МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ «НЕЙРОННЫЕ СЕТИ»

для студентов специальности
6.091503 "Специализированные компьютерные системы" (КСД)

Рассмотрено на заседании кафедры
«Автоматизированные системы управления»
Протокол № 4 от 21 октября 2010 г.

Утверждено на заседании
учебно – издательского совета ДонНТУ
Протокол № 5 от 06.12.2010 г.

Донецьк 2010

Методические указания к лабораторным работам по дисциплине «Нейронные сети» для студентов специальности 6.091503 «Специализированные компьютерные системы» (КСД). / Составили: Скобцов Ю.А., Васяева Т.А., Хмелевой С.В. – Донецк: ДНТУ, 2010. – 48 с.

Методические указания содержат краткие теоретические сведения, методические рекомендации и задания для выполнения лабораторных работ по дисциплине «Нейронные сети» с использованием пакета MATLAB. Изложена методика выполнения каждой из лабораторных работ, требования к содержанию отчетов, контрольные вопросы, список рекомендуемой литературы.

Утверждено методической комиссией специальности 6.091503 «Специализированные компьютерные системы».

Составители: д.т.н., проф. Скобцов Ю.А.
к.т.н. Васяева Т.А.
к.т.н. Хмелевой С.В.

Рецензент: к.т.н., доц. Привалов М.В.

Ответственный за выпуск: зав. каф. АСУ Скобцов Ю.А.

СОДЕРЖАНИЕ

Введение……. 4
Лабораторная робота № 1……..……... 5
Лабораторная робота № 2……..……... 17
Лабораторная робота № 3……..……... 26
Лабораторная робота № 4……..……... 34
Список литературы…..……. 42
Приложение А. Список функций Neural Network Toolbox...……. 43

ВВЕДЕНИЕ

В последние десятилетия в мире бурно развивается новая прикладная область математики, специализирующаяся на искусственных нейронных сетях (НС). Актуальность исследований в этом направлении подтверждается массой различных применений НС. Это автоматизация процессов распознавания образов, адаптивное управление, аппроксимация функционалов, прогнозирование, создание экспертных систем, организация ассоциативной памяти и многие другие приложения. С помощью НС можно, например, предсказывать показатели биржевого рынка, выполнять распознавание оптических или звуковых сигналов, создавать самообучающиеся системы, способные управлять автомашиной при парковке или синтезировать речь по тексту.

Методическое пособие содержит основные положения НС и их приложения при решении различных технических задач. Предлагается выполнить четыре лабораторные работы, направленные на решение различных типов задач. Выполняются работы в пакете MATLAB. Методическое пособие содержит теоретические сведения по функциям пакета и примеры выполнения подобных задач.

Система MATLAB – матричная лаборатория создана и выпускается фирмой Math Works (USA), содержит базовую систему и десятки пакетов расширения в самых разных областях компьютерной математики. Система MATLAB в настоящее время принята в качестве официального средства оформления инженерной документации и научных публикаций. Язык программирования системы прост, он содержит несколько десятков операторов, и большое количество процедур и функций, содержание которых понятно пользователю. MATLAB предоставляет широкие возможности для работы с НС.

Лабораторная работа №1

Тема: Классификация с помощью перцептрона

Цель работы: изучение модели нейрона перцептрона и архитектуры перцептронной однослойной нейронной сети; создание и исследование моделей перцептронных нейронных сетей в системе MATLAB.

Нейронные сети: основные положения

Основу каждой НС составляют относительно простые, в большинстве случаев – однотипные, элементы (ячейки), имитирующие работу нейронов мозга. Далее под нейроном будет подразумеваться искусственный нейрон, то есть ячейка НС. Каждый нейрон характеризуется своим текущим состоянием по аналогии с нервными клетками головного мозга, которые могут быть возбуждены или заторможены. Он обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов, а также имеет аксон – выходную связь данного нейрона, с которой сигнал (возбуждения или торможения) поступает на синапсы следующих нейронов. Общий вид нейрона приведен на рисунке 1.1. Каждый синапс характеризуется величиной синаптической связи или ее весом w_i, который по физическому смыслу эквивалентен электрической проводимости.

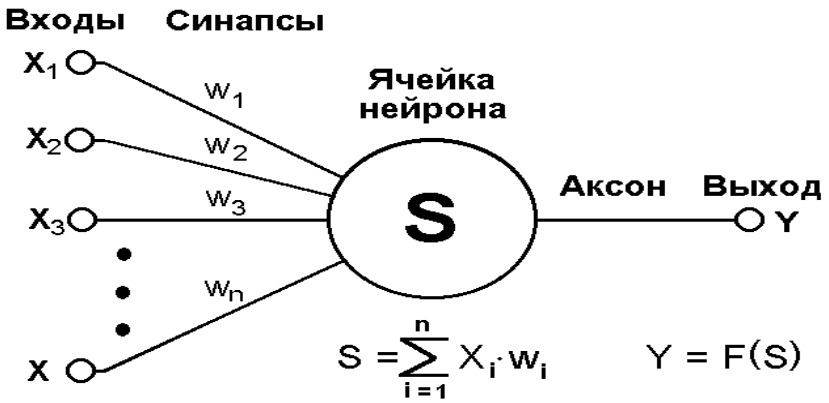

Рис.1.1.– Искусственный нейрон

Текущее состояние нейрона определяется, как взвешенная сумма его входов:

$$s = \sum_{i=1}^{n} x_i \cdot w_i \qquad (1.1)$$

Выход нейрона есть функция его состояния:

$$y = f(s) \qquad (1.2)$$

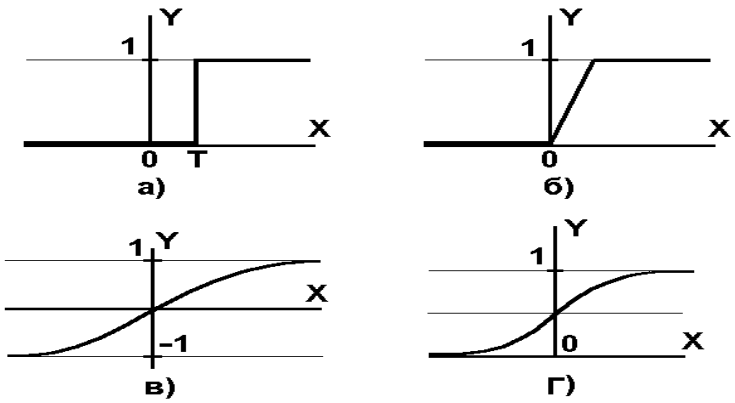

Рис. 1.2.- а) функция единичного скачка; б) линейный порог (гистерезис); в) сигмоид – гиперболический тангенс; г) сигмоид – формула (1.3)

Нелинейная функция *f* называется активационной и может иметь различный вид, как показано на рисунке 1.2. Одной из наиболее распространенных является нелинейная функция с насыщением, так называемая логистическая функция или сигмоид (т.е. функция *S*-образного вида):

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \qquad (1.3)$$

При уменьшении α сигмоид становится более пологим, в пределе при $\alpha=0$ вырождаясь в горизонтальную линию на уровне 0.5, при увеличении α сигмоид приближается по внешнему виду к функции единичного скачка с порогом T в точке $x = 0$. Из выражения для сигмоида очевидно, что выходное значение нейрона лежит в диапазоне [0,1].

Всем НС присущ принцип параллельной обработки сигналов, который достигается путем объединения большого числа нейронов в так называемые

слои и соединения определенным образом нейронов различных слоев, а также, в некоторых конфигурациях, и нейронов одного слоя между собой, причем обработка взаимодействия всех нейронов ведется послойно.

В качестве примера простейшей НС рассмотрим трехнейронный перцептрон (рис.1.3), На *n* входов поступают некие сигналы, проходящие по синапсам на 3 нейрона, образующие единственный слой этой НС и выдающие три выходных сигнала:

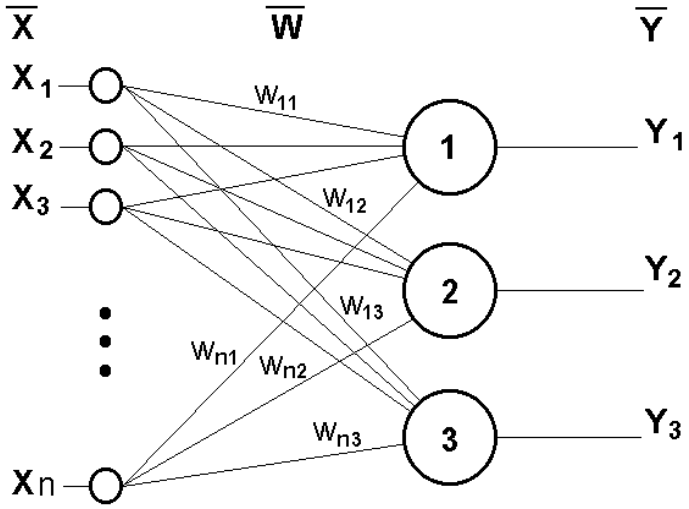

Рис.1.3 Однослойный перцептрон

$$y_j = f\left[\sum_{i=1}^{n} x_i \cdot w_{ij}\right], \quad j=1...3 \tag{1.4}$$

Очевидно, что все весовые коэффициенты синапсов одного слоя нейронов можно свести в матрицу *W*, в которой каждый элемент w_{ij} задает величину *i*-ой синаптической связи *j*-ого нейрона. Таким образом, процесс, происходящий в НС, может быть записан в матричной форме:

$$Y=F(XW) \tag{1.5}$$

где *X* и *Y* – соответственно входной и выходной сигнальные векторы, *F(V)* – активационная функция, применяемая поэлементно к компонентам вектора *V*.

Теоретически число слоев и число нейронов в каждом слое может быть произвольным, однако фактически оно ограничено ресурсами компьютера

или специализированной микросхемы, на которых обычно реализуется НС. Чем сложнее НС, тем масштабнее задачи, подвластные ей.

Выбор структуры НС осуществляется в соответствии с особенностями и сложностью задачи. Для решения некоторых отдельных типов задач уже существуют оптимальные конфигурации. Если же задача не может быть сведена ни к одному из известных типов, разработчику приходится решать сложную проблему синтеза новой конфигурации.

Очевидно, что процесс функционирования НС, то есть сущность действий, которые она способна выполнять, зависит от величин синаптических связей, поэтому, задавшись определенной структурой НС, отвечающей какой-либо задаче, разработчик сети должен найти оптимальные значения всех переменных весовых коэффициентов (некоторые синаптические связи могут быть постоянными).

