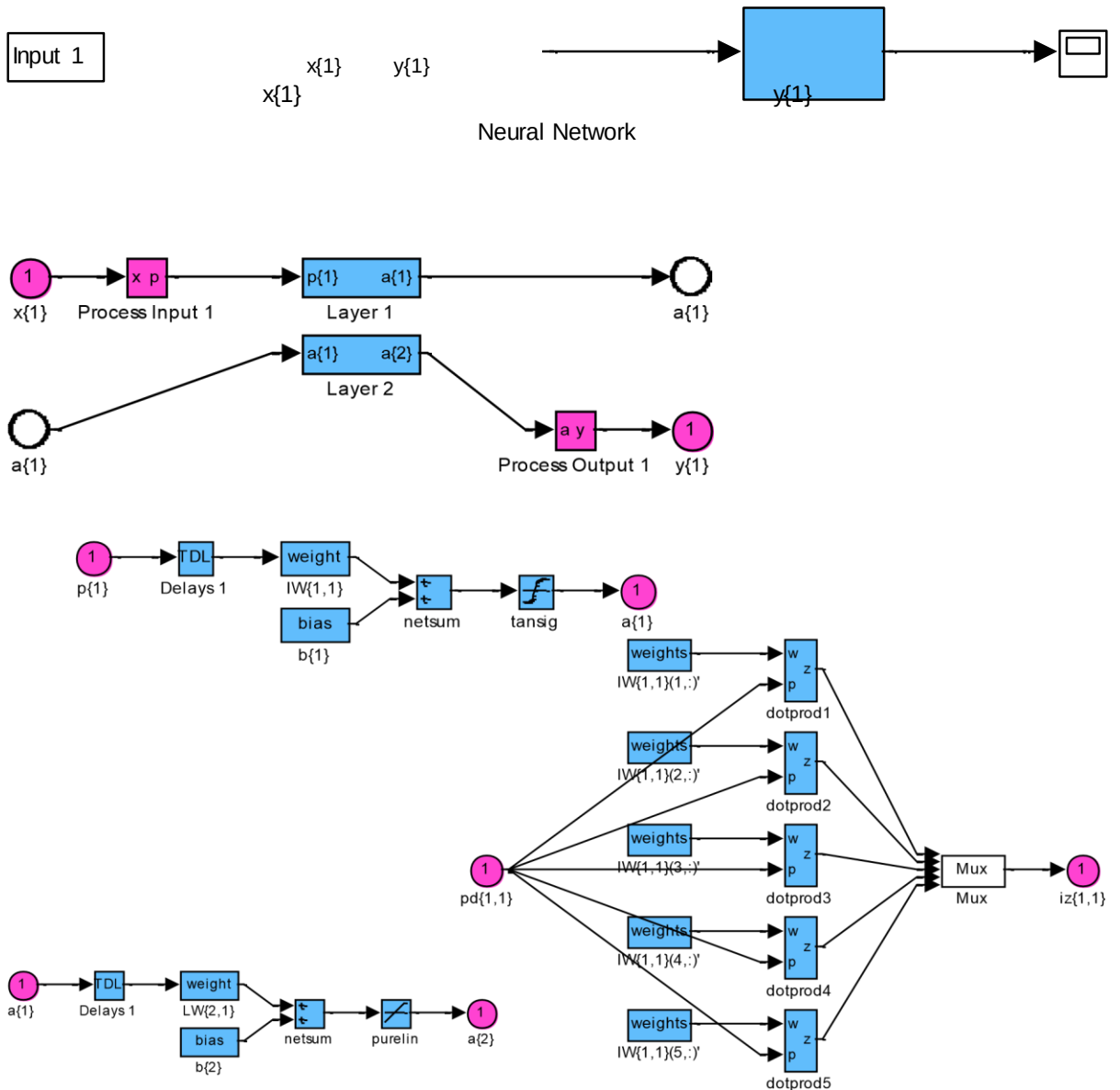
При такой формализации задачи обучения предполагаются известными

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.



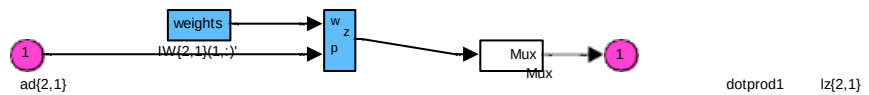


Рисунок 3.10. Структурная схема модели нейронной сети

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

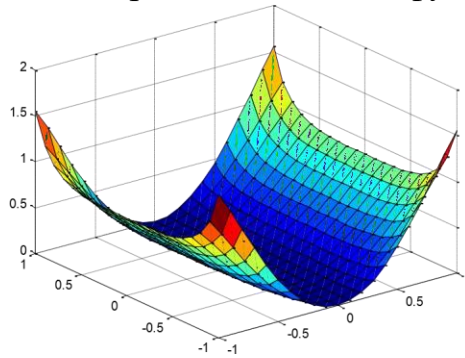


Рисунок 3.11. График поверхности целевой функции

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

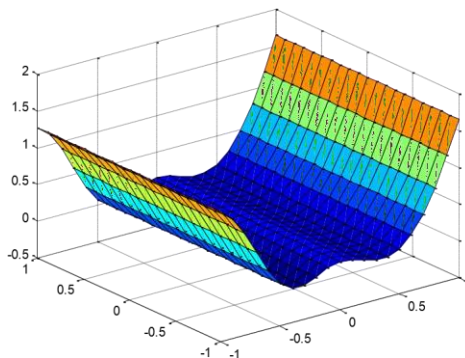


Рисунок 3.12. График поверхности функции аппроксимируемой нейронной сетью

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 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.

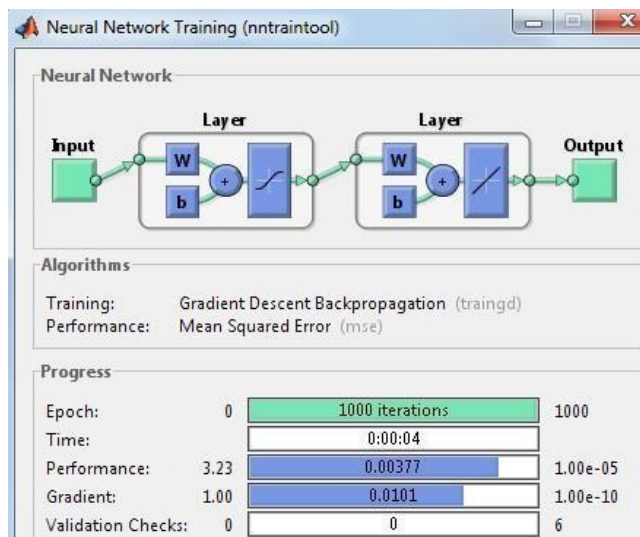


Рисунок 3.13. Окно обучения нейронной сети

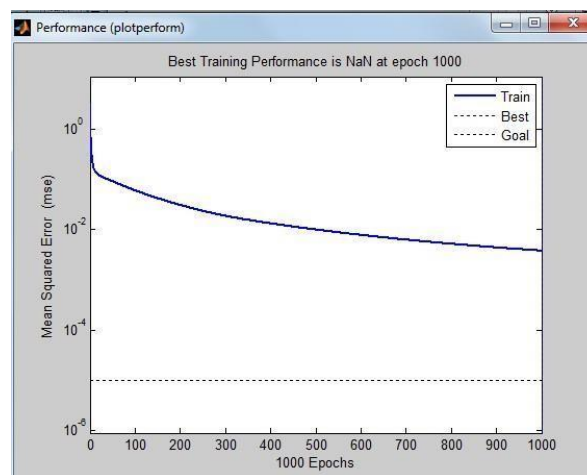


Рисунок 3.14. График изменения ошибки в зависимости от числа выполненных циклов обучения

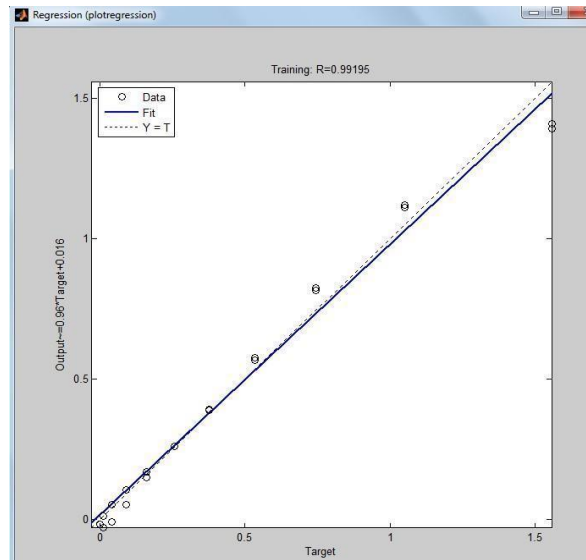


Рисунок 3.15. Окно линейной регрессии между выходом НС и целями

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

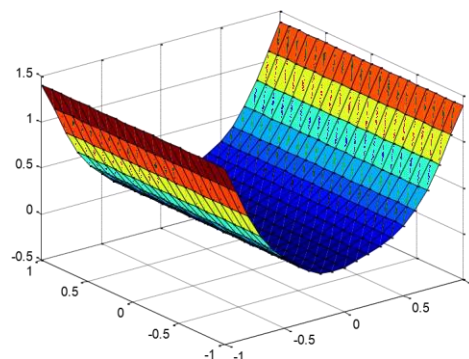


Рисунок 3.16. График поверхности функции аппроксимируемой нейронной сетью

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:

$$\Delta w_k = mc \Delta w_{k-1} + lr g_k$$

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='learnngdm'; % задание имени функции
% настройки
net.biases{1,1}.learnFcn, соответствующее
% алгоритму градиентного спуска с
% возмущением – M-функция learnngdm
net.biases{2,1}.learnFcn='
```

```
learnngdm';
```

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

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

```
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
```

=	c
n	o
u	s
m	(
2	y
c	.
e	^
l	2
l	)
(	;
p	t=num2cell(t,1);
,	
1	
)	
;	
t	
=	
s	
i	
n	
(	
x	
.	
^	
2	
)	
.	
/	

(3.1)

```
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
```

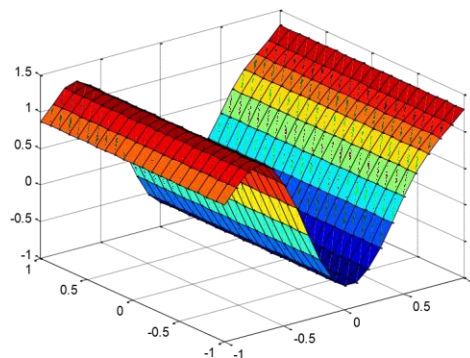


Рисунок 3.17. График поверхности функции аппроксимируемой нейронной сетью

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 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 $J_N \geq 1.04J_{N-1}$, то параметр возмущения `ms` следует установить в 0.

```
net = newff([-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.

```

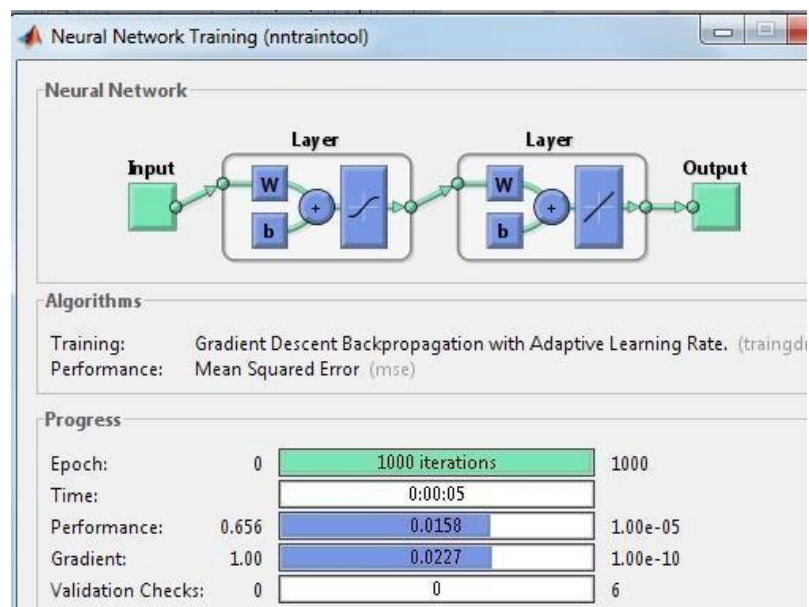


Рисунок 3.18. Окно обучения нейронной сети

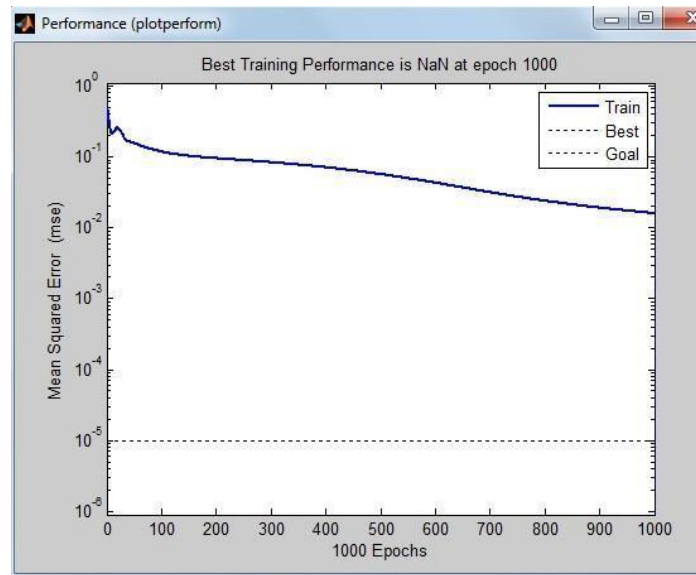


Рисунок 3.19. График изменения ошибки обучения в зависимости от числа выполненных циклов

обучения

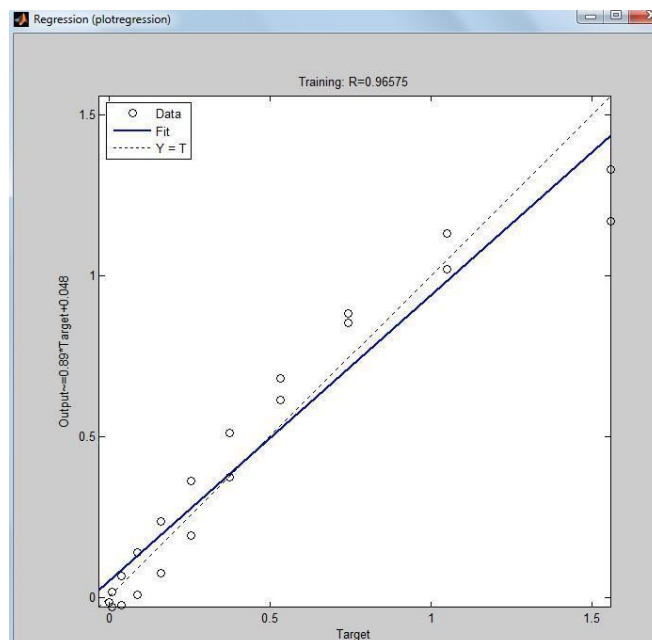


Рисунок 3.20. Окно линейной регрессии между выходом НС и целями

a=s

im(net,p);

e=t-a;

mse(e)

ans = 0.0158

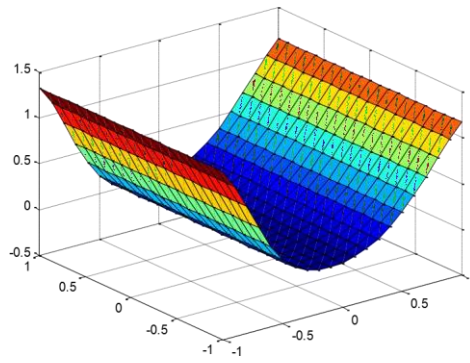


Рисунок 3.21. График поверхности функции аппроксимируемой нейронной сетью

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.

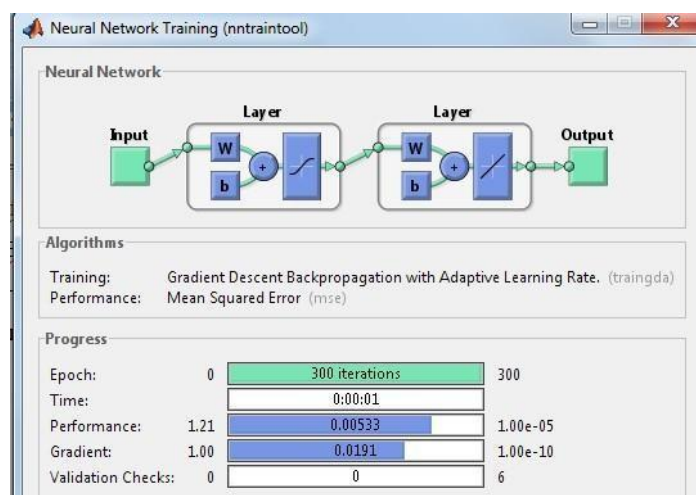


Рисунок 3.22. Окно обучения нейронной сети

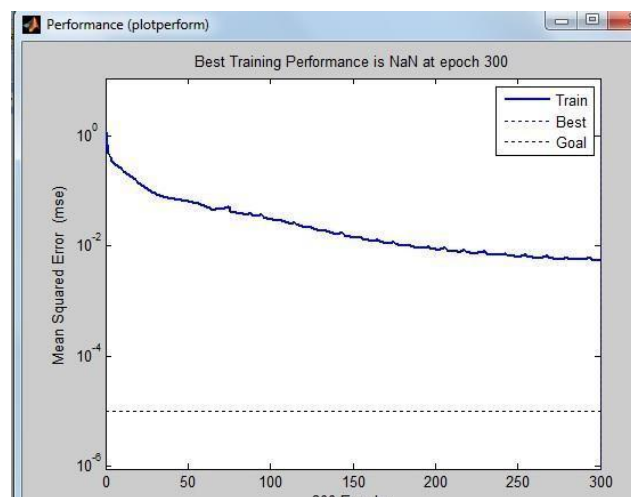


Рисунок 3.23. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

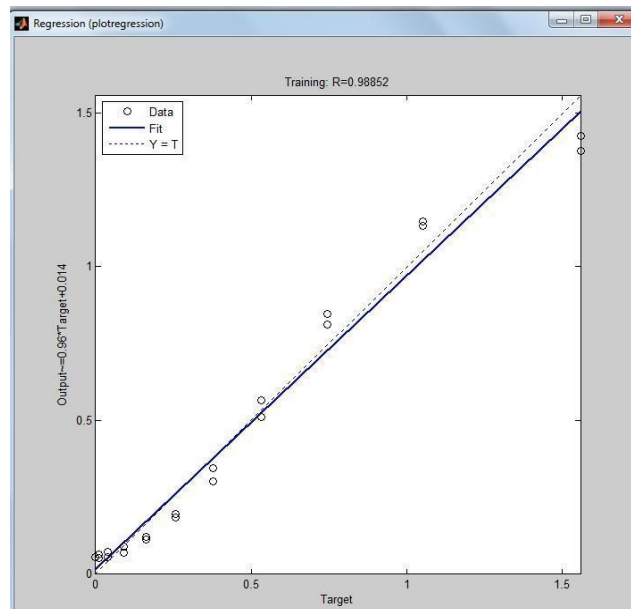


Рисунок 3.24. Окно линейной регрессии между выходом НС и целями

$a=s$

$\text{im}(\text{net}, p);$

$e=t-a;$

$\text{mse}(e)$

0.0053

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

Демонстрационная программа `nnd12v1` 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 `trainp`:

10;

300;

5;

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

следующие значения этих
параметров:

```
net.trainParam.show =
```

```
net.trainParam.epochs =
```

```
net.trainParam.goal = 1e-
```

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

```
y
```

```
=
```

```
x
```

```
;
```

```
p
```

```
=
```

```
[
```

```
x
```

```
;
```

```
y
```

```
]
```

```
;
```

```
t=sin(x
```

```
.^2)./co
```

```
s(y.^2)
```

```
; tic,
```

```
net=tr
```

`ain(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.

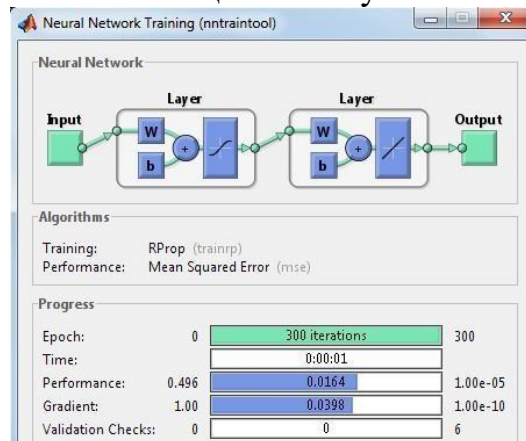


Рисунок 3.26. Окно обучения нейронной сети

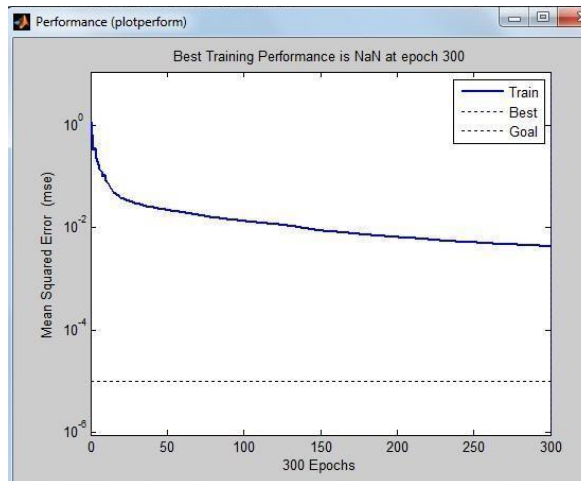


Рисунок 3.26. График изменения ошибки обучения в зависимости от числа выполненных циклов обучения

a=s

im(net,p);

e=t-a;

mse(e)

0.0043

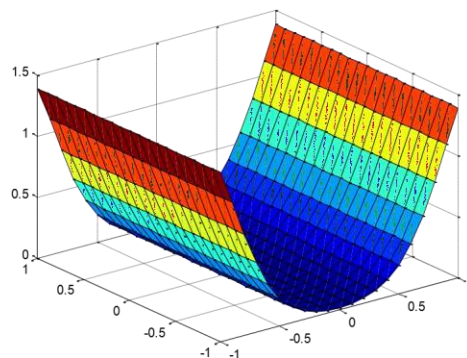


Рисунок 3.27. График поверхности функции аппроксимируемой нейронной сетью

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 обучают 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.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 6

Применение нейронных сетей для распознавания образов

Цель: изучить интерфейс пакета Neural Pattern Recognition Tool.

Порядок и последовательность выполнения работы

1. Постановка задачи распознавания образов
2. Интерфейс пакета распознавания образов

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

1. Постановка задачи распознавания образов

Задача распознавания образов состоит в отнесении входного набора данных, представляющего распознаваемый объект, к одному из заранее известных классов. Это может быть задача распознавания рукописных и печатных символов при оптическом вводе в ЭВМ, распознавание типов клеток крови, распознавание речи и пр.

Распознавание образов - это отнесение исходных данных к определенному классу с помощью выделения существенных признаков, характеризующих эти данные, из общей массы несущественных данных.

Постановка задачи распознавания образов. Дано множество объектов. Относительно них необходимо провести классификацию. Множество представлено подмножествами, которые называются классами. Заданы: информация о классах, описание всего множества и описание информации об объекте, принадлежность которого к определенному классу неизвестна. Требуется по имеющейся информации о классах и описании объекта установить - к какому классу относится этот объект.

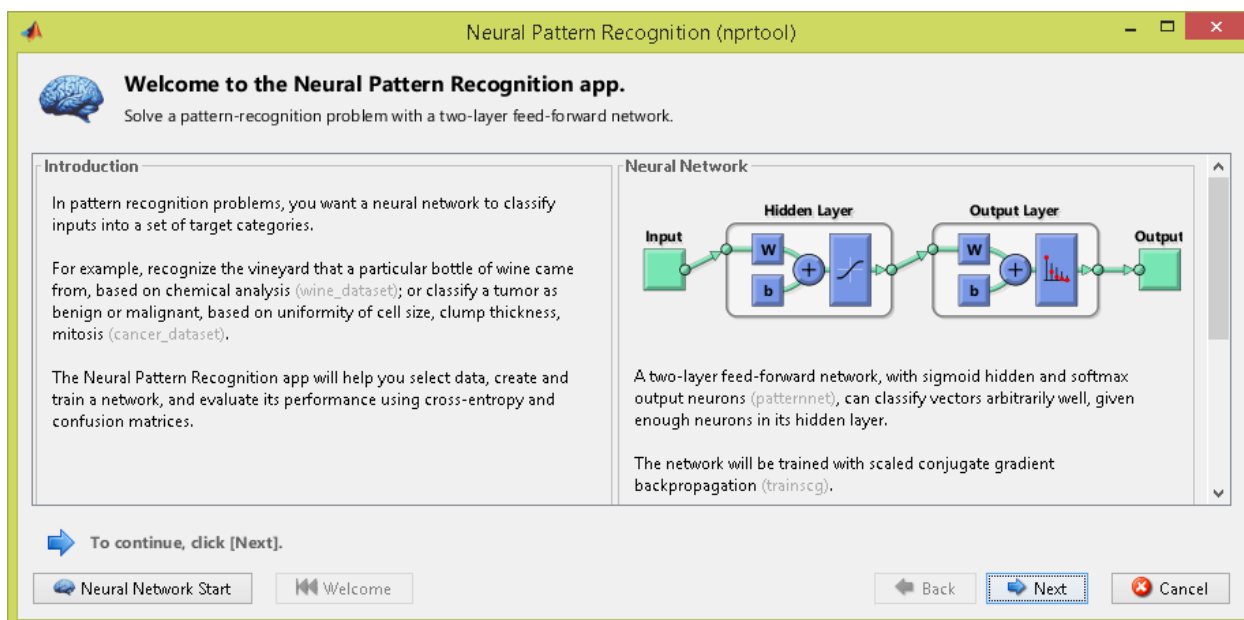
2. Интерфейс пакета распознавания образов

Распознавание образов является областью наиболее эффективного применения нейронных сетей. Рассмотрим инструментальные средства пакета Neural Pattern Recognition Tool пакета MATLAB, предназначенные для распознавания образов. Так и в случае приближения функций, рассмотрим визуальные средства распознавания образов и возможность генерирования созданных компьютерных моделей нейронных сетей распознавания образов в скрипты.

Для запуска мастера создания нейронной сети распознавания образов можно нажать кнопку окна выбора инструментальных средств нейронных сетей Pattern recognition app (рисунок 1.1) или в окне команд набрать:

```
>>nprtool,
```

после этого будет выведено окно Neural Pattern Recognition, показанное на рисунке 4.1.



Нажатие кнопки Next вызывает следующее окно мастера Select Data, посвященное заданию исходных данных (рисунок 4.2).

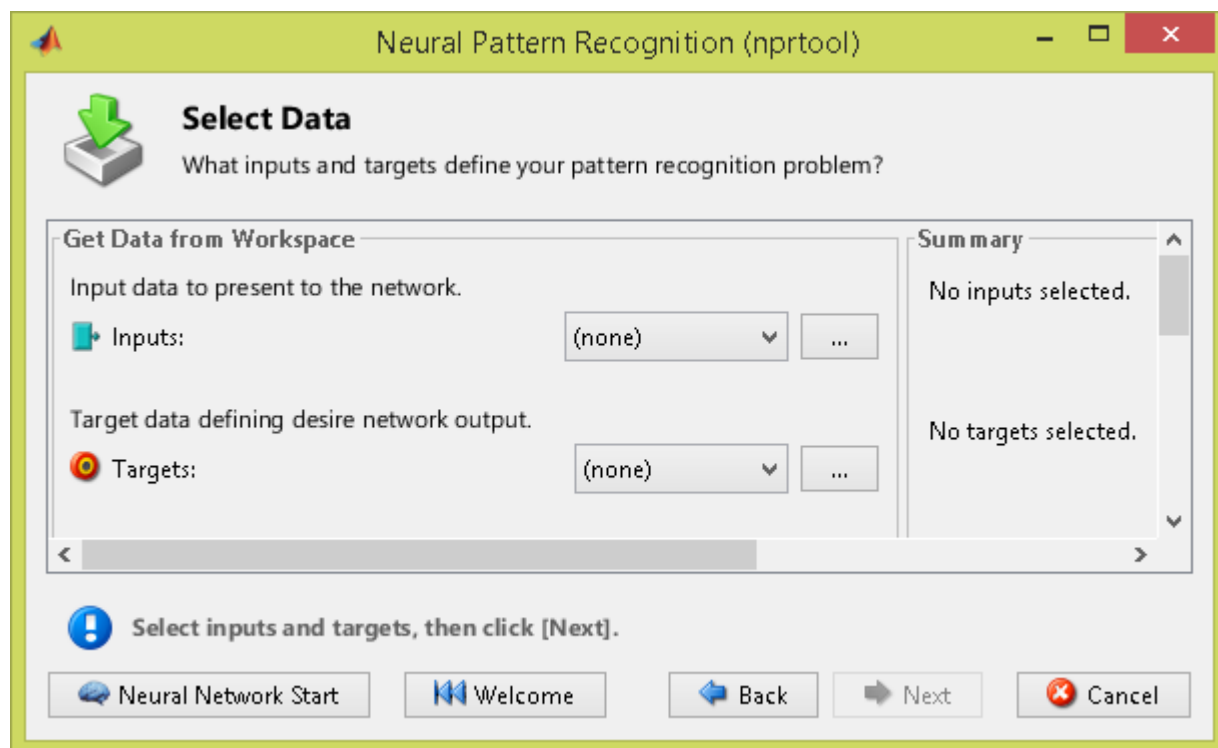
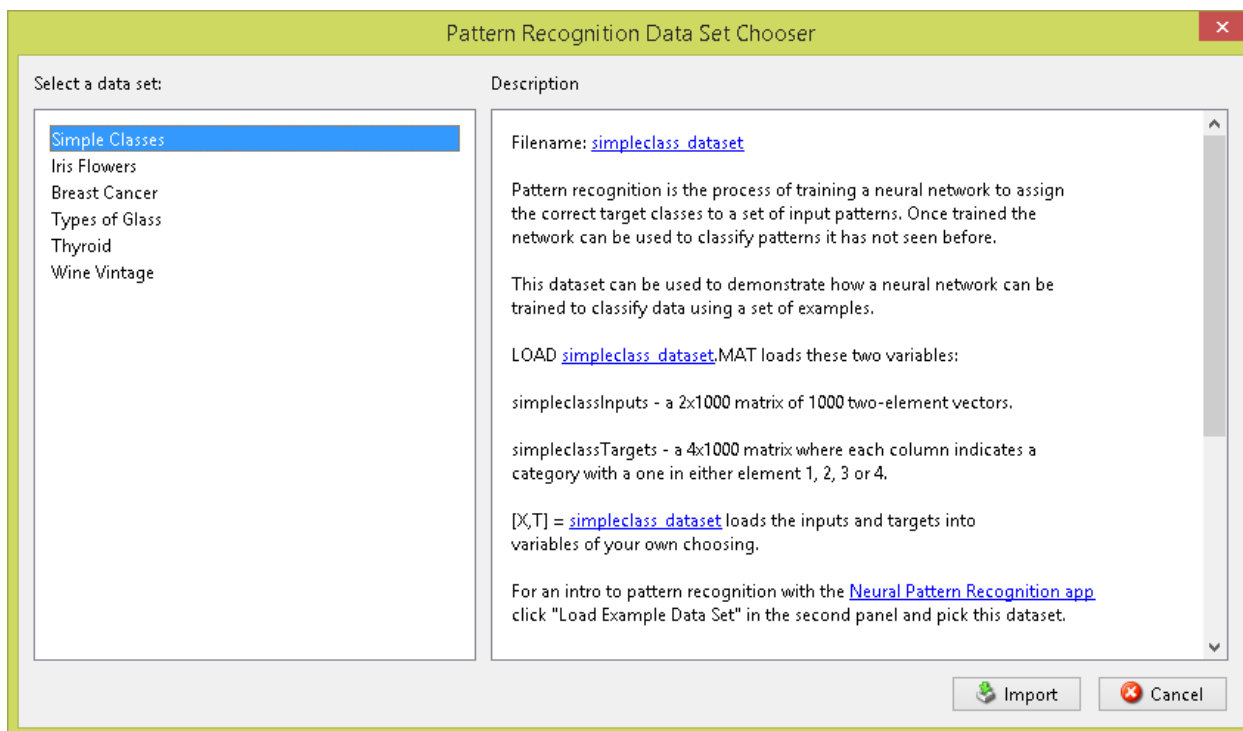


Рисунок 4.2 – Окно задания исходных данных мастера

Работу мастера создания нейронной сети распознавания образов

рассмотрим на демонстрационном примере. Для выбора одного из демонстрационных примеров необходимо нажать кнопку Load Example Data Set окна мастера выбора исходных данных. При этом будет выведено окно Pattern Recognition Data Set Chooser, показанное на рисунке 4.3, в левой половине которого выделен демонстрационный пример простых классов (Simple Classes).



Импорт исходных данных, распределение проверочных тестовых и данных обучающей выборки, задание параметров архитектуры нейронной сети обучение сети осуществляется по аналогии с работой мастера создания нейронной сети аппроксимации функции. После нажатия кнопки Plot Confusion выводиться результаты анализа функционирования сети с графиком правильности распознавания для данных обучающей выборки, проверочных, тестовых и обобщенных наборах (рисунок 4.4). По диагонали в зеленых клетках указан процент правильного распознавания для каждого образа, в красных – ошибочных, в синей, нижней правой клетке – общей процент правильного распознавания. Из графиков видно, что сеть обучена правильно.

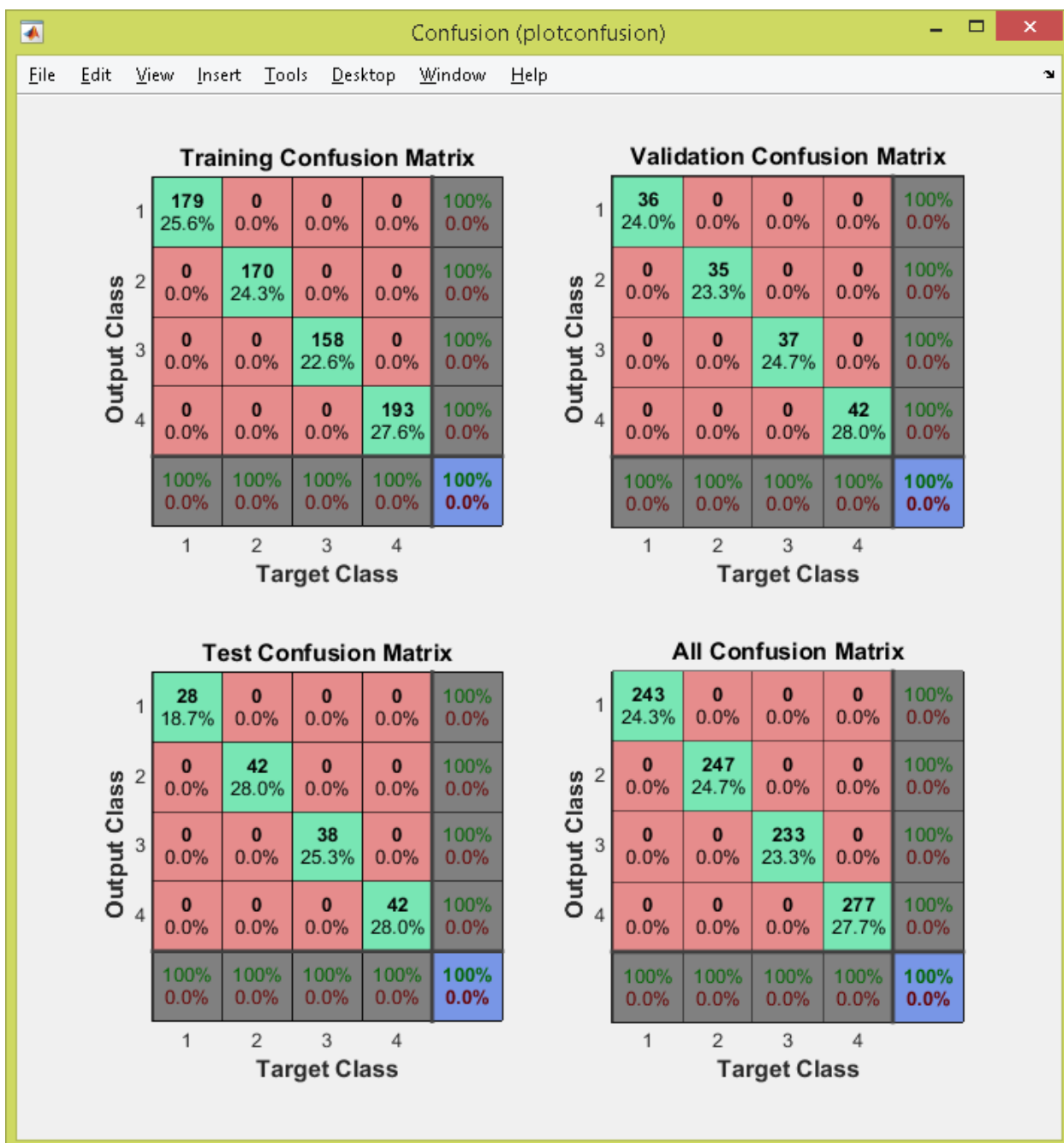


Рисунок 4.4 – График результатов обучения нейронной сети распознавания образов

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить компьютерную модель нейронной сети распознавания образов на основе демонстрационного примера, обучить ее и сохранить полученные результаты в различном виде. Просимулировать сеть исходными данными и сравнить с данными демонстрационных примеров. Рассчитать среднеквадратическую ошибку на всех образах обучающей выборки. Попробуйте изменить архитектуру сети. Сделать вывод о работе сети и качестве исходных данных обучающей выборки.

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Тема, программа и цель работы.
2. Задание для лабораторной работы.
3. Структура нейронной сети распознавания образов. Графики погрешности распознавания. Значение ошибки. Результаты симулирования сети.
4. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каком образом ставиться задача распознавания образов?
2. Какой пакет Matlab используется для моделирования нейронных сетей распознавания образов?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 7

Разработка нейросетевого алгоритма классификации цветов ириса

Цель:

1. Закрепить знания, полученные на лекции, посвященной нейросетевым методам распознавания образов.
2. Разработать нейросетевой алгоритм классификации цветов ириса на основе демонстрационного примера.

Порядок и последовательность выполнения работы

1. Создание компьютерной модели нейронной сети классификации цветов ириса.
2. Обучение нейронной сети и исследование ее работы.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

- 1. Создание компьютерной модели нейронной сети классификации цветов ириса.**

Рассматриваемый демонстрационный пример позволяет создать нейросетевой алгоритм классификации цветов ириса.

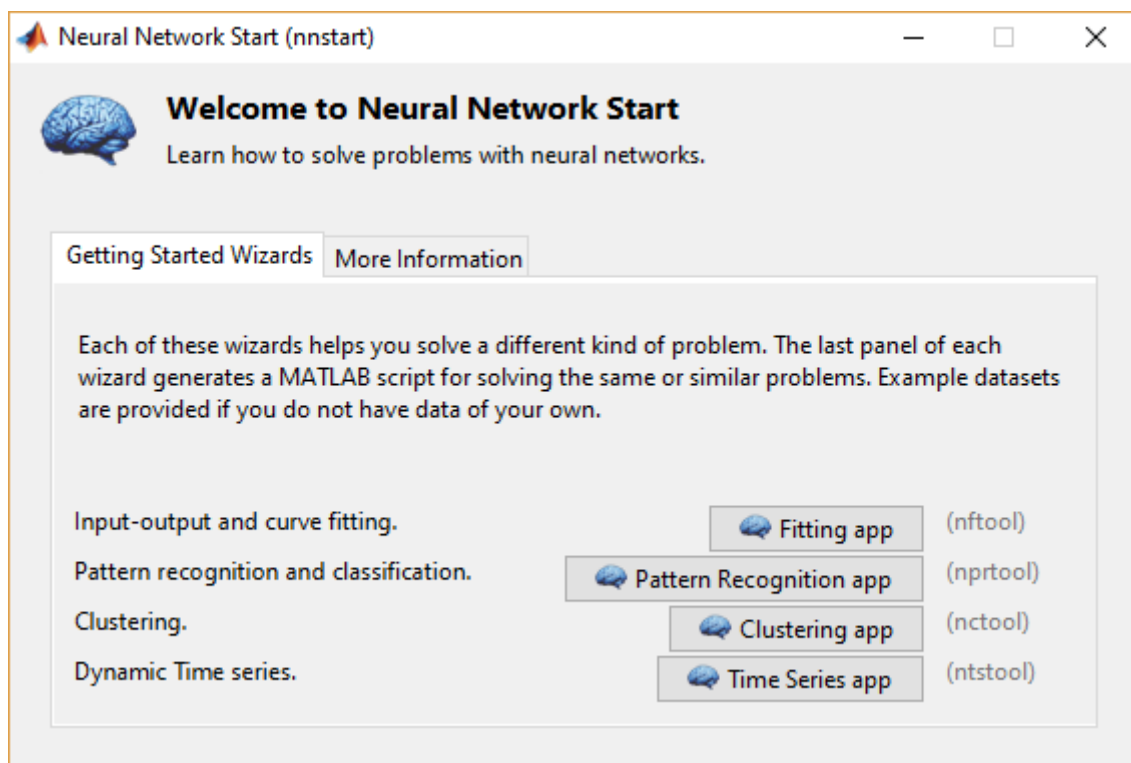
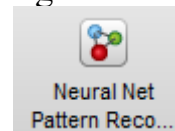


Рисунок 1 – Мастер запуска нейросетевых приложений

Для запуска приложения Neural Pattern Recognition необходимо на

вкладке APPS системного меню выбрать иконку



команд набрать

