Федеральное агентство по образованию

Московский государственный горный университет

КАФЕДРА ЭЛЕКТРОТЕХНИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

Рекомендовано УМК по специальности
230201 – Информационные системы и технологии

Степанян И.В.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Интеллектуальные информационные системы

для направления
230200 – Информационные системы
специальности 230201 – Информационные системы и технологии

Москва 2007

Посвящается И. М. Сеченову, П. К. Анохину,
И. П. Павлову, В. М. Бехтереву, А. А. Ухтомскому,
и Дмитрию Витальевичу Салонину,
который открывает нам путь в науку XXI века.

Содержание

Введение

Сегодня всё настоятельнее требуются системы, которые способны не только выполнять однажды запрограммированную последовательность действий над заранее определенными данными, но способны и сами анализировать информацию, находить в ней закономерности, производить прогнозирование. Требуются системы, наделённые элементами интеллекта при обработке колоссального объёма информации и в то же время работающие в темпе управляемых процессов. В этой области приложений самым лучшим образом зарекомендовали себя искусственные нейронные сети – самообучающиеся системы, имитирующие деятельность коры головного мозга. Нейронные сети применяются в таких прикладных областях деятельности человека, как космология, молекулярная биология, гидрология, охрана окружающей среды, медицина, экономика и прочих.

Нейрокомпьютеры – это ЭВМ, качественно отличающиеся от других классов вычислительных систем тем, что для решения задач они используют не заранее разработанные алгоритмы, а специальным образом подобранные примеры, на которых они учатся. В результате обучения может быть построена сеть, которая обеспечит правильное решение, когда на вход будет подан сигнал, который отличается от тех, которые использовались в процессе обучения. В основной части курса лабораторных работ упор сделан на нейронные сети как на одно из наиболее перспективных на сегодняшний день направлений искусственного интеллекта и теории интеллектуальных информационных систем.

Лабораторные работы выполняются на эмуляторах нейронных сетей в среде ППП MatLab NNT. Для того чтобы было проще ориентироваться в материале, в описаниях лабораторных работ приведены примеры программ. Диапазоны значений некоторых переменных (количество нейронов, слоёв и прочие параметры) студенты должны подобрать самостоятельно, ориентируясь на приведенные примеры. Практика показала, что для защиты лабораторных работ нужно проверять понимание студентов выполненной работы. И тем не менее для практической отработки навыков нейропрограммирования был выбран язык программирования Matlab, а не готовые нейросетевые эмуляторы. Большинство лабораторных работ построено на основе справочника [1], который рекомендуется иметь как студентам, так и преподавателю.

В пособии приведено шесть работ. Наибольшие трудности у студентов вызывает вторая лабораторная работа, где, несмотря на в общем-то несложный математический аппарат, приходится выполнять множество простых вычислений, что часто ведет к ошибкам. Далее по сложности идут 4 и 5 работы, где важно правильно понять смысл задания и способ его выполнения: а именно, что является обучающим, что контрольным множеством. Автор выражает благодарность студентам группы ИС-2 за помощь в составлении шестой лабораторной работы, в которой они придумали пример на основе книги [2].

Следует отметить, что алгоритм нечеткого логического вывода Mamdani, используемый в шестой лабораторной работе во многом сходен с алгоритмом Sugeno, который в свою очередь является нечеткой нейронной сетью [2]. Это хорошо демонстрирует связь нечетких систем и нейронных сетей и способствует углубленному пониманию теории искусственного интеллекта.

Целью лабораторных работ является приобретение навыков работы с алгоритмами нейроинформатики, построение экспертных систем на основе нечеткой логики, знакомство с системами искусственного интеллекта и их возможностями в рамках курса “Интеллектуальные информационные системы”.

Лабораторная работа № 1

Модель нейрона и функции активации

Цель лабораторной работы:

Знакомство с эмулятором нейронных сетей ППП MatLab NNT, с простейшими нейронами и их работой.

Теоретическая часть

Основатель кибернетики Норберт Винер предопределил фундаментальность кибернетики, назвав свой главный труд «Кибернетика или управление и связь в животном и машине». Этим он подчеркнул, что законы управления являются общими для живой и неживой природы.

Центральным звеном в биологических системах управления является мозг, состоящий из более чем 100 млрд. нервных клеток – нейронов, каждая из которых имеет в среднем 10000 связей. В коре больших полушарий человека на один кубический миллиметр приходится почти 40 тысяч нейронов. Информацию об окружающем мире и о внутренней среде организма человек получает с помощью сенсорных систем, названных Павловым анализаторами.

Несмотря на большое разнообразие вариантов искусственных нейронных сетей, все они имеют общие черты. Так, все они, так же, как и мозг человека, состоят из большого числа связанных между собой однотипных элементов – нейронов, которые имитируют нейроны головного мозга.

Прежде чем перейти к архитектуре искусственного нейрона, обратимся к биологическому прототипу. Биологический нейрон имеет тело, дерево входов – дендритов, и выход – аксон (рис. 1). На дендритах располагаются окончания других нервных клеток. Каждое такое окончание называется синапсом. Каждый синапс имеет вес, который определяет, насколько соответствующий вход нейрона влияет на его состояние. Этот вес называется синаптическим коэффициентом и может меняться в процессе функционирования синапса. Проходя через синапс, электрический сигнал меняет свою амплитуду:

увеличивает или уменьшает. Это можно интерпретировать как умножение амплитуды сигнала на синаптический коэффициент. Взвешенные в дендритном дереве входные сигналы суммируются, и затем на аксонном выходе генерируется выходной импульс.