Этот этап называется обучением НС, и от того, насколько качественно он будет выполнен, зависит способность сети решать поставленные перед ней проблемы во время эксплуатации. На этапе обучения кроме параметра качества подбора весов важную роль играет время обучения. Как правило, эти два параметра связаны обратной зависимостью и их приходится выбирать на основе компромисса.

Обучение НС может вестись с учителем или без него. В первом случае сети предъявляются значения как входных, так и желательных выходных сигналов, и она по некоторому внутреннему алгоритму подстраивает веса своих синаптических связей. Во втором случае выходы НС формируются самостоятельно, а веса изменяются по алгоритму, учитывающему только входные и производные от них сигналы.

Однослойный перцептрон

Однослойный перцептрон – однослойная нейронная сеть, все нейроны которой имеют жесткую пороговую функцию активации.

Перцептрон простейшая нейронная сеть, веса и смещения которой могут быть настроены таким образом, чтобы решить задачу классификации входных векторов.

Однослойный перцептрон имеет простой алгоритм обучения и способен решать лишь самые простые задачи. Эта модель вызвала к себе большой интерес в начале 1960-х годов и стала толчком к развитию искусственных нейронных сетей.

Нейрон, используемый в модели перцептрона, имеет пороговую функцию активации *hardlim* с жесткими ограничениями.

Каждое значение элемента вектора входа перцептрона умножено на соответствующий вес, и сумма полученных взвешенных элементов является входом функции активации. Если вход функции активации $n \geq 0$, то нейрон перцептрона возвращает 1, если $n < 0$, то 0.

Функция активации с жесткими ограничениями придает перцептрону способность классифицировать векторы входа, разделяя пространство входов на две области. Пространство входов делится на две области разделяющей линией. Эта линия перпендикулярна к вектору весов w и смещена на величину b. Векторы входа выше линии соответствуют положительному потенциалу нейрона, и, следовательно, выход перцептрона для этих векторов будет равен 1; векторы входа ниже линии соответствуют выходу перцептрона, равному 0.

При изменении значений смещения и весов граница линии изменяет свое положение.

Перцептрон без смещения всегда формирует разделяющую линию, проходящую через начало координат; добавление смещения формирует линию, которая не проходит через начало координат.

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

Грубо говоря, работа сети сводится к классификации (обобщению) входных сигналов, принадлежащих *n*-мерному гиперпространству, по некоторому числу классов. С математической точки зрения это происходит путем разбиения гиперпространства гиперплоскостями (запись для случая однослойного перцептрона)

$$\sum_{i=1}^{n} x_i \cdot w_{ik} = T_k, \quad k=1...m \tag{1.6}$$

Каждая полученная область является областью определения отдельного класса. Число таких классов для одной НС перцептронного типа не превышает 2^m, где m – число выходов сети. Однако не все из них могут быть разделимы НС.

Например, однослойный перцептрон, состоящий из одного нейрона с двумя входами, не способен разделить плоскость (двумерное гиперпространство) на две полуплоскости так, чтобы осуществить классификацию входных сигналов по классам A и B (см. таблицу 1.1).

Таблица 1.1.

X_1 X_2	0	1
0	A	B
1	B	A

Уравнение сети для этого случая

$$x_1 \cdot w_1 + x_2 \cdot w_2 = T \tag{1.7}$$

является уравнением прямой (одномерной гиперплоскости), которая ни при каких условиях не может разделить плоскость так, чтобы точки из множества входных сигналов, принадлежащие разным классам, оказались по разные стороны от прямой (рис. 1.4).

Если присмотреться к таблице 1, можно заметить, что данное разбиение на классы реализует логическую функцию исключающего ИЛИ для входных сигналов. Невозможность реализации однослойным перцептроном этой функции получила название проблемы исключающего ИЛИ.

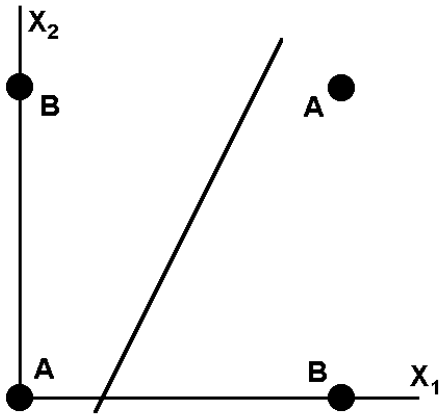

Рис.1.4 Визуальное представление работы НС

Функции, которые не реализуются однослойной сетью, называются линейно неразделимыми. Решение задач, подпадающих под это ограничение, заключается в применении 2-х и более слойных сетей или сетей с нелинейными синапсами, однако и тогда существует вероятность, что корректное разделение некоторых входных сигналов на классы невозможно.

Обучения НС – на примере перцептрона представленного на рисунке 1.3.

1. Проинициализировать элементы весовой матрицы (обычно небольшими случайными значениями).

2. Подать на входы один из входных векторов, которые сеть должна научиться различать, и вычислить ее выход.

3. Если выход правильный, перейти на шаг 4.

Иначе вычислить разницу между идеальным и полученным значениями выхода:

$$\delta = Y_I - Y \tag{1.8}$$

Модифицировать веса в соответствии с формулой:

$$w_{ij}(t+1) = w_{ij}(t) + \nu \cdot \delta \cdot x_i \tag{1.9}$$

где t и $t+1$ – номера соответственно текущей и следующей итераций; ν – коэффициент скорости обучения, $0<\nu<1$; i – номер входа; j – номер нейрона в слое.

Очевидно, что если $Y_I > Y$ весовые коэффициенты будут увеличены и тем самым уменьшат ошибку. В противном случае они будут уменьшены, и Y тоже уменьшится, приближаясь к Y_I.

4. Цикл с шага 2, пока сеть не перестанет ошибаться.

На втором шаге на разных итерациях поочередно в случайном порядке предъявляются все возможные входные вектора. К сожалению, нельзя заранее определить число итераций, которые потребуется выполнить, а в некоторых случаях и гарантировать полный успех.

Пример. Построить перцептронную нейронную сеть, которая производит классификацию на 2 класса, таким образом, что A_1[0;0] - 1 класс; A_2[1;1] – 2 класс; A_3[-1;-1] – 1 класс;

1. Построение обучающей выборки.

На координатной плоскости покажем проверочные точки

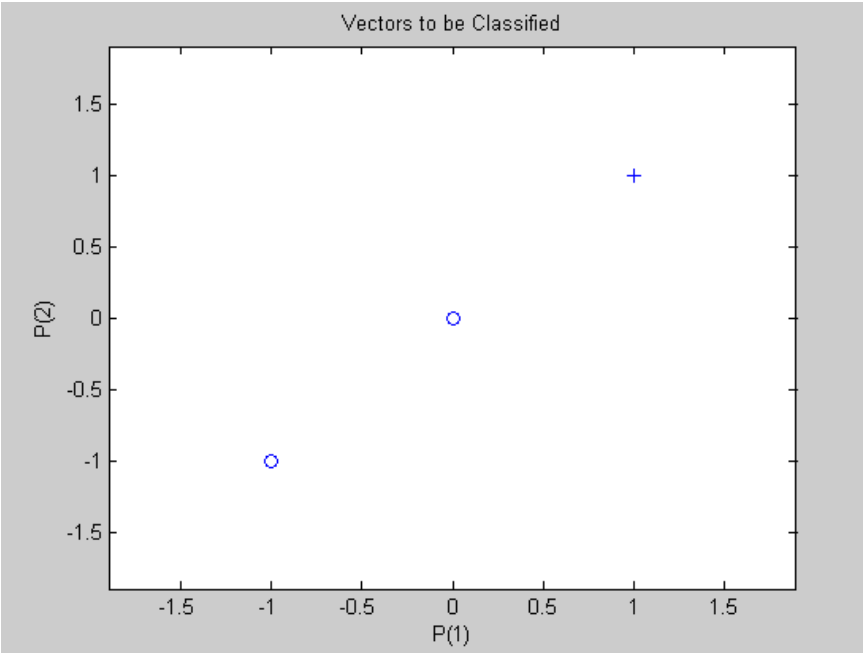

Рис. 1.5. Проверочные точки.

Добавим в их окрестности обучающие (по 3 на каждый класс). Например так:

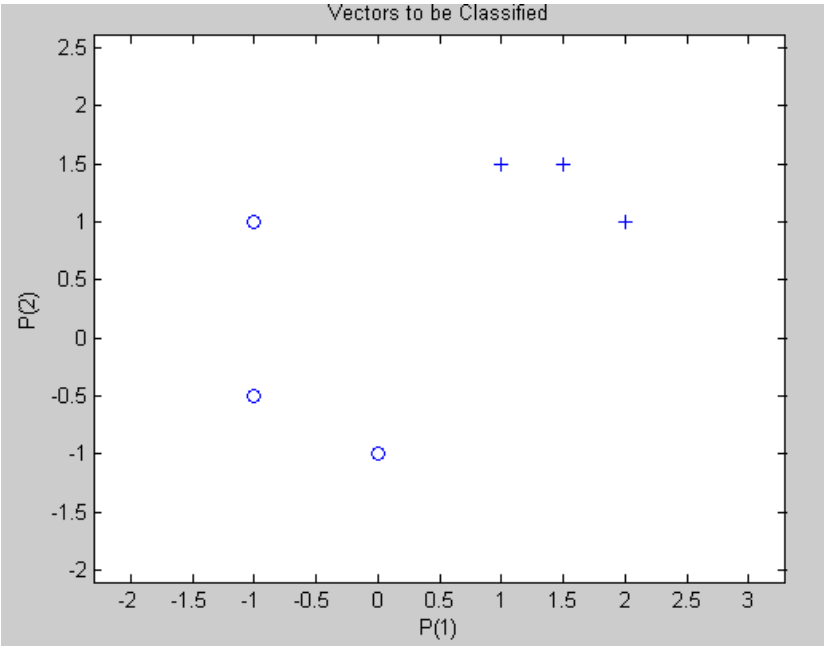

Рис. 1.5. Обучающие точки.

Обучающие точки не содержат проверочные, классы линейно разделимы.

Фрагмент программы формирования обучающего множества.

```
P=[-1 0 -1 1.5 1 2; -0.5 -1 1 1.5 1.5 1];%массив с
координатами обучающих точек
T=[0 0 0 1 1 1];% массив результатов классификации
figure(1); plotpv(P,T);  % отображение на экране
```

2. Архитектура сети.