```
>> nprtool,
```

либо запустив окно запуска нейросетевых приложений (рисунок 1)

```
>> nnstart
```

выбрать Pattern Recognition app.

Для загрузки данных этого демонстрационного примера, на основе которого будет создан нейросетевой алгоритм классификации цветов ириса, необходимо выбрать демонстрационный пример (Load Example Data Set) Iris Flowers (рисунок 2) приложения Neural Pattern Recognition.

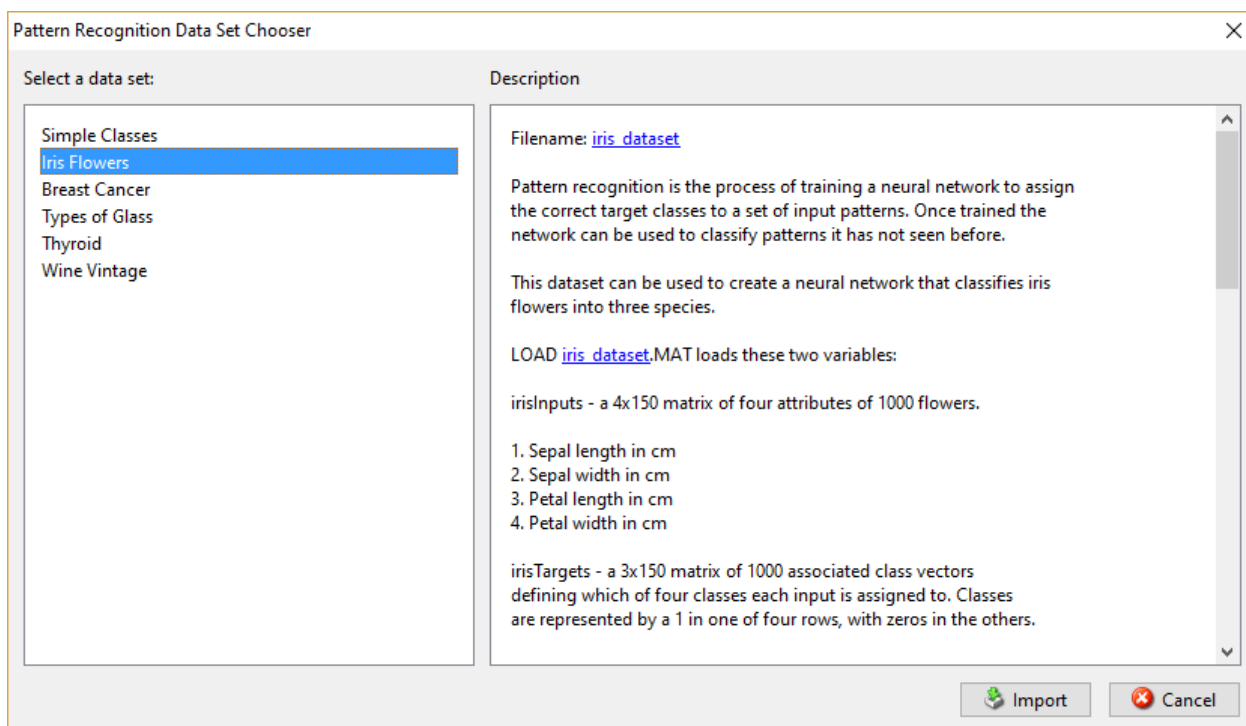


Рисунок 2 – Выбор демонстрационного примера

Исходные данные рассматриваемого демонстрационного примера содержит обучающую выборку из 150 образов, каждый из которых содержит четыре характеристики цветов ириса:

1. Длина чашелистиков (в см),
2. Ширина чашелистиков (в см),
3. Длина лепестка (в см),
4. Ширина лепестка (в см).

Описанные данные содержатся в матрице `irisInputs`. Фрагмент этой матрицы, содержащий первые 10 образов обучающей выборки приведен на рисунке 3.

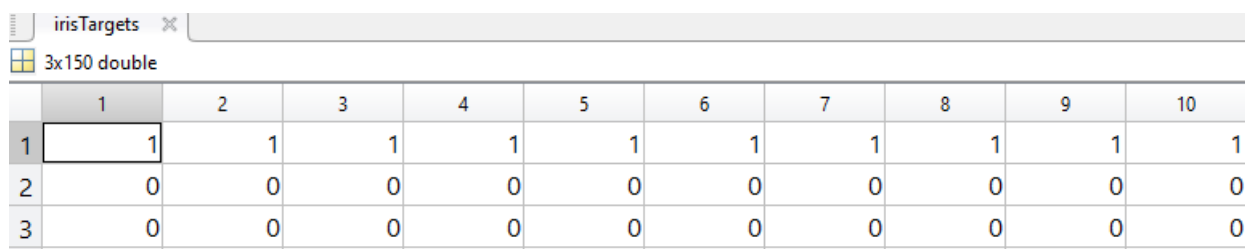
irisInputs										
4x150 double										
	1	2	3	4	5	6	7	8	9	10
1	5.1000	4.9000	4.7000	4.6000	5	5.4000	4.6000	5	4.4000	4.9000
2	3.5000	3	3.2000	3.1000	3.6000	3.9000	3.4000	3.4000	2.9000	3.1000
3	1.4000	1.4000	1.3000	1.5000	1.4000	1.7000	1.4000	1.5000	1.4000	1.5000
4	0.2000	0.2000	0.2000	0.2000	0.2000	0.4000	0.3000	0.2000	0.2000	0.1000

Рисунок 3 – Исходные данные

В качестве выходных данных обучающая выборка содержит цель – принадлежность цветка к одному из трех классов. Описанные данные содержатся в матрице `IrisTargets`. Матрица содержит три строки. Значения матрицы – «0» или «1», причем «1» обозначает принадлежность цветка к

классу, соответствующему номеру строки, в которой находится «1». Фрагмент

матрицы IrisTargets приведен на рисунке 4.



	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0

Рисунок 4 – Цель

Для создания компьютерной модели нейронной сети распознавания образов необходимо при помощи мастера определить разбиение обучающей выборки (рисунок 5), затем задать архитектуру сети, указав количество нейронов в скрытом слое, по умолчанию – 10 (рисунок 6).

Для обучения сети на следующем шаге работы мастера необходимо нажать кнопку Train (рисунок 7).

Результаты обучения сети отображаются на форме Neural Network Training, представленной на рисунке 8. Видно, что нейронная сеть обучена за 15 итераций.

На следующем шаге работы мастера можно улучшить сеть: переобчив её (Train Again), изменив архитектуру сети (Adjust Network Size) или загрузив данные другой обучающей выборки (Import Larger Data Set). Эти элементы управления находятся в левой части формы мастера. Кроме этого, имеется возможность тестирования работы сети на всех образах обучающей выборки. Для этого на правой стороне формы улучшения сети необходимо выбрать матрицы исходных данных и цели обучающей выборки, и нажать кнопку Test Network.

После того что будет активирована кнопка Plot Confusion, при нажатии на которую будет выведено окно с результатами тестирования сети (рисунок 10).

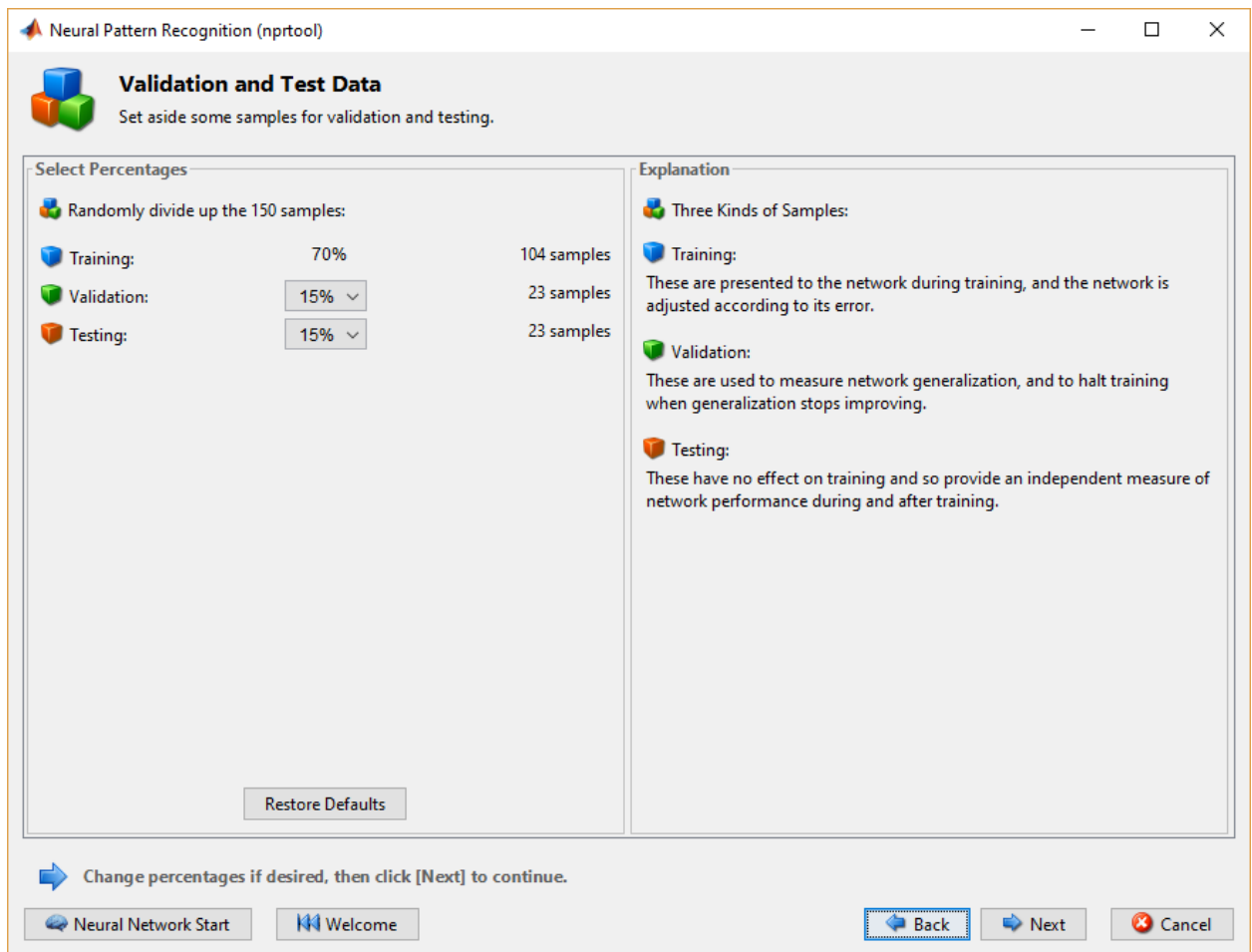


Рисунок 5 – Разбиение данных обучающей выборки

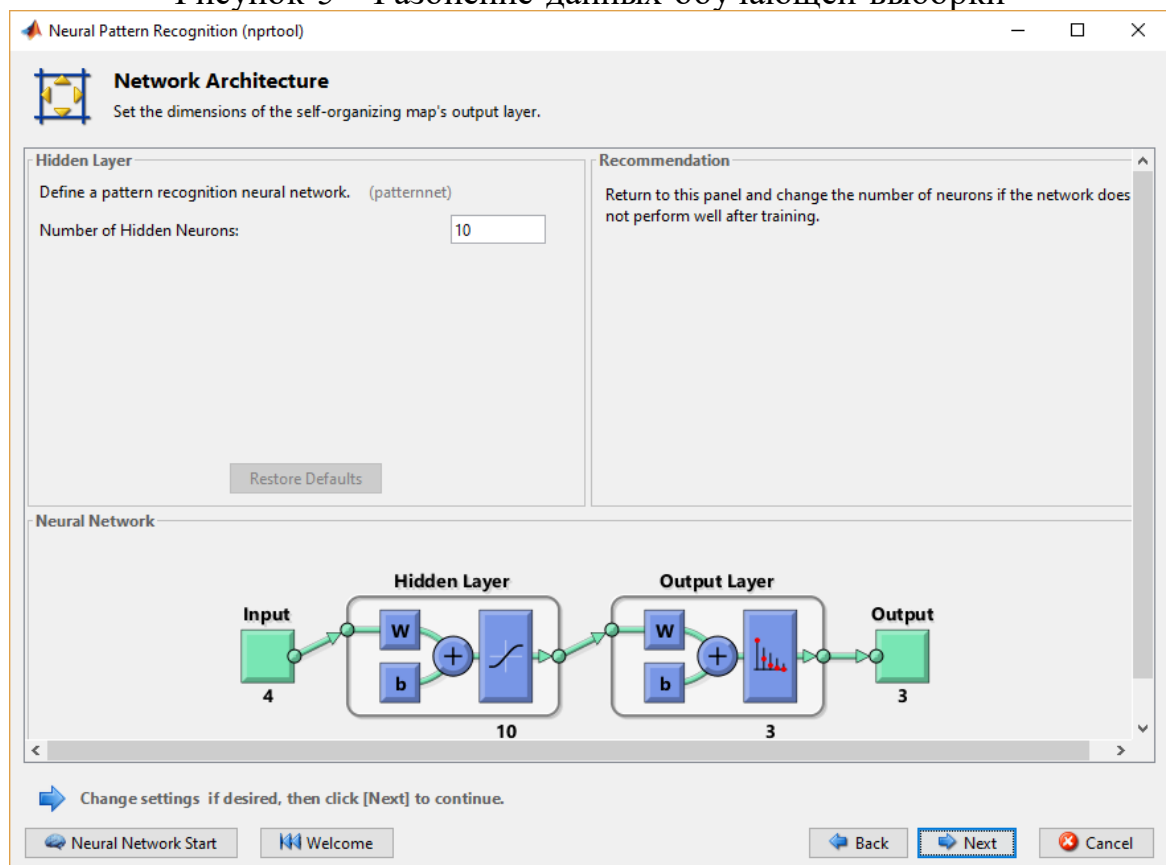


Рисунок 6 – Архитектура нейронной сети

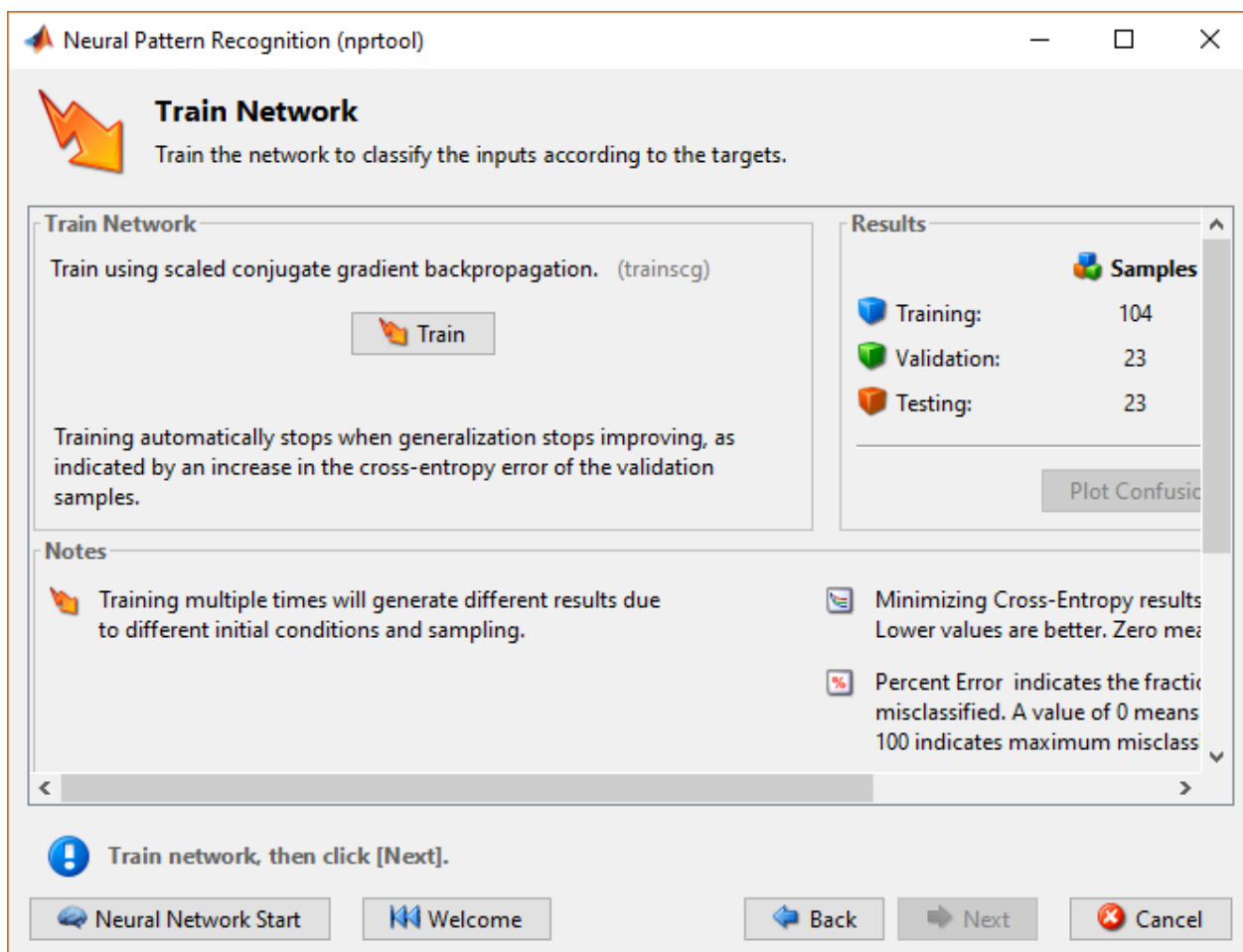


Рисунок 7 – Обучение нейронной сети распознавания образов

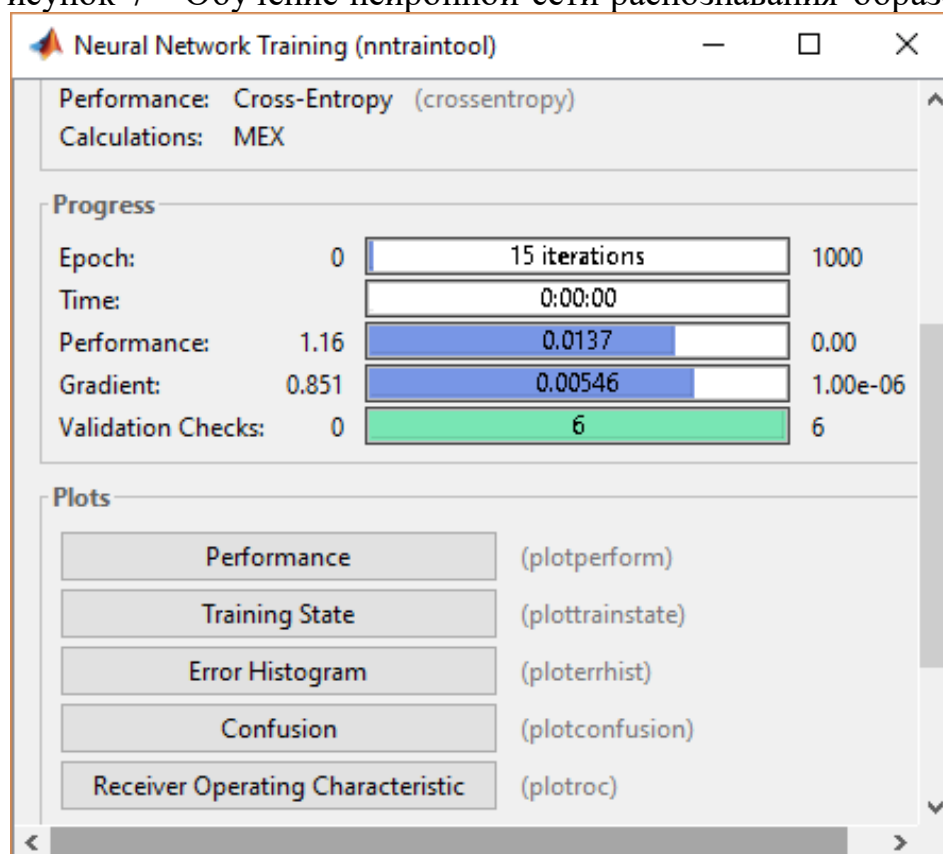


Рисунок 8 – Результаты обучения нейронной сети

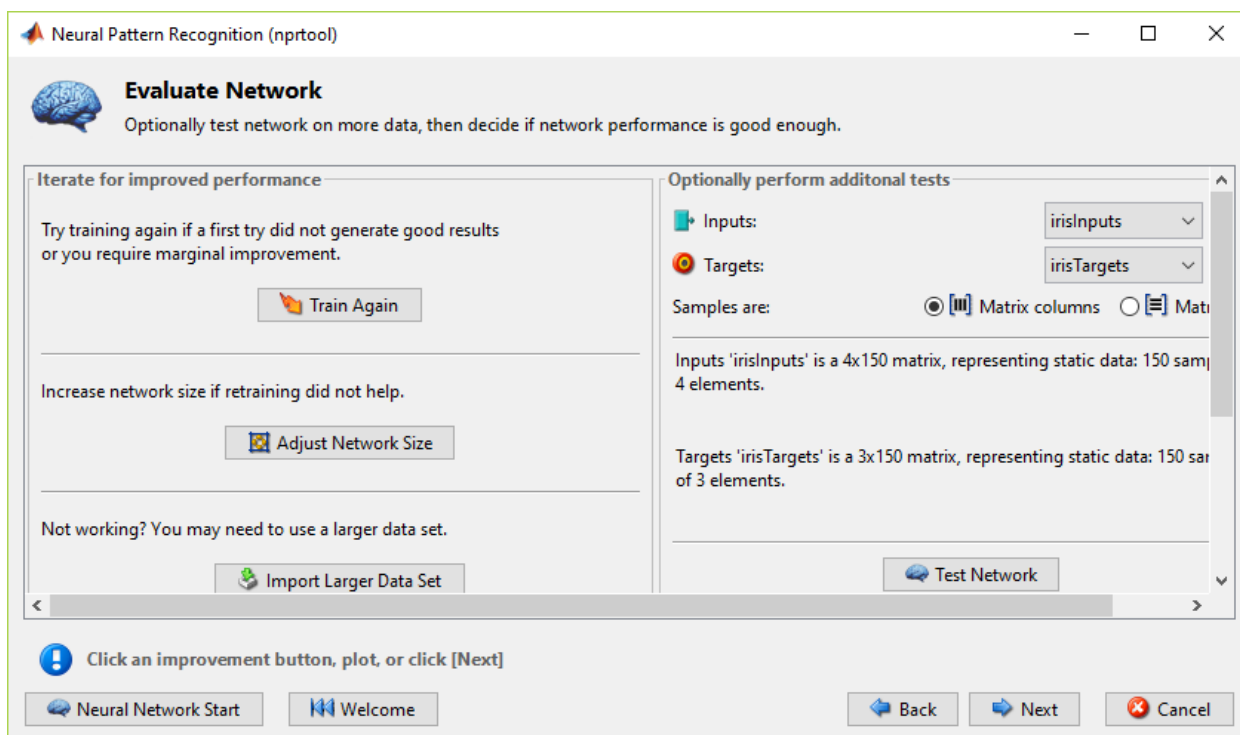


Рисунок 9 – Форма улучшения сети

Confusion Matrix				
Output Class	1	2	3	
	50 33.3%	0 0.0%	0 0.0%	100% 0.0%
	0 0.0%	49 32.7%	1 0.7%	98.0% 2.0%
	0 0.0%	1 0.7%	49 32.7%	98.0% 2.0%
	1	2	3	
Target Class				

Рисунок 10 – Результаты тестирования сети

Анализ полученных результатов показывает, что два образа обучающей выборки распознаны неправильно.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить компьютерную модель нейронной сети нейросетевой алгоритма классификации цветов ириса на основе демонстрационного примера, обучить ее, протестировать, и сохранить полученные результаты в различном виде. Сделать вывод о работе сети и качестве исходных данных обучающей выборки.

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы оформляется отчет, в котором

должны быть отражены:

1. Тема, программа и цель работы.
2. Задание для лабораторной работы.
3. Структура нейронной сети распознавания образов. Результаты тестирования сети.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каком образом ставиться задача распознавания образов?
2. Какой пакет Matlab используется для моделирования нейронных сетей распознавания образов?
3. Какая нейросетевая парадигма используется в задачах распознавания образов?
4. Какой алгоритм обучения нейронной сети используется для распознавания образов?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 8

Распознавание букв

Цель занятия:

1. Расширить и углубить знания, полученные на лекциях по основам теории нейронных сетей.
2. Приобрести первичные навыки по проведению исследований и экспериментов с использованием компьютерных моделей информационных систем.

Порядок проведения занятия

1. Создание компьютерной модели нейронной сети.
2. Обучение сети распознаванию образов.
3. Проверка эффективности функционирования сети.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

1. Создание компьютерной модели нейронной сети

Часто полезно иметь устройство, которое выполняет распознавание образов. В частности, очень эффективны и выгодны машины, которые могут читать символы. Машина, которая читает банковские чеки, может выполнять за то же самое время намного больше проверок, чем человек. Этот вид приложений сохраняет время и деньги, а также устраняет условия, при которых человек выполняет монотонную, периодически повторяющуюся работу. Демонстрационная программа сценарий `aprcr1` иллюстрирует, как распознавание символов может быть выполнено в сети с обратным распространением.

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 5x7. Например, символ А может быть представлен, как это показано на рисунок 1.

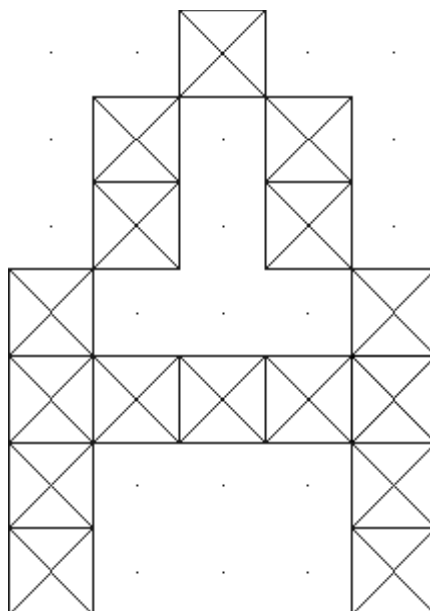


Рисунок 1. – Шаблон символа А

Однако система считывания символов обычно работает неидеально и отдельные элементы символов могут оказаться искаженными (рисунок 2).

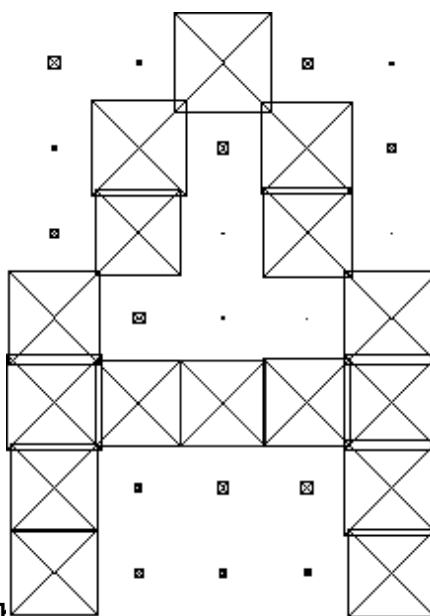


Рисунок 2. – Шаблон символа А

Проектируемая нейронная сеть должна точно распознавать идеальные

векторы вход и с максимальной точностью воспроизводить зашумленные векторы. М-функция `prprol` 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;`

```

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

```

Определим шаблон для символа А, который является первым элементом алфавита

```

[alphabet, targets] = prprob;
i = 2;
ti = alphabet(:, i);
letter{i} = reshape(ti, 5, 7)';
letter{i}

```

```

ans =
00100
01010
01010

10001

11111

10001

10001

```

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. Кроме 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 `compret`, 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 35×26 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)` и представлена на рисунок 3.

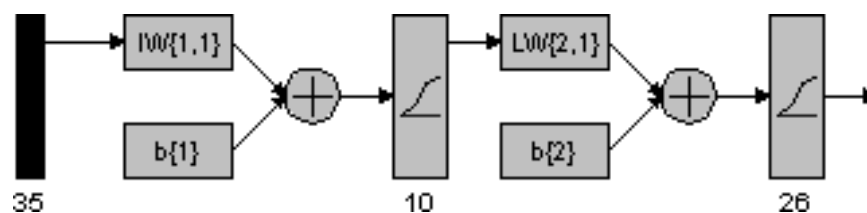


Рисунок 3 – Структура нейронной сети

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 `trainbrx`, 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 4);

```

P = alphabet;
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;

net.trainParam.epochs = 5000;

net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);

```

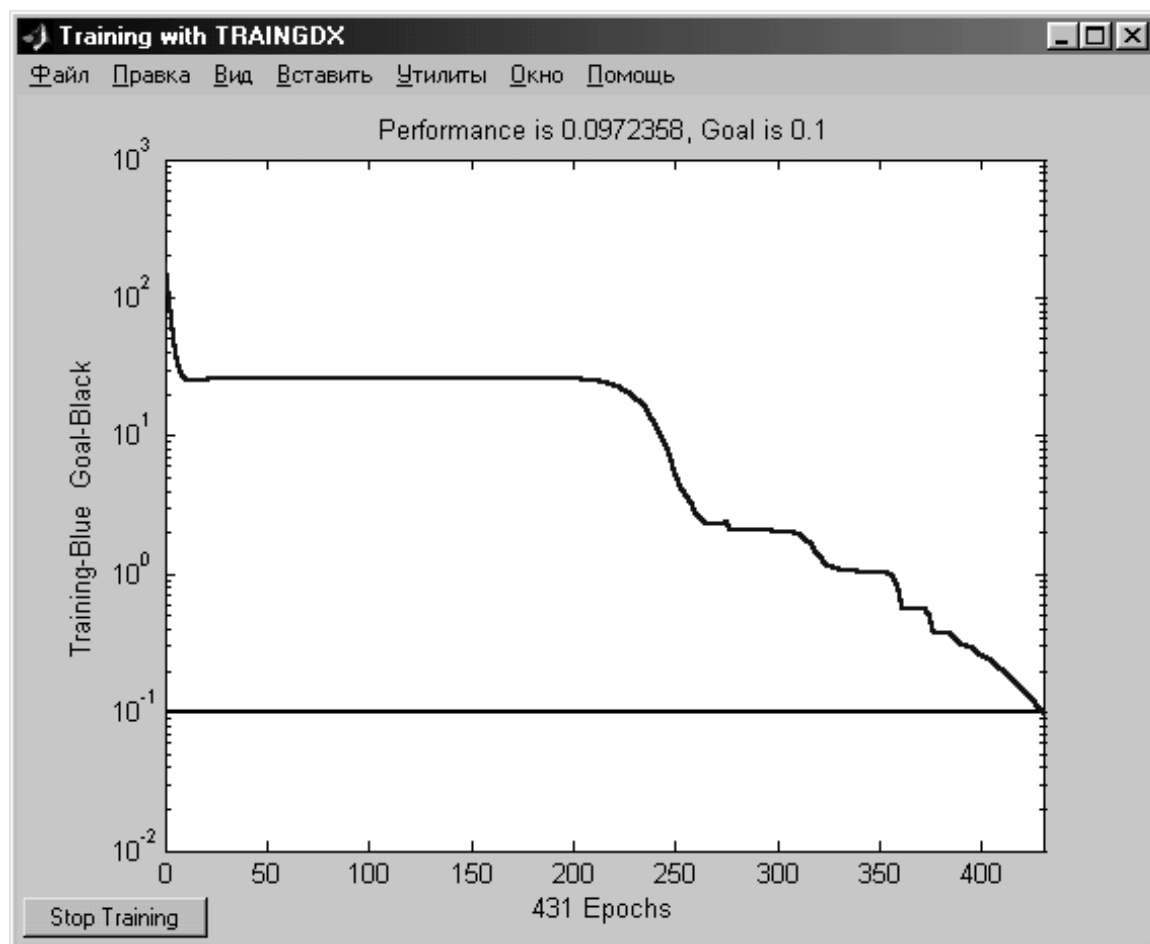


Рисунок 4 – Обучение нейронной сети в отсутствии шума

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 (рисунок 5):

```

netn      =      net;

```

```
netn.trainParam.goal = 0.6;
```

```

netn.trainParam.epochs = 300;
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

```

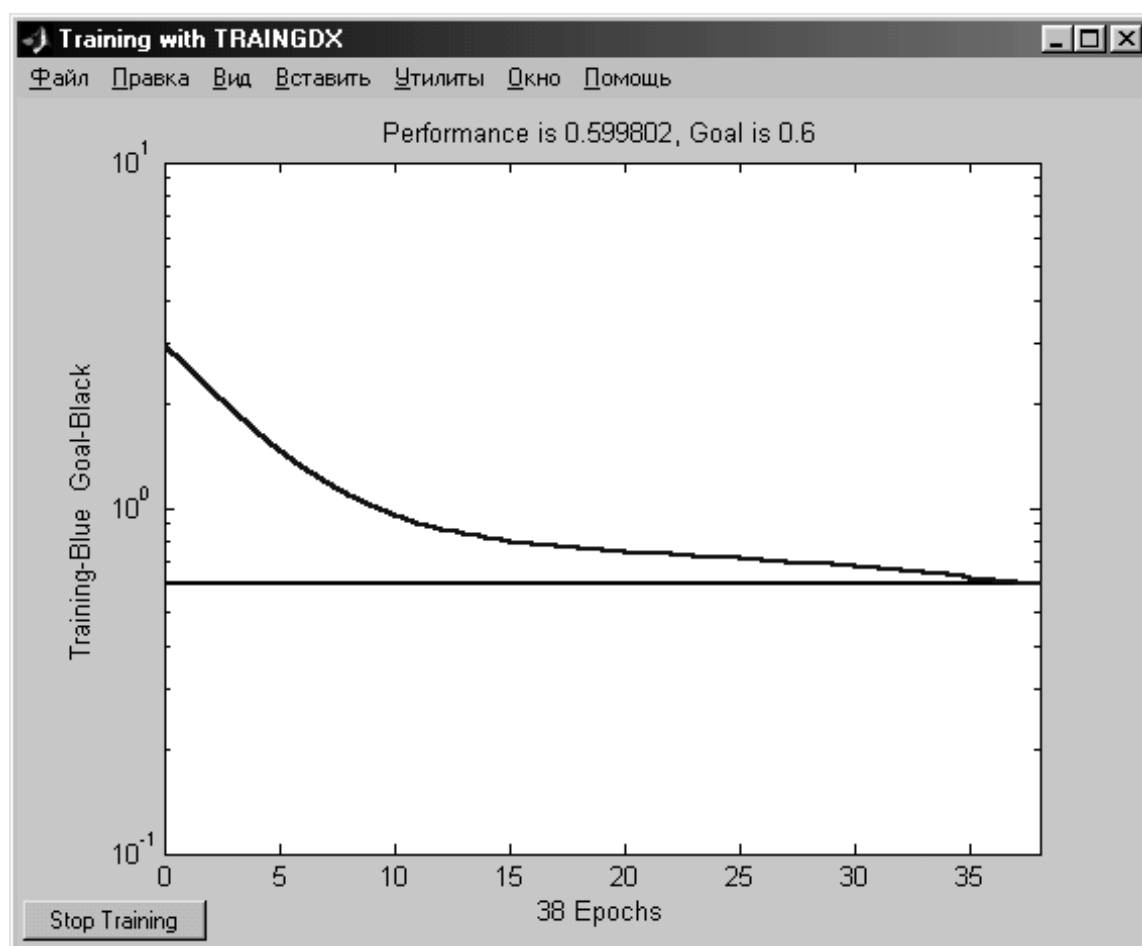


Рисунок 5 – Обучение сети с шумом

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;
netn.trainParam.epochs = 500;
net.trainParam.show    =5;
[netn, tr] = train(netn, P, T);

```

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
    fprintf('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];
    network2 = [network2 errors2/26/100];
end
plot(noise_range,network1*100,'--',noise_range,network2*100);
title('Процент ошибочных классификаций');
xlabel('Уровень шума');
ylabel('Сеть 1 - - Сеть 2 ---');
```

Тестирование 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 последовательностей для каждого символа вычисляется выход сети. Выходной сигнал обрабатывается М-функцией `compet` с той целью, чтобы выбрать только один из 26 элементов вектора выхода. После этого оценивался количество ошибочных классификаций и вычисляется процент ошибки. Соответствующий график погрешности сети от уровня входного шума показан на рисунке 6.

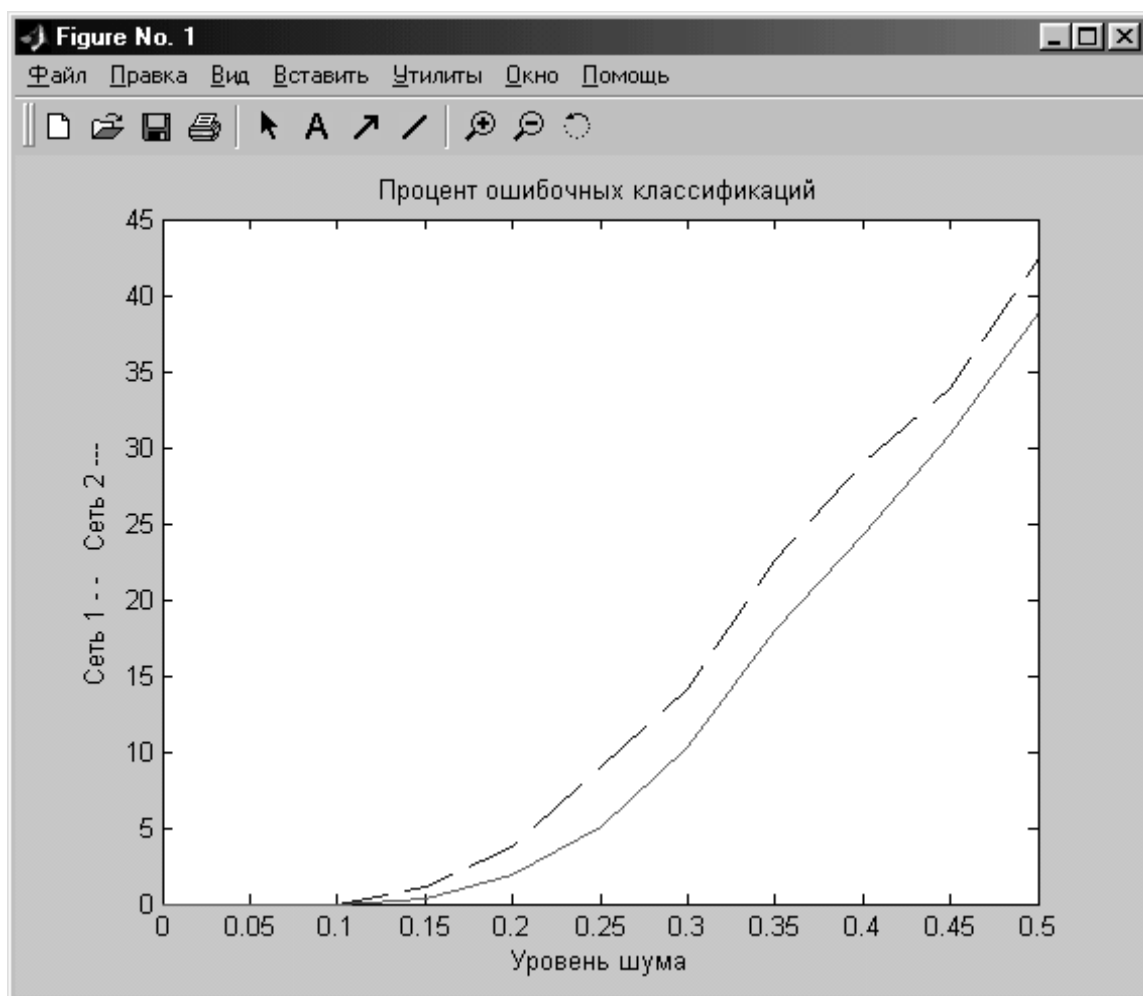


Рисунок 6 – График сравнения эффективности распознавания образов нейронными сетями, обученных с шумом и без

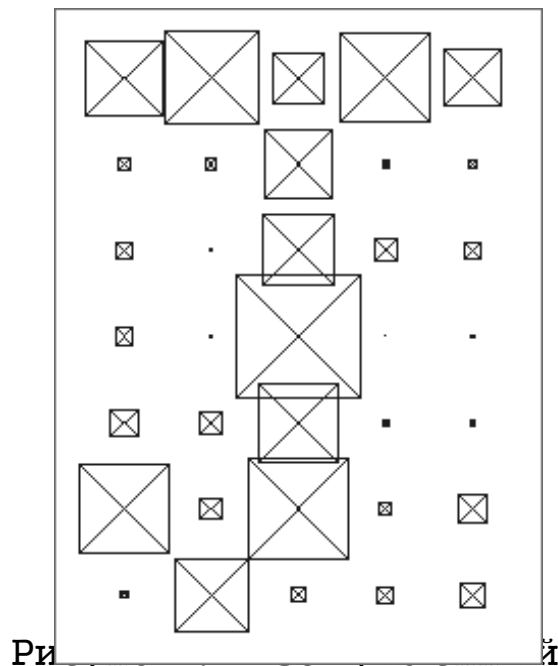
Сеть 1 обучена на идеальных векторах входа, а сеть 2 - на зашумленных. Обучение сети на зашумленных векторах входа значительно снижает погрешность распознавания реальных векторов входа. Сети имеют очень малые погрешности, если среднеквадратичное значение шума находится в пределах от 0.00 до 0.05. Когда 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 (рисунок 7):

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

```
plotchar(noisyJ); % Зашумленный символ J (рис. 7)
```



```
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 8).

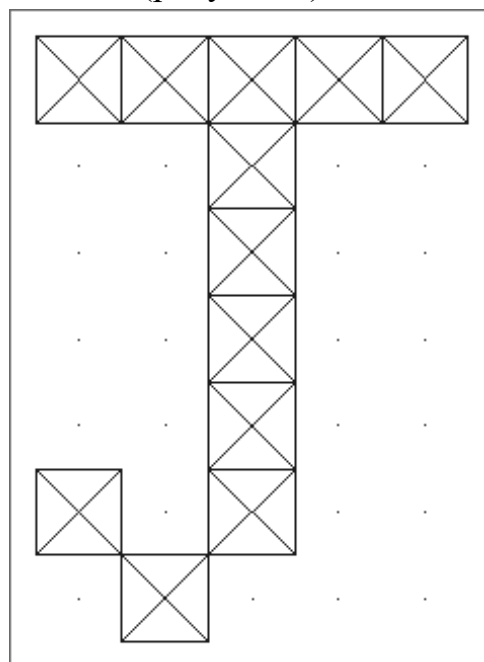


Рисунок 8 – Распознанный символ J

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, искаженными шумами, что характерно для реальной практики.

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

При помощи пакета Neural Network ToolBox создайте модель нейронной сети распознавания букв английского алфавита. Обучение нейронную сеть распознаванию образов в условиях шума и без него. Исследуйте правильность распознавания образов этими сетями при различных уровнях шума. Сравните эффективность распознавания букв этими сетями сделайте выводы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая модель нейронной сети создана на занятии?
2. Чем обусловлена разница эффективности распознавания сетей, обученных с шумом и без него?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 9

Использование нейронной сети для приближения функций

Порядок и последовательность выполнения работы

1. Создание компьютерной модели нейронной сети приближения функции.
2. Обучение и симулирование сети. Получение погрешности приближения.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

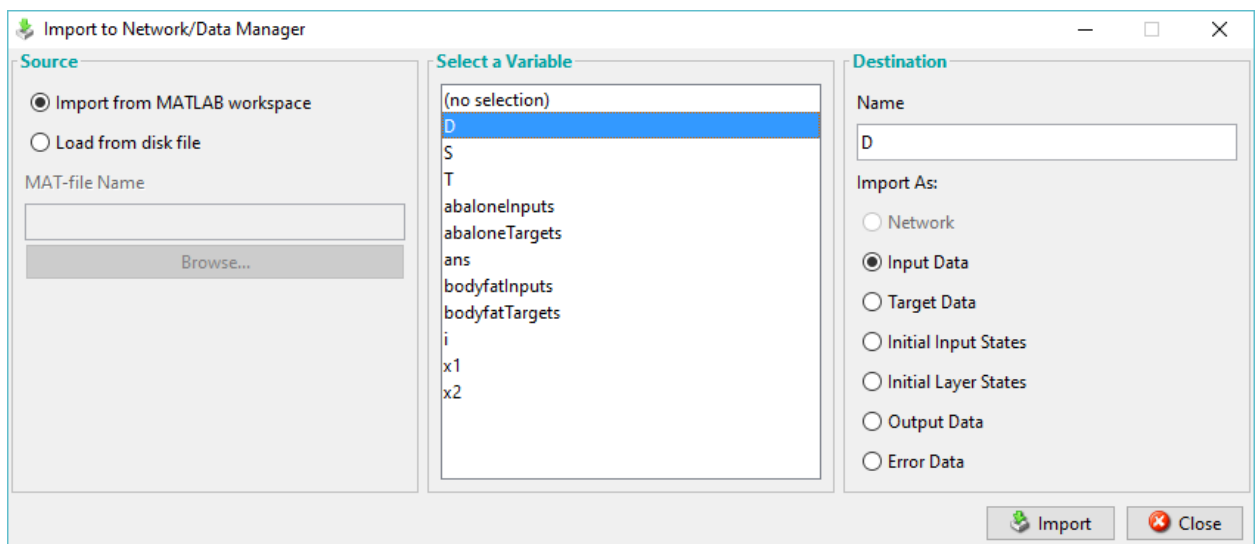
Пусть необходимо аппроксимировать полином вида

$$Y = x_1^{21} - x_1^2 + x_1 x_2$$

Для формирования вектора обучающих примеров в окне команд или в m-файле необходимо набрать следующие команды:

```
i=1;
for x1=-1:.1:1;
for x2=-1:.1:1;
D(1,i)=x1;
D(2,i)=x2;
T(i)=x1^2-x2^2+x1*x2;
i=i+1;
end;
end;
```

После чего в рабочей области будут сформированы матрица входных данных D и вектор целей T. Импортируем данные из рабочей области: вектор целей T и матрицу входных данных D (рисунок 1).



нейронных сетей 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 (например логистическая – LOGSIG), второго – линейная (PURELIN). На рисунке 2 представлено окно Network/Data Manager с выбором параметров компьютерной модели создаваемой нейронной сети.

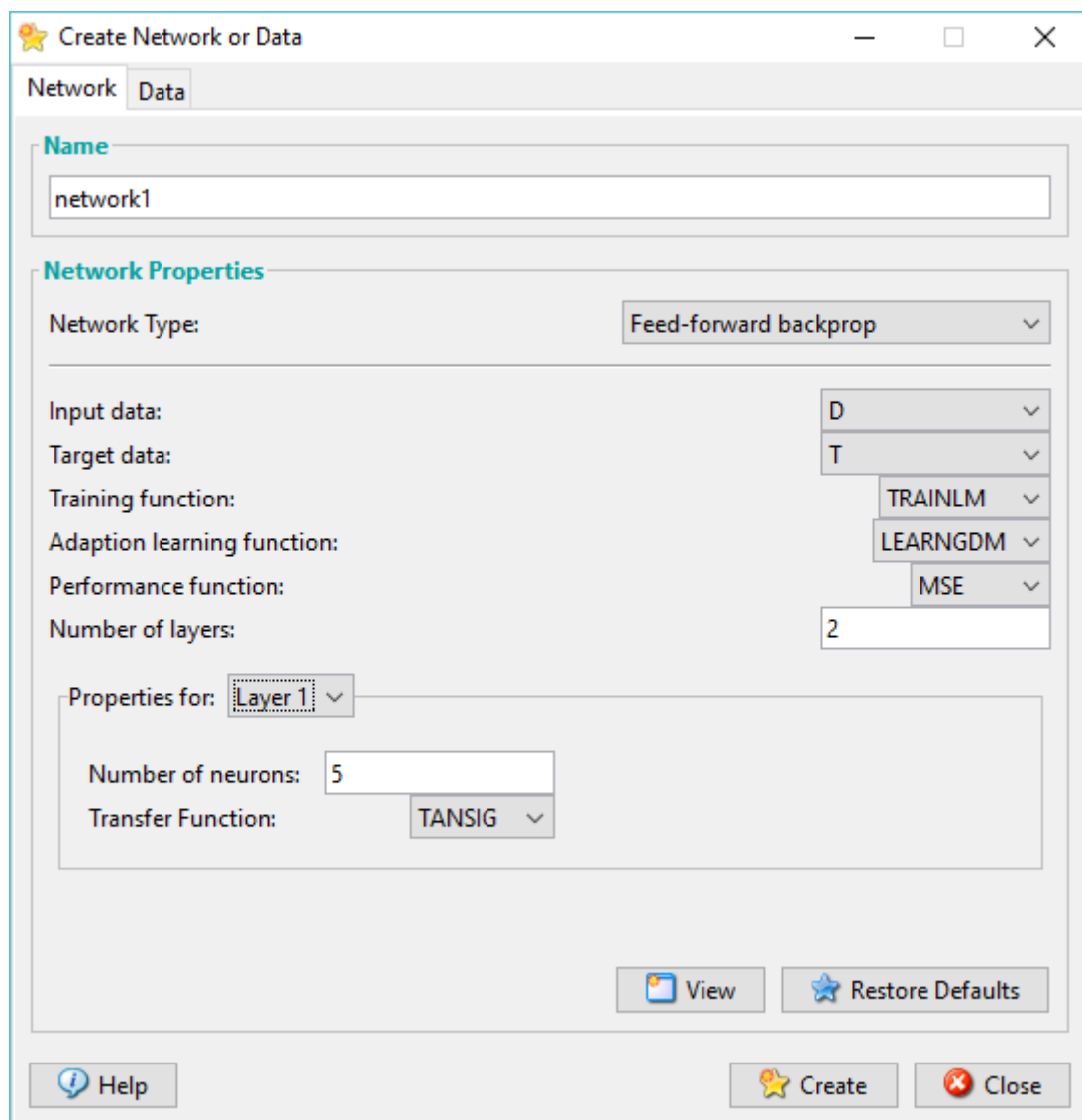


Рисунок 2 – Окно Network/Data Manager с выбором параметров компьютерной модели создаваемой нейронной сети приближения функции

Полученная сеть в среде Simulink представлена на рисунке 3.

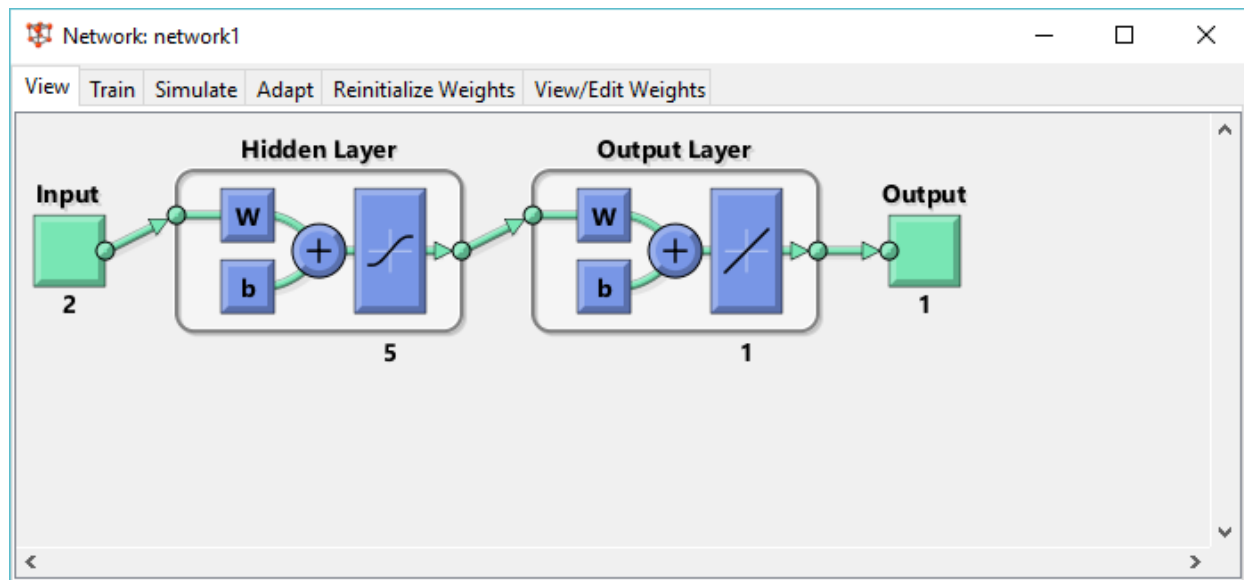
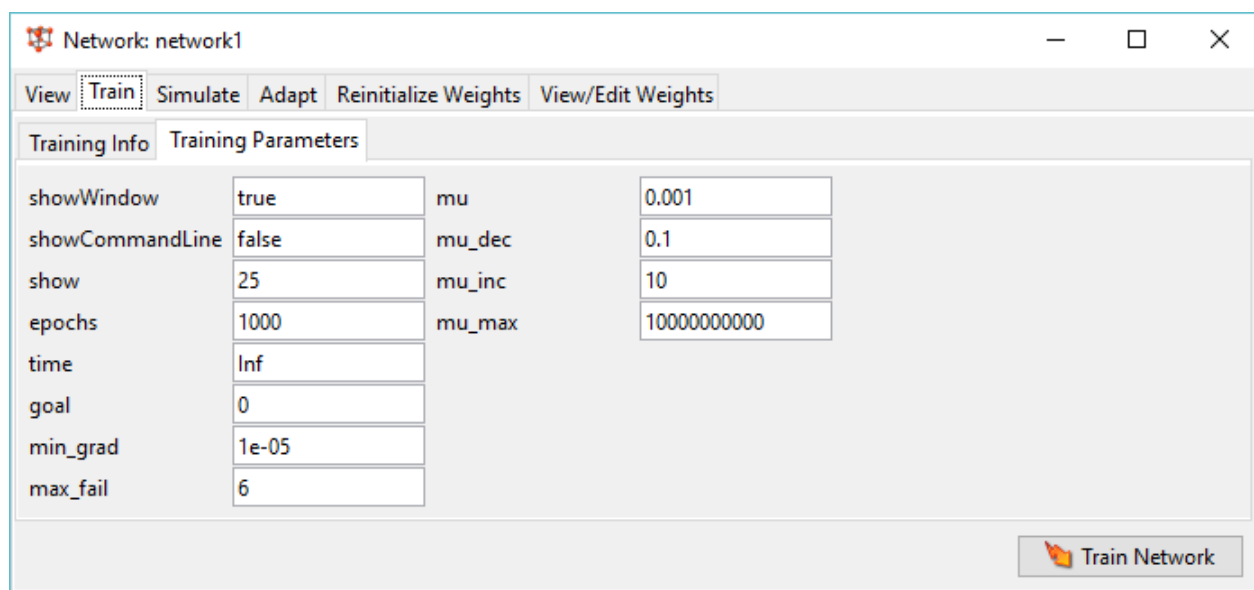


Рисунок 3 – Структура созданной нейронной сети

Обучим сеть, при этом в качестве входных данных необходимо использовать матрицу D, а в качестве цели вектор T (рисунок 4).



Training Parameters			
showWindow	true	mu	0.001
showCommandLine	false	mu_dec	0.1
show	25	mu_inc	10
epochs	1000	mu_max	10000000000
time	Inf		
goal	0		
min_grad	1e-05		
max_fail	6		

Train Network

Рисунок 4 – Задание параметров обучения нейронной сети

Процесс обучения нейронной сети представлен на рисунке 5. Сеть обучена за 98 итераций до ошибки $9.7 \cdot 10^{-6}$.

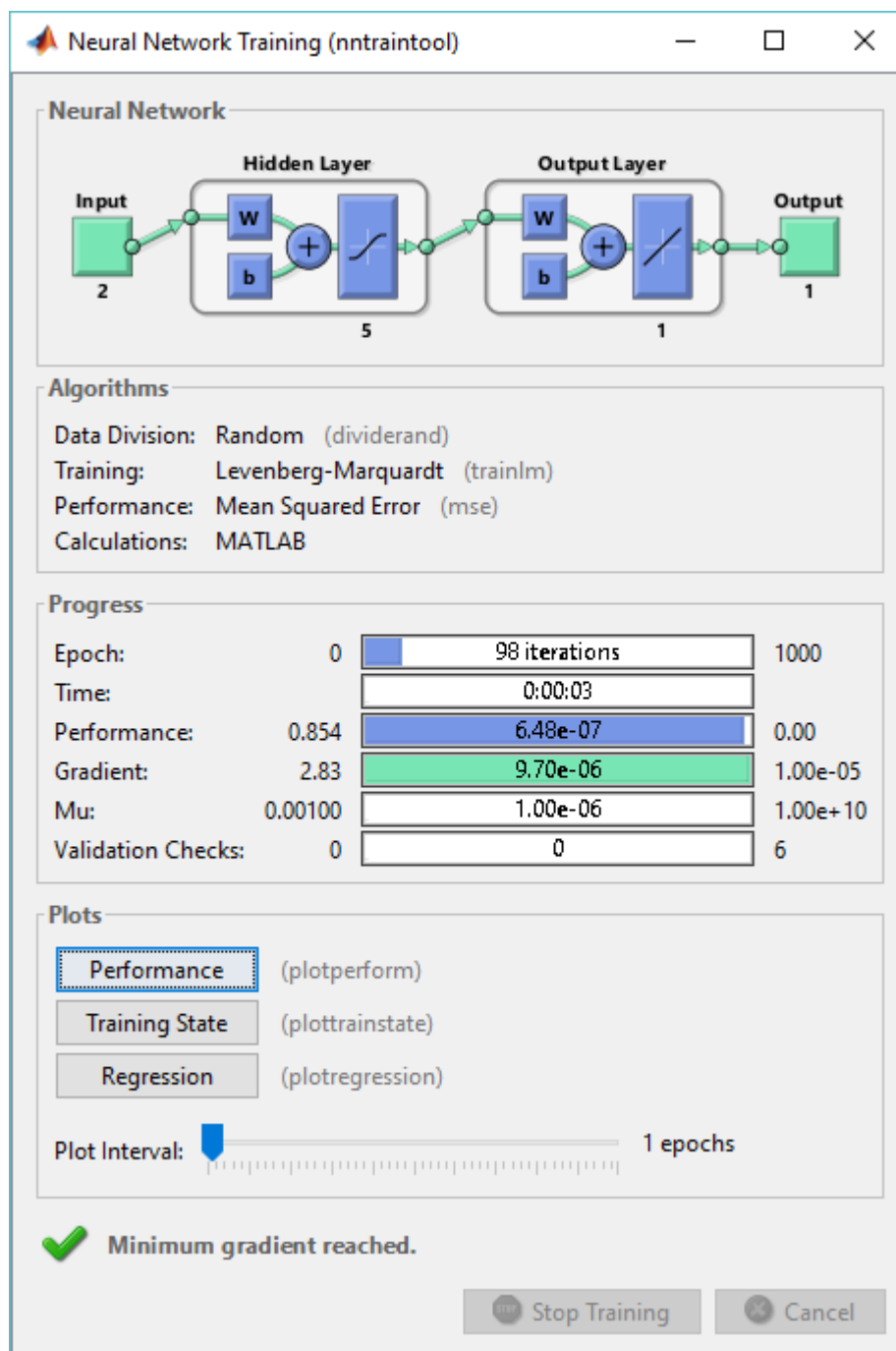


Рисунок 5 – Процесс обучения нейронной сети

Обученную сеть симулируем вектором данных, получив выход сети `network1_outputs` на основе входных данных матрицы `D` (рисунок 6).

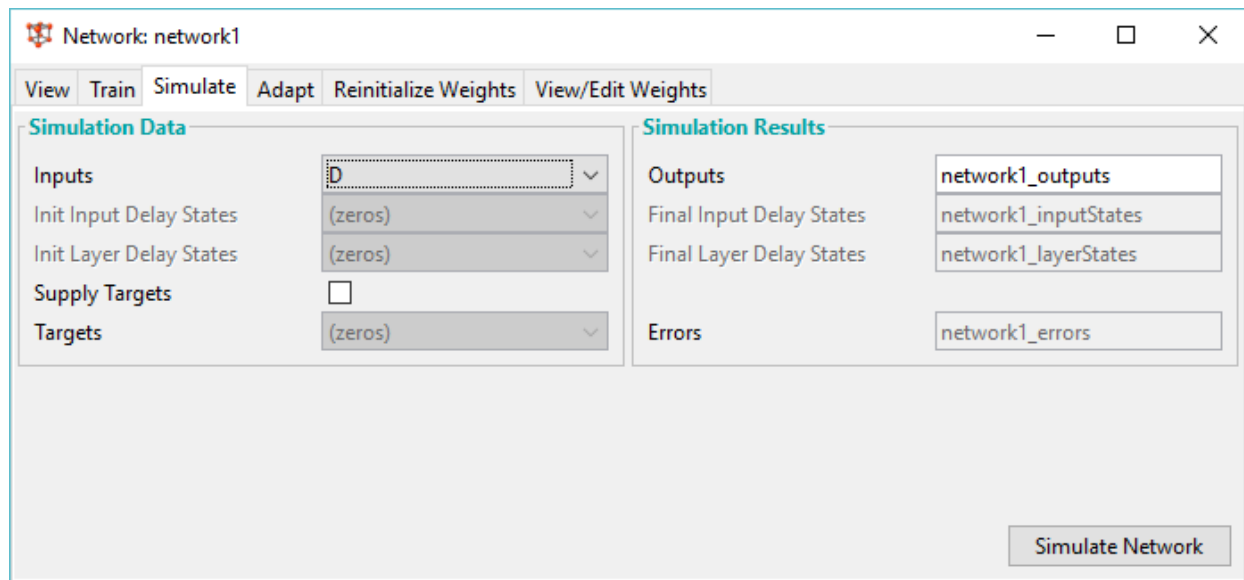


Рисунок 6 – Симулирование нейронной сети

После этого необходимо экспортировать вектор `network1_outputs` в рабочую область системы Matlab (рисунок 7).

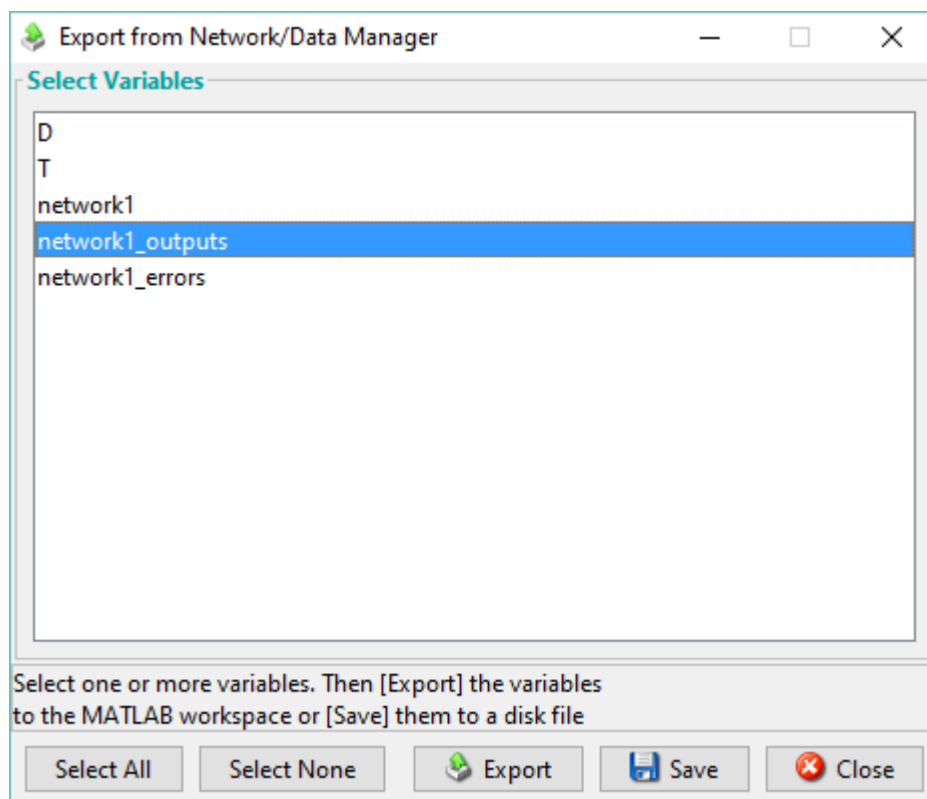


Рисунок 7 – Экспорт данных из пакета NNTool в рабочую область системы Matlab

Абсолютную ошибку сети можно найти, получив разность выхода обученной сети при симуляции `network1_outputs` и цели `T` в рабочей области: $error = network1_outputs - T$; График ошибки представлен на рисунке

8.