Так как количество классов равно 2, достаточно одного сеть будет состоять из одного нейрона. Входами которого будут координаты X и Y. Выход – номер класса.

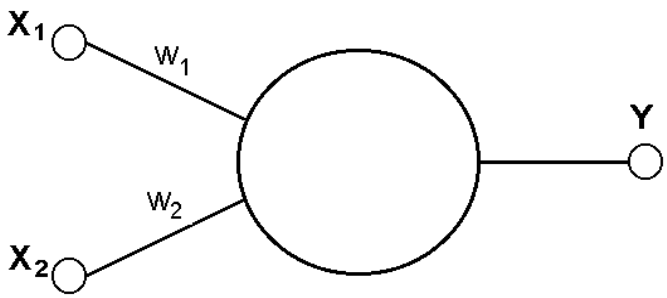

Рис.1.6 .Однонейронный перцептрон X_1 – коор. X; X_2 – коор. Y. Y-выход (номер класса).

Фрагмент программы:

```
net=newp([-1.5 1.5; -1.5 1.5;],1); % сознание сети - перцептрон
net.adaptparam.passes=10; % количество шагов юбучения
net=adapt(net,P,T);       % обучение сети
figure(2); plotpv(P,T);    % отображение на экране точек
plotpc(net.IW{1},net.b{1});% отображение на экране прямой
```

3. Моделирование работы НС.

Фрагмент программы:

```
A= [0 1 -1; 0 1 -1]; % массив проверочных точек
 a=sim(net,A);        % моделирование работы НС
 figure(3); plotpv(A,a); % отображение результатов классификации
```

Перечень функций Neural Network Toolbox в приложении А.

Порядок выполнения работы

1. Построить нейронную сеть, которая производит классификацию на заданное количество классов с помощью функции newp. Произвести начальную инициализацию весов нейронной сети с помощью функции init.

2. Построить обучающую выборку, которая позволяет правильно классифицировать заданную проверочную выборку. Координаты точек обучающей выборки должны быть подобраны геометрически. Для этого первоначально нанести точки проверочной выборки на график, ориентировочно построить вокруг каждой из этих точек несколько точек обучающей выборки, затем записать координаты этих точек в массив обучающей выборки. На каждую из точек проверочной выборки должно приходиться не менее 3 точек обучающей выборки.

3. Показать обучающую выборку на графике с помощью функции plotpv.

4. Произвести обучение нейронной сети на составленной обучающей выборке с использованием функции adapt. В процессе обучения

показывать изменение линий разбиения на классы в нейронной сети (с помощью функции plotpc).

5. Используя обученную нейронную сеть, произвести классификацию индивидуального для каждого варианта проверочного множества, с помощью функции sim. Поскольку номер класса для каждой точки может меняться при каждом новом обучении сети, номера классов в таблице наведены условно. Точки с разными номерами классов в таблице при классификации должны иметь разные номера классов, точки с одинаковыми – одинаковые. Привести график классификации точек (с помощью функций plotpc, plotpv). Вывести результаты работы сети на график. Для графика возможно применение различных цветов (команды findobj, set).

Таблица 1.2.

Индивидуальные задания

№ вв	Кол-во классов для классификации	Координаты точек проверочного множества и номер класса, к которому принадлежит каждая точка	Контрольный вопрос
1	3	[0;0]-1; [1;1]-2; [-1;-1]-3;	1
2	4	[2;1]-1; [1;0]-2; [0;-1]-3;	2
3	3	[0;0]-1; [1;1]-1; [-1;-1]-3;	3
4	4	[1;0]-1; [-1;1]-2; [-1;-1]-2;	4
5	3	[0;0]-1; [1;1]-2; [-1;-1]-3;	5
6	4	[0;0]-1; [1;1]-1; [-1;-1]-1;	6
7	3	[-1;0]-1; [-1;1]-2; [-1;-1]-3;	7
8	4	[0;1]-1; [1;-1]-2; [-1;-1]-3;	1
9	3	[2;0]-1; [1;1]-2; [-1;1]-2;	2
10	4	[0;0]-1; [1;1]-2; [-1;-1]-1;	3
11	3	[0;0]-1; [1;1]-1; [-1;-1]-2;	4
12	4	[0;0]-1; [1;1]-2; [-1;-1]-3;	5
13	3	[0;0]-1; [1;1]-1; [-1;-1]-1;	6
14	4	[0;1]-1; [1;-1]-2; [-1;-1]-3;	7

Содержание отчета

1. Титульный лист.
2. Задание, учитывая свой вариант.
3. Теоретические сведения.
4. Представить графическое и табличное представление обучающего множества. Показать их линейную разделимость.
5. Представить графическое и табличное представление проверочных точек.
6. Представить программу, написанную в среде MATLAB, и результаты классификации.
7. Представить программу, написанную без применения функций Neural Network Toolbox, и результаты классификации.
8. Выводы.

Контрольные вопросы

1. Структура перцептронного нейрона.
2. Правило нахождения количества нейронов в персептроне для распознавания заданного числа классов.
3. Построение линий классификации персептрона на основании его весов.
4. Процесс обучения персептрона.
5. Параметры функций newp, adapt, plotpc, plotpv, sse.
6. Анализ информации, выдаваемой функцией display.
7. Результат, который возвращает функция sim.

Лабораторная работа №2.

Тема: аппроксимация функции.

Цель работы: изучение возможности аппроксимации с помощью нейронной сети прямого распространения.

Теоретические сведения

Аппроксимация – замена одних математических объектов другими, в том или ином смысле близкими к исходным. Аппроксимация позволяет исследовать числовые характеристики и качественные свойства объекта, сводя задачу к изучению более простых или более удобных объектов (например, таких, характеристики которых легко вычисляются или свойства которых уже известны).

Аппроксимация может сводиться к нахождению функциональной зависимости по имеющемуся набору точек. В данном случае, функциональная зависимость будет представлена в нейросетевом базисе, т.е. через комбинацию активационных функций нейронных сетей.

Типы активационных функций

1. Линейная $f(S) = k * S$, рис. 2.1.а.
2. Пороговая $f(S) = \begin{cases} 1, S > 0 \\ 0, S \le 0 \end{cases}$, рис. 2.1.б.
3. Логистическая (сигмоидальная) $f(S) = \frac{1}{1 + e^{-\alpha s}}$, рис. 2.1.в.
4. Гиперболический тангенс $f(S) = \frac{e^{\alpha s} - e^{-\alpha s}}{e^{\alpha s} + e^{-\alpha s}}$, рис. 2.1.д.
5. Функция радиального базиса $f(S) = e^{-s^2}$, рис. 2.1.е.
6. Знаковая $f(S) = \begin{cases} 1, S > 0 \\ -1, S \le 0 \end{cases}$, рис. 2.1.з.
7. Полулинейная $f(S) = \begin{cases} k * S > 0 \\ 0, S \le 0 \end{cases}$, рис. 2.1.к.
8. Экспоненциальная $f(S) = e^{\alpha s}$

9. Синусоидальная $f(S) = \sin(S)$

10. Синусоидальная-рациональная $f(S) = \frac{S}{dt|S|}$

11. Квадратичная $f(S) = S^2$

12. Модульная $f(S) = |S|$

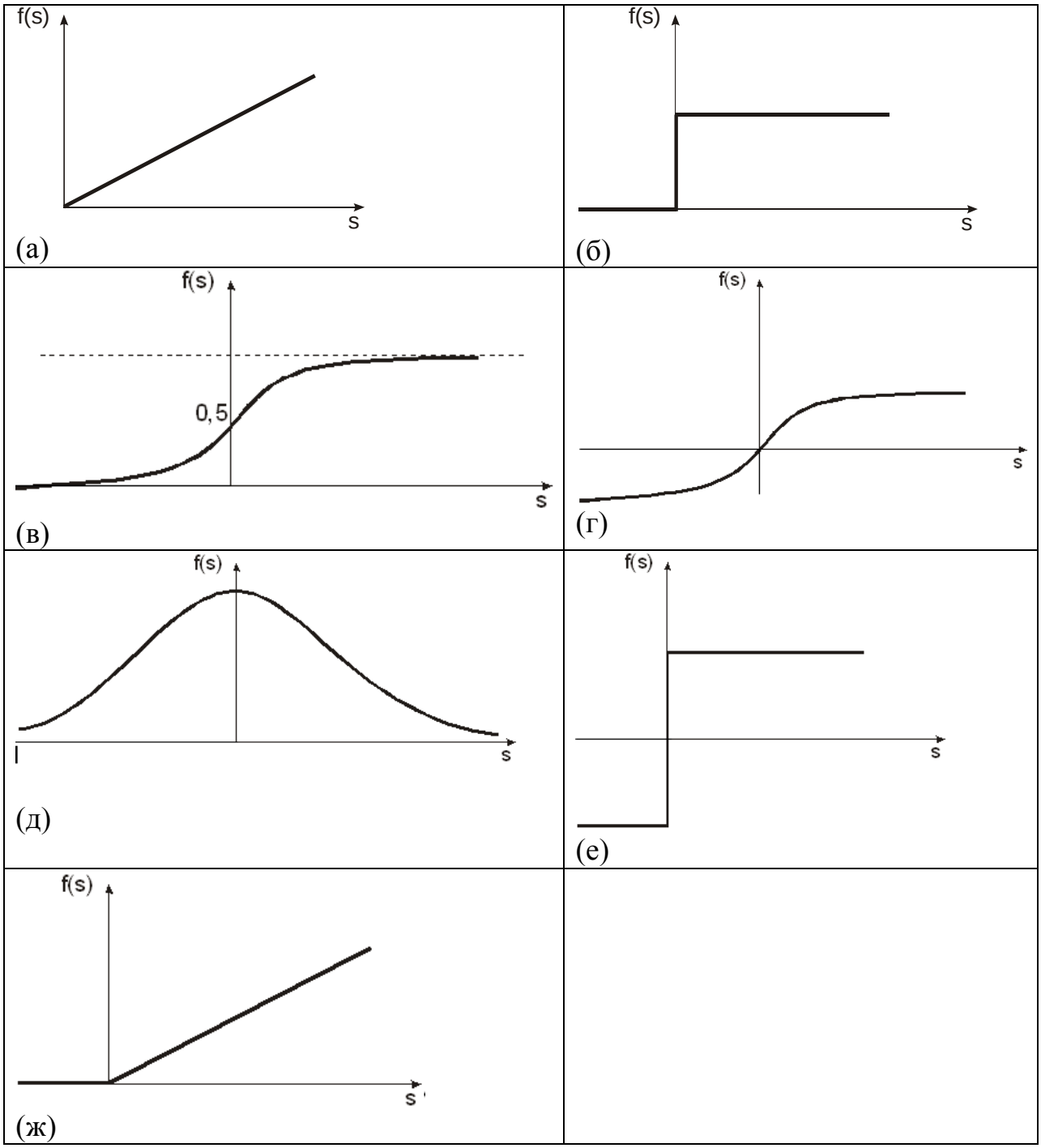

Рис. 2.1. Активационные функции нейронов: а – линейная; б – пороговая; в – логистическая (сигмоидальная); г – гиперболический тангенс; д – функция радиального базиса; е – знаковая; ж – полулинейная.