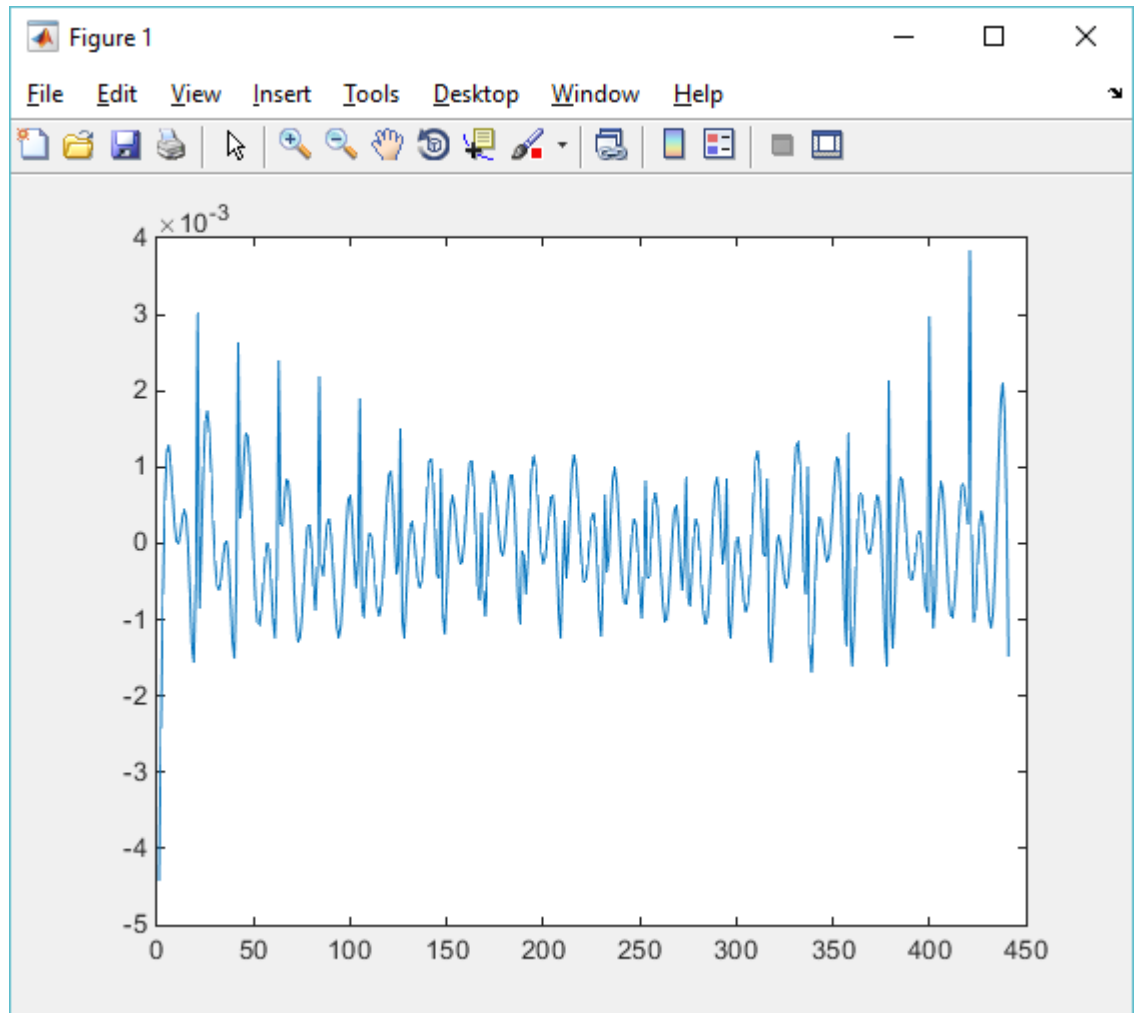


Рисунок 8 – Погрешность приближения функции нейронной сетью

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

5. Программа и цель работы.
6. Задание для лабораторной работы по каждому пункту.
7. Структура нейронной сети приближения функции. Графики погрешности приближения функции.
8. Выводы по работе.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить модель нейронной сети, аппроксимирующей полином вида

$$Y = a x^2 + b x^2 + c x x + d x + e x + f$$

1 2 1 2

на интервале $[-1, 1]$, коэффициенты полинома заданы в таблице 1.

Вариант соответствует порядковому номеру по журналу.

Таблица 1 – Варианты заданий

№ варианта	a	b	c	d	e	f
1.	-1	1	2	0	0	0
2.	1	1	1	1	1	1
3.	2	1	1	2	2	-8
4.	0	1	0	1	0	-2
5.	1	0	1	0	1	-3
6.	2	1	-2	1	0	-7
7.	1	2	3	-3	-4	-1
8.	2	1	2	-1	-3	-2
9.	2	1	1	-2	-2	-1
10.	-1	-1	-1	-1	2	2
11.	-2	1	-1	2	1	-2
12.	-1	3	-1	2	-1	-1
13.	-2	-2	2	2	-2	2
14.	-3	2	-2	3	-1	1
15.	-2	2	-2	2	-2	2
16.	-1	-1	-1	1	1	1
17.	0	1	0	2	0	-3
18.	2	0	2	3	1	0
19.	2	-2	4	-4	3	-3
20.	-5	-3	3	3	2	1
21.	-2	-3	2	3	0	-1
22.	2	6	1	1	-3	-4
23.	3	2	1	2	1	-2
24.	-2	-1	-3	2	3	2
25.	1	1	2	2	-3	-3
26.	-2	-3	-4	4	3	2
27.	2	5	-5	-2	1	-1

Вычертить в отчете структуру нейронной сети, построить графики аппроксимируемой функции, выход нейронной сети и функции погрешности

приближения. Уменьшить шаг дискретизации при задании обучающих примеров вдвое, построить для этого случая функцию ошибок. В каких точках функция ошибки сети равна нулю? Сделайте выводы по полученным результатам.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Постановка задачи обучения нейронной сети.

3. Алгоритм обучения нейронной сети на основе обратного распространения ошибки.
4. Обучение нейронной сети на основе самоорганизации.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 10

Разработка нейросетевого алгоритма аппроксимации функции

Порядок и последовательность выполнения работы

1. Создание компьютерной модели нейронной сети приближения функции.
2. Обучение и симулирование сети. Получение погрешности приближения.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

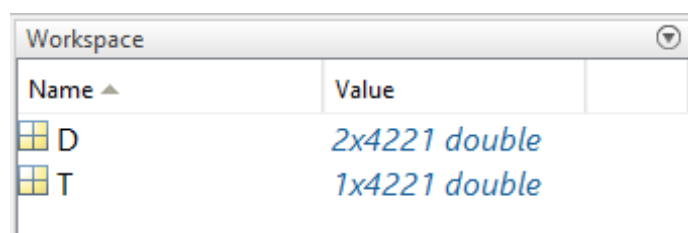
Пусть необходимо аппроксимировать полином вида

$$Y = x^2 - x^2 + x_1 x_2 \quad (1)$$

Для формирования вектора обучающих примеров в окне команд или в m-файле необходимо набрать следующие команды:

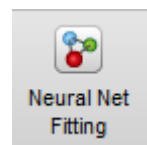
```
i=1;  
for x1=-1:0.1:1;  
    for x2=-1:.01:1;  
        D(1,i)=x1;  
        D(2,i)=x2;  
        T(i)=x1^2-x2^2+x1*x2;  
        i=i+1;  
    end;  
end;
```

После чего в рабочей области будут сформированы матрица входных данных D и вектор целей T (рисунок 1).



Name ▲	Value
D	2x4221 double
T	1x4221 double

Рисунок 1 - Рабочая область с сформированной обучающей выборкой



Запустим пакет Neural Net Fitting воспользовавшись иконкой вкладки APPS или набрав nftool в командной строке. Запуститься мастер создания нейронной сети прямого распространения с обратным распространением ошибки (см. предыдущее практическое занятие). При выборе данных обучающей выборки (Select Data) необходимо для входных данных (Inputs:) выбрать матрицу D, а для выходных данных (Targets). С распределением данных для обучения нейронной сети, проверки и тестирования, предложенным мастером, можно согласиться. Для двух входов нейронной сети достаточно в скрытом слое пяти нейронов. Архитектура созданной нейронной сети представлена на рисунке 2.

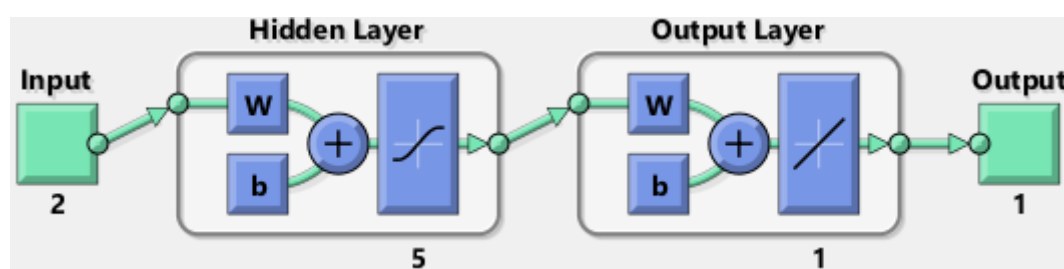


Рисунок 2 – Архитектура созданной нейронной сети

Обучим нейронную сеть. После обучения сети можно посмотреть графики распределения ошибок (Plot Error Gistogram, рисунок 3) и распределение аппроксимируемых данных (Plot Regression, рисунок 4).

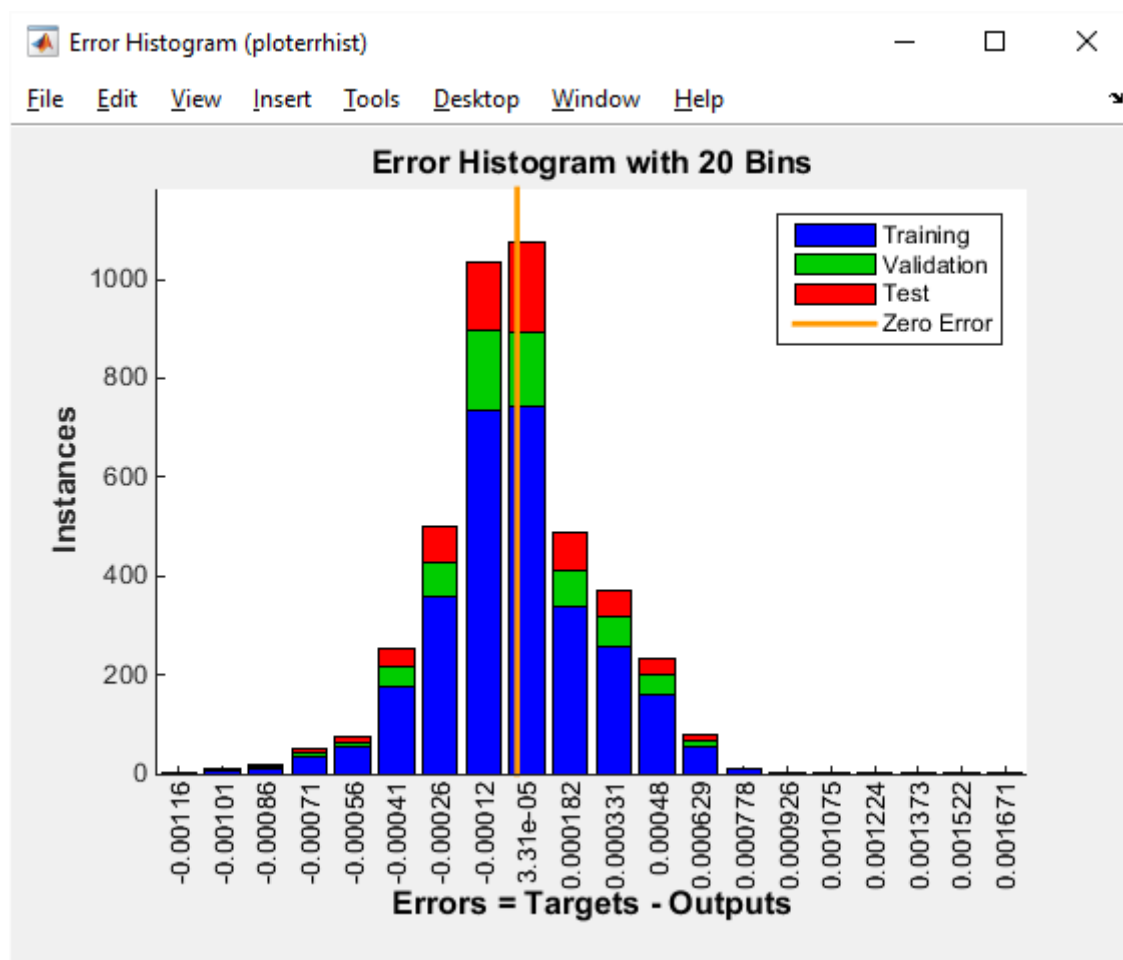


Рисунок 3 – Распределение ошибок

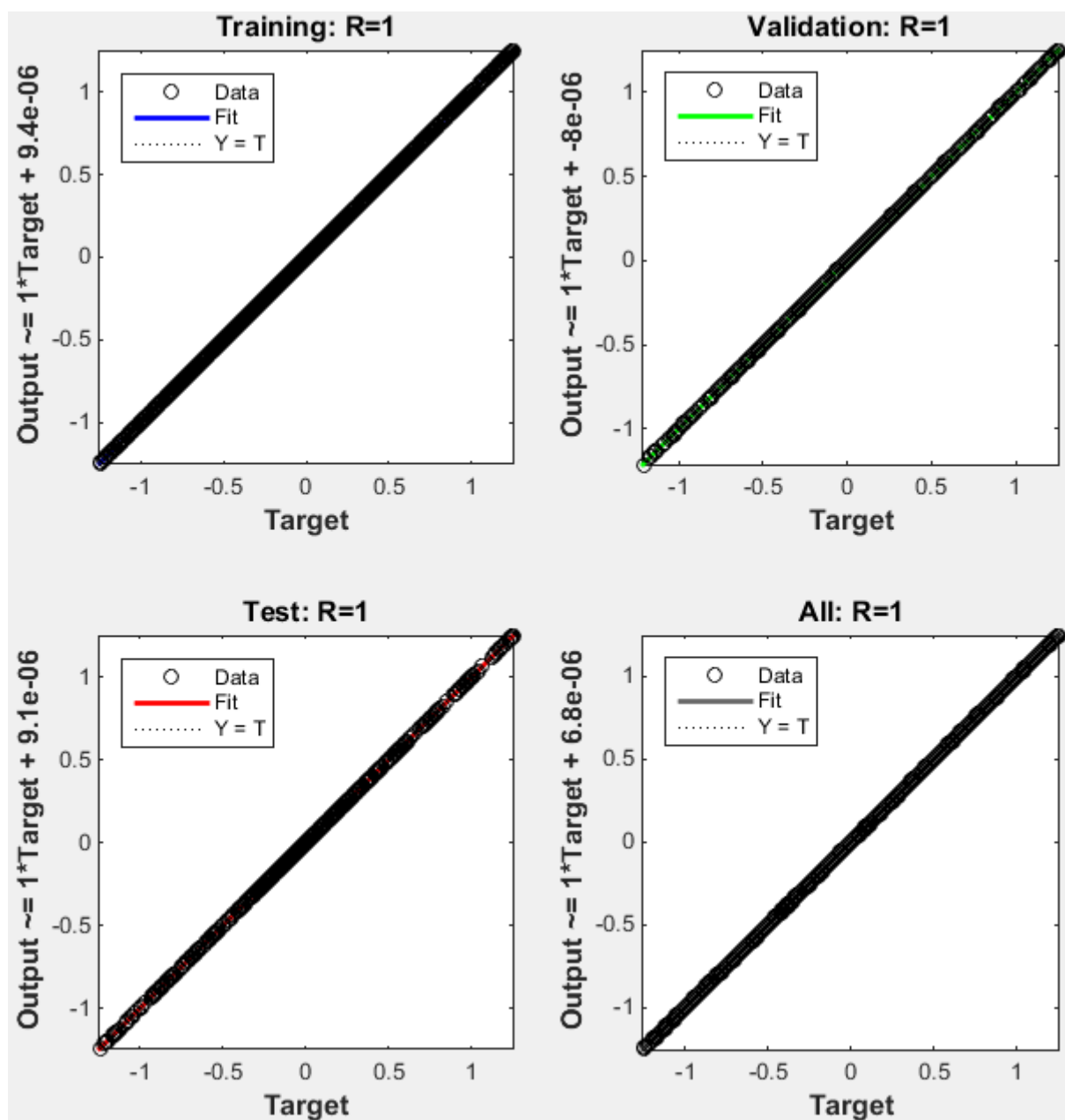


Рисунок 4 – Регрессия аппроксимируемой функции

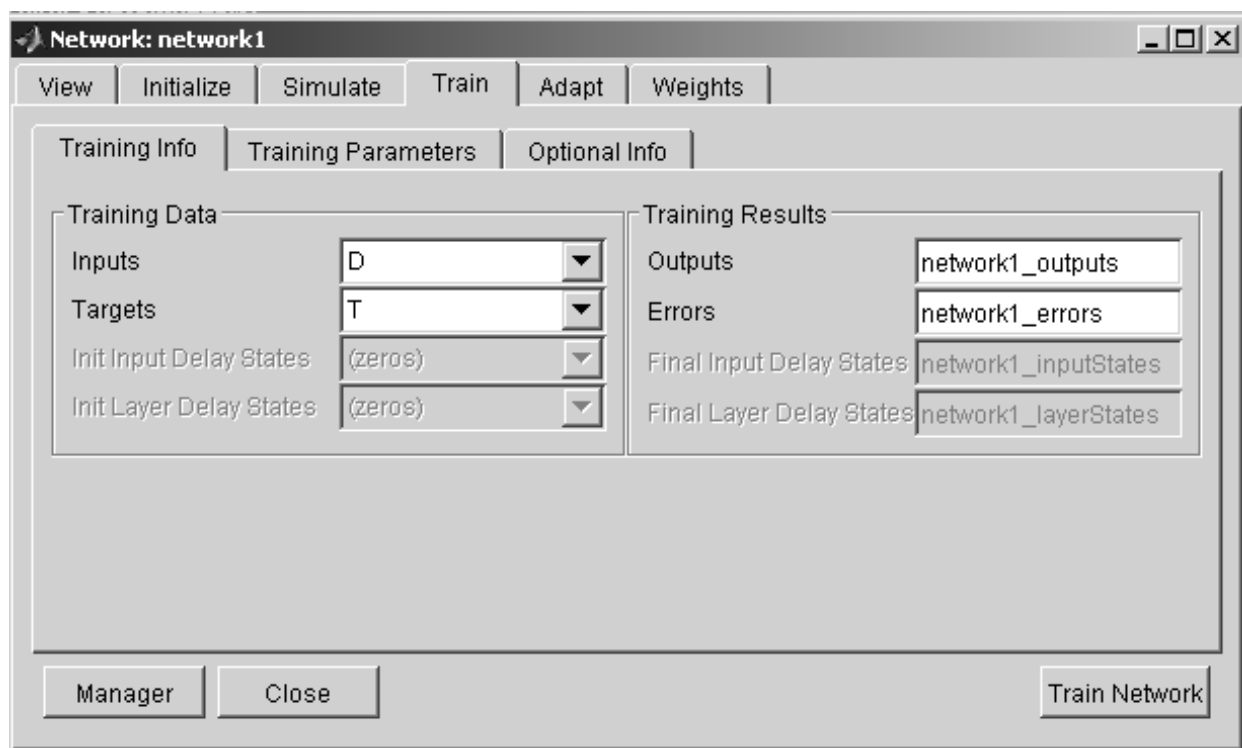


Рисунок 4 – Задание параметров обучения нейронной сети

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Программа и цель работы.
2. Задание для лабораторной работы по каждому пункту.
3. Структура нейронной сети приближения функции. Графики погрешности приближения функции.
4. Выводы по работе.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить модель нейронной сети, аппроксимирующей полином вида

$$Y = ax^2 + bx^2 + cx_1x_2 + dx_1 + ex_2 + f$$

на интервале $[-1, 1]$, коэффициенты полинома заданы в таблице 1.

Вариант соответствует порядковому номеру по журналу.

Таблица 1 – Варианты заданий

№ варианта	a	b	c	d	e	f
1.	-1	1	2	0	0	0

2.	1	1	1	1	1	1
3.	2	1	1	2	2	-8
4.	0	1	0	1	0	-2
5.	1	0	1	0	1	-3
6.	2	1	-2	1	0	-7

7.	1	2	3	-3	-4	-1
8.	2	1	2	-1	-3	-2
9.	2	1	1	-2	-2	-1
10.	-1	-1	-1	-1	2	2
11.	-2	1	-1	2	1	-2
12.	-1	3	-1	2	-1	-1
13.	-2	-2	2	2	-2	2
14.	-3	2	-2	3	-1	1
15.	-2	2	-2	2	-2	2
16.	-1	-1	-1	1	1	1
17.	0	1	0	2	0	-3
18.	2	0	2	3	1	0
19.	2	-2	4	-4	3	-3
20.	-5	-3	3	3	2	1
21.	-2	-3	2	3	0	-1
22.	2	6	1	1	-3	-4
23.	3	2	1	2	1	-2
24.	-2	-1	-3	2	3	2
25.	1	1	2	2	-3	-3
26.	-2	-3	-4	4	3	2
27.	2	5	-5	-2	1	-1

Вычертить в отчете структуру нейронной сети, построить графики аппроксимируемой функции, выход нейронной сети и функции погрешности приближения. Уменьшить шаг дискретизации при задании обучающих примеров вдвое, построить для этого случая функцию ошибок. Сделайте выводы по полученным результатам.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Постановка задачи обучения нейронной сети.

2. Алгоритм обучения нейронной сети на основе обратного распространения ошибки.
3. Обучение нейронной сети на основе самоорганизации.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 11

Прогнозирование энергопотребления здания с использованием нейронной сети

Цель: Создать нейросетевой алгоритм прогнозирования энергопотребления здания.

Порядок и последовательность выполнения работы

1. Создание компьютерной модели нейронной сети и обучение ее.
2. Симулирование произвольными данными.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

Рассматриваемый демонстрационный позволяет повести прогноз уровня энергопотребления здания в зависимости от даты, времени суток, погоды и др.

Для загрузки этого демонстрационного примера, на основе которого будет создан нейросетевой алгоритм прогнозирования уровня энергопотребления здания, необходимо выбрать демонстрационный пример (Load Example Data Set) Building Energy (рисунок 1) приложения Neural Net Fitting.

Исходные данные рассматриваемого демонстрационного примера содержит обучающую выборку из 4208 образцов, и состоит из реальных данных, и обобщает следующие данные:

- 1-10. Код дня недели, времени суток
11. Температура
12. Влажность
13. Солнечная сила
14. Ветер

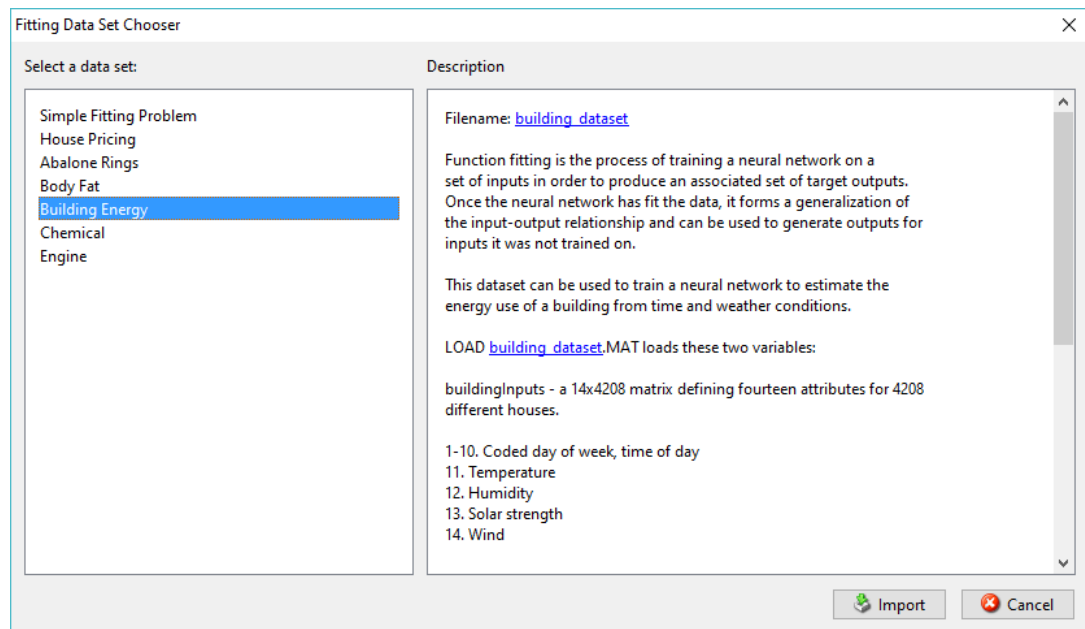


Рисунок 1 – Выбор демонстрационного примера

Описанные данные содержатся в матрице BildingInputs (рисунок 2).

buildingInputs										
14x4208 double										
	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0.8333	0.7500	0.6667	0.5833	0.5000	0.4167	0.3333	0.2500	0.1667	0.0833
9	-0.8261	-0.7391	-0.6522	-0.5652	-0.4783	-0.3913	-0.3043	-0.2174	-0.1304	-0.0435
10	1	1	1	1	1	1	1	1	1	1
11	0.8190	0.8070	0.7970	0.7900	0.7890	0.7850	0.7870	0.8020	0.8340	0.8650
12	0.7360	0.7480	0.7760	0.7880	0.7960	0.8080	0.8280	0.8360	0.8040	0.7120
13	0.0094	0.0094	0.0094	0.0095	0.0095	0.0099	0.0542	0.2179	0.4214	0.5536
14	0.2721	0.2836	0.2757	0.2171	0.2029	0.1382	0.1907	0.2211	0.2654	0.3546

Рисунок 2 – Обучающая выборка

Уровень энергопотребления зданиям обучающей выборки содержит матрица BildingTargets (рисунок 3).

buildingTargets

3x4208 double

1	2	3	4	5	6	7	8	9	10
0.3434	0.3441	0.3438	0.3424	0.3448	0.3579	0.4081	0.5225	0.5754	0.5908
0.6150	0.6075	0.6075	0.6075	0.6000	0.6000	0.6000	0.6300	0.6675	0.6675
0.1227	0.1330	0.1330	0.1432	0.1432	0.1432	0.1432	0.1432	0.1432	0.1330

Рисунок 3 – Цель

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить компьютерную модель нейронной сети прогнозирования уровня энергопотребления зданием на основе демонстрационного примера, обучить ее и сохранить полученные результаты в различном виде. Просимулировать сеть исходными данными и сравнить с данными демонстрационных примеров. Рассчитать среднеквадратическую ошибку на всех образах обучающей выборки. Попробуйте изменить архитектуру сети. Сделать вывод о работе сети и качестве исходных данных обучающей выборки.

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Тема, программа и цель работы.

2. Задание для лабораторной работы.
3. Структура нейронной сети приближения функции. Графики погрешности приближения функции. Значение ошибки. Результаты симулирования сети.
4. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие рекуррентные нейронные сети известны Вам?
2. Какие функции активации используются в нейронной сети Хопфилда?
3. Какое количество образов способна хранить нейронная сеть Хопфилда?
4. Каким образом рассчитываются синаптические веса нейронной сети Хопфилда?
5. Каким образом функционирует нейронная сеть Хопфилда?
6. Сколько слоев нейронов содержит нейронная сеть Хопфилда?
7. Сколько слоев нейронов содержит нейронная сеть Хэмминга?
8. Каким образом рассчитываются синаптические веса нейронной сети Хэмминга?
9. Каким образом функционирует нейронная сеть Хэмминга?
10. Сколько слоев нейронов содержит нейронная сеть двунаправленной ассоциативной памяти?
11. Каким образом функционирует нейронная сеть двунаправленной ассоциативной памяти?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 12

Разработка нейросетевого алгоритма управления химической реакцией

Цель: создать нейросетевой алгоритм управления химической реакцией.

Порядок и последовательность выполнения работы

1. Создание компьютерной модели нейронной сети и обучение ее.
2. Симулирование произвольными данными.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

Рассматриваемый демонстрационный позволяет создать нейросетевой алгоритм управления количеством выхода действующего вещества при химической реакции в зависимости от количества восьми других компонентов.

Для загрузки этого демонстрационного примера, на основе которого будет создан нейросетевой алгоритм управления количеством выхода действующего вещества при химической реакции, необходимо выбрать демонстрационный пример (Load Example Data Set) Chemical (рисунок 1) приложения Neural Net Fitting.

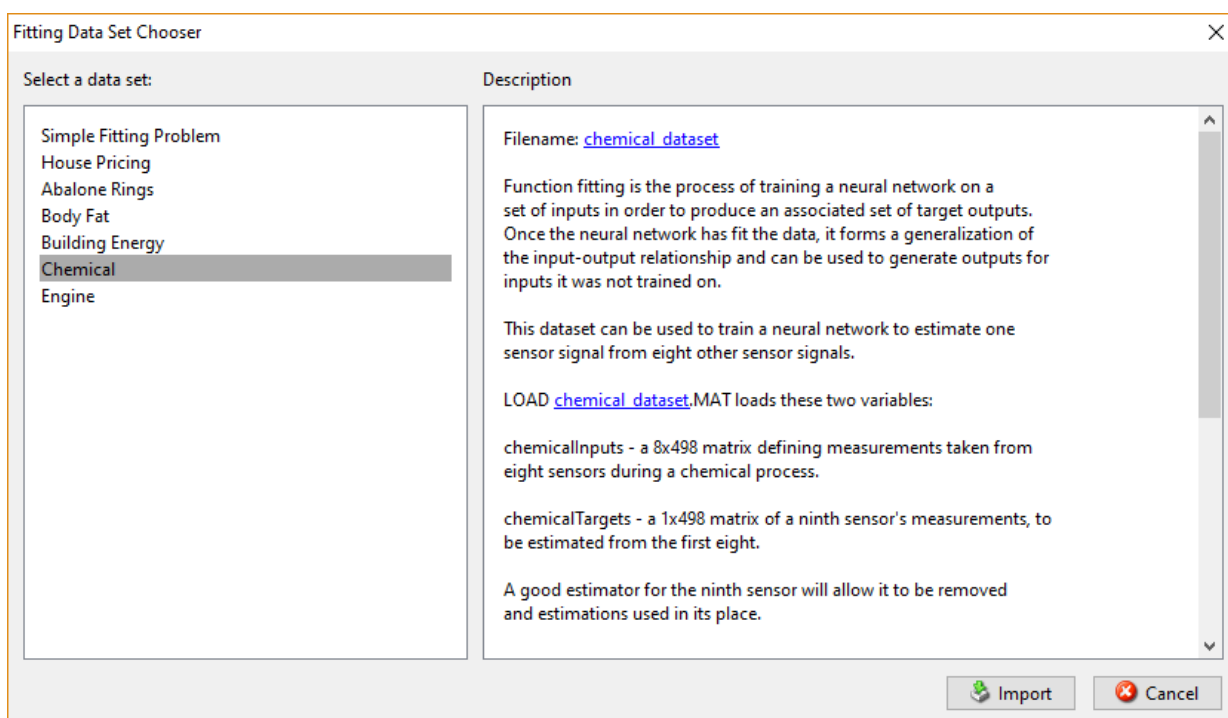


Рисунок 1 – Выбор демонстрационного примера

Исходные данные рассматриваемого демонстрационного примера содержит обучающую выборку из 498 образов, и содержит восемь строк показаний сенсоров. Описанные данные содержатся в матрице Chemical Input (рисунок 2).

chemicalInputs ✕

8x498 double

1	2	3	4	5	6	7	8	9	10
157	155	154	154	152	153	151	156	154	155
9596	9487	9551	9637	9486	9633	9637	9668	9482	9516
4714	5049	5070	5087	5065	5319	4987	5009	5017	5205
376	381	374	382	380	377	379	387	381	388
2.5800	2.2800	2.9800	2.5700	3.0400	2.7200	3.0700	2.5900	2.9700	2.1900
407	411	406	408	408	408	408	410	407	412
564	567	563	565	567	565	563	570	565	574
510354	504718	456972	512311	489312	495420	474552	470842	421062	522021

Рисунок 2 – Исходные данные

В качестве выходных данных обучающая выборка содержит цель. Описанные данные содержатся в матрице ChemicalTargets (рисунок 3).

chemicalTargets ✕

1x498 double

	1	2	3	4	5	6	7	8	9	10
1	514	516	512	516	515	513	512	517	515	518

Рисунок 3 – Обучающая выборка

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить компьютерную модель нейронной сети управления количеством выхода действующего вещества при химической реакции в зависимости от количества восьми других компонентов на основе демонстрационного примера, обучить ее и сохранить полученные результаты в различном виде. Просимулировать сеть исходными данными и сравнить с данными демонстрационных примеров. Рассчитать среднеквадратическую ошибку на всех образах обучающей выборки. Попробуйте изменить архитектуру сети. Сделать вывод о работе сети и качестве исходных данных обучающей выборки.

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Тема, программа и цель работы.

2. Задание для лабораторной работы.

3. Структура нейронной сети приближения функции. Графики погрешности приближения функции. Значение ошибки. Результаты симулирования сети.
4. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая нейросетевая парадигма используется в задаче распознавания речи?
2. Какая нейросетевая парадигма используется в задачах маршрутизации?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 13

Разработка нейросетевого алгоритма оценки стоимости дома

Цель: создать нейросетевой алгоритм оценки стоимости дома на основе демонстрационного примера.

Порядок и последовательность выполнения работы

1. Создание компьютерной модели нейронной сети и обучение ее.
2. Симулирование произвольными данными.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

Рассматриваемый демонстрационный пример позволяет создать нейросетевой алгоритм оценки стоимости дома в зависимости от статистики продаж домов.

Для загрузки этого демонстрационного примера, на основе которого будет создан нейросетевой алгоритм повести оценки крутящего момента и выбросов в зависимости от скорости и расхода топлива двигателем автомобиля, необходимо выбрать демонстрационный пример (Load Example Data Set) House Pricing (рисунок 1) приложения Neural Net Fitting.

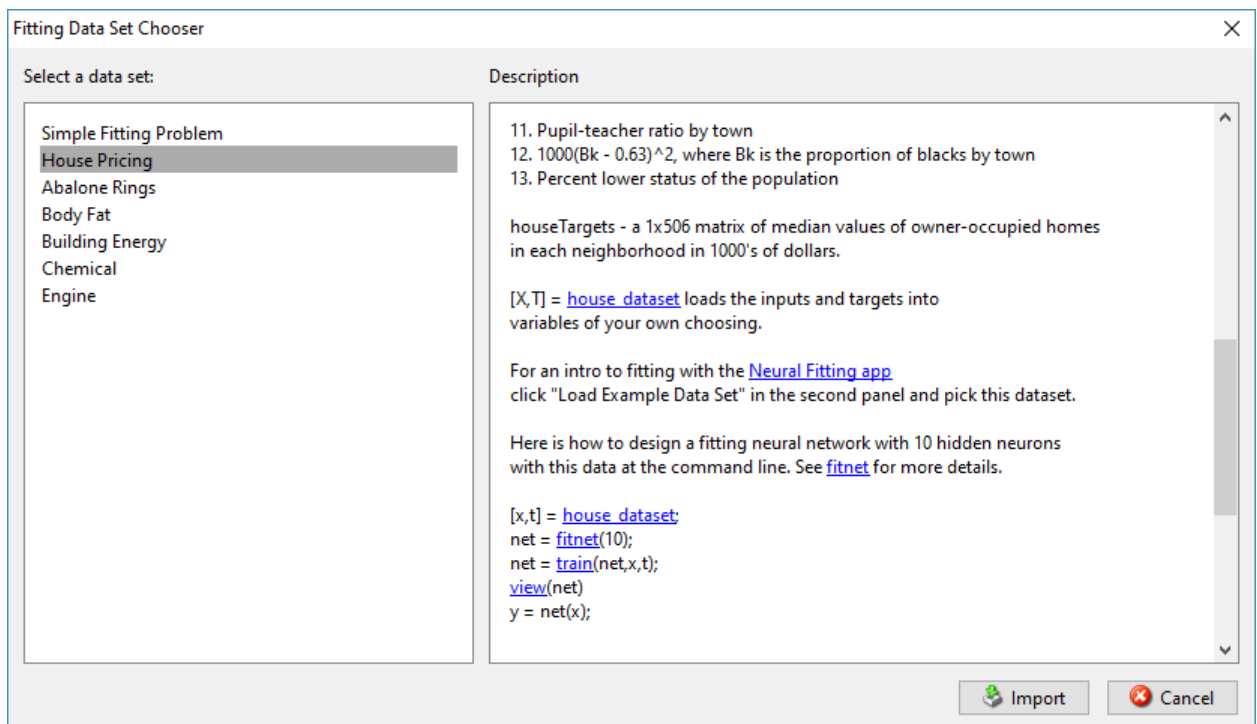


Рисунок 1 – Выбор демонстрационного примера

Исходные данные рассматриваемого демонстрационного примера содержит обучающую выборку из 506 образов, и содержит тринадцать строк статистики продажи домов:

1. Уровень преступности на душу населения в городе
 2. Доля жилых земельных участков, предназначенных для лотов свыше 25 000 кв. метров.
 3. доля розничного бизнеса в городе
 4. 1 если есть река, 0 - в противном случае
 5. Концентрация окислов азота
 6. Количество комнат в жилом помещении
 7. Доля домов, построенных до 1940 г.
 8. Расстояния до центра занятости
 9. Индекс доступности до магистрали
 10. Полная ставка налога
 11. Соотношение ученик-учитель, по городу
 12. Процент негров по городу
 13. Процентов населения, живущих ниже уровня бедности
- Описанные данные содержатся в матрице House Input (рисунок 2).

houseInputs										
13x506 double										
	1	2	3	4	5	6	7	8	9	10
1	0.0063	0.0273	0.0273	0.0324	0.0691	0.0299	0.0883	0.1446	0.2112	0.1700
2	18	0	0	0	0	0	12.5000	12.5000	12.5000	12.5000
3	2.3100	7.0700	7.0700	2.1800	2.1800	2.1800	7.8700	7.8700	7.8700	7.8700
4	0	0	0	0	0	0	0	0	0	0
5	0.5380	0.4690	0.4690	0.4580	0.4580	0.4580	0.5240	0.5240	0.5240	0.5240
6	6.5750	6.4210	7.1850	6.9980	7.1470	6.4300	6.0120	6.1720	5.6310	6.0040
7	65.2000	78.9000	61.1000	45.8000	54.2000	58.7000	66.6000	96.1000	100	85.9000
8	4.0900	4.9671	4.9671	6.0622	6.0622	6.0622	5.5605	5.9505	6.0821	6.5921
9	1	2	2	3	3	3	5	5	5	5
10	296	242	242	222	222	222	311	311	311	311
11	15.3000	17.8000	17.8000	18.7000	18.7000	18.7000	15.2000	15.2000	15.2000	15.2000
12	396.9000	396.9000	392.8300	394.6300	396.9000	394.1200	395.6000	396.9000	386.6300	386.7100
13	4.9800	9.1400	4.0300	2.9400	5.3300	5.2100	12.4300	19.1500	29.9300	17.1000

Рисунок 2 – Исходные данные

В качестве выходных данных обучающая выборка содержит цель – стоимость дома. Описанные данные содержатся в матрице HouseTargets

(рисунок 3).

houseTargets X										
1x506 double										
1	2	3	4	5	6	7	8	9	10	
24	21.6000	34.7000	33.4000	36.2000	28.7000	22.9000	27.1000	16.5000	18.9000	

Рисунок 3 – Обучающая выборка

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Построить компьютерную модель нейронной сети оценки стоимости дома на основе демонстрационного примера, обучить ее и сохранить полученные результаты в различном виде. Просимулировать сеть исходными данными и сравнить с данными демонстрационных примеров. Рассчитать среднеквадратическую ошибку на всех образах обучающей выборки. Попробуйте изменить архитектуру сети. Сделать вывод о работе сети и качестве исходных данных обучающей выборки.