Согласно теореме, доказанной в 1989г. Л.Фунахаши, произвольно сложная функция может быть аппроксимирована двуслойной нейронной сетью, имеющей произвольные активационные функции. Поэтому классически для решения задачи аппроксимации используют многослойный персептрон. В данной лабораторной работе для решения задачи аппроксимации предлагается использование трехслойной нейронной сети, содержащей входной, один скрытый и выходной слои. Первый (входной) слой содержит число нейронов, равное количеству входных факторов.

Выходной слой содержит один нейрон, выдающий выходное значение чаще всего с линейной активационной функцией. Число нейронов второго (скрытого) уровня первоначально берется завышенное, например, равным полусумме числа обучающих данных:

$$n^{(2)} = \frac{\sum_p P \cdot k_p}{2} \qquad (2.1)$$

где P-число входных факторов, k_p - глубина ретроспективной выборки по p-ому фактору. Затем число нейронов второго слоя уменьшается до тех пор, пока ошибка аппроксимации не начнет возрастать. Функцией активации традиционно являются сигмоидальный тангенс или гиперболический тангенс, так как они являются гладкими, дифференцируемыми, а их производная выражается через саму функцию, что важно для обучения сети. Сигмоидальный тангенс предпочтительнее в связи с несимметричностью области определения [0;1] сигмоиды, что уменьшает скорость обучения.

Для простейшего случая аппроксимации – аппроксимации одномерной функции $y = f(x)$ нейронная сеть выполняет функцию нахождения преобразования $f(x)$ через активационные функции. Число входов в случае одномерной функции равно 1 (x), число выходов – 1 (y). Нейронная сеть в таком случае может быть представлена рис. 2.2.

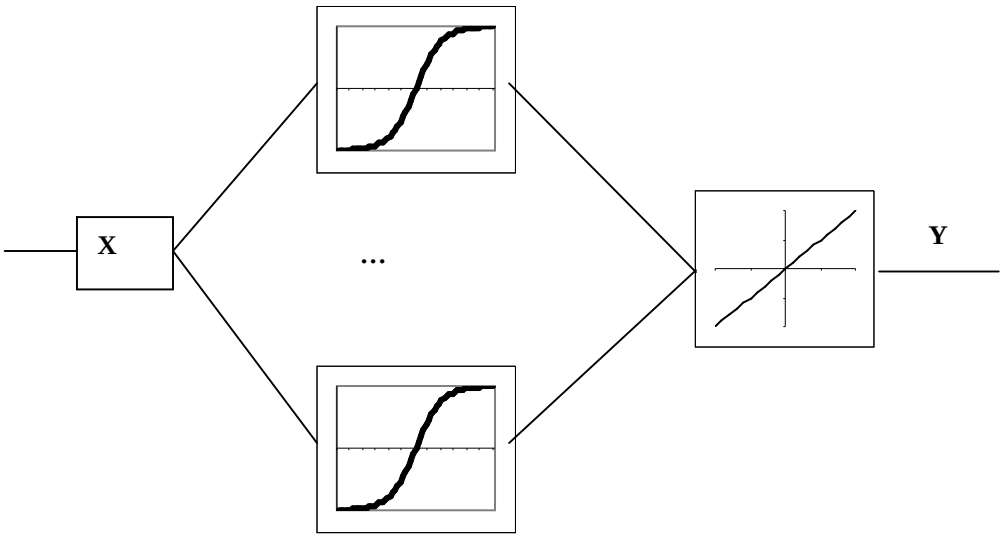

Рис. 2.2. структура нейронной сети.

Обучение многослойного персептрона чаще всего происходит с помощью алгоритма обратного распространения ошибки или его модификации.

Алгоритм обратного распространения ошибки

Долгое время не было теоретического обоснования обучения многослойных НС. А так как возможности НС сильно ограничены (несмотря на то, что алгоритмы обучения этих сетей были известны), то это сильно сдерживало развитие НС. Разработка алгоритма обратного распространения сыграла очень важную роль возрождения интереса к исследованию НС.

Обратное распространение – системный метод обучения многослойных НС. Он имеет серьезное математическое обоснование. Несмотря на некоторые ограничения, этот алгоритм сильно расширил область проблем, где могут быть использованы НС.

Этот алгоритм был фактически заново открыт и популяризован Румельхарт Макклеланд (1986) из знаменитой группы по изучению распределенных процессов.

На этапе обучения происходит вычисление синаптических коэффициентов *w*. При этом в отличие от классических методов в основе лежат не аналитические вычисления, а методы обучения по образцам с помощью примеров сгруппированных в обучаемом множестве. Для каждого образа из обучающей выборки считается известным требуемое значение выхода НС (обучение с учителем). Этот процесс можно рассматривать как решение оптимизационной задачи. Ее целью является минимизация функции ошибки или невязки E на обучающем множестве путем выбора значений синоптических коэффициентов *w*.

$$E = \frac{1}{2}\sum_{i}^{p}(d_i - y_i)^2 , \tag{2.2}$$

d_i – требуемое (желаемое) значение выхода на j-м образце выборки;

y_i – реальное значение;

p – число образцов обучающей выборки.

Минимизация ошибки E обычно осуществляется с помощью градиентных методов. При этом изменение весов происходит в направлении обратном направлению наибольшей крутизны функции ошибки.

$$W(t+1) = W(t) - \eta \frac{\partial E}{\partial W}, \tag{2.3}$$

$0 < \eta \le 1$ - определяемый пользователем параметр.

Существует два подхода к обучению. В первом из них веса w пересчитываются после подачи всего обучающего множества, и ошибка имеет вид:

$$E = \frac{1}{2}\sum_{i}^{p}(d_i - y_i)^2 \tag{2.4}$$

Во втором подходе ошибка пересчитывается после каждого образца:

$$E_i = \frac{1}{2}(d_i - y_i)^2 \tag{2.5}$$

$$E(W) = \frac{1}{2}\sum_{j,p}(d_{jp} - y_{jp})^2 \tag{2.6}$$

Пусть

$$\Delta W_{ij}(t) = -\eta \frac{\partial E}{\partial W_{ij}} \tag{2.7}$$

$$W_{ij}(t+1) = Wij(t) + \Delta W_{ij}(t) \tag{2.8}$$

Тогда

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial y_{ij}} * \frac{\partial y_i}{\partial S_i} * \frac{\partial S_j}{\partial W_{ij}} \tag{2.9}$$

$y_j(S_j)$ – активационная функция. Для

$$y_j = \bar{f}(S) = \frac{1}{1+e^{-Sj}} \tag{2.10}$$

$$\bar{f}'(S) = \frac{\partial}{\partial S_j}\left(\frac{1}{1+e^{-Sj}}\right) = \frac{1}{(1+e^{-Sj})^2}(-e^{-Sj}) = \frac{1}{1+e^{-Sj}} * \frac{-e^{-Sj}}{1+e^{-Sj}} = y_j(1-y_j) \tag{2.11}$$

$$\frac{dy_i}{dS_j} = (1-y_j)^2 \tag{2.12}$$

рассмотрим третий сомножитель:

$$S_j = \sum W_{ij} * y_i^{n-1}, \frac{\partial S_i}{\partial W_{ij}} = y_i^{(n-1)} \tag{2.13),(2.14}$$

Можно показать, что

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} * \frac{dy_k}{dS_k} * \frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} * \frac{dy_k}{dS_k} * W_{jk}^{(n+1)} \tag{2.15}$$

при этом суммирование к идет среди нейронов *n*-го слоя.

Введем новое обозначение:

$$\delta_j = \frac{\partial E}{\partial y_k} * \frac{dy_j}{dS_j} \tag{2.16}$$

$$\delta_j^{n-1} = \left[\sum_k \delta_k^n * W_{jk}^n\right] * \frac{dy_j}{dS_j} \tag{2.17}$$

Для внутреннего нейрона (2.17)

$$\delta_j^n = \left(d_j^n - y_j^n\right) * \frac{dy_j}{dS_j} \tag{2.18}$$

Для внешнего нейрона (2.18)

$$\Delta W_{ij}^n = -\eta \delta^{(n)} * y_j^{n-1} \tag{2.19}$$

$$W_{ij}(t+1) = W_{ij}(t) + \Delta W_{ij} \qquad (2.20)$$

Таким образом, полный алгоритм обучения обратного распространения строится так:

1. Подать на входы сети один из возможных образцов и в режиме обычного функционирования НС, когда сигналы распространяются от входа к выходам рассчитать значения выходов всех нейронов (обычно начальное значение веса составляют малые значения).

2. Рассчитать значения δ_j^n для нейронов выходного слоя по формуле Для внешнего нейрона (2.18).

3. Рассчитать значения δ_j^n для всех внутренних нейронов по формуле (2.17).

4. С помощью формулы (2.19) для всех связей найти приращения весовых коэффициентов ΔW_{ij}.

5. Скорректировать синаптические веса: $W_{ij}(t+1) = W_{ij}(t) + \Delta W_{ij}$

6. Повторить шаги 1-5 для каждого образа обучающей выборки пока ошибка *E* не станет достаточно маленькой.

Реализация многослойной сети в MATLAB

Создание нейронной сети происходит с помощью функции newff:

```
net=newff(PR, [S1, S2, …, SN],{TF1, TF2, … , TFn}, BTF, BLF, PF);
```

Аргументы функции:

PR – матрица минимальных и максимальных значений *R* входных элементов;

S_i – размер *i*-го слоя, для *n* слоев;

TF_i – функции активации нейронов *i* – го слоя, по умолчанию – «*tansig*»;

BTF – функция обучения сети, по умолчанию – «*traingd*»;

BLF – функция настройки весов и смещений, по умолчанию – «*learngdm*»;

PF – функция ошибки, по умолчанию – «mse».

В качестве примера можно привести создание нейронной сети с 5 входами, 10 нейронами на скрытом слое и 1 выходным нейроном:

```
net=newff([-2 2; -2 2; -2 2; -2 2; -2 2],
[10 1],{'tansig' 'purelin'});
```

После создания нейронной сети её необходимо проинициализировать малыми случайными значениями

```
init(net);
```

Обучение сети производится функцией train:

```
[net,TR,Y,E]=train(net,xtrain,ytrain);
```

Аргументы функции:

TR – набор записей для каждой эпохи обучения (эпоха и погрешность)

Y – набор реальных (не эталонных) выходов, полученных нейронной сетью для обучающих данных;

E – массив ошибок для каждого примера;

x_{train},y_{train} – массивы входов и выходов обучающей выборки.

После обучения сети необходимо её использовать на тестовой выборке:

```
ytest1=sim(net,xtest);
```

Массив y_{test1} содержит апроксимированные точки, полученные нейронной сетью для входных значений x_{test}.

Ход работы

Для заданной функции построить ее табличные значения (количество значений должно быть достаточным для того, чтобы аппроксимированная функция визуально совпадала с табличными значениями). Использовать нейронную сеть для аппроксимации этих табличных значений и нахождения аппроксимированной функции. Вывести на график табличные значения (точками) и аппроксимированную функцию (линией). Найти численное значение погрешности результата аппроксимации, вывести на экран. Обучение нейронной сети выполнять с помощью функции train.

Таблица 2.1.

Индивидуальные задания

Вариант	Вид функции	Промежуток нахождения решения
1	(1.85-x)*cos(3.5x-0.5)	x∈[-10,10]
2	cos(exp(x))/sin(ln(x))	x∈[2,4]
3	sin(x)/x2	x∈[3.1,20]
4	sin(2x)/x2	x∈[-20,-3.1]
5	cos(2x)/x2	x∈[-20,-2.3]
6	(x-1)cos(3x-15)	x∈[-10,10]
7	ln(x)cos(3x-15)	x∈[1,10]
8	cos(3x-15)/abs(x)=0	x∈[-10,-0.3),(0.3,10] x∈[-0.3,0.3]
9	cos(3x-15)*x	x∈[-9.6,9.1]
10	sin(x)/(1+exp(-x))	x∈[0.5,10]
11	cos(x)/ (1+exp(-x)	x∈[0.5,10]
12	(exp(x)-exp(-x))cos(x)/ (exp(x)+exp(-x))	x∈[-5,5]
13	(exp(-x)-exp(x))cos(x)/ (exp(x)+exp(-x))	x∈[-5,5]
14	cos(x-0.5)/abs(x)	x∈[-10,0),(0,10] , min
15	cos(2x)/abs(x-2)	x∈[-10,2),(2,10], max

Контрольные вопросы:

1. Что такое аппроксимация?
2. Параметры обучения при использовании для обучения функции train.
3. Виды прекращения обучения сети при использовании функции обучения train.
4. Способы нахождения погрешности результата.

Лабораторная работа №3

Тема: Классификация с помощью слоя Кохонена

Цель работы: изучение модели слоя Кохонена и алгоритма обучения без учителя; создание и исследование модели слоя Кохонена в системе MATLAB.

Основные положения

В естественной биологической системе трудно себе представить наличие учителя и сам процесс обучения. Тем не менее, человек (и не только) способен выполнять классификацию без учителя, что дает основания полагать, что объективно существуют алгоритмы обучения без учителя. То есть в обучающей выборке для образов неизвестны правильные (желаемые) выходные реакции. Главная черта, делающая обучение без учителя привлекательным – это его самостоятельность. Процесс обучения, как и в случае обучения с учителем заключается в подстройке синаптических коэффициентов сети.

Для реализации этого подхода необходимо решить две основные проблемы: 1) разработать методы разбиения образов на классы без учителя – этап обучения; 2) выработать правила отнесения текущего входного образа к некоторому классу – этап распознавания. Очевидно, разбиение на классы должно быть основано на использовании достаточно общих свойств классифицируемых объектов. В один класс должны попасть в некотором смысле похожие образы. При этом, как правило, используют компактность образов. В этом случае каждому классу в пространстве признаков соответствует обособленная группа точек. Тогда задача классификации сводится к разделению в многомерном пространстве на части множества точек. Процесс разбиения множества образов на классы называют кластеризацией. Визуально в двумерном (или трехмерном) пространстве

разделение множества точек часто не представляет особых проблем. Необходимо формализовать способ проведения границы между классами.

Существует тип нейронных сетей, который основан на конкуррентном обучении (competitive learning), где нейроны выходного слоя соревнуются за право активации. При этом активным становится обычно один нейрон - победитель в сети нейронов. Таким образом, здесь реализуется принцип “победитель получает все”.

Обучение по Хеббу

Очевидно, что подстройка синапсов может проводиться только на основании информации доступной в нейроне, т.е. его состояние Y_j и уже имеющихся коэффициентов W. Исходя из этих соображений и по аналогии с известными принципами самоорганизации нервных клеток построен алгоритм обучения Хебба, который основан на следующем правиле коррекции весов:

$$W_{ij}(t) = W_{ij}(t-1) + \alpha y_i^{(n-1)} y_j^{(n)} \qquad (3.1)$$

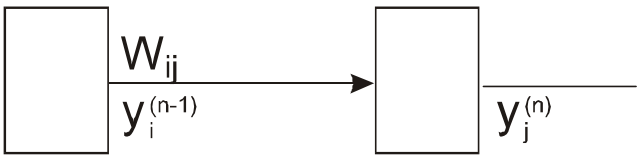

Рис. 3.1. Нейронная сеть.

$y_j^{(n-1)}$ – выходное значение i-го нейрона (n-1)-го слоя

$y_j^{(n)}$ – выходное значение j-го нейрона n-го слоя

$W_{ij}(t), W_{ij}(t-1)$ - весовые коэффициенты синапса, соединяющего i-й и j-й нейроны на итерациях t и t-1 соответственно.

α – коэффициент обучения.

Очевидно, что при обучении данным методом усиливаются связи между соседними нейронами.

Существует также дифференциальный метод обучения Хебба, в котором коррекция весов производится по формуле:

$$W_{ij}(t) = W_{ij}(t-1) + \alpha[y_i^{(n-1)}(t) - y_i^{(n-1)}(t-1)][y_j^{(n)}(t) - y_j^{(n)}(t-1)] \quad (3.2)$$

Из формулы видно, что сильнее всего обучаются синапсы соединяющие те нейроны выходы которых наиболее динамично изменяются.

Полный алгоритм обучения выглядит следующим образом:

1. На стадии инициализации всем весовым коэффициентам присваивается небольшое случайное значение.
2. На входы сети подается входной образ и сигнал возбуждения распространяется по всем слоям сети.
3. На основании поученных выходных значений Y_i, Y_j по формулам (3.1) и (3.2) производится изменение весовых коэффициентов.
4. Переход на шаг 2, пока выходы сети не стабилизируются с заданной точностью. При этом на втором шаге цикла попеременно предъявляются все образы обучающего множества.

Вид отклика на каждый класс входных образов неизвестен заранее и представляет собой произвольное сочетание состояний нейронов обусловленное случайным распределением весов на стадии инициализации. Несмотря на это сеть обобщает похожие образы, относя их к одному классу.

Обучение по Кохонену

Этот алгоритм предусматривает подстройку синаптических коэффициентов на основании их значений на предыдущей итерации

$$W_{ij}(t) = W_{ij}(t-1) + \alpha[y_j^{(n-1)} - W_{ij}(t-1)] \quad (3.3)$$

Видно, что здесь обучение сводится к минимизации разности между входными синапсами нейронов поступающих с выходов нейронов предыдущего слоя и весовыми коэффициентами его синапсов.

Полный алгоритм обучения имеет приблизительно такую же структуру, как и алгоритм Хебба, но на шаге 3 из всего слоя выбирается 1 нейрон, значение синапсов которого максимально подходят на входной образ и подстройка весов этого нейрона выполняется по формуле 3. Таким образом,

здесь реализуется принцип «победитель получает все». Часто такой подход называют конкурирующим обучением.

Приведенная процедура разбивает множество входных образов на кластеры присущие входным данным.

В правильно обученной сети для текущего входного образа *X* активизируется только один выходной нейрон-победитель, соответствующий кластеру, который попадает в текущий входной образ *X*.

Методы определения нейрона-победителя

Существует два метода определения нейрона победителя.

Первый основан на скалярном произведении $\overline{X} * \overline{W}$, т.е. максимальное значение дает выигравший нейрон. При этом обычно вектора $\overline{X}$ и $\overline{W}$ нормализуются.

Для каждого выходного нейрона вычисляется скалярное произведение

$$y_0 = \sum_i W_{i0} * x_i = \overline{W}_0^t * \overline{x} \qquad \forall o \neq k, y_o \leq y_k \tag{3.4}$$

и выбирается нейрон победитель.

Далее устанавливаются значения: для каждого нейрона победителя: $y_k = 1$, для остальных: $y_{o \neq k} = 1$.

Как только определяется победитель, его веса корректируются следующим образом:

$$\overline{W}_k(t+1) = \frac{\overline{W}_k(t) + \alpha\left[\overline{x}(t) - \overline{W}_k(t)\right]}{\left\|\overline{W}_k + \alpha\left[x(t) - W_k(t)\right]\right\|} \tag{3.5}$$

где деление сохраняет вектора $\overline{W}$ нормализованными.

Согласно этому выражению вектор весов вращается («подкручивается») в сторону вектора *X*.

В процессе обучения, каждый раз, когда предъявляется текущий вектор $\overline{x}$, определяется ближайший к нему весовой вектор нейрона победителя, который «подкручивается» по направлению к $\overline{x}$. В результате вектора весов

нейронов вращаются по направлению к тем областям, где находится много входных образов (т.е. кластеров), что показано на рисунке 3.3.

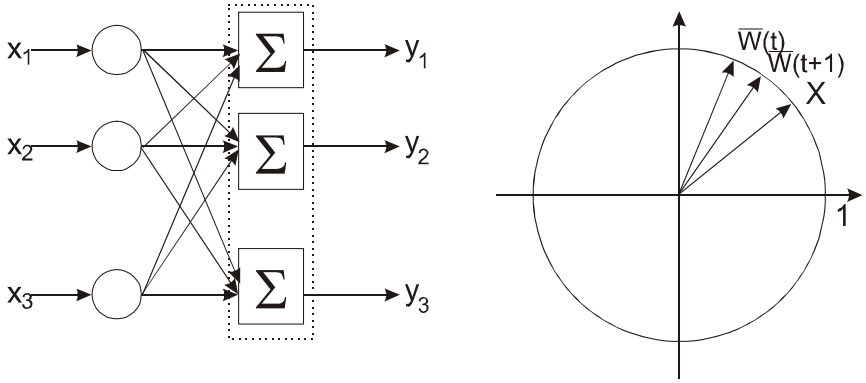

Рис. 3.2.Вращение весового вектора в процессе обучения.