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Тема, программа и цель работы.
2. Задание для лабораторной работы.
3. Структура нейронной сети приближения функции. Графики погрешности приближения функции. Значение ошибки. Результаты симулирования сети.
4. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая нейросетевая парадигма используется в задаче распознавания речи?
2. Какая нейросетевая парадигма используется в задачах маршрутизации?

Практическое занятие 14. Изучение радиальных базисных, сетей регрессии, вероятностных нейронных сетей

Лабораторная работа 15. Исследование сети Кохонена и алгоритма обучения без учителя

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 14

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.

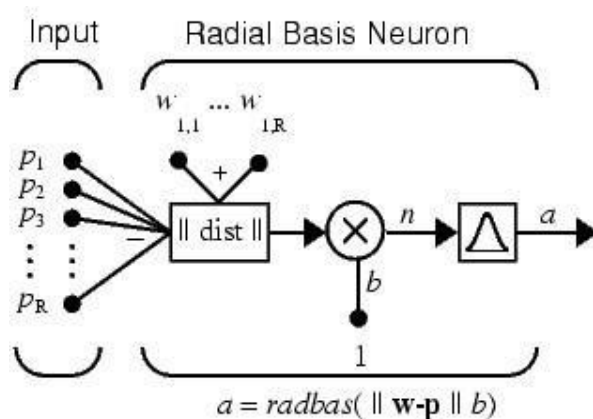


Рисунок 6.1. Радиальный базисный нейрон с R входами

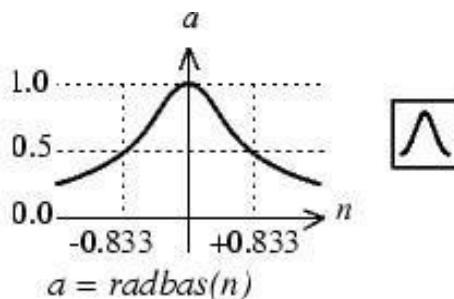


Рисунок 6.2. График функции активации

Радиальные базисные нейронные сети состоят из большого количества нейронов, чем стандартные сети с прямой передачей сигналов и обучением методом обратного распространения ошибки, но на их создание требуется значительно 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 $n=0$ $a=\exp(-0^2)=1$; при $n=0,833$ $a=\exp(-0,833^2)=0,5$; при $n=-0,833$ $a=\exp(0,833^2)=0,5$ (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 S^1 нейронов, и выходного линейного слоя, имеющего S^2 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 S^1 элементов. Выход блока $\|dist\|$ умножается поэлементно на вектор смещения b^1 и формирует вход функции активации.

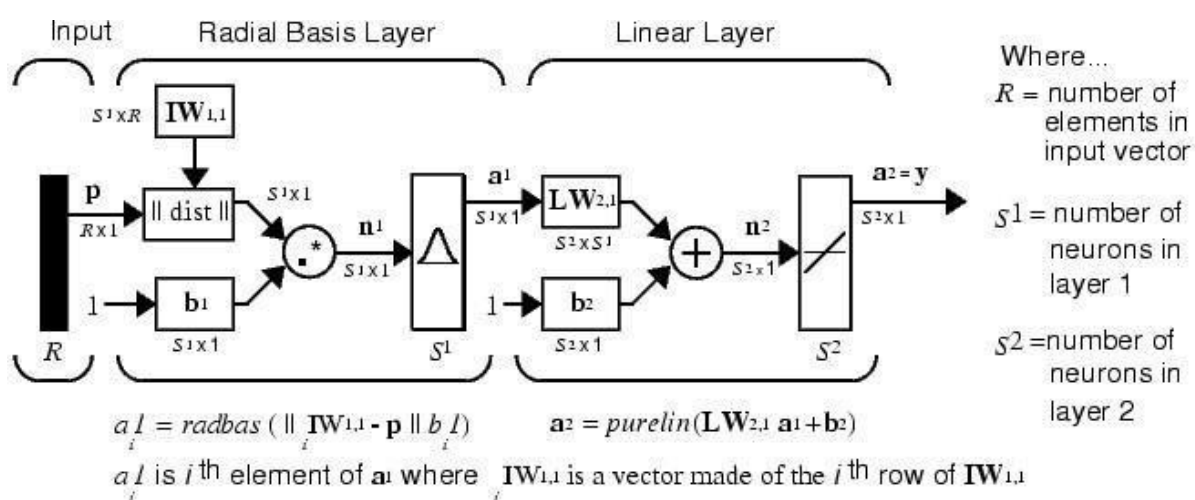


Рисунок 6.3. Схема архитектуры радиальной базисной сети

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:

$$f(x) = \sum_i w_i \varphi(\|x - c_i\|) \quad (6.3)$$

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

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

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 [57]:

`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 LW_{21}

и вектора входа a_1 в соответствии со следующим соотношением: (6.5)

$$n^2 = \frac{LW_{21}^T a^1}{\text{sum}(a^1)}$$

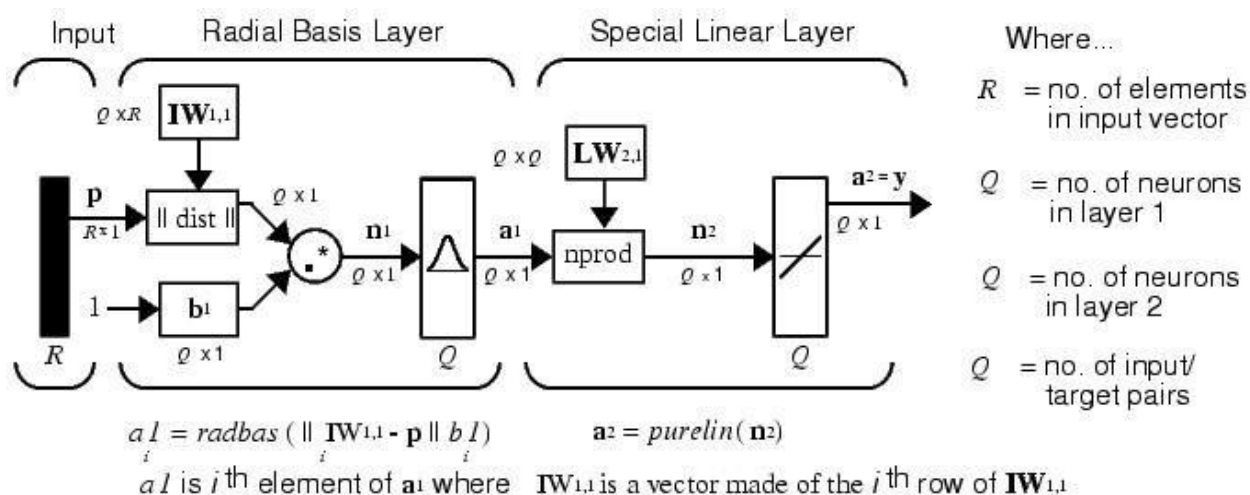


Рисунок 6.4. Схема архитектуры сети GRNN

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^T , 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 p_i , близкий к одному из

векторов входа p из обучающего множества. Этот вход p_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.

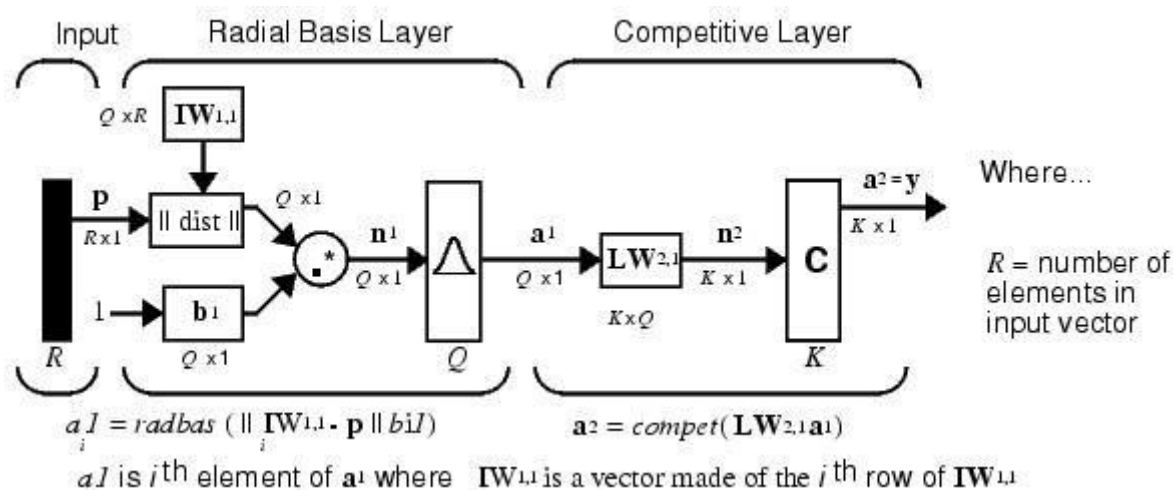


Рисунок 6.5. Схема архитектуры сети PNN

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.

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 $\mathbf{T}$ 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 $\mathbf{IW}^{11}$ ($\text{net.IW}\{1,1\}$) формируется с использованием векторов входа из обучающего множества в виде матрицы $\mathbf{p}$. Когда подается новый вход, блок $\|\text{dist}\|$ 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 $\mathbf{a}^1$ числом близким к 1.

Весовая матрица второго слоя $\mathbf{LW}^{21}$ ($\text{net.LW}\{2,1\}$) соответствует матрице связности $\mathbf{T}$, построенной для данной обучающей последовательности. Эта операция может быть выполнена с помощью функции ind2vec , которая преобразует вектор целей в матрицу связности $\mathbf{T}$. Произведение $\mathbf{T} \cdot \mathbf{a}^1$ определяет элементы вектора $\mathbf{a}^1$, 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 $\mathbf{n}^2$ и 0 в остальных случаях. Таким образом, сеть PNN выполняет классификацию векторов входа по K классам.

Вероятностную нейронную сеть 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 [57]:

```
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 функции
```

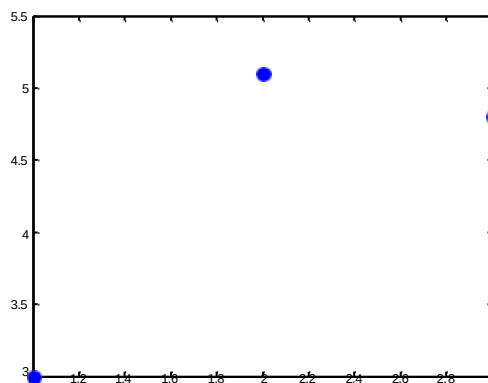


Рисунок 6.6. График аппроксимируемой функции

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

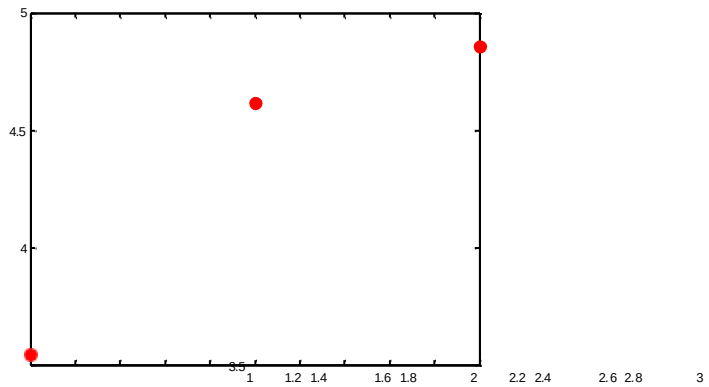


Рисунок 6.7. Выходы необученной нейронной сети

```
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 функции
```

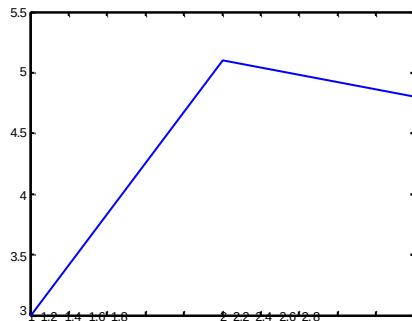


Рисунок 6.8 – Изображение аппроксимируемой функции

```
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 последовательность входов P2
```

```
plot(P2, A2, 'linewidth', 4, 'color', [1 0 0]); % отклик НС регрессии
```

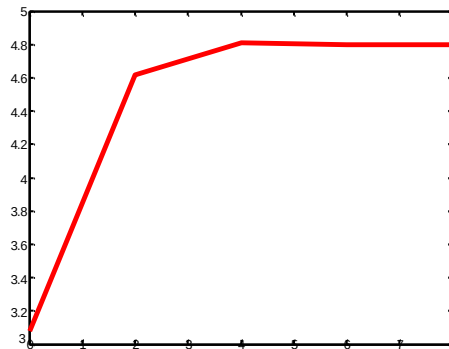


Рисунок 6.9. Отклик нейронной сети регрессии

```
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
```

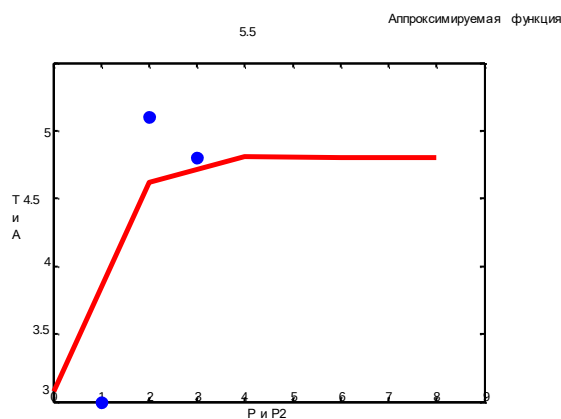


Рисунок 6.10. Аппроксимация точек с помощью нейронной сети регрессии

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

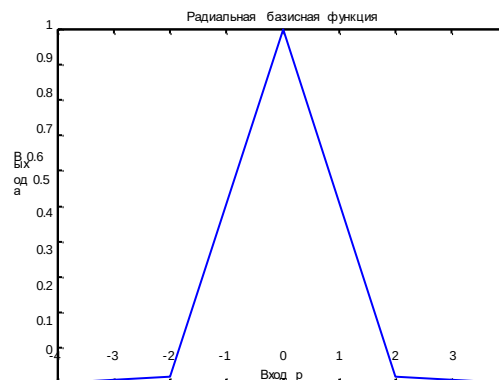


Рисунок 6.11. Радиальная базисная функция

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 (рис. 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

```

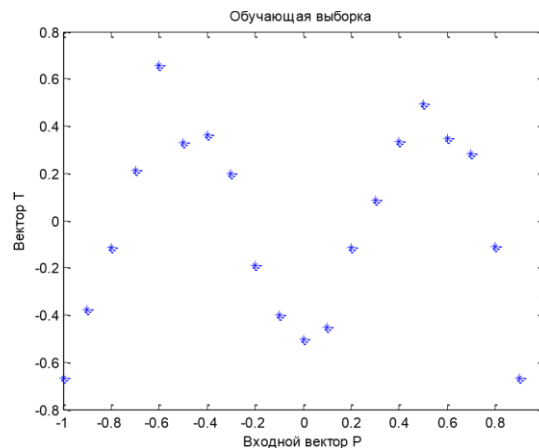


Рисунок 6.12. Обучающая выборка

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

```

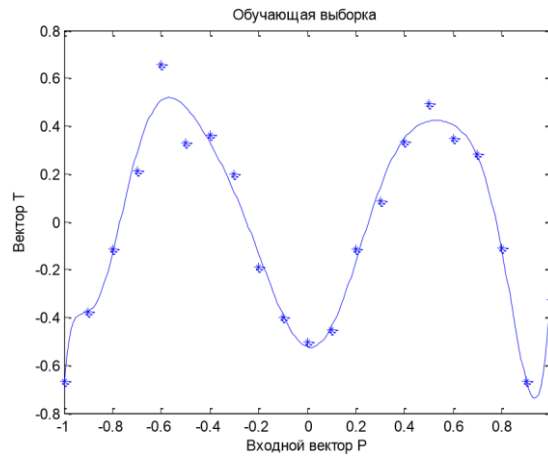


Рисунок 6.13. Результат аппроксимации с РБНС

4. Пример решения аппроксимации функции двух переменных

Пусть функция $z = \exp(x^2) \cdot \cos(-y^3)$ задана на промежутках $x=[-1; 1]$, $y=[-1.5; 1.5]$; 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)

```

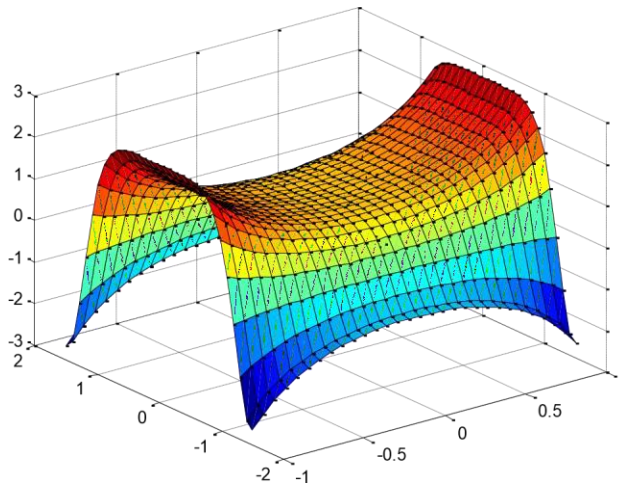


Рисунок 6.14. График исходной функции двух переменных $z = \exp(x^2) \cos(y^3)$

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

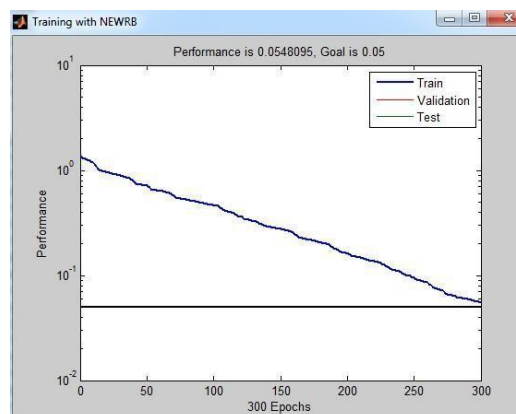


Рисунок 6.15. Характеристика точности обучения в зависимости от количества эпох обучения

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

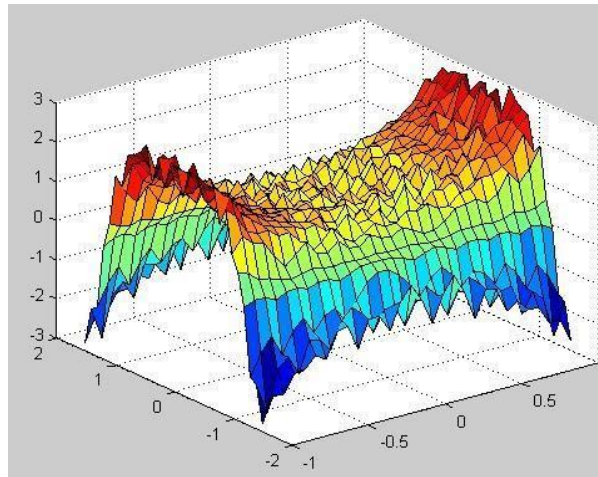


Рисунок 6.16. Результат моделирования исходной функции

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

$T = \text{ind2vec}(V)$; % конвертирование индексов в векторы, содержащие 1 в индексных позициях

$\text{net} = \text{newpnn}(P, T)$; % создание вероятностной НС

$Y = \text{sim}(\text{net}, P)$; % имитация работы вероятностной НС

$W = \text{vec2ind}(Y)$; % 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 P и целей T 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, аппроксимирующей функцию, приведенную в табл. 6.2.

Таблица 6.2. Варианты аппроксимируемых функций

№ варианта	$z = f(x, y)$
1.	$z = \cos(x) \cos(y), x, y \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
2.	$z = \frac{\sin(x)}{x} \frac{\sin(y)}{y}, x, y \in [-1, 1]$
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.	$z = x^2 \cos(y) - 1, x \in [-1.1, 1.1], y \in [-1.5, 1.5]$

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.	$z = x^2 \cos(y) + 1, x \in [-1,1], y \in [-\frac{\pi}{4}, \frac{\pi}{4}]$
12.	$z = x^3 \cos(y) + 1, x \in [-0.5,0.5], y \in [-\frac{\pi}{4}, \frac{\pi}{4}]$
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.	$z = \frac{\cos(x)}{x^2 + 1}, x \in [-5,5], y \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
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.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 15

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 $x_i = (x_{i1}, x_{i2}, \dots, x_{in})$, 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 x попарно сравнивается со всеми векторами 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 $d(x, w_c)$ между входным вектором x , характеризующим некоторый образ, к векторам w_c максимально, объявляется «победителем». При этом образ, характеризующийся вектором x , будет отнесен к классу, который представляется нейроном-«победителем».

Рассмотрим алгоритм самообучения сетей Кохонена. Обозначим функцию близости $z = \|x - w\|$. Выигрывает нейрон c

$$\|x - w_c\| = \min \|x - w_j\| \quad (7.1)$$

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

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

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

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

где $w_j(t)$ – значение весового вектора на шаге t самообучения сети, $d_{cj}(t)$ – функция близости между нейронами слоя Кохонена $\eta(t)$ – 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.

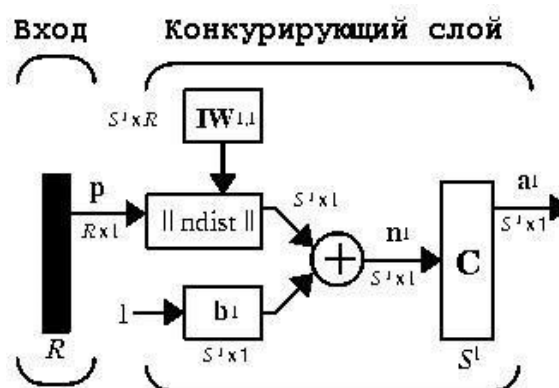


Рисунок 7.1. Архитектура слоя Кохонена

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):

$\text{net} = \text{newc}(\text{PR}, \text{S}, \text{KLR}, \text{CLR}),$

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 $e \cdot s$. 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 последовательности 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 поступает 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 незадействованными, 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 создании 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*:

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

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 [54]

```
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.

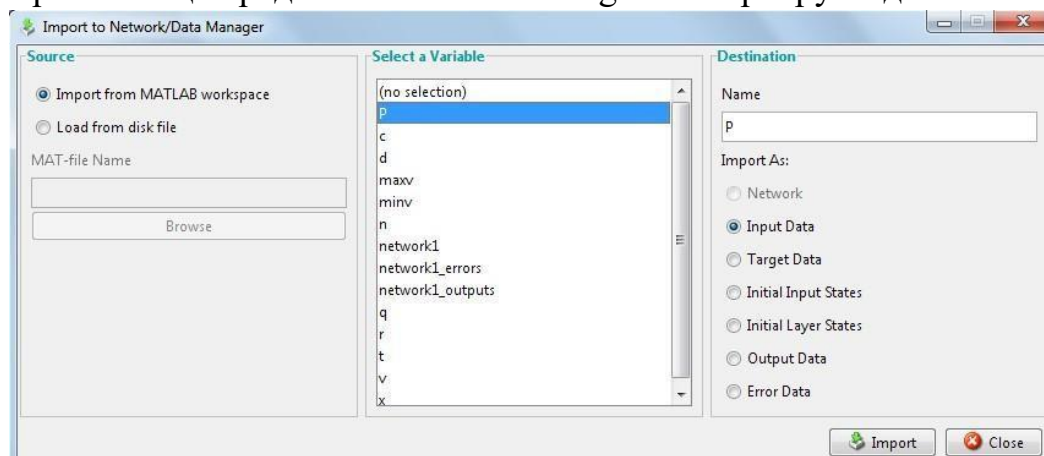


Рисунок 7.2. Окно для импорта и загрузки данных

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

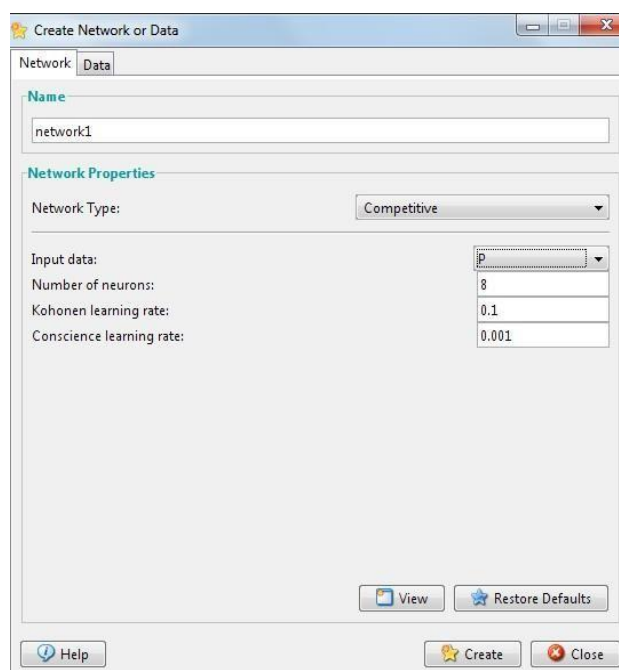


Рисунок 7.3. Окно создания нейронной сети

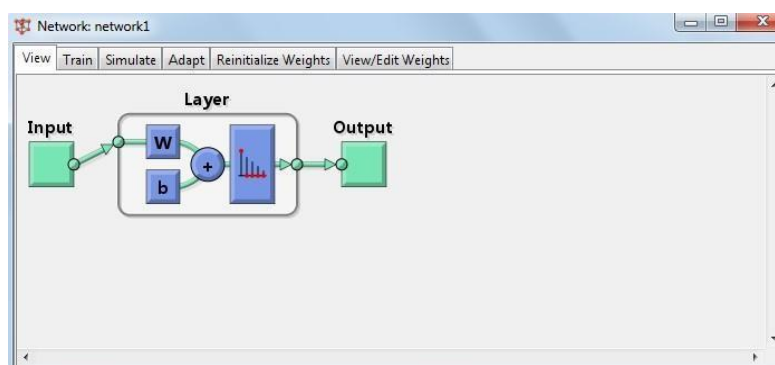


Рисунок 7.4. Диалоговая панель Network

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

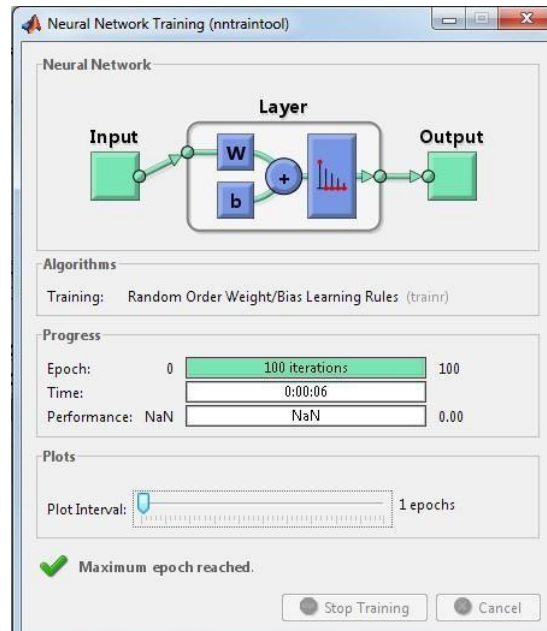


Рисунок 7.5. Окно обучения нейронной сети

Для иллюстрации обучения построим расположение входных данных.

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

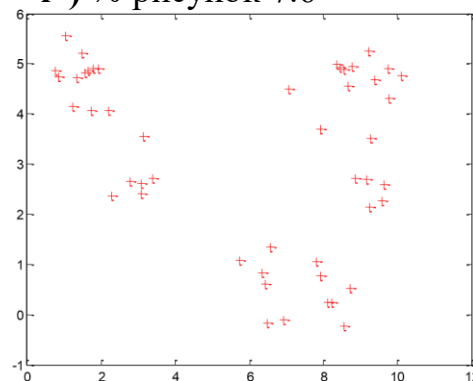
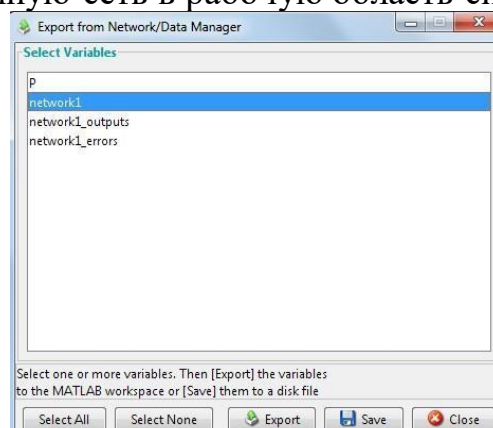
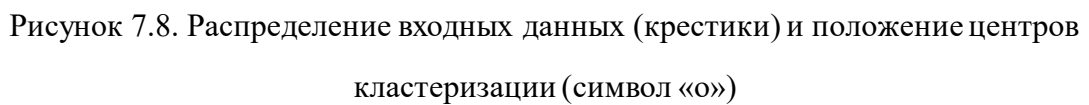


Рисунок 7.6. Распределение входных данных

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



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



The screenshot shows the "Network/Data Manager" application window. It has three main sections: "Input Data:" containing "p", "data1", and "data2"; "Target Data:" which is currently empty; and "Output Data:" containing "network1_outputs", "network3_outputs", and "network2_outputs". A small "Networks" panel above the output section lists "network1". An "Error Data:" section at the bottom right contains "network1_errors". A modal dialog titled "Data: network2_outputs" is open in the center, displaying a "Value" field with a list of numbers: [0., 0., 1., 0., 0., 0., 0., 0.]. The dialog has "OK" and "Cancel" buttons at the bottom.

```
data2 = [1.7; 8.7];
```

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

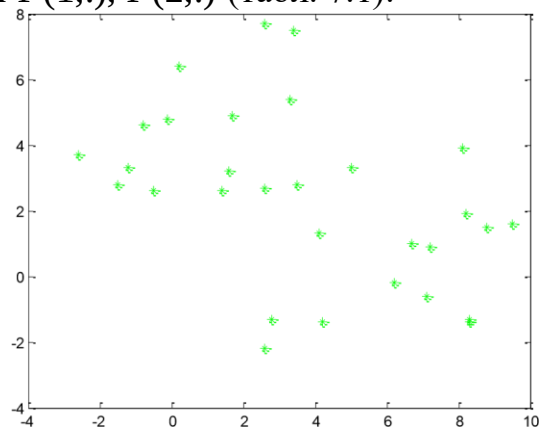
Yc = vec2ind(Y) % конвертирование выхода сети в номер класса; Y состоит из векторов, содержащих только по одной единице. Функция *vec2ind* показывает индексы единичных позиций.

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).



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

Рисунок 7.10. Распределение входных векторов

Таблица 7.1. Массив входных данных

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 (звёзды).

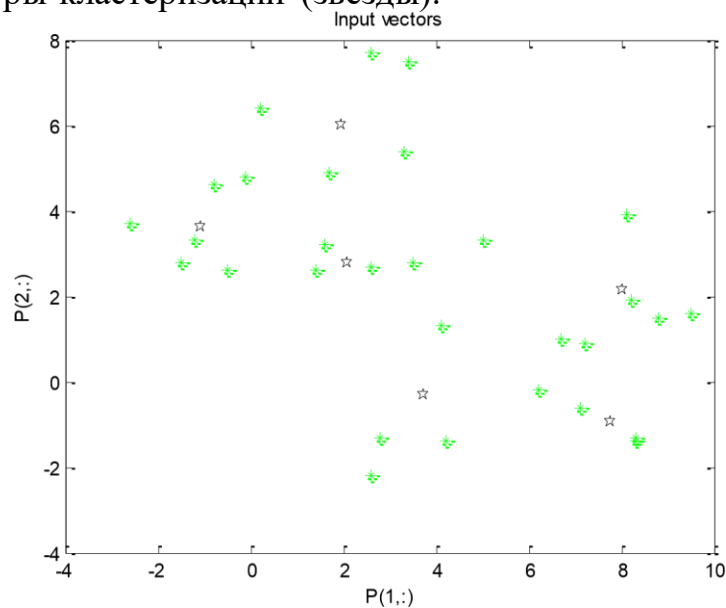


Рисунок 7.11. Распределение входных данных (крестики) и положение центров кластеризации (звёздочки)

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

```

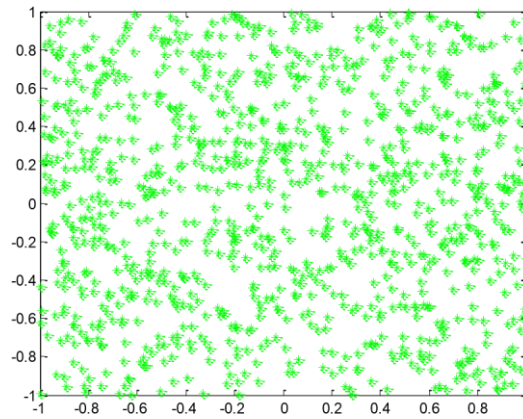


Рисунок 7.12. Множество исходных векторов для кластеризации

```
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 показывает 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.

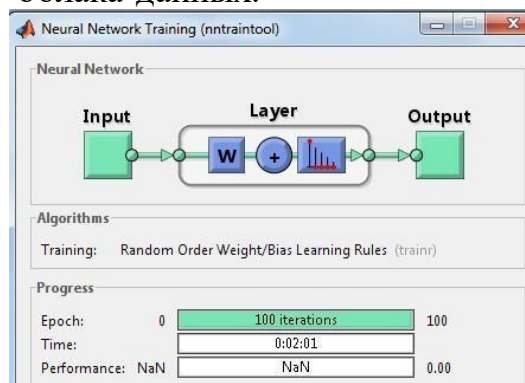


Рисунок 7.13. Окно обучения нейронной сети

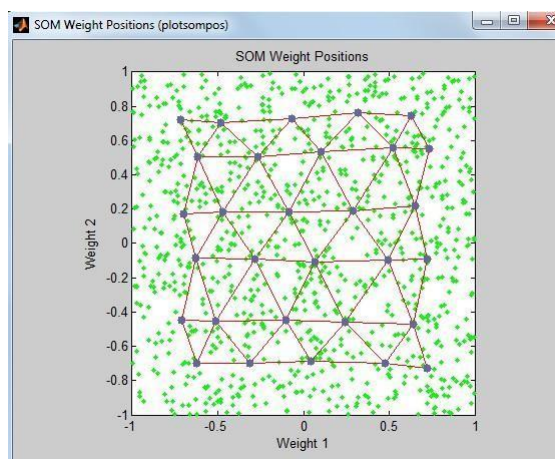


Рисунок 7.14. Расположение точек данных и весовых векторов

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.