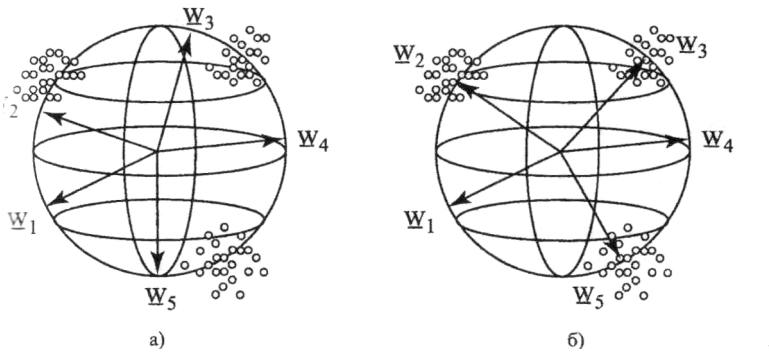

Рис. 3.3 Установление весовых векторов нейронов в центры кластеров.

Второй метод основан на вычислении евклидова расстояния. Т.е. выбирается *k*-й нейрон для которого евклидово расстояние удовлетворяет условию:

$$K : \left\| \overline{W}_k - \bar{x} \right\| \le \left\| \overline{W}_o - \bar{x} \right\| \forall o \qquad (3.6)$$

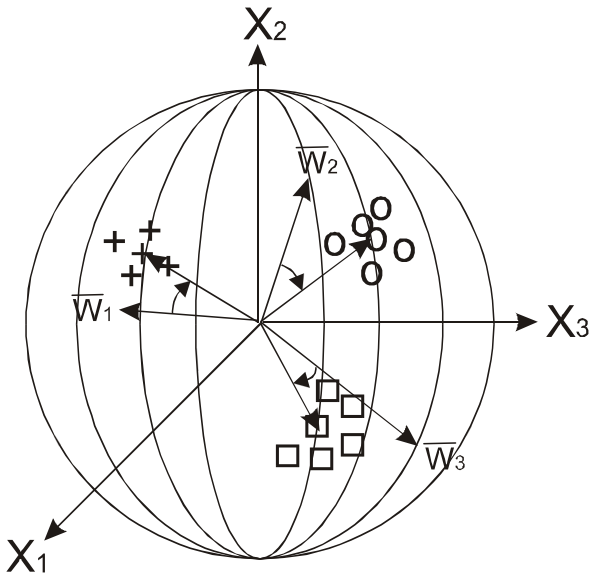

Рис. 3.4. Установление весовых векторов нейронов в центры кластеров.

Очевидно, что если вектора $\overline{X}$ и $\overline{W}$ нормализованы, то этот метод дает те же результаты, что и первый.

После определения победителя его веса корректируются по той же формуле

$$\overline{W_k}(t+1) = \overline{W_k}(t) + \alpha\left(\overline{x}(t) - \overline{W_k}(t)\right) \qquad (3.7)$$

Пример. Фрагменты программы создания и обучения сети:

```
P = [.1 .8   .1 .9; .2 .9 .1 .8]; %массив обучающих точек
…
net = newc([0 1; 0 1],2);          % создание сети
…
net = train(net,P);                % обучение сети
…
A= [0 1 -1; 0 1 -1];                 % массив проверочных точек
a=sim(net,A);                       % моделирование работы НС
…
```

Порядок выполнения работы

1. Построить нейронную сеть, которая производит классификацию на основе слоя Кохонена с помощью функции newc. Произвести начальную инициализацию весов нейронной сети с помощью функции init.

2. Построить обучающую выборку, которая позволяет правильно классифицировать заданную проверочную выборку. Координаты точек обучающей выборки должны быть подобраны геометрически. Для этого первоначально нанести точки проверочной выборки на график, ориентировочно построить вокруг каждой из этих точек несколько точек обучающей выборки, затем записать координаты этих точек в массив обучающей выборки. На каждую из точек проверочной выборки должно приходиться не менее 3 точек обучающей выборки.

3. Показать обучающую выборку на графике с помощью функции plot. На графике должны быть координатные оси, подписи по осям и название самого графика.

4. Произвести обучение нейронной сети на составленной обучающей выборке с использованием функции train.

5. Используя обученную нейронную сеть, произвести классификацию проверочного множества, с помощью функции sim. Показать результат классификации в командном окне matlab и на графике.

Таблица 3.1.

Индивидуальные задания

№ варианта	Кол-во классов для классификации	Координаты точек проверочного множества, номер класса, к которому принадлежит данная точка
1	3	[0;0]-1; [1;1]-2; [-1;-1]-3;
2	4	[2;1]-1; [1;0]-2; [0;-1]-3;
3	5	[0;0]-1; [1;1]-1; [-1;-1]-3;
4	6	[1;0]-1; [-1;1]-2; [-1;-1]-2;
5	7	[0;0]-1; [1;1]-2; [-1;-1]-3;
6	8	[0;0]-1; [1;1]-1; [-1;-1]-1;
7	3	[-1;0]-1; [-1;1]-2; [-1;-1]-3;
8	4	[0;1]-1; [1;-1]-2; [-1;-1]-3;
9	5	[2;0]-1; [1;1]-2; [-1;1]-2;
10	6	[0;0]-1; [1;1]-2; [-1;-1]-1;
11	7	[0;0]-1; [1;1]-1; [-1;-1]-2;
12	8	[0;0]-1; [1;1]-2; [-1;-1]-3;
13	3	[0;0]-1; [1;1]-1; [-1;-1]-1;
14	4	[0;1]-1; [1;-1]-2; [-1;-1]-3;

Содержание отчета

1. Титульный лист.
2. Задание, учитывая свой вариант.
3. Теоретические сведения.
4. Представить графическое и табличное представление обучающего множества.
5. Представить графическое и табличное представление проверочных точек.
6. Представить программу, написанную в среде MATLAB, и результаты классификации.
7. Выводы.

Контрольные вопросы

1. Архитектура нейронной сети.
2. Правило нахождения количества нейронов в сети для распознавания заданного числа классов.
3. Алгоритм обучения.
4. Анализ информации, выдаваемой функцией display.
5. Результат, который возвращает функция sim.
6. Параметры функций newc.

Лабораторная работа №4

Тема: Прогнозирование временных рядов с использованием нейронных сетей

Цель работы: изучение способа прогнозирования временных ряда с помощью НС в системе MATLAB.

Понятие прогнозирования временных рядов

Ряд наблюдений $x(t_1)$, $x(t_2)$,…, $x(t_N)$ анализируемой величины x(t), произведенных в последовательные моменты времени t_1, t_2,..., t_N, называется временным рядом (ВР). В общем виде моменты наблюдений могут быть произвольными, но классически считается, что во временных рядах равноотстоящие моменты наблюдений.

Задача одношагового прогнозирования может быть сформулирована так: имеется временной ряд $\bar{Y}$, представленный своими значениями, лагами (ЛАГ – от англ. lag — запаздывание, временной лаг – показатель, отражающий отставание или опережение во времени одного явления по сравнению с другими) в предшествующие моменты времени Y_{t-k}, Y_{t-k+1}, Y_t, где t-текущий момент времени, k-глубина исторической выборки. Необходимо найти величину изменения ее значения, которое произойдет в следующий момент времени, на основе анализа прошлых значений, а также истории изменения других факторов, влияющих на динамику прогнозируемой величины. При этом важным может являться как максимально низкая величина ошибки прогнозирования, так и правильное определение характера дальнейшего изменения значения прогнозируемой величины (увеличение или уменьшение).

При решении задачи прогнозирования требуется найти значение Y (или несколько величин, вектор) в будущий момент времени на основе известных предыдущих значений $\bar{Y}, \bar{V}, \bar{C}$: $Y_{t+1} = g(\bar{Y}, \bar{V}, \bar{C})$

$$Y_{t+1} = g(Y_{t-k} \dots Y_t, V1_{t-k} \dots V1_t, \dots, Vn_{t-k} \dots Vn_t, C1_{t-k}, \dots C1_t, \dots, Cm_{t-k} \dots Cm_t) \quad (4.1)$$

Выражение (4.1) является основным соотношением, определяющим прогноз.

Определение структуры нейронной сети

Определение структуры нейронной сети предполагает задание ее архитектуры (число слоев, число нейронов на каждом слое), а также функции активации нейронов.

При использовании многослойного перцептрона для решения задач аппроксимации функций (в т.ч. и задач прогнозирования) нет необходимости в числе скрытых слоев больше одного. Таким образом, предлагается использовать нейронную сеть, имеющую в своем составе входной, один скрытый и выходной слои. Число нейронов первого (то есть входного) слоя $n^{(1)}$ зависит от количества входных факторов и глубины ретроспективной выборки по каждому фактору и определяется по формуле:

$$n^{(1)} = \sum_p P \cdot k_p \quad (4.2)$$

где P-число входных факторов, k_p - глубина ретроспективной выборки по p-ому фактору.

Таким образом, число нейронов входного слоя равно числу элементов входных данных.

Выходной слой содержит один нейрон, выход которого представляет собой полученное в результате работы нейронной сети значение прогноза. При необходимости получения одновременно нескольких выходных значений (например, котировок продажи и покупки ценной бумаги) предпочтительнее не увеличивать число выходных нейронов, а обучать несколько отдельных нейронных сетей, каждая из которых имеет один выход. В пользу предлагаемого подхода свидетельствуют результаты исследований. Так, в частности, в результате экспериментов было установлено, что точность прогноза нейросети с двумя выходами хуже, чем

двух нейросетей с единственным выходом при обучении на одних и тех же данных.

Число нейронов второго (скрытого) уровня $n^{(2)}$ первоначально предлагается принимать равным полусумме числа нейронов входного и выходного слоев:

$$n^{(2)} = \frac{\sum_p P \cdot k_p + 1}{2} \tag{4.3}$$

В процессе определения и настройки параметров нейронной сети число скрытых нейронов может варьироваться с целью увеличения точности прогноза. При этом порядок их количества можно приблизительно оценить с помощью теоремы Колмогорова и следствия из нее. В соответствии с этими теоретическими результатами непрерывную функцию можно аппроксимировать с любой точностью при помощи трехслойной нейронной сети, которая имеет *N* входных, *2*N+1* скрытых и один выходной нейрон. Таким образом, число *2*N+1* теоретически можно считать верхней границей для числа нейронов на скрытых уровнях. Однако данная оценка справедлива для наилучшей сети, соответствующей глобальному минимуму адаптивного рельефа ошибки в пространстве весов. На практике достичь глобального минимума обычно не удается. Кроме того, имея *2*N+1* скрытых нейронов, сеть способна в точности запомнить обучающие примеры. В то же время в реальности от нее требуется их обобщить, выявив скрытые закономерности. Таким образом, оценка Колмогорова по сути сделана для другой задачи и в данном случае служит лишь ориентиром. В ходе экспериментов эмпирическим путем было установлено, что число скрытых нейронов может составлять от *N/2* до *3N*.

В качестве функции активации нейронов предлагается использовать биполярную сигмоиду. Сигмоидная функция традиционно используется в сетях типа «многослойный перцептрон» так как она является гладкой, дифференцируемой, а ее производная выражается через саму функцию, что важно для обучения сети. В тоже время рациональная сигмоида с областью

определения [0;1] имеет недостаток, связанный с ее несимметричностью относительно оси абсцисс, который уменьшает скорость обучения. Поэтому в данной лабораторной работе предлагается применять биполярную сигмоидную функцию. Эксперименты показали, что область определения биполярной функции не оказывает существенного влияния на процесс обучения. Для определенности в дальнейшем будет использоваться функция с областью определения [-1;1].

Формирование обучающих множеств

Для обучения нейронной сети необходимо сформировать из массива данных обучающие примеры, каждый из которых содержит вход и требуемый выход нейросети, который сеть должна научиться выдавать в ответ на данный вход. При этом входные данные содержат историю предыдущих значений по факторам, использующимся при построении прогноза, а в качестве требуемого выхода используется будущее значение прогнозируемой величины (на этапе обучения оно известно). В качестве иллюстрации ниже приведен фрагмент реальных данных, и на их основе сформирован обучающий пример:

Таблица 4.1

Исходные данные:

Название фактора	Значение нормированной котировки на дату					Требуемый выход сети
	04.06.97	05.06.97	06.06.97	09.06.97	10.06.97	
US Treas-5	0	-1.552	0.789	-0.052	0.29	
Dow Jones			-1.304	-0.393	-.058	
ОГВВЗ-5	-0.124	-0.124	-0.358	-0.245	0	0.462

Таблица 4.2

Вид сформированного примера.

0	-1.552	0.789	-0.052	0.29	-1.304	-0.393	-0.058	-0.124	-0.124	-0.358	-0.245	0	0.462

Как видно, в обучающем примере исходные данные вытягиваются в строку вида <вход нейросети><выход нейросети> (в данном случае первые пять значений – US Treasuries 5, затем три значения – Dow Jones, потом пять значений ОГВВЗ 5 и завершает строку значение требуемого выхода сети (то есть будущая котировка ОГВВЗ 5). Количество нейронов входного слоя сети соответствует числу элементов входных данных, содержащихся в обучающем примере, а число нейронов выходного слоя – соответственно числу эталонных выходов (в данном случае – один).

Совокупность обучающих примеров создается из массива данных с информацией о значениях в предшествующие моменты времени с помощью метода “скользящего окна”. Для формирования обучающего примера на массив исходных данных накладывается временное “окно”. При этом данные о значениях входных факторов, попавшие в него, формируют обучающий пример. Следующий пример формируется путем сдвига “окна” на один временной интервал вперед. В качестве ширины «окна» по каждому фактору используется ранее определенная глубина ретроспективной выборки.

Сформированные примеры разделяются на два подмножества: обучающее и контрольное. Первое подмножество используется для непосредственного обучения сети, второе – для обучения не используется и служит для оценки способности сети работать с незнакомыми данными. Обычно число примеров обучающего множества относится к числу примеров контрольного множества как 2:1 или 3:1.

Построение программы в среде Matlab, использующей нейронные сети для прогнозирования временных рядов

Для построения программы, решающей задачу прогнозирования, будем считать, что временной ряд изменяется по определенному закону

(представленному таблицей вариантов), на который влияют также случайные помехи. В таком случае программа состоит из следующих этапов:

1. Получение исходного временного ряда. Для этого формируют массив, содержащий временные отсчеты на заданном диапазоне изменения, и на их основании получают сами значения временного ряда (с добавлением шума). Добавление шума производится функцией rand

```
y=y+rand(size(t))*0.2;
```

2. Построение из временного ряда последовательности, подаваемой на вход нейронной сети в случае одного фактора происходит исключительно «вытягиванием» временного ряда в массив с помощью «плавающего окна». Для глубины значащей выборки равной 5 лагам это происходит следующим образом:

```
ytrain=y(6:i);
 y1=y(5:i-1);
 y2=y(4:i-2);
 y3=y(3:i-3);
 y4=y(2:i-4);
 y5=y(1:i-5);
 xn=x(5:i-1);
```

3. Разделение массива примеров на обучающую и тестовую выборку (ОВ и ТВ).
4. Построение нейронной сети (содержащей 5 входов для 5 лагов).
5. Обучение сети.
6. Получение погрешности прогнозирования сети на тестовой выборке. Для среднеквадратической ошибки это производится следующим образом:

```
ytest1=sim(net,xtest);
sum((ytest1-ytest).*(ytest1-ytest))
```

Ход работы

1. Построить временной ряд, который представляет из себя функцию. Вид функции указан в вариантах задания.
2. К полученному временному ряду добавить шум в размере максимум 20% от амплитуды сигнала.
3. Построить выборку для обучения. Для этого на основании временного ряда строится 5 рядов с задержкой от 1 до 5. Для построения ряда с задержкой 5 берутся от 1 до n-5 элементы выборки, с задержкой 4 берутся от 2 до n-4 элементы выборки, с задержкой 3 берутся от 3 до n-3 элементы выборки, с задержкой 2 берутся от 4 до n-2 элементы выборки, с задержкой 1 берутся от 5 до n-1 элементы выборки. Здесь n – длина исходного временного ряда.
4. Построить проверочную выборку. Поскольку длина выборки для обучения на 5 элементов меньше длины исходного временного ряда, для построения взять элементы от 6 до n-го.
5. Временной ряд разбить на 2 части: использующиеся для обучения сети и для проверки. Размеры массивов должны относиться друг к другу приблизительно как 3:1.
6. Построить нейронную сеть для прогнозирования. Число слоев – 2. Активационная функция первого слоя – гиперболический тангенс, второго – линейная. Число нейронов первого слоя взять достаточным для удовлетворительного прогнозирования (10-100), число нейронов второго слоя – 1.
7. Произвести обучение сети на обучающем множестве, привести график исходного ряда и спрогнозированного, а также погрешности прогнозирования.

8. Произвести проверку работы сети на проверочном множестве, привести график исходного ряда и спрогнозированного, а также погрешности прогнозирования.

Таблица 4.3.

Вид временного ряда и диапазон его изменения.

Вариант	Вид функции	Промежуток нахождения решения
1	(1.85-t)*cos(3.5t-0.5)	$t \in [0,10]$
2	cos(exp(t))	$t \in [2,4]$
3	sin(t)*t	$t \in [3.1,20]$
4	sin(2t)*t	$t \in [3.1,20]$
5	cos(2t)*t	$t \in [2.3,20]$
6	(t-1)cos(3t-15)	$t \in [0,10]$
7	cos(3t-15)	$t \in [1,10]$
8	cos(3t-15)/abs(t)=0	$t \in [0,0.3),(0.3,10]$
9	cos(3t-15)*t	$t \in [09.1]$
10	(exp(t)-exp(-t))cos(t)/ (exp(t)+exp(-t))	$t \in [0,5]$
11	(exp(-t)-exp(t))cos(t)/ (exp(t)+exp(-t))	$t \in [0,5]$
14	cos(t-0.5)*abs(t)	$t \in (0,10]$
15	cos(2t)*abs(t-2)	$t \in (2,10]$

Контрольные вопросы

1. Что такое временной ряд?
2. С помощью какой НС выполняется прогнозирование временного ряда?
3. Принципы формирования обучающей выборки?
4. Принципы формирования проверочной выборки?

СПИСОК ЛИТЕРАТУРЫ

1. Руденко О.Г. Искусственные нейронные сети / Руденко О.Г., Бодянский Е.В. – Учебное пособие. – Харьков: ООО «Компания СМИТ», 2005. – 408 с.

2. Комарцова Л.Г. Нейрокомпьютеры: учебное пособие для вузов / Комарцова Л.Г., Максимов А.В. – М.: Издательство МГТУ им. Н.Э. Баумана, 2002. – 320с.

3. Круглов В.В. Искусственные нейронные сети. Теория и практика / Круглов В.В., Борисов В.В. – М.: Горячая линия – Телеком, 2001. – 384 с.

4. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Рутковская Д., Пилинский М., Рутковский Л. [Пер. с польск. Рудинского И.Д.]. – М.: Горячая линия – Телеком, 2006. – 452 с.

5. Медведев В.С. Нейронные сети. MATLAB 6 / Медведев В.С., Потемкин В.Г. [Под общ. ред. к.т.н. В.Г. Потемкина]. – М.: ДИАЛОГ-МИФИ, 2002. – 496 с.

6. Хайкин Саймон. Нейронные сети: полный курс, 2-е издание / Хайкин Саймон. – М.: Издательский дом «Вильямс», 2006. – 1104 с.

7. Тихонов Э. Е. Методы прогнозирования в условиях рынка: учебное пособие. / Тихонов Э. Е. – Невинномыск, 2006. – 221 с.

8. С.И. Татаренко Методы и модели анализа временных рядов / Метод. указания к лаб. работам / сост. С.И. Татаренко. – Тамбов: Изд-во Тамб. гос.техн. ун-та, 2008. – 32 с.

Приложение А.