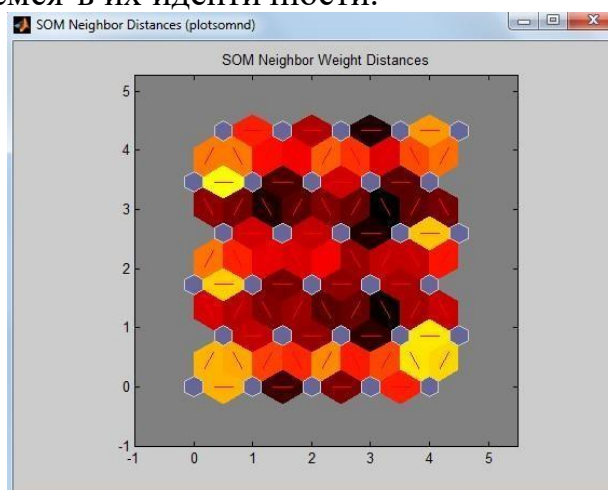


Рисунок 7.15. Сила взаимодействия (веса нейронов) соседних нейронов в карте Кохонена

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.

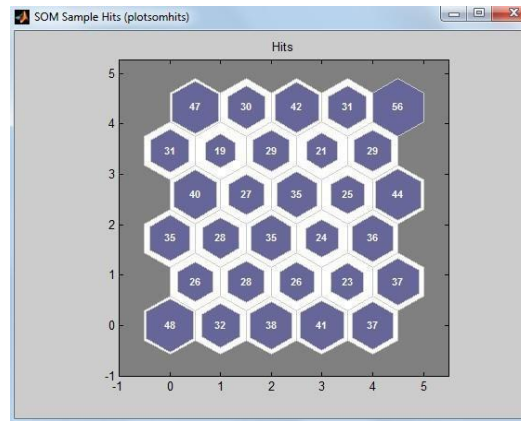


Рисунок 7.16. Количество точек данных, связанных с каждым нейроном

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.

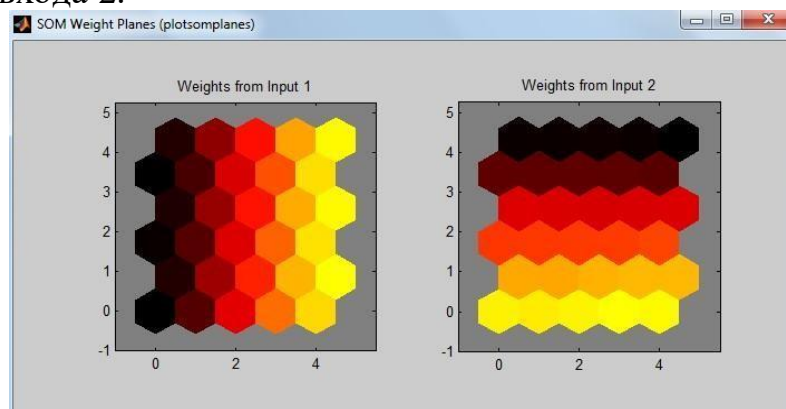


Рисунок 7.17. Визуализация весов, соединяющих каждый вход с каждым из

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

SOM

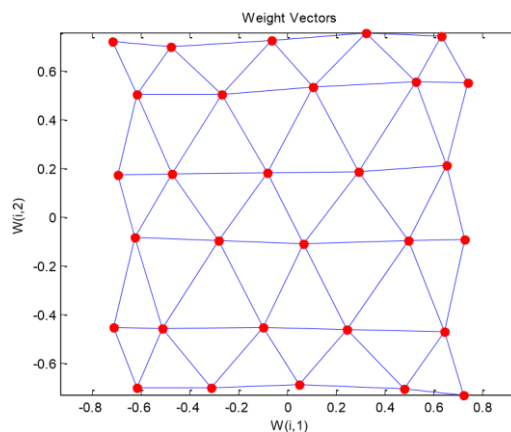


Рисунок 7.18. Самоорганизующаяся карта кластеризации

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

1. 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)];

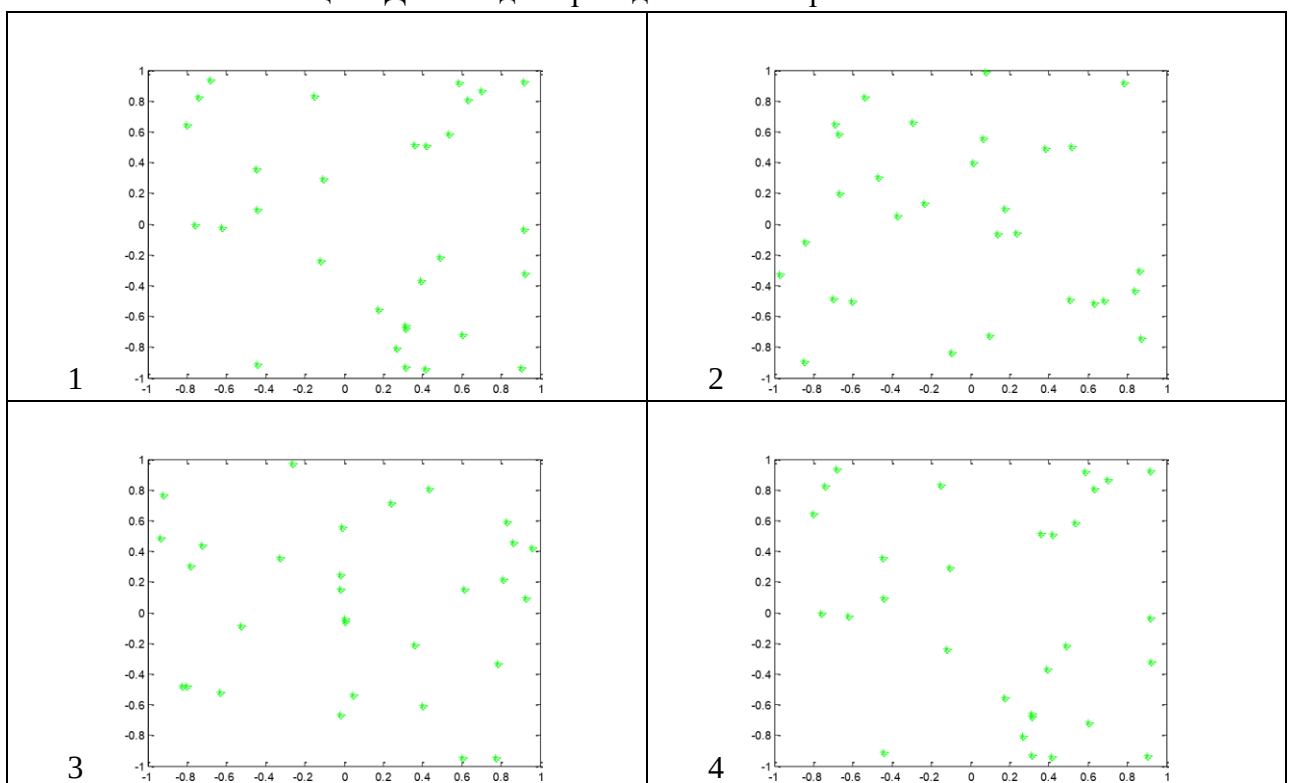
```

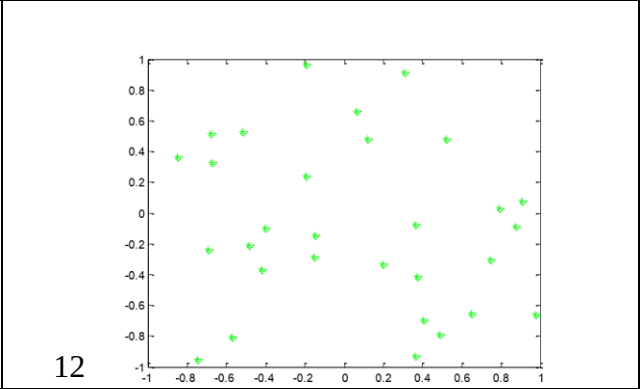
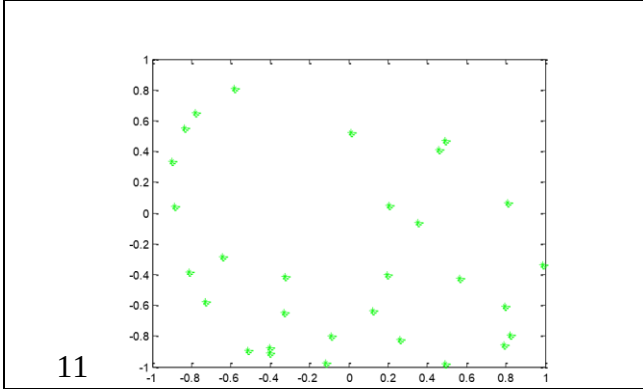
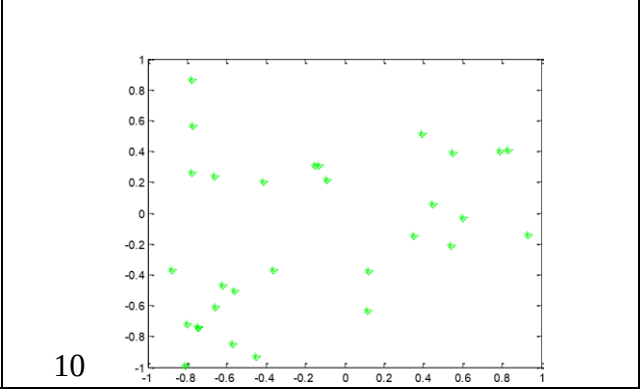
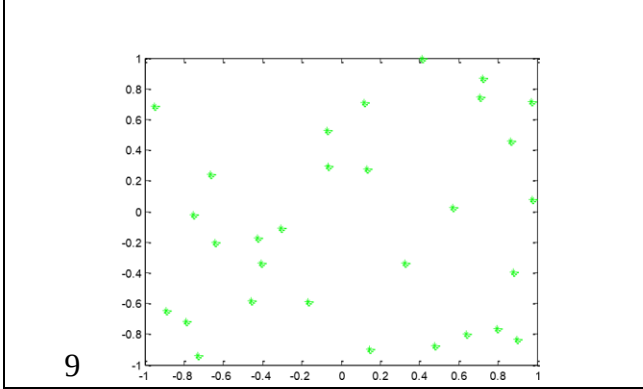
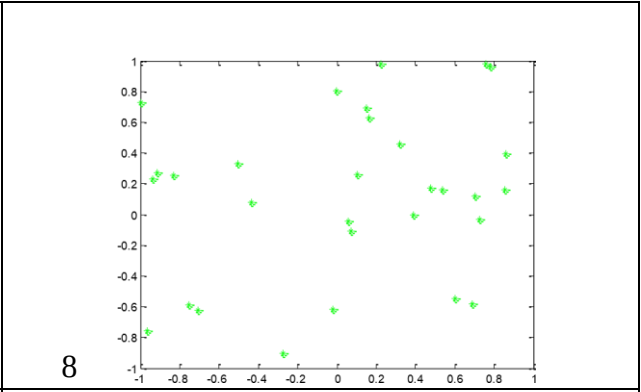
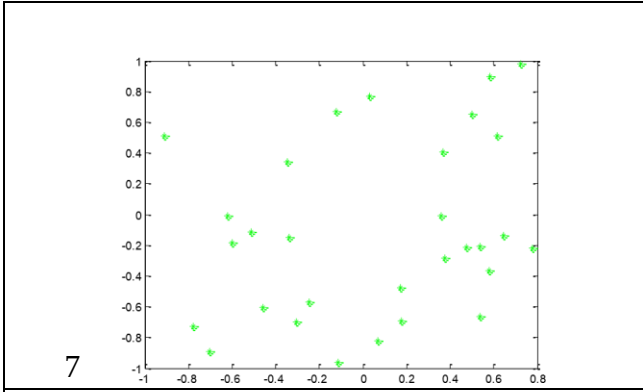
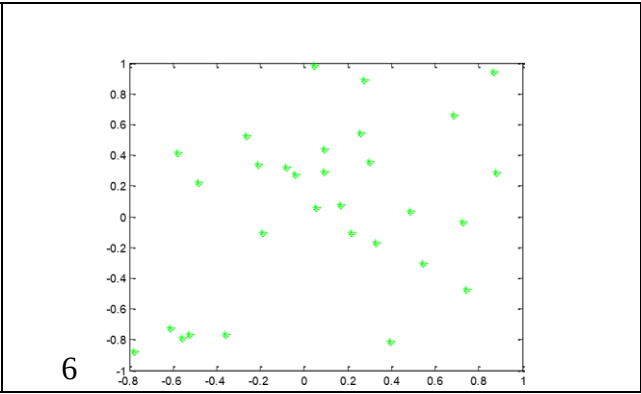
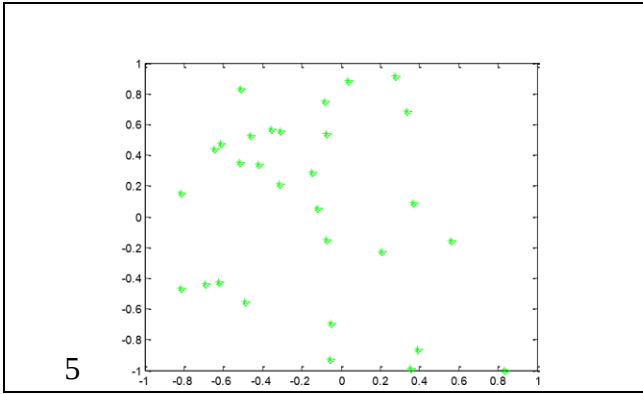
plot(P(1, 1:10:end), P(2, 1:10:end), '*8')
hold on
net=newsom([0 1 ;0 1], [10]);
net.trainparam.epochs=2000;
net.trainparam.show=100;
[net,tr]=train(net,P);
plotsom(net.iw{1,1}, net.layers{1}.distances) figure(2)
a=sim(net,P) % – моделирование на обучающем
% множестве и построение
% столбцовой диаграммы.
a=sim(net,[1;0]) % – отнесен к 10-му
кластеру

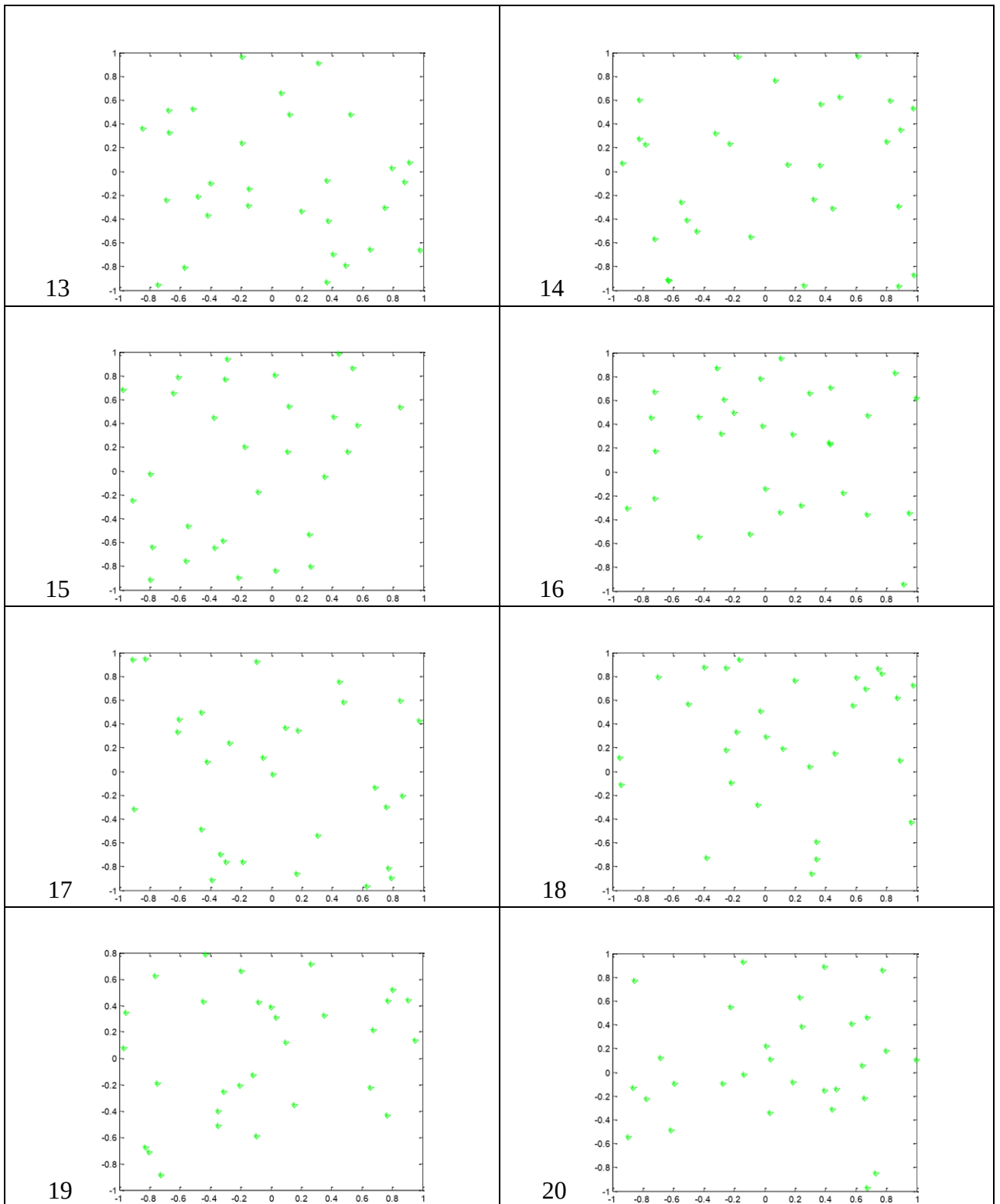
```

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.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 16

Решение оптимизационных задач с использованием сети Хопфилда

Цель занятия:

1. Углубить и закрепить знания, полученные на лекциях темы по основным понятиям нейронных сетей.
2. Выработать у студентов практические навыки при исследовании возможностей применения нейронных сетей для решения оптимизационных задач при помощи пакета прикладных программ MATLAB.

Порядок и последовательность выполнения работы

1. Создание исходных данных.
2. Создание компьютерной модели рекуррентной нейронной сети на основе исходных данных..
2. Симулирование сети произвольными данными.

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.

Конфигурации сетей с обратными связями

На рис. 1 показана сеть с обратными связями, состоящая из двух слоев. Способ представления несколько отличается от использованного в работе Хопфилда и других, но эквивалентен им с функциональной точки зрения, а также хорошо связан с сетями, рассмотренными в предыдущих главах. Нулевой слой, как и на предыдущих рисунках, не выполняет вычислительной функции, а лишь распределяет выходы сети обратно на входы. Каждый нейрон первого слоя вычисляет взвешенную сумму своих входов, давая сигнал NET, который затем с помощью нелинейной функции F преобразуется в сигнал OUT. Эти операции сходны с нейронами других сетей.

Бинарные системы

В первой работе Хопфилда функция F была просто пороговой функцией. Выход такого нейрона равен единице, если взвешенная сумма выходов с других нейронов больше порога T_j , в противном случае она равна нулю. Он вычисляется следующим образом:

$$NET_j = \sum_{i \neq j} w_{ij} OUT_i + IN_j, \quad (1)$$

$$OUT_j = 1, \text{ если } NET_j > T_j,$$

$$OUT_j = 0, \text{ если } NET_j < T_j,$$

$$OUT \text{ не изменяется, если } NET_j = T_j,$$

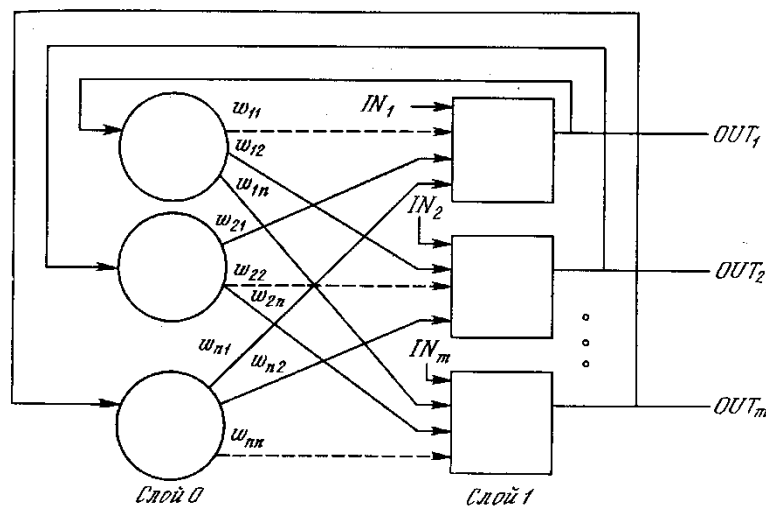


рис. 1. Однослойная сеть с обратными связями

Пунктирные линии обозначают нулевые веса

Состояние сети – это просто множество текущих значений сигналов OUT 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 сигналом OUT некоторого нейрона.

Функционирование сети легко визуализируется геометрически. На рис. 2 показан случай двух нейронов в выходном слое, причем каждой вершине квадрата соответствует одно из четырех состояний системы (00, 01, 10, 11). На рис. 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.

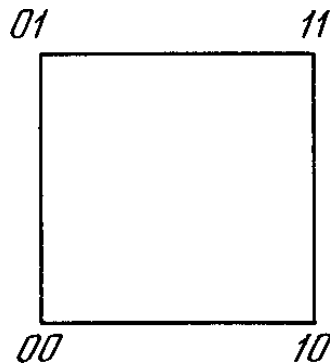


Рис. 2. Два нейрона порождают систему с четырьмя состояниями

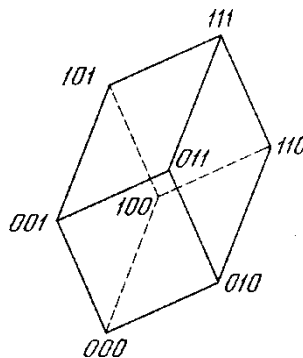
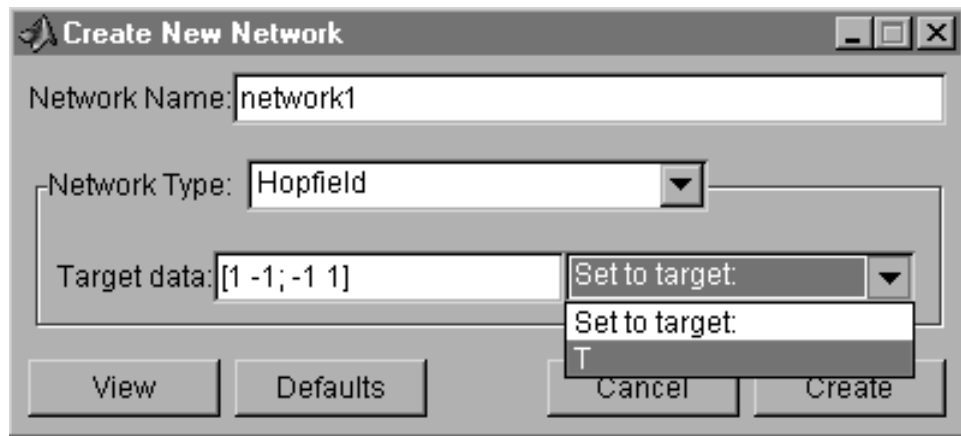


Рис. 3. Три нейрона порождают систему с восемью состояниями. Когда подается новый входной вектор, сеть переходит из вершины в вершину, пока не стабилизируется. Устойчивая вершина определяется сетевыми весами, текущими входами и величиной порога. Если входной вектор частично неправилен или неполон, то сеть стабилизируется в вершине, ближайшей к желаемой.

Пример 2. Для создания нейронной сети Хопфилда необходимо в рабочей среде MatLab создать вектор, определяющий стабильные состояния сети

$$T=[1 \ -1; \ -1 \ 1]$$

Импортируем полученный вектор как цель в NNTools Manager и создадим сеть



Зададим случайное состояние сети
 $a = \{\text{rand}(2,1)\}$

Симулируем сеть

```
[Y, Pf, Af] = sim (network1, {1,20}, {}, a);
```

Подготовим данные для иллюстрации этого примера

```
plot (T(1,:),T(2,:), '*r')
```

```
hold on
```

```
s = cell2mat(a);
```

```
r = [cell2mat(a) cell2mat(Y)];
```

```
plot (s(1,1),s(2,1),'+b',r(1,:),r(2,:))
```

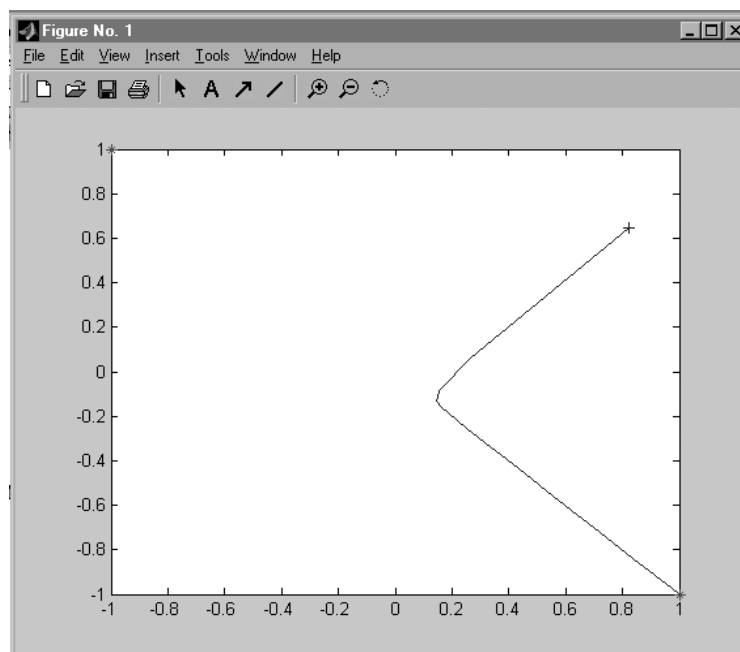


Рис. 5. – График обучения нейронной сети Хопфилда

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ:

Построить модель нейронной сети Хопфилда, состоящей из трех нейронов, имеющую два устойчивых состояния. Симулировать ее случайным вектором. Построить график обучения нейронной сети.

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Программа и цель работы.
2. Задание на работу.
3. Структура сети, график обучения нейронной сети Хопфилда.
4. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какую нейронную сеть называют рекуррентной?
2. Какие рекуррентные нейронные сети Вы знаете?
3. Как определяется матрица весовых коэффициентов нейронной сети Хопфилда?
4. Как происходит обучение нейронной сети Хопфилда?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 17

Нейросетевые системы управления

Цель занятия:

Исследовать компьютерные модели нейросетевых систем управления.

Порядок проведения занятия

1. Регулятор с предсказанием.
2. Регулятор NARMA-L2.
3. Регулятор на основе эталонной модели.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

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.

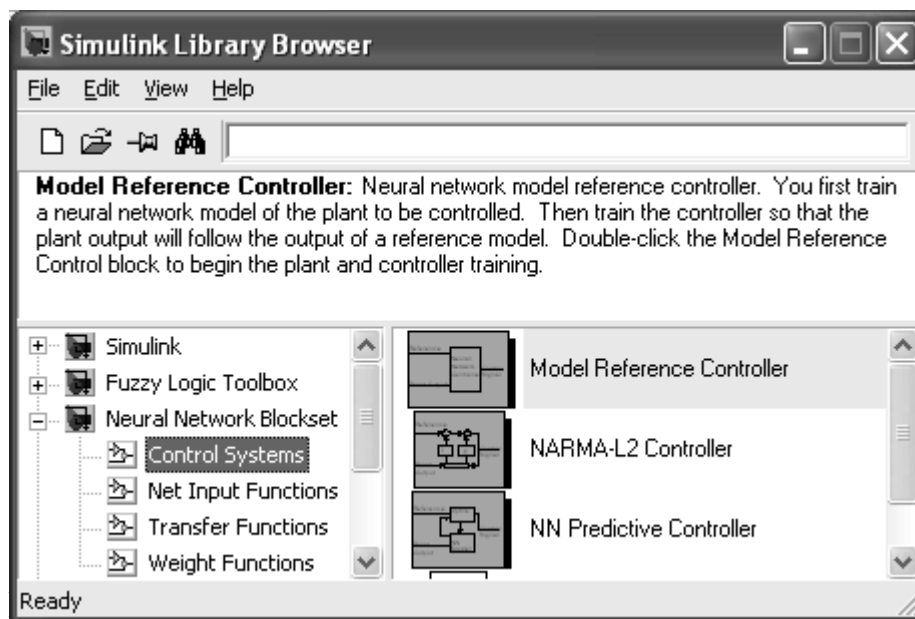
При управлении на основе модели авторегрессии со скользящим

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.

Каждый вопрос занятия 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 Simulink.

Динамические модели систем управления с нейросетевыми регуляторами размещены в специальном разделе Control Systems набора блоков Neural Network Blocksets (рисунок 1) и включают три упомянутые выше модели регуляторов, а также блок построения графиков.



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. Из 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 состояний, 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 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 $e = u_p - u_m$, на последовательность пробных сигналов u .

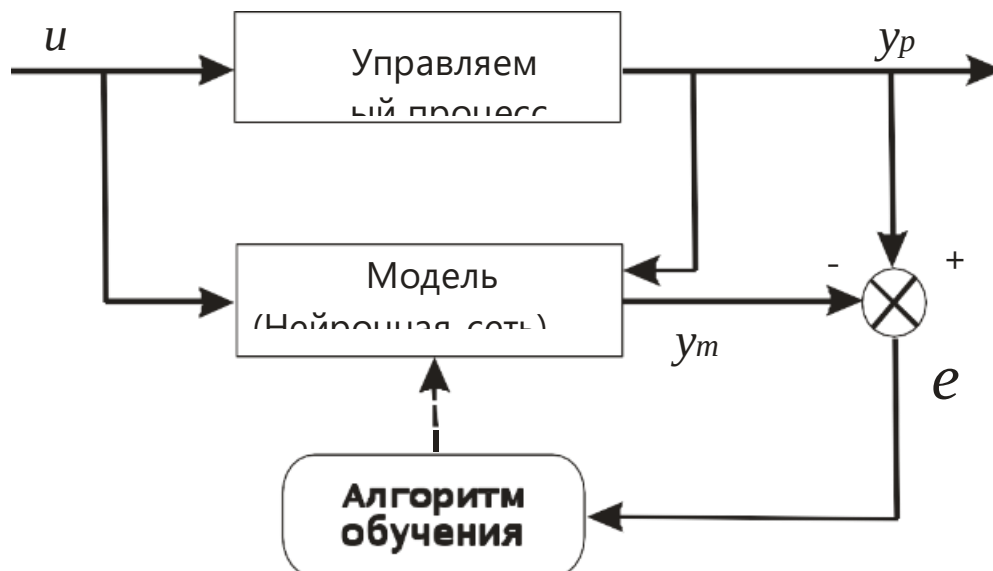


Рисунок 2 – Схема подсистемы идентификации

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.

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:

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

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 на рисунке 3 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.

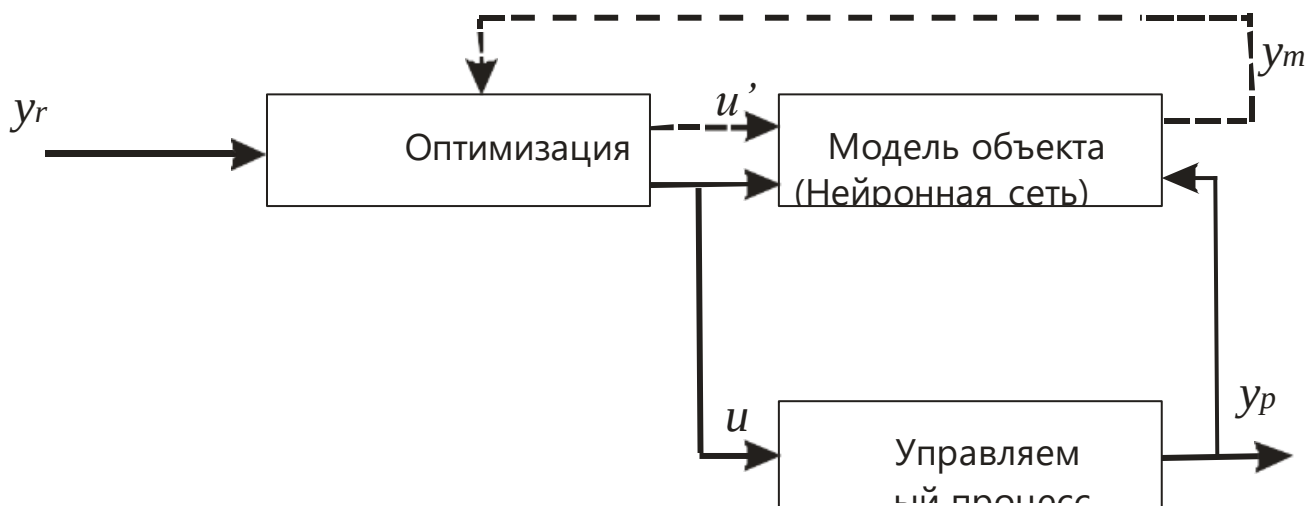


Рисунок 3 – Процесс управления с предсказанием

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 4.

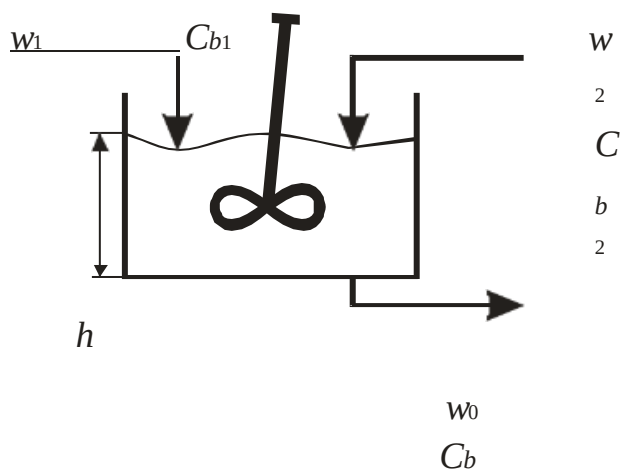


Рисунок 4 – Схема каталитического реактора с непрерывным смешиванием

Динамическая модель управляемого процесса описывается следующими обыкновенными нелинейными дифференциальными уравнениями:

$$\begin{cases}
 \frac{dh}{dt} = w_1(t) + w_2(t) - 0,2 \sqrt{h}; \\
 \frac{dC_b}{dt} = \frac{w_1(t) C_{b1} + w_2(t) C_2 - w_0 C_b}{h}
 \end{cases} \quad (2)$$

$$= (C_{b1} - C_b) h + (C_{b2} - C_b) h - \frac{h}{(1 + k_2 C_b)^2}$$

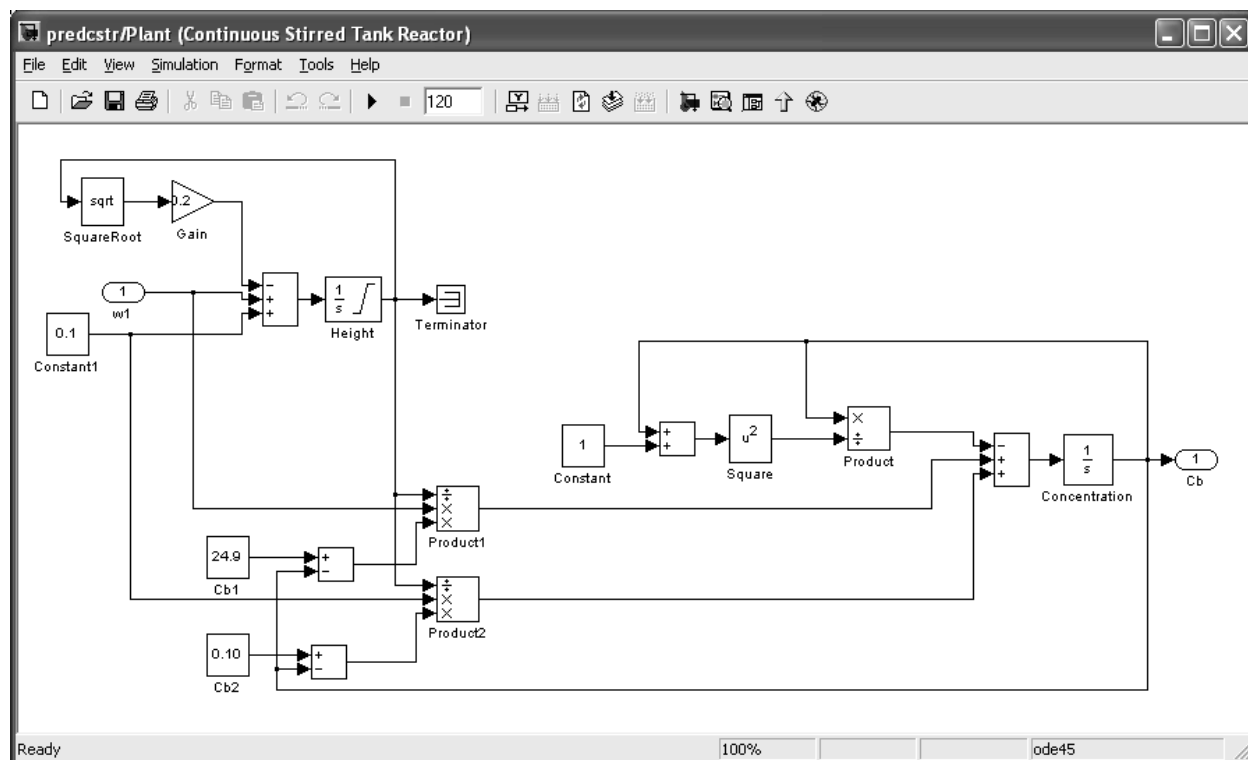
где h – уровень жидкости в резервуаре; $w_1(t)$ - скорость потока продукта с концентрацией C_{b1} ; $w_2(t)$ – скорость потока разбавленного продукта с концентрацией C_{b2} ; C_b – концентрация продукта на выходе объекта.