Список функций Neural Network Toolbox

Функции для анализа

- **rrsurf** – поверхность ошибки нейрона с единственным входом
- **maxlinlr** – максимальная скорость обучения для линейного нейрона

Функции отклонения

- **boxdist** – расстояние между двумя векторами
- **dist** – евклидова весовая функция отклонения
- **linkdist** – связанная фунция отклонения
- **mandist** – весовая функция отклонения Манхеттена

Функции графического интерфейса

- **nntool** – вызов графического интерфейса пользователя

Функции инициализации слоя

- **initnw** – функция инициализации Нгуен-Видроу (Nguyen-Widrow)
- **initwb** – функция инициализации по весам и смещениям

Функции обучения

- **learncon** – обучающая функция смещений
- **learngd** – обучающая функция градиентного спуска
- **learngdm** – обучающая функция градиентного спуска с учетом моментов
- **learnh** – обучащая функция Хэбба
- **learnhd** – обучающая функция Хэбба с учетом затухания
- **learnis** – обучающая функция instar
- **learnk** – обучающая функция Кохонена
- **learnlv1** – LVQ1 обучающая функция
- **learnlv2** – LVQ2 обучающая функция
- **learnos** – outstar обучающая функция
- **learnp** – обучающая функция смещений и весов перцептрона
- **learnpn** - обучающая функция нормализованных смещений и весов перцептрона
- **learnsom** – обучающая функция самоорганизующейся карты весов
- **learnwh** – правило обучения Уидроу-Хоффа (Widrow-Hoff)

Линейные функции поиска

- **srchbac** – одномерная минимизация с использованием поиска с возвратом
- **srchbre** – одномерная локализация интервала с использованием метода Брента (Brent)
- **srchcha** - одномерная минимизация с использованием метода Караламбуса (Charalambous)
- **srchgol** - одномерная минимизация с использованием золотого сечения
- **srchhyb** - одномерная минимизация с использованием гибридного бисекционного поиска

Функции вычисления производных от входов сети

- **dnetprod** – вычисление производной от входов сети с перемножением входов
- **dnetsum** – вычисление производной от входов сети с суммированием входов

Входные функции сети

- **netprod** – функция произведения входов
- **netsum** – функция суммирования входов

Функции инициализации сети

- **initlay** – функция послойной инициализации сети

Функции использования сети

- **adapt** – разрешает адаптацию сети
- **disp** – отображает свойства нейронной сети
- **display** – отображает имена переменных и свойства сети
- **init** – инициализация нейронной сети
- **sim** – моделирование нейронной сети
- **train** – тренировка нейронной сети

Функции создания новой сети

- **network** – создание нейронной сети пользователя
- **newc** – создание конкурентного слоя
- **newcf** – создание каскадной направленной сети
- **newelm** – создание сети обратного распространения Элмана (Elman)
- **newff** – создание однонаправленной сети
- **newfftd** – создание однонаправленной сети с входными задержками
- **newgrnn** – создание обобщенной регрессонной нейронной сети
- **newhop** – создание рекуррентной сети Хопфилда
- **newlin** – создание линейного слоя
- **newlind** – конструирование линейного слоя
- **newlvq** – создание квантованной сети
- **newp** – создание перцептрона
- **newpnn** – конструирование вероятностной нейронной сети
- **newrb** – конструирование сети с радиальным базисом
- **newrbe** – конструирование точной сети с радиальными базисными функциями
- **newsom** – создание самоорганизующейся карты

Функции производных функционирования

- **dmae** – средняя абсолютная ошибка вычисления производной
- **dmse** – средне-квадратичная ошибка производной
- **dmsereg** - средне-квадратичная ошибка производной w/reg
- **dsse** – суммарная квадратичная ошибка производной

Функции выполнения

- **mae** - средняя абсолютная ошибка
- **mse** – средне-квадратичная ошибка
- **msereg** - средне-квадратичная ошибка w/reg
- **sse** - суммарная квадратичная ошибка

Функции графики

- **hintonw** – график Хинтона для матрицы весов
- **hintonwb** – график Хинтона для матрицы весов и векторов смещений
- **plotbr** – график функционирования сети при регулярной тренировке (Bayesian)
- **plotep** – изображение положений весов и смещений на поверхности ошибки
- **plotes** – изображение поверхности ошибок единичного входного нейрона
- **plotpc** – изображение линии классификации в векторном пространстве перцептрона
- **plotperf** – графическое представление функционирования сети
- **plotpv** - графическое представление входных целевых векторов
- **plotsom** – графическое представление самоорганизующейся карты
- **plotv** – графическое представление векторов в виде линий, выходящих из начала координат
- **plotvec** - графическое представление векторов различными цветами

Функции предварительной и пост обработки

- **postmnmx** – ненормализованные данные, которые были нормализованы посредством prenmmx
- **postreg** – линейный регрессионный анализ выходов сети по отношению к целевым значениям обучающего массива
- **poststd** – ненормированные данные, которые были нормированы с помощью функции prestd
- **premnmx** – нормирование данных в диапазоне от –1 до +1
- **prepca** – анализ главных компонент для входных данных
- **prestd** – нормирование данных к единичному стандартному отклонению и нулевому среднему
- **tramnmx** – преобразование данных с предварительно вычисленными минимумом и максимумом
- **trapca** – преобразование данных с использованием PCA матрицы, вычисленной с помощью функции prepca
- **trastd** – преобразование данных с использованием предварительно вычисленных значений стандартного отклонения и среднего

Функции поддержки Simulink

- **gensim** – генерация блока Simulink для моделирования нейронной сети

Топологические функции

- **gridtop** – топологическая функция в виде сеточного слоя
- **hextop** – топологическая функция в виде гексагонального слоя
- **randtop** – топологическая функция в виде случайного слоя

Функции тренировки

- **trainb** – пакетная тренировка с использованием правил обучения для весов и смещений
- **trainbfg** – тренировка сети с использованием квази–Ньютоновского метода BFGS
- **trainbr** – регуляризация Bayesian
- **trainc** – использование приращений циклического порядка
- **traincgb** – метод связанных градиентов Пауэлла-Била (Powell-Beale)
- **traincgf** - метод связанных градиентов Флетчера-Пауэлла (Fletcher-Powell)
- **traincgp** - метод связанных градиентов Полака-Рибира (Polak-Ribiere)
- **traingd** – метод градиентного спуска
- **traingda** – метод градиентного спуска с адаптивным обучением
- **traingdm** - метод градиентного спуска с учетом моментов
- **traingdx** - метод градиентного спуска с учетом моментов и с адаптивным обучением
- **trainlm** – метод Левенберга-Маркара (Levenberg-Marquardt)
- **trainoss** – одноступенчатый метод секущих
- **trainr** – метод случайных приращений
- **trainrp** – алгоритм упругого обратного распространения
- **trains** – метод последовательных приращений
- **trainscg** - метод шкалированных связанных градиентов

Производные функций активации

- **dhardlim** – производная ступенчатой функции активации
- **dhardlms** – производная симметричной ступенчатой функции активации
- **dlogsig** – производная сигмоидной (логистической) функции активации
- **dposlin** – производная положительной линейной функции активации
- **dpurelin** – приозводная линейной функции активации
- **dradbas** – производная радиальной базисной функции активации
- **dsatlin** – производная насыщающейся линейной функции активации
- **dsatlins** – производная симметричной насыщающейся функции активации
- **dtansig** – производная функции активации гиперболический тангенс
- **dtribas** – производная треугольной функции активации

Функции активации

- **compet** – конкурирующая функция активации
- **hardlim** – ступенчатая функция активации
- **hardlims** – ступенчатая симметричная функция активации
- **logsig** – сигмоидная (логистическая) функция активации
- **poslin** – положительная линейная функция активации
- **purelin** – линейная функция активации
- **radbas** – радиальная базисная функция активации
- **satlin** – насыщающаяся линейная функция активации
- **satlins** – симметричная насыщающаяся линейная функция активации
- **softmax** – функция активации, уменьшающая диапазон входных значений
- **tansig** – функция активации гиперболический тангенс
- **tribas** – треугольная функция активации

Полезные функции

- **calca** – вычисляет выходы сети и другие сигналы
- **calca1** – вычисляет сигналы сети для одного шага по времени
- **calce** – вычисляет ошибки слоев
- **calce1** - вычисляет ошибки слоев для одного шага по времени
- **calcgx** – вычисляет градиент весов и смещений как единственный вектор
- **calcjejj** – вычисляет Якобиан
- **calcjx** – вычисляет Якобиан весов и смещений как одну матрицу
- **calcpd** – вычисляет задержанные входы сети
- **calcperf** – вычисление выходов сети, сигналов и функционирования
- **formx** – формирует один вектор из весов и смещений
- **getx** – возвращает все веса и смещения сети как один вектор
- **setx** – устанавливает все веса и смещения сети в виде одного вектора

Векторные функции

- **cell2mat** – объединяет массив элементов матриц в одну матрицу
- **combvec** – создает все комбинации векторов
- **con2seq** – преобразует сходящиеся векторы в последовательные векторы
- **concur** – создает сходящиеся векторы смещений
- **ind2vec** – преобразование индексов в векторы
- **mat2cell** – разбиение матрицы на массив элементов матриц
- **minmax** – вычисляет минимальные и максимальные значения строк матрицы
- **normc** – нормирует столбцы матрицы
- **normr** – нормирует строки матрицы
- **pnormc** – псевдо-нормировка столбцов матрицы
- **quant** – дискретизация величины
- **seq2con** – преобразование последовательных векторов в сходящиеся векторы
- **sumsqr** – сумма квадратов элементов матрицы
- **vec2ind** – преобразование векторов в индексы

Функции инициализации весов и смещений

- **initcon** – "сознательная" функция инициализации
- **initzero** – инициализация с установкой нулевых значений весов и смещений
- **midpoint** – инициализация с установкой средних значений весов
- **randnc** – инициализация с установкой нормализованных значений столбцов весовых матриц
- **randnr** - инициализация с установкой нормализованных значений строк весовых функций
- **rands** – инициализация с установкой симметричных случайных значений весов и смещений
- **revert** – возвращение весам и смещениям значений, соответствующих предыдущей инициализации

Функции весовых производных

- **ddotprod** – производная скалярного произведения

Весовые функции

- **dist** – Евклидово расстояние
- **dotprod** – весовая функция в виде скалярного произведения
- **mandist** – весовая функция – расстояние Манхеттена
- **negdist** – весовая функция – отрицательное расстояние
- **normprod** – нормированное скалярное произведение

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

По выполнению лабораторных работ

по курсу «НЕЙРОННЫЕ СЕТИ»

Для студентов специальности 6.091503, «Специализированные компьютерные системы».

Составители: Юрий Александрович Скобцов д.т.н., професор

Татьяна Александровна Васяева к.т.н.

Сергей Владимирович Хмелевой к.т.н.

Формат 60x84, Ус. печ. лист. __2.04__

Тираж - электронный вариант

83000, м. Донецк, ул. Артема 58, ДонНТУ.