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 каталитического реактора, соответствующая уравнениям (2), показана на рисунке 5.



Выполнить запуск демонстрационного примера можно несколькими способами:

- в окне запуска приложений Launch Pad выбрать опцию Demos для пакета Neural Network Toolbox;
- ввести команду `predcstr`, используя командное окно системы MATLAB.

В любом случае открывается окно системы Simulink со следующей структурой системы управления (рисунок 6).

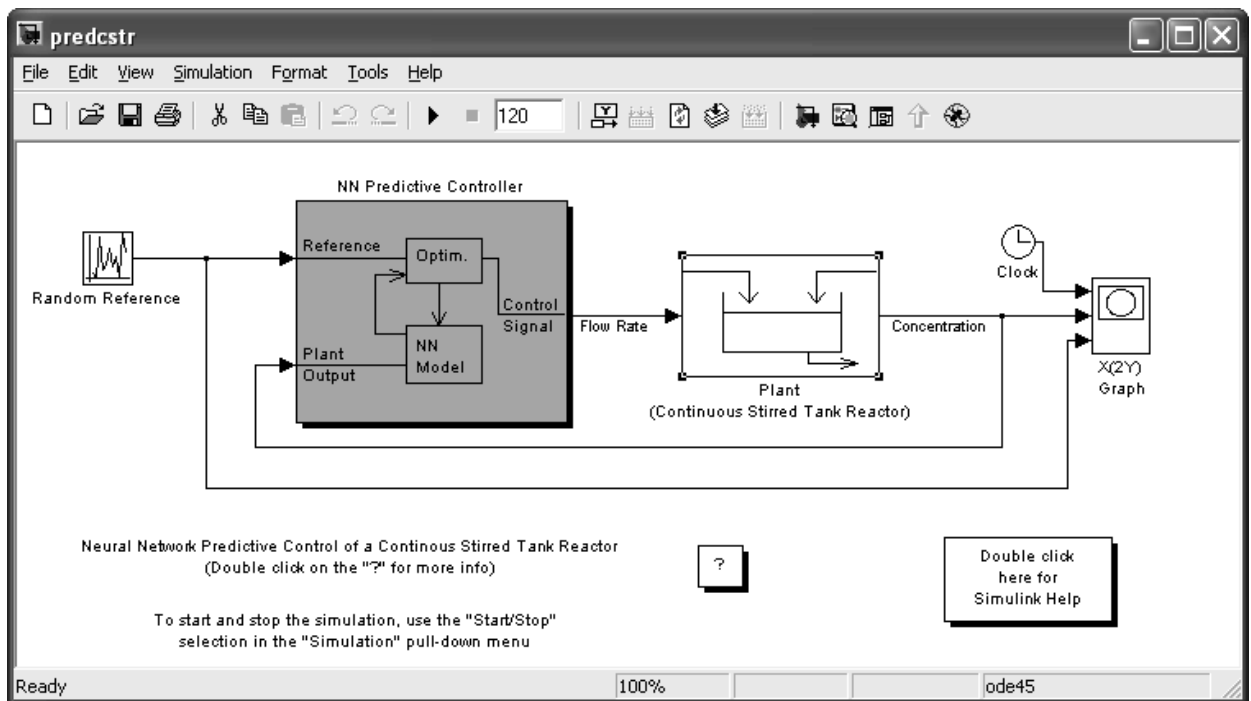


Рисунок 6 – Структура системы управления в Simulink

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. Для того чтобы начать работу, необходимо активизировать блок NN Predictive Controller двойным щелчком левой кнопки мыши. Появится окно, показанное на рисунке 7.

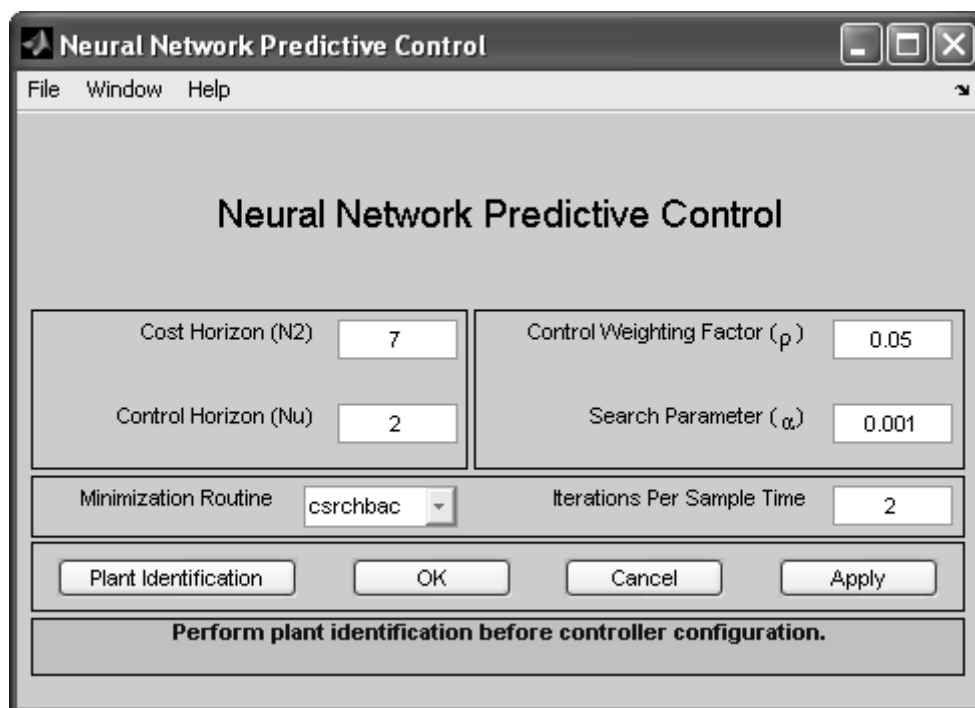
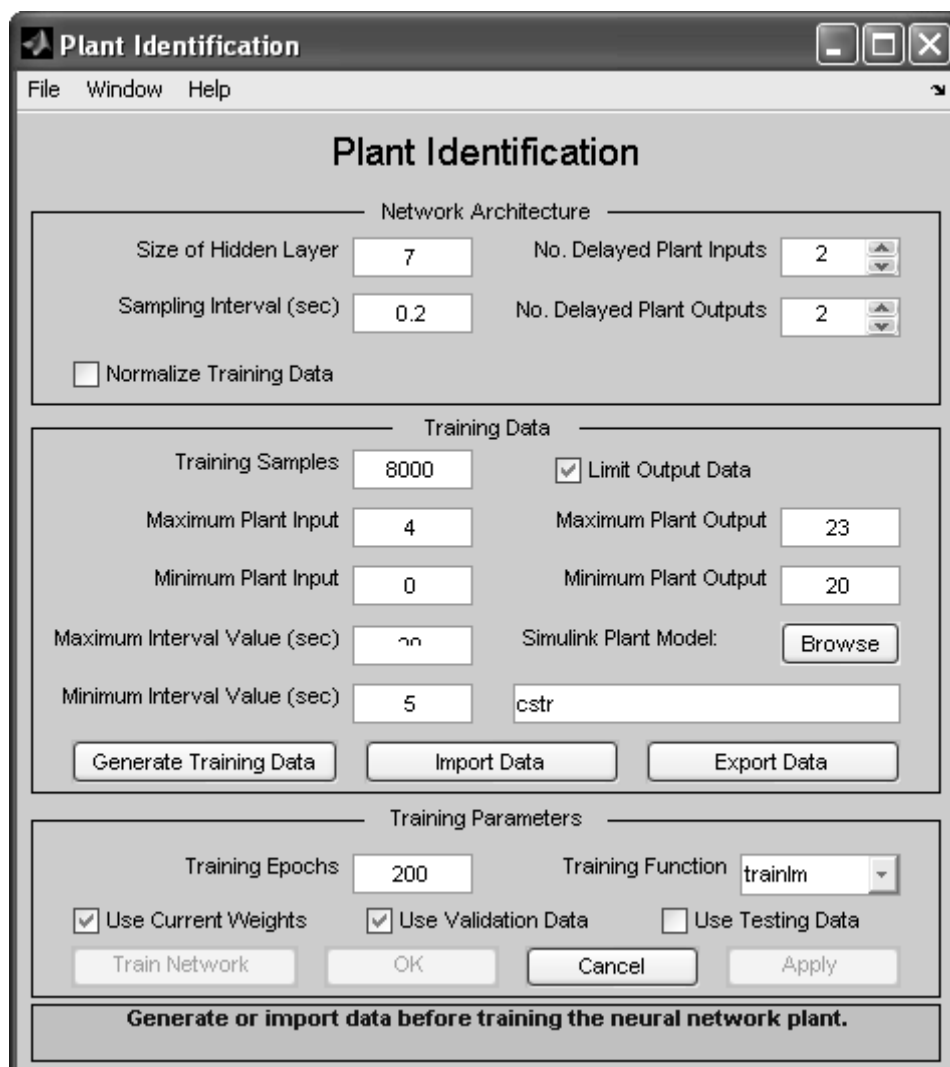


Рисунок 7 – Графический интерфейс пользователя нейросетевой системы управления с предсказанием

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.

Вид окна Plant Identification приведен на рисунке 8. 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 каталитического реактора CSTR.



Plant Identification

File Window Help

Network Architecture

Size of Hidden Layer: 7

No. Delayed Plant Inputs: 2

Sampling Interval (sec): 0.2

No. Delayed Plant Outputs: 2

☐ Normalize Training Data

Training Data

Training Samples: 8000

☒ Limit Output Data

Maximum Plant Input: 4

Maximum Plant Output: 23

Minimum Plant Input: 0

Minimum Plant Output: 20

Maximum Interval Value (sec): inf

Simulink Plant Model: Browse

Minimum Interval Value (sec): 5

cstr

Generate Training Data Import Data Export Data

Training Parameters

Training Epochs: 200

Training Function: trainlm

☒ Use Current Weights ☒ Use Validation Data ☐ Use Testing Data

Train Network OK Cancel Apply

Generate or import data before training the neural network plant.

Рисунок 8 – Окно процедуры идентификации

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).

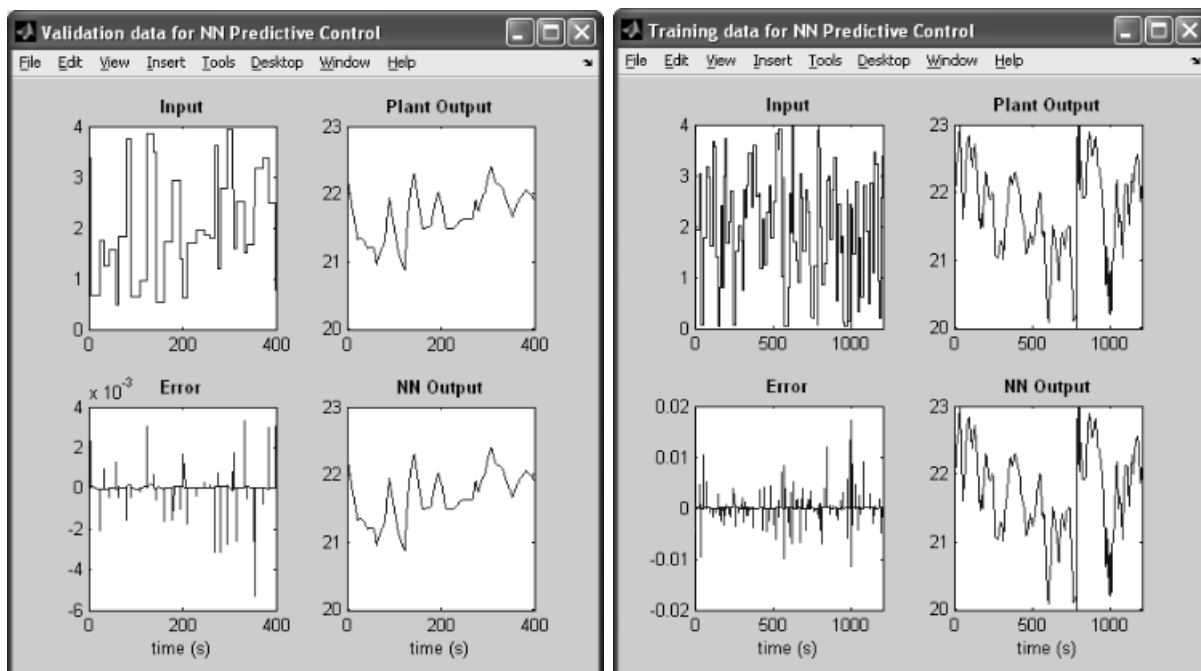


Рисунок 9 – Графики входного и выходного сигналов объекта управления

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. После завершения обучения его результаты отображаются на графиках, как это показано на рисунке 9, где построены соответственно результаты обучения и тестирования на контрольном множестве.

Текущее состояние отмечено в окне Plant Identification (рисунок 9) 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 10).

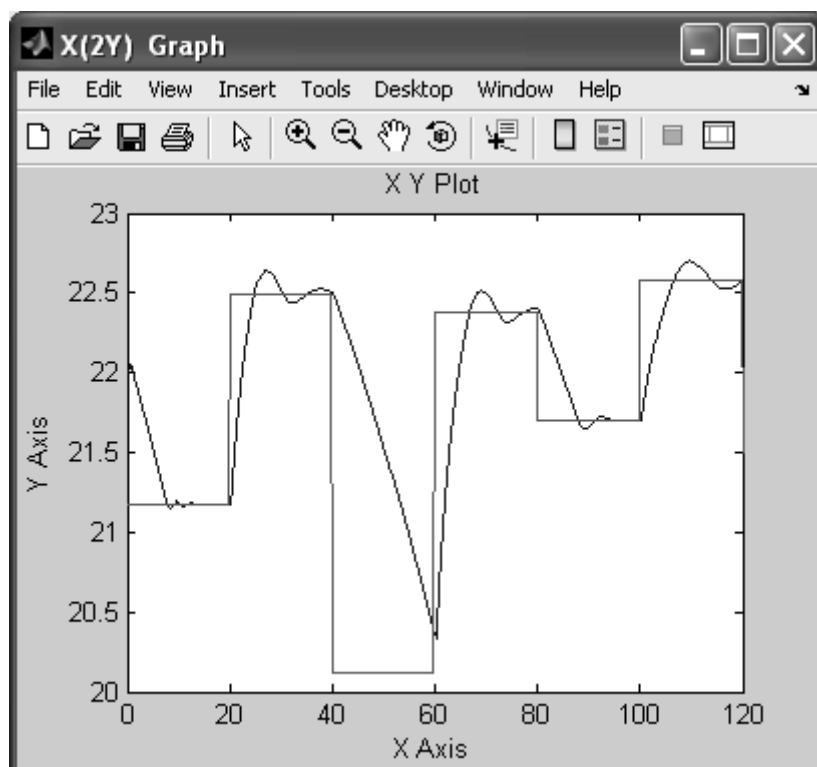


Рисунок 10 – Процесс моделирования

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 20 с все воздействия эффективно отрабатываются. Таким образом, регулятор с предсказанием, реализованный в виде нейронной сети, можно использовать для управления каталитическим реактором с непрерывным перемешиванием.

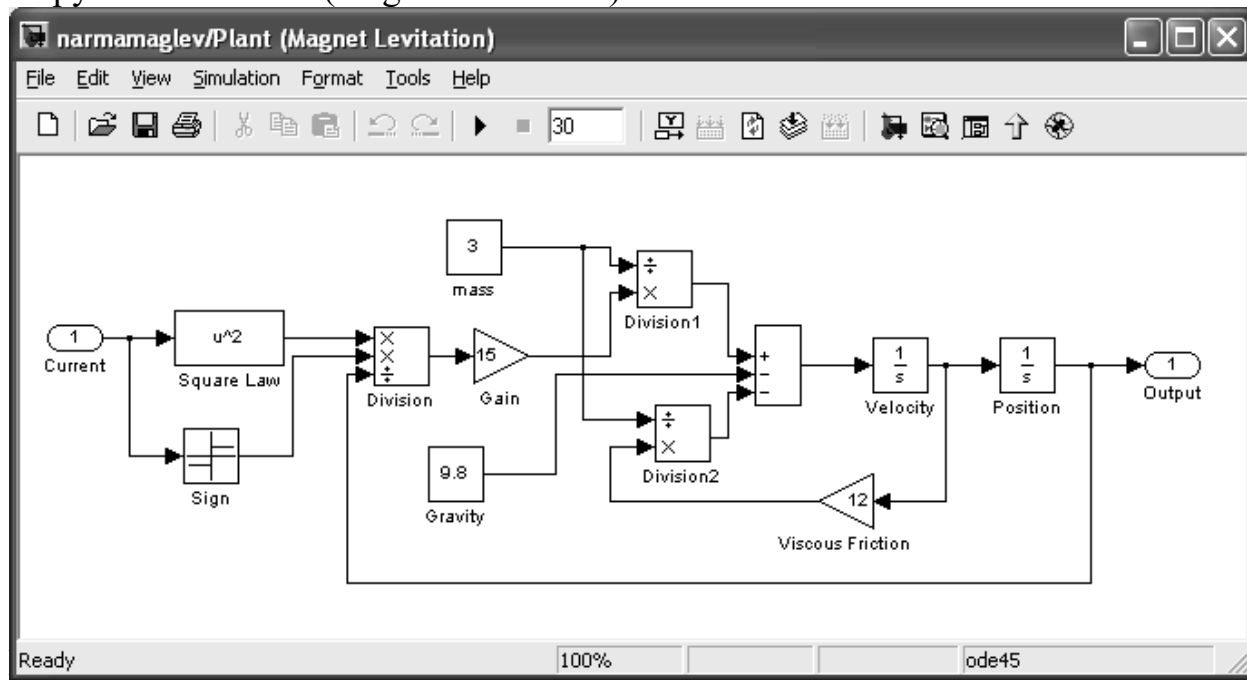
2. Регулятор NARMA-l2

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 11.

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 13. 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.



Рассматриваемая динамическая система является нелинейной.

Построим ее дискретную нелинейную модель, как авторегрессионную модель со скользящим средним, или NARMA-модель, в форме

$$y(k+d) = N[y(k), y(k-1), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-n+1)], \quad (4)$$

где $y(k)$ - выход модели; d - число тактов предсказания; $u(k)$ - вход модели.

На этапе идентификации необходимо построить нейронную сеть для NARMA-модели вида (4). Эта процедура аналогична описанной выше процедуре идентификации для регулятора с предсказанием.

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

$$y(k+d) = y_r(k+d), \quad (5)$$

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

$$u(k) = G[y(k), y(k-1), \dots, y(k-n+1), y_r(k+d), u(k-1), \dots, u(k-m+1)]. \quad (6)$$

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.

Выполнить запуск демонстрационного примера можно несколькими способами:

- в окне запуска приложений Launch Pad выбрать опцию Demos для ППП Neural Network Toolbox;
- ввести команду `parmamaglev` в командном окне системы MATLAB.

Для того чтобы начать работу, необходимо активизировать блок NARMA-L2 Controller двойным щелчком левой кнопки мыши. Появится окно, показанное на рисунке 14.

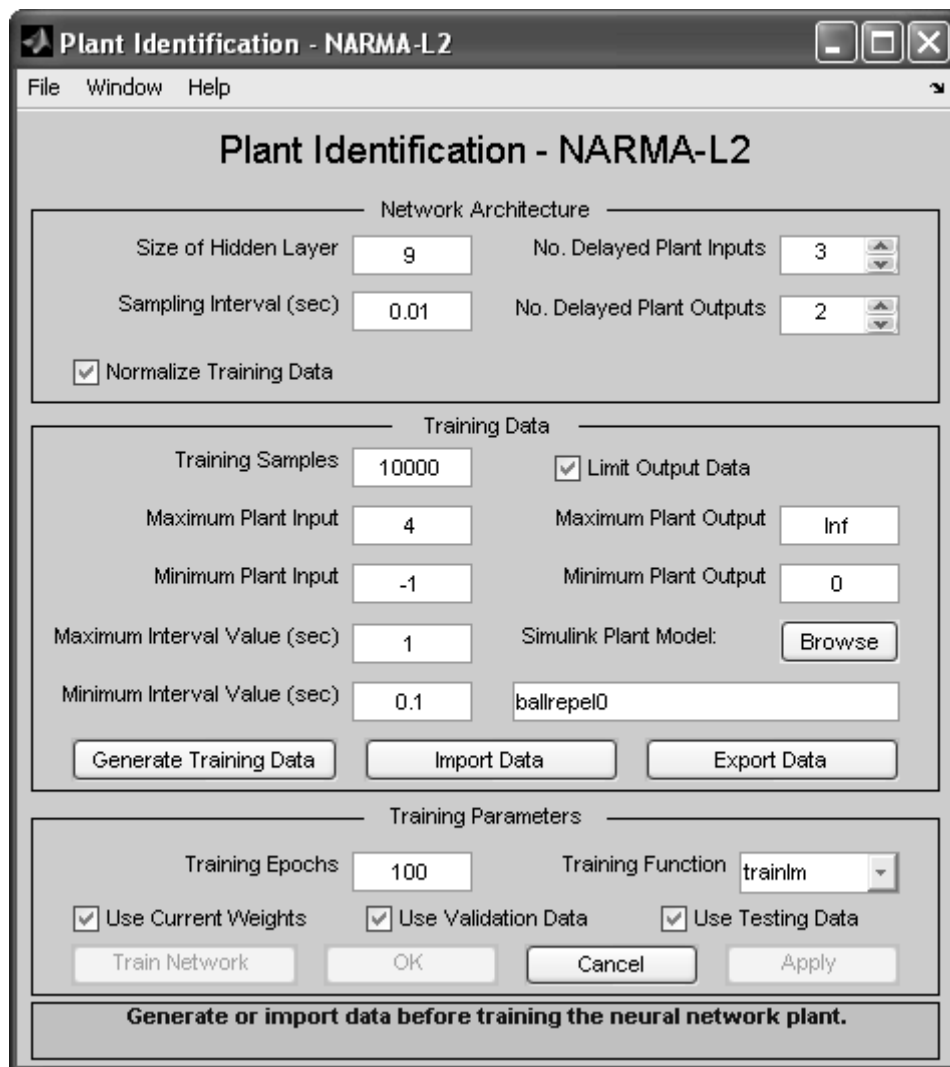


Рисунок 14 - Окно примера NARMA-L2

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.

После окончания обучения нажать на клавишу ОК для ввода данных регулятора в модель Simulink. Возвращаемся к модели Simulink (см. рисунок 11) и начинаем моделирование, выбрав опцию Start из меню Simulation. Графики задающего сигнала и выхода системы приведены на рисунке 15.

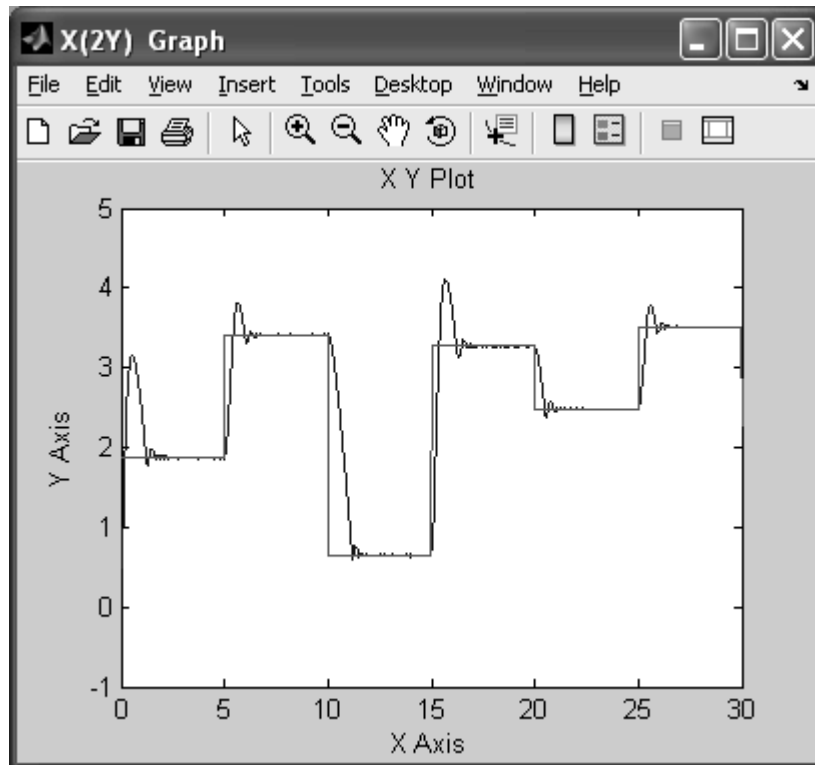


Рисунок 15 - Задающий и выходной сигналы системы управления

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 5 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 работа показана на рисунке 16.

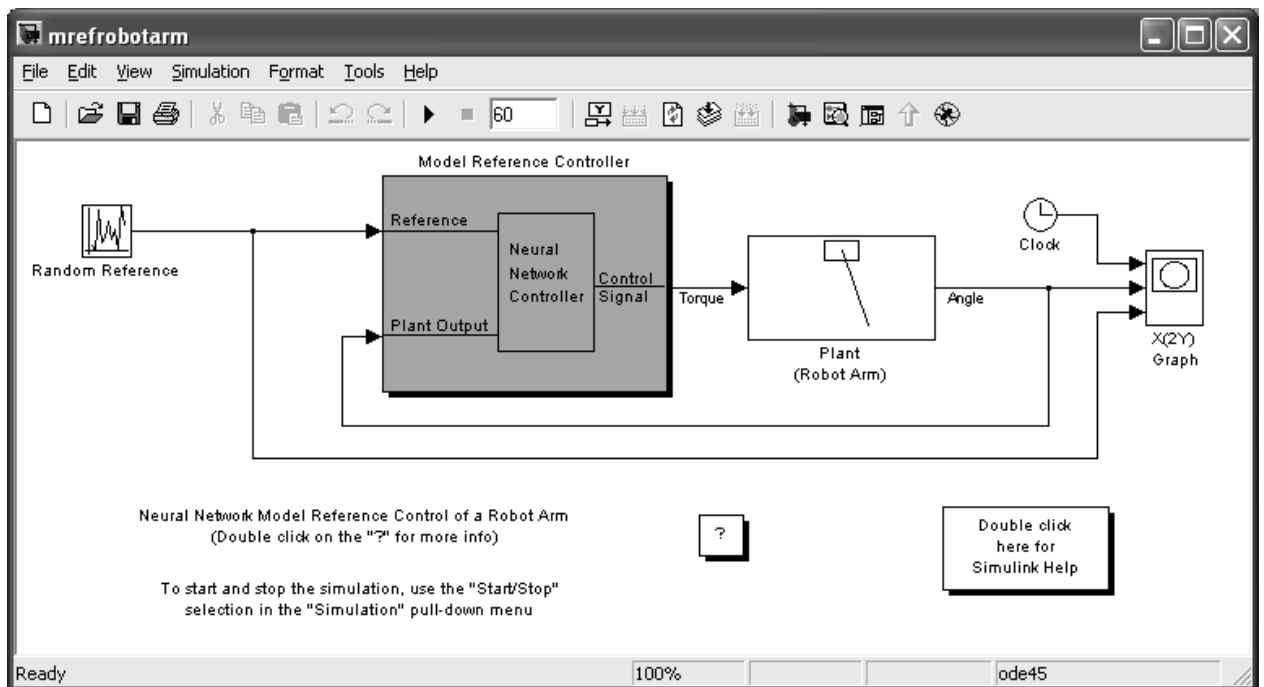


Рисунок 16 - Схема демонстрационного примера регулятора на основе эталонной модели

В этом демонстрационном примере цель состоит в управлении движением одного звена робота, как это показано на рисунке 17.

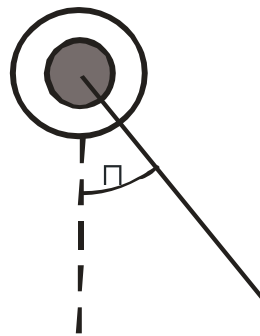


Рисунок 17 - Схема управления движением звена робота

Уравнения движения звена:

$$\frac{d^2\varphi}{dt^2} = -10\sin\varphi - 2\frac{d\varphi}{dt} + u,$$

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

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

$$\frac{d^2y}{dt^2} = \frac{dy}{dt}$$

$$\frac{d^2 r}{dt^2} = -9r - 6 \frac{dr}{dt} + u, \quad (8)$$

где y_r - выход эталонной модели; r - задающий сигнал на входе модели.
 Соответствующая динамическая модель, реализованная в системе

Simulink, показана на рисунке 18.

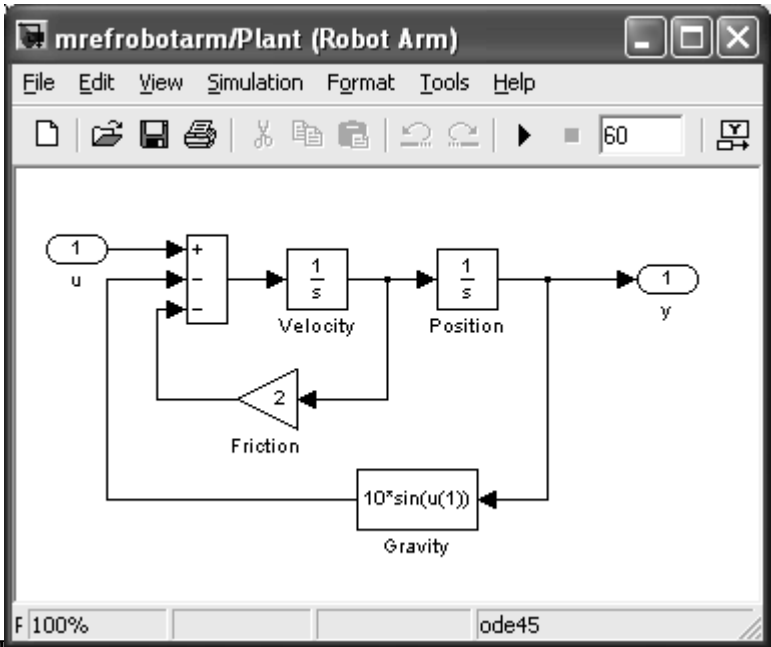


Рисунок 18 - Динамическая модель звена робота

Структурная схема, поясняющая принцип построения системы управления с эталонной моделью, показана на рисунке 19.

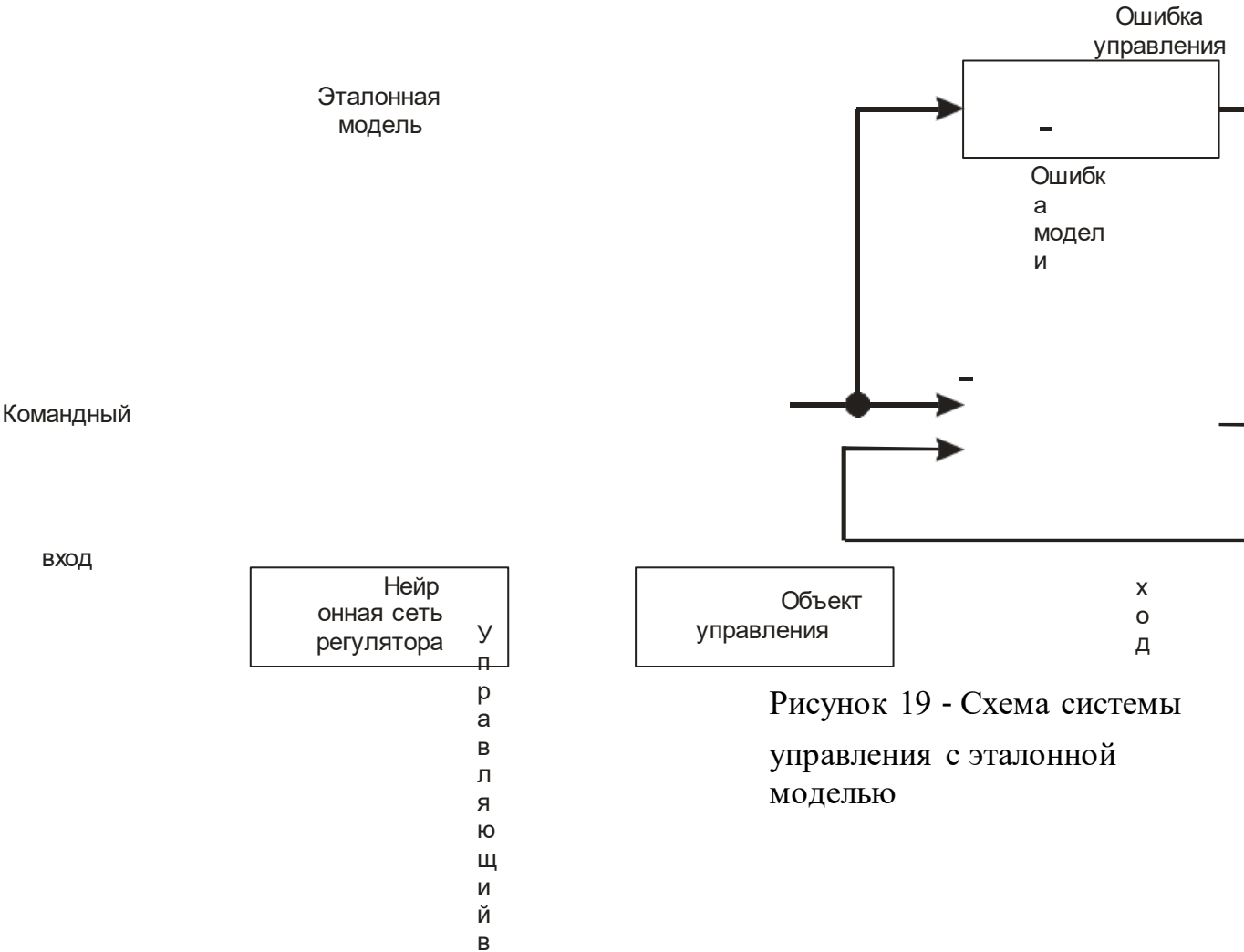


Рисунок 19 - Схема системы управления с эталонной моделью

Выход объекта

В ней следует выделить эталонную модель, которая задает желаемую траекторию движения звена робота, удовлетворяющую дифференциальному уравнению (7), а также нейронные сети, реализующие регулятор и модель объекта управления.

Архитектуру нейронной сети регулятора можно описать профилем 5- 13- 1 (5 входов, 13 нейронов скрытого слоя и 1 выход).

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 20.

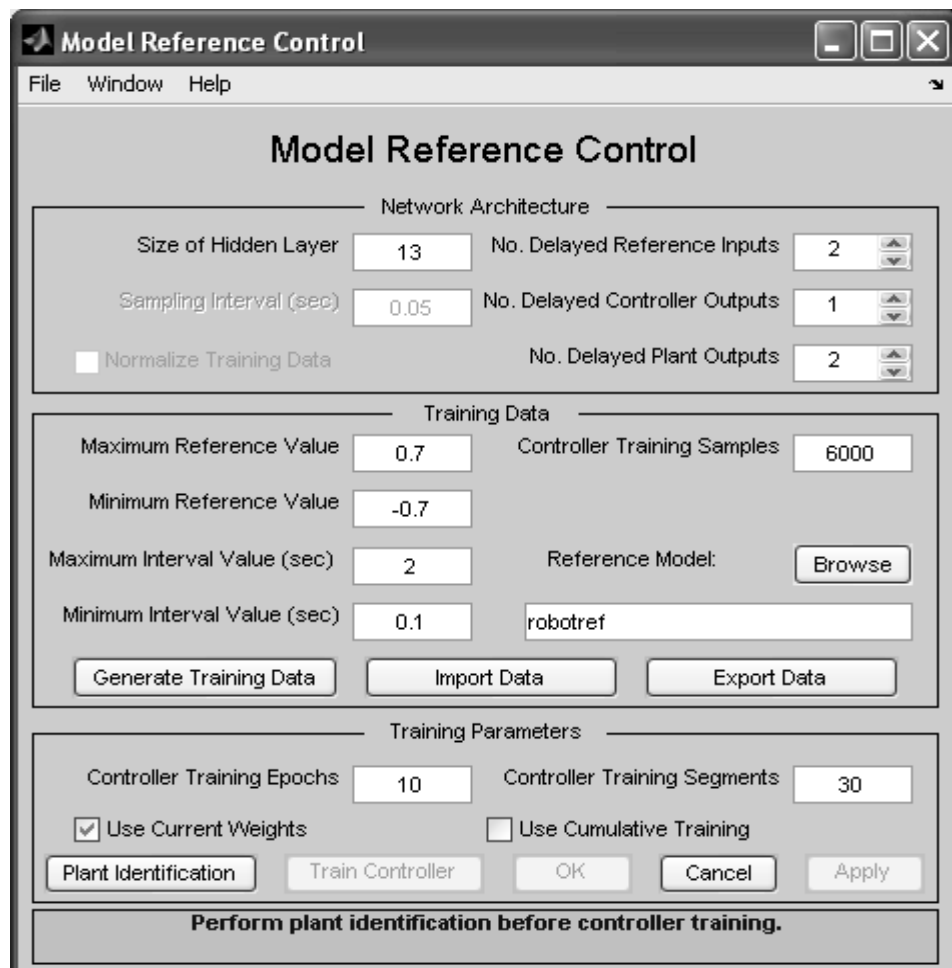


Рисунок 20 - Окно системы управления на основе эталонной модели

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 20, выбрать кнопку Plant Identification. При этом откроется окно Plant Identification, которое показано на рисунке 21.

Plant Identification

File Window Help

Plant Identification

Network Architecture

Size of Hidden Layer: 10 No. Delayed Plant Inputs: 2

Sampling Interval (sec): 0.05 No. Delayed Plant Outputs: 2

☐ Normalize Training Data

Training Data

Training Samples: 10000 ☒ Limit Output Data

Maximum Plant Input: 15 Maximum Plant Output: 3.1

Minimum Plant Input: -15 Minimum Plant Output: -3.1

Maximum Interval Value (sec): 2 Simulink Plant Model: Browse

Minimum Interval Value (sec): 0.1 robotarm

Generate Training Data Import Data Export Data

Training Parameters

Training Epochs: 300 Training Function: trainlm

☒ Use Current Weights ☒ Use Validation Data ☒ Use Testing Data

Train Network OK Cancel Apply

Generate or import data before training the neural network plant.

Рисунок 21 - 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 20).

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 вам 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 (см. рисунок 16) и начать моделирование, выбрав опцию Start из меню Simulation. Из анализа полученных данных (рисунок 22) следует, что реакция системы на ступенчатые воздействия со случайной амплитудой носит монотонный характер и отрабатывается в пределах 3 с. Это свидетельствует о хорошем качестве регулятора Model Reference Controller для управления звеном робота-манипулятора.

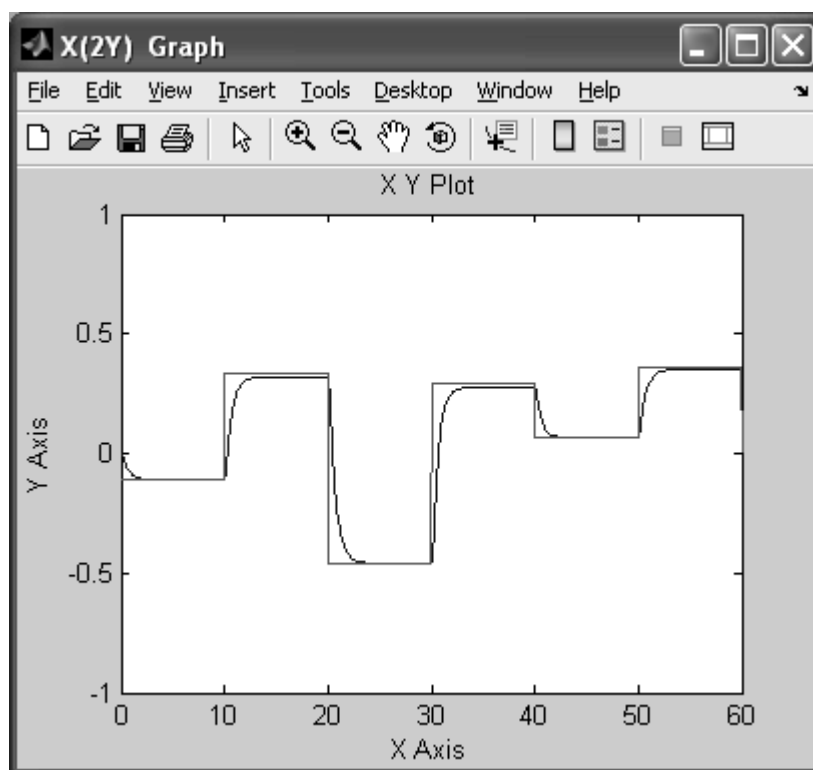


Рисунок 22 - Задающий и выходной сигналы системы управления на основе эталонной модели

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

При помощи пакета Neural Network ToolBox создайте три моделей нейросетевых систем управления, иницируйте входное воздействие и проведите моделирование. Полученный результат перенесите в отчет.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие модели нейронных систем управления рассмотрены в каждом вопросе лабораторной работе?
2. Какие физические процессы лежат смоделированы в каждом вопросе?

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 18-19

Применение нейронных сетей для предсказания временного процесса

Цель занятия:

1. Расширить и углубить знания, полученные на лекциях по основам теории нейронных сетей.
2. Приобрести первичные навыки по проведению исследований и экспериментов с использованием компьютерных моделей информационных систем.

ПОРЯДОК И ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ РАБОТЫ

1. Создание компьютерной модели нейронной сети предсказания временного процесса.
2. Обучение и адаптация сети. Получение погрешности предсказания.

МЕТОДИЧЕСКИЕ ПОЯСНЕНИЯ И РЕКОМЕНДАЦИИ

Пусть необходимо предсказать функцию $y(t) = (t^2 - 4 \cdot t) \sin(t^2 - 5 \cdot t)$.

Сформируем в рабочей области вектор `time`, описывающий временной отрезок, и вектор данных `F` предсказываемой функции:

```
time=0:.1:10;
```

```
F=(time.^2-4.*time).*sin(time.^2-5.*time);
```

и последовательность целей

```
P=F;
```

Обучающая последовательность должна быть представлена в виде последовательности ячеек

```
F=con2seq(F);
```

```
P=F;
```

Импортируем данные из рабочей области `NNTools`: вектор, описывающий предсказываемый процесс. Построим при помощи инструментального средства `Neural Network Tools` `Network/Data Manager` модель нейронной сети – линейный слой (обучаемый) `linear layer (train)`. Количество нейронов в слое – 1. Область изменения входных данных необходимо взять из последовательности `F`. Количество элементов задержки задается вектором в области `Input delay vector`. Точность обучения определяется в области `Learning rate`, в рассматриваемом примере 0.01

(рисунок 1).

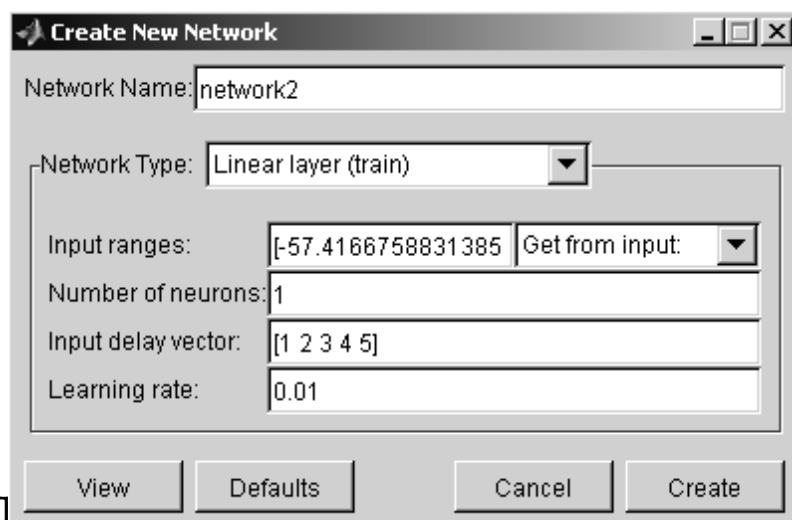


Рисунок 1 – Создание нейронной сети предсказания. Для предсказания функции, представленной в виде последовательности F , необходимо провести адаптацию сети (рисунок 2).

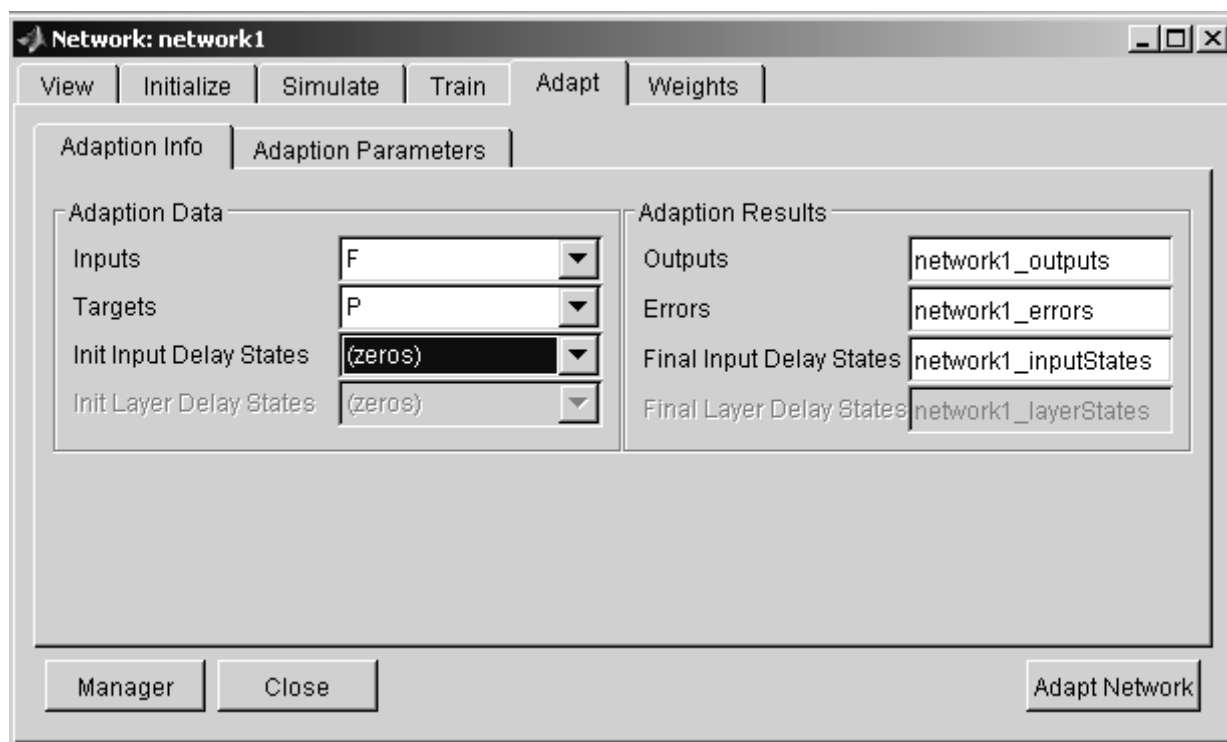


Рисунок 2 – Адаптация сети

Предсказанные значения функции будут сформированы в последовательности `network1_outputs`. Для извлечения информации необходимо экспортировать последовательность `network1_outputs` в рабочую область системы MatLAB и преобразовать ее в вектор

```
E=cat(2,network1_outputs{:});
```

и построить графики ошибки, функции и предсказания.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ:

Построить модель нейронной сети предсказания функции $f(t)$ на промежутке $[0, 10]$, заданной согласно таблице 1. Вычертить в отчете структуру нейронной сети прогнозирования, построить графики предсказываемой функции, выход нейронной сети и функции погрешности предсказания. Исследуйте влияние глубины предсказания на погрешность.

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

22.	$(t^2 - t) \sin(t^2 - 5 \cdot t)$
23.	$t \cdot \sin(t^2 - 10 \cdot t)$
24.	$(t^2 - 5 \cdot t) \sin(t^2 - 10 \cdot t)$
25.	$(t^2 - 8 \cdot t) \sin(t^2 - 4 \cdot t)$

Вариант соответствует порядковому номеру

по журналу. Таблица 1 – Варианты заданий

26.	$(t^2 - 10 \cdot t) \sin(t^2 - 2 \cdot t)$
27.	$(t^2 - 10 \cdot t) \sin(t^2 - 8 \cdot t)$

ОТЧЕТНОСТЬ ЗА ЗАНЯТИЕ

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены:

1. Программа и цель работы.
2. Задание на работу.
3. Структура сети, графики предсказываемого, полученного временного процесса и ошибки предсказания.
4. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие области эффективного применения нейронных сетей Вам известны?
2. Какую модель используют для формализации нейрона?
3. Какие функции активации используются в модели формального нейрона?
4. Зачем необходима функция активации?
5. Перечислите требования, предъявляемые к функциям активации.

Учебно-методическое и информационное обеспечение дисциплины.

Перечень основной литературы:

1. Нейросетевые технологии обработки информации : учеб. пособие : Направление подготовки 010200.62 - Математика и компьютерные науки. Профиль подготовки - Вычислительная математика, информатика и компьютерные технологии. Бакалавриат / Аведьян Э. Д., Галушкин А. И., Червяков Н. И., Сахнюк П. А. ; Севю-Кав. федер. ун-т. МФТИ. - Ставрополь : СКФУ, 2014. - 283 с. : ил. - Гриф: Рек. УМО. - Библиогр.: с. 277-281

2. Барский, А.Б. Введение в нейронные сети
Электронный ресурс : учебное пособие / А.Б. Барский. - Введение в нейронные сети, 2019-12-01. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 358 с. - Книга находится в базовой версии ЭБС IPRbooks.

Перечень дополнительной литературы:

1. Барский, А.Б. Логические нейронные сети
Электронный ресурс : учебное пособие / А.Б. Барский. - Логические нейронные сети, 2020-07-28. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 492 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-94774-646-4

2. Применение искусственных нейронных сетей и системы остаточных классов в криптографии / [авт. кол.: Червяков Н.И., Евдокимов А.А., и др.]. - Москва : ФИЗМАТЛИТ, 2012. - 280 с. : ил. - Прил.: с. 269-279. - Библиогр.: с. 269-279 (191 назв.) и в конце гл. - ISBN 978-5-9221-1386-1

Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине

1. Методические указания для обучающихся по организации и проведению самостоятельной работы по дисциплине «Нейросетевые технологии обработки информации» (электронный вариант)

2. Методические указания по выполнению лабораторных работ по дисциплине «Нейросетевые технологии обработки информации» (электронный вариант)

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. www.aiportal.ru/ Портал искусственного интеллекта
2. www.compres.ru. Сайт журнала «Компьютер пресс».
3. www.ibxt.ru. Новости вычислительной техники.
4. www.ibxt.ru. Новости вычислительной техники.

5. www.intuit.ru. Интернет-университет информационных технологий ИНТУИТ.
6. www.osp.ru. Издательство «Открытые системы».
7. www.radiotec.ru/catalog.php?cat=jr7 Журнал «Нейрокомпьютеры: разработка, применение».

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

для обучающихся по организации и проведению самостоятельной работы
по дисциплине «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ КОМАНДНОЙ РАБОТЫ И
ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ»
для студентов направления подготовки

направленность (профиль):

Ставрополь
2026

1. Введение

Самостоятельная работа – планируемая учебная, учебно – исследовательская, научно –исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения. Федеральным образовательным стандартом предусматривается, как правило, 50% и более часов из общей трудоемкости дисциплины на самостоятельную работу студентов (далее СРС). В связи с этим, обучение в ВУЗе включает в себя две, практически одинаковые по объему и взаимовлиянию части – процесса обучения и процесса самообучения. Поэтому СРС должна стать эффективной и целенаправленной работой студента.

Концепцией модернизации российского образования определены основные задачи высшего образования - "подготовка квалифицированного работника соответствующего уровня и профиля, конкурентоспособного на рынке труда, компетентного, ответственного, свободно владеющего своей профессией и ориентированного в смежных областях деятельности, способного к эффективной работе по специальности на уровне мировых стандартов, готового к постоянному профессиональному росту, социальной и профессиональной мобильности".

Решение этих задач невозможно без повышения роли самостоятельной работы студентов над учебным материалом, усиления ответственности преподавателей за развитие навыков самостоятельной работы, за стимулирование профессионального роста студентов, воспитание творческой активности и инициативы.

Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

В результате изучения дисциплины у обучающихся формируются следующие компетенции (в соответствии с образовательным стандартом):

Код	Формулировка
ОПК-1	Способен оценивать роль информации, информационных технологий и информационной безопасности в современном обществе, их значение для обеспечения объективных потребностей личности, общества и государства

ПК-8	Способен создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники
------	--

2. Общая характеристика самостоятельной работы студента при изучении дисциплины «Нейросетевые технологии обработки информации»

Коды реализ уемых компет енций	Вид деятельности студентов	Итоговый продукт самостояте льной работы	Средства и технологии оценки	Объем часов, в том числе (астр.)		
				СРС	Контактная работа с преподавателе м	Всег о
3 семестр						
ОПК-1 ПК-8	Подготовка к лекциям	Конспект лекций	Собеседо вание	9,0	9,0	18,0
ОПК-1 ПК-8	Подготовка к практическим занятиям	Отчет в электронн ом виде	Собеседо вание	9,0	9,0	18,0
ОПК-1 ПК-8	Самостоятел ьное изучение литературы	Конспект	Собеседо вание	9,0	9,0	18,0
ОПК-1 ПК-8	Самотестир ование, подготовка к тестированию	Тестиров ание	Тестовые задания	9,0	9,0	18,0
Итого за 3 семестр				36,0	36,0	72,0
4 семестр						
ОПК-1 ПК-8	Подготовка к лекциям	Конспект лекций	Собеседо вание	9,0	10,0	19,0
ОПК-1 ПК-8	Подготовка к практическим занятиям	Отчет в электронно м виде	Собеседо вание	9,0	10,0	19,0
ОПК-1 ПК-8	Самостоятел ьное изучение литературы	Конспект	Собеседо вание	9,0	10,0	19,0
ОПК-1 ПК-8	Самотестир ование, подготовка к тестированию	Тестиров ание	Тестовые задания	9,0	10,0	19,0
Итого за 4 семестр				36,0	40,0	76,0
Итого				176,0	76,0	76,0

3. Методические рекомендации по изучению теоретического материала

Изучение литературы и источников необходимо начинать с прочтения

соответствующих глав учебных изданий, учебных пособий или литературы, рекомендованной в качестве основной или дополнительной по дисциплине «Математическое моделирование биологических процессов», которые прямо или косвенно относятся к отрабатываемой теме.

Изучая литературу и источники, студенту рекомендуется вести краткий конспект. Однако не следует переписывать все содержание изучаемой темы, нужно выписывать лишь основные идеи и главные на ваш взгляд мысли. В отдельных случаях, когда встречаются важные определения, понятия, необходимый фактический материал и примеры, статистическая информация, имеющие отношение к изучаемой теме, студенту следует выписать их в виде цитат с полным указанием библиографических источников.

Конспектирование рекомендуемой литературы и источников необходимо вести с распределением собранных материалов по отдельным главам и параграфам согласно учебно-тематическому плану. Необходимо выписывать все выходные данные по используемой литературе и источникам.

Основные виды систематизированной записи прочитанного:

1. Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;
2. Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;
3. Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;
4. Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;
5. Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические указания по составлению конспекта

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и

дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

Проверка освоения теоретического материала проводится в рамках собеседований по темам дисциплины.

ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ

по дисциплине «Нейросетевые технологии обработки информации».

3 семестр

1. Какая нейронная сеть называется рекуррентной?
2. Объясните особенности функционирования нейронной сети прямого распространения.
3. Напишите уравнения, характеризующие динамику рекуррентной нейронной сети в дискретном времени (скалярная форма).
4. Напишите векторно-матричные уравнения, характеризующие динамику рекуррентной нейронной сети в дискретном времени.
5. В чем состоит задача распознавания образов? Какая информация должна быть представлена об объекте при решении задачи распознавания образов?
6. Какую задачу решает сеть Хемминга?
7. В какой форме представлены характеристики объектов при решении задачи распознавания образов с помощью сети Хемминга?
8. Каким способом заданы классы при решении задачи распознавания образов с помощью сети Хемминга?
9. Сформулируйте критерий оптимальности отнесения объекта к одному из классов при применении сети Хемминга.
10. Какое содержание имеют выходные значения нейронов рабочего слоя сети Хемминга?
11. Сколько нейронов содержит рабочий слой сети Хемминга? Каковы их синаптические коэффициенты?
12. Как связаны расстояние по Хеммингу между эталонным представителем и классифицируемым входным вектором и потенциал соответствующего нейрона сети Хемминга?
13. Какая активационная характеристика используется в сети Хемминга? Как она может быть реализована?
14. Что называется автоассоциативной памятью? Приведите пример.
15. Что называется аттрактором динамической системы? Объясните принцип применения динамических систем с множеством аттракторов для построения ассоциативной памяти.
16. Напишите уравнения динамики сети Хопфилда.
17. Какова активационная характеристика нейронов сети Хопфилда?
18. Чему равно начальное состояние нейронов сети Хопфилда?
19. Объясните различие синхронного и асинхронного режимов функционирования рекуррентной нейронной сети. Какой из режимов функционирования используется в сети Хопфилда?
20. Каким выражением определен энергетический функционал в процессе работы сети Хопфилда?

21. Почему время достижения сетью Хопфилда одного из аттракторов из произвольного начального состояния конечно?
22. Как рассчитывается матрица синаптических коэффициентов сети Хопфилда? Какими свойствами она обладает?
23. Как приближенно оценивается объем памяти сети Хопфилда?
24. На какие типы можно разделить множество аттракторов сети Хопфилда? Что такое «ложная память»?
25. Как применяется сеть Хопфилда для кластеризации данных? Как интерпретируются кластеры?
26. Как устанавливается класс принадлежности вектора признаков при использовании нескольких (по числу классов) сетей Хопфилда, «настроенных» на разные классы?
27. Как организован в лабораторной работе процесс оценивания размеров бассейнов аттракторов сети Хопфилда?
28. В чем состоит проблема кластеризации данных? Какие выборочные данные необходимы для решения этой задачи?
29. Где в соревновательном слое нейронов хранится информация о прототипах кластеров?
30. Дайте описание работы нейронов соревновательного слоя.
31. В чем состоит избирательность нейронов соревновательного слоя по отношению к данным обучающей выборки?
32. Как оценивается расстояние между выборочным примером и вектором синаптических коэффициентов нейрона?
33. Почему именно потенциалы нейронов соревновательного слоя используются для выявления нейрона-победителя?
34. Объясните правило самообучения нейронов соревновательного слоя.
35. Что называется «эпохой» самообучения соревновательного слоя нейронов?
36. Как влияет выбор параметра скорости самообучения на качество процесса настройки синаптических коэффициентов? Какие практические рекомендации можно дать для выбора этого параметра?

4 семестр

1. Что такое «соревновательная» активационная характеристика?
2. Какие негативные особенности можно ожидать при реализации самообучения соревновательного слоя нейронов?
3. Каким способом можно избежать появления «мертвых» нейронов соревновательного слоя?
4. По каким причинам может возникать укрупнение кластеров при обучении соревновательного слоя нейронов?
5. Как практически можно оценить качество обучения соревновательного слоя нейронов? Каким свойством обладает оптимально обученный соревновательный слой?
6. Какое основное качество отличает карту Кохонена от соревновательного слоя нейронов?

7. Приведите примеры оценки расстояния между нейронами в топографической карте. Какими в этих примерах являются линии точек, равноудаленных от фиксированного нейрона?
8. В чем состоит правило обучения нейронов карты Кохонена? Каково его отличие от правила обучения нейронов соревновательного слоя?
9. В чем состоит «раскраска» топографической карты отдельными признаками?
10. Что представляет собой одномерная карта Кохонена?
11. Какой вы можете привести пример практического применения карты Кохонена?
12. Напишите уравнения функционирования многослойной нейронной сети.
13. Объясните состав данных таблицы обучающей выборки, используемой для настройки параметров нейронной сети.
14. Какие параметры многослойной нейронной сети настраиваются в процессе ее обучения?
15. Какой критерий используется для организации обучения многослойной нейронной сети?
16. Чем характеризуется эффект «паралича» при обучении многослойной нейронной сети?
17. В чем состоит и как реализуется процесс инициализации при обучении нейронной сети?
18. В чем состоит и как проверяется эффект «генерализации данных» в нейронной сети?
19. Какие параметры режима обучения многослойной нейронной сети доступны пользователю для настройки?
20. В чем состоит процедура валидации многослойной нейронной сети?
21. В чем состоит процедура тестирования обученной многослойной нейронной сети?
22. Какие переменные вычисляются с помощью метода обратного распространения ошибки и как они используются в процессе обучения многослойной нейронной сети?
23. Чем отличаются пакетный и стохастический режимы обучения многослойной нейронной сети?
24. По какому правилу производится модификация значений параметров многослойной нейронной сети при ее обучении методом обратного распространения ошибки?
25. Объясните содержание задачи аппроксимации функции многих переменных на многослойной нейронной сети.
26. В чем состоит этап подготовки данных для обучения многослойной нейронной сети в задаче аппроксимации функции многих переменных?
27. Объясните содержание задачи прогноза временных рядов на многослойной нейронной сети.
28. В чем состоит этап подготовки данных для обучения многослойной нейронной сети в задаче прогноза временных рядов?
29. Объясните содержание задачи классификации данных на многослойной нейронной сети.
30. В чем состоит этап подготовки данных для обучения многослойной нейронной сети в задаче классификации данных?

31. Какой критерий используется при решении задачи классификации данных с помощью многослойной нейронной сети?
32. Какие правила останова процесса обучения МНС применяются в практических приложениях?
33. В чем состоит функциональная задача дополнительного блока принятия решения на выходе МНС при использовании ее в качестве классификатора данных?
34. Почему чрезмерное увеличение числа нейронов скрытых слоев в МНС при решении задачи аппроксимации может ухудшить точностные показатели сети? Какое свойство нейронной сети имеется в виду?
35. Почему на задачи классификации данных и прогноза временного ряда распространяются те же рекомендации по выбору архитектуры МНС, что и в задаче аппроксимации функции многих переменных?
36. Какие вы можете предложить нейросетевые решения задачи прогноза временного ряда одновременно на несколько последовательных временных тактов?
37. Предложите вариант построения нейросетевого классификатора при числе классов 4 с использованием многослойных нейросетевых классификаторов на 2 класса.

4. Методические указания (по видам работ, предусмотренных рабочей программой дисциплины)

По дисциплине «Нейросетевые технологии обработки информации» предусмотрены следующие виды работ: лекции, лабораторные работы, самостоятельная работа, курсовая работа, зачет с оценкой.

Правила и приемы конспектирования лекций

1. Конспектирование лекций ведется в специально отведенной для этого тетради, каждый лист которой должен иметь поля (4-5 см) для дополнительных записей.
2. Необходимо записывать тему и план лекций, рекомендуемую литературу к теме. Записи разделов лекции должны иметь заголовки, подзаголовки, красные строки. Для выделения разделов, выводов, определений, основных идей можно использовать цветные карандаши и фломастеры.
3. Названные в лекции ссылки на первоисточники надо пометить на полях, чтобы при самостоятельной работе найти и вписать их.
4. В конспекте дословно записываются определения понятий, формулировки теорем, основные формулы для вычисления и т.п. Остальное должно быть записано своими словами.
5. Каждому студенту необходимо выработать и использовать допустимые сокращения наиболее распространенных терминов и понятий. Однако чрезмерное увлечение сокращениями может привести к тому, что со временем в них будет трудно разобраться. С целью лаконичности и правильности записи текстов математического содержания рекомендуется грамотно использовать математическую символику.

В конспект следует заносить всё, что преподаватель пишет на доске, а также рекомендуемые схемы, таблицы, диаграммы и т.д. Надо иметь в виду, что изучение и отработка прослушанных лекций без промедления значительно экономит время и способствует лучшему усвоению материала. Идеальным и самым эффективным методом отработки лекции является ее отработка в день прочтения лекции.

Критерием того, что материал лекции усвоен, является умение самостоятельно

изложить суть вопроса, поясняя его графиками и чертежами, сделать необходимые математические доказательства, не прибегая к помощи конспекта. Нельзя конспекты лекций читать как художественную литературу.

Методические указания по подготовке к лабораторным работам

Для того чтобы лабораторные работы приносили максимальную пользу, необходимо помнить, что вопросы и задания составляются по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях), он будет закрепляться на лабораторных работах как в результате обсуждения и анализа лекционного материала, так и с процессе выполнения лабораторных работ. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

Для более эффективного выполнения лабораторных работ необходимо повторить соответствующий теоретический материал, а на занятиях, прежде всего, внимательно ознакомиться с содержанием работы и оборудованием.

К каждой лабораторной работе разработаны письменные инструкции.

Весь процесс выполнения лабораторных работ включает в себя теоретическую подготовку, числовую обработку результатов лабораторного эксперимента и сдачу отчета по выполненной работе.

Для самоконтроля в каждой работе приведены контрольные вопросы, на которые обучающийся обязан дать четкие, правильные ответы.

Теоретическая подготовка завершается предварительным составлением отчета.

В результате выполненной работы заполняется бланк отчета, в котором должны быть отражены (как правило):

1. Тема, программа и цель работы.
2. Задание для лабораторной работы.
3. Структура нейронной сети. Графики погрешности. Значение ошибки. Результаты симулирования сети.
4. Выводы по работе.

Методические указания для обучающихся по освоению дисциплины

На первом этапе необходимо ознакомиться с рабочей программой дисциплины, в которой рассмотрено содержание тем практических занятий, темы и виды самостоятельной работы. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности.

Для успешного освоения дисциплины, необходимо выполнить следующие виды самостоятельной работы, используя рекомендуемые источники информации

Критерии оценки:

Оценка «отлично» выставляется студенту, если он, в полном объёме ориентируется в основных понятиях, связанных с обеспечением информационно психологической безопасности личности, общества и государства, понятиях информационного противоборства, информационной войны и форм их проявлений в современном мире, в полном объёме анализирует источники и даёт классификацию и оценку угрозам информационной безопасности, отлично знает как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных

технологий, программирования и компьютерной техники, отлично умеет создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей со-временных информационных технологий, программирования и компьютерной техники, отлично владеет как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники.

Оценка «хорошо» выставляется студенту, если он, ориентируется в основных понятиях, связанных с обеспечением информационно психологической безопасности личности, общества и государства, понятиях информационного противоборства, информационной войны и форм их проявлений в современном мире, анализирует источники и даёт классификацию и оценку угрозам информационной безопасности, знает как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники, умеет создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники, владеет как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей со-временных информационных технологий, программирования и компьютерной техники.

Оценка «удовлетворительно» выставляется студенту, если он, в основном ориентируется в основных понятиях, связанных с обеспечением информационно психологической безопасности личности, общества и государства, понятиях информационного противоборства, информационной войны и форм их проявлений в современном мире, в основном анализирует источники и даёт классификацию и оценку угрозам информационной безопасности, плохо знает как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники, плохо умеет создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники, плохо владеет как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники.

Оценка «неудовлетворительно» выставляется студенту, если он, с трудом ориентируется в основных понятиях, связанных с обеспечением информационно психологической безопасности личности, общества и государства, понятиях информационного противоборства, информационной войны и форм их проявлений в современном мире, с трудом анализирует источники и даёт классификацию и оценку угрозам информационной безопасности, не знает как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники, не умеет создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники, не владеет как создавать и исследовать новые математические модели в естественных науках, промышленности и бизнесе, с учетом возможностей современных информационных технологий, программирования и компьютерной техники. Компетенции ОПК-1 и ПК-8 не освоены.

Описание шкалы оценивания

Максимально возможный балл за весь текущий контроль устанавливается равным 55.

Текущее контрольное мероприятие считается сданным, если студент получил за него не менее 60% от установленного для этого контроля максимального балла. Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

Уровень выполнения контрольного задания	Рейтинговый балл (в % от максимального балла за контрольное задание)
Отличный	100
Хороший	80
Удовлетворительный	60
Неудовлетворительный	0

5. Список рекомендуемой литературы. Перечень

основной литературы:

1. Нейросетевые технологии обработки информации : учеб. пособие : Направление подготовки 010200.62 - Математика и компьютерные науки. Профиль подготовки - Вычислительная математика, информатика и компьютерные технологии. Бакалавриат / Аведьян Э. Д., Галушкин А. И., Червяков Н. И., Сахнюк П. А. ; Севю-Кав. федер. ун-т. МФТИ. - Ставрополь : СКФУ, 2014. - 283 с. : ил. Гриф: Рек. УМО. - Библиогр.: с. 277-281
2. Барский, А.Б. Введение в нейронные сети Электронный ресурс : учебное пособие / А.Б. Барский. - Введение в нейронные сети, 2019-12-01. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 358 с. - Книга находится в базовой версии ЭБС IPRbooks.

перечень дополнительной литературы:

1. Барский, А.Б. Логические нейронные сети Электронный ресурс : учебное пособие / А.Б. Барский. - Логические нейронные сети, 2020-07-28. - Москва : Интернет- Университет Информационных Технологий (ИНТУИТ), 2016. - 492 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-94774-646-4
2. Применение искусственных нейронных сетей и системы остаточных классов в криптографии / [авт. кол.: Червяков Н.И., Евдокимов А.А., и др.]. - Москва : ФИЗМАТЛИТ, 2012. - 280 с. : ил. - Прил.: с. 269-279. - Библиогр.: с. 269-279 (191 назв.) и в конце гл. - ISBN 978-5-9221-1386-1

Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине

1. Методические указания по организации самостоятельной работы студентов по дисциплине «Применение искусственных нейронных сетей и глубокого обучения в стеганоанализе». Ставрополь : СКФУ, 2025.

2. Методические указания к практическим занятиям по дисциплине «Применение искусственных нейронных сетей и глубокого обучения в стеганоанализе». Ставрополь : СКФУ, 2025.

**Перечень ресурсов информационно-телекоммуникационной сети «Интернет»,
необходимых для освоения дисциплины**

1. <http://biblioclub.ru/> – Университетская библиотека online "Библиоклуб"
2. <https://4brain.ru/liderstvo/> – Лидерство: уроки эффективного руководителя
3. <https://spravochnick.ru/psihologiya/> – Справочник по психологии