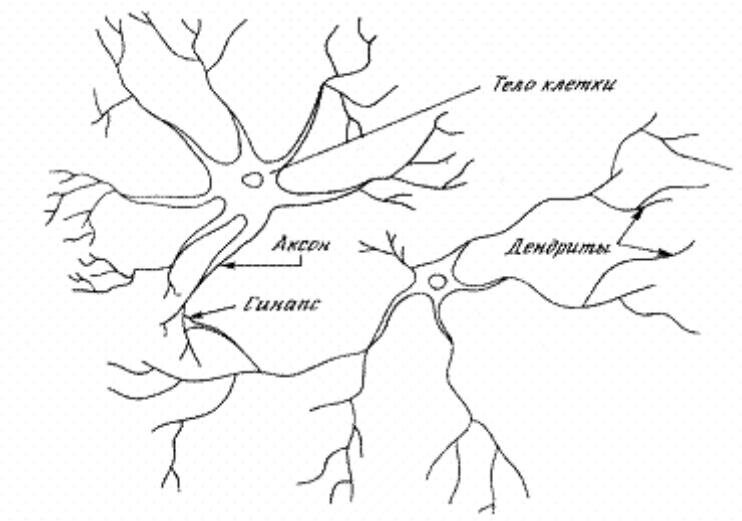


Рис.1 Биологический нейрон. В нервной клетке мембранный потенциал покоя равен примерно 70 мВ

Аксон – разветвленная структура, контактирующая с сотнями нейронов, расположенными как вблизи него, так и на значительном от него удалении (от одного до сотен миллиметров). Выходной сигнал нейрона проходит по ветви аксона и достигает синапсов, которые соединяют аксон с дендритными деревьями других нейронов. Через синапсы сигнал трансформируется в новый входной сигнал смежных нейронов. Приведённое описание принципа обработки информации позволяет сделать вывод, что техническая кибернетика вплотную подошла к решению задач при помощи методов, отшлифованных за миллионы лет эволюции.

Для описания алгоритмов и устройств в нейроинформатике разработана специальная "схемотехника", в которой элементарные устройства – сумматоры, синапсы, нейроны и т.п. объединяются в сети, предназначенные для решения задач искусственного интеллекта [2]. Сумматор вычисляет скалярное произведение вектора входного сигнала x на вектор параметров (весовых коэффициентов) α. На схемах он обозначается так, как показано на рис. 2. Сумматор называют адаптивным из-за наличия вектора настраиваемых параметров α. В большинстве задач полезно иметь неоднородную линейную функцию выходных сигналов. Ее вычисление можно представить с помощью адаптивного сумматора, имеющего n+1 вход и получающего на 0-й вход постоянный единичный сигнал, который называют смещением (рис. 3).

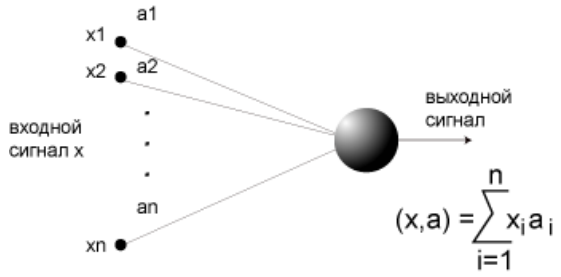


Рис. 2. Адаптивный сумматор

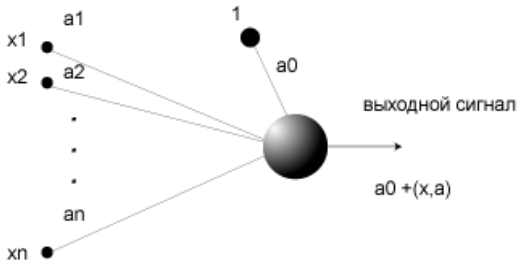


Рис. 3. Неоднородный адаптивный сумматор

Рис. 4. Нелинейный преобразователь сигнала

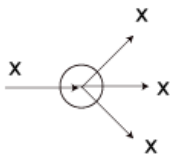


Рис. 5. Точка ветвления

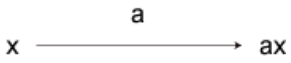


Рис. 6. Синапс

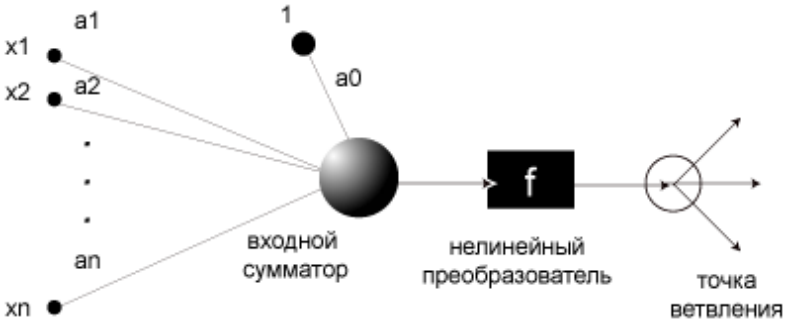


Рис. 7. Формальный нейрон

Нелинейный преобразователь сигнала – активационная характеристика – изображен на рис. 4. Он получает скалярный входной сигнал x и переводит его в функционал f(x). Точка ветвления служит для рассылки одного сигнала по нескольким адресатам (рис. 5). Она получает скалярный входной сигнал x и передает его всем своим выходам. Формальный нейрон составлен из входного сумматора, нелинейного преобразователя и точки ветвления (рис. 7). Линейная связь - синапс - отдельно не встречается, однако для некоторых рассуждений бывает удобно выделить этот элемент (рис. 6). Он умножает входной сигнал x на «вес синапса» α. Часто бывает полезно «присоединить» связи не к входному сумматору, а к точке ветвления. В результате получаем элемент, двойственный с адаптивным сумматором и называемый «выходная звезда». Его выходные связи производят умножение сигнала на свои веса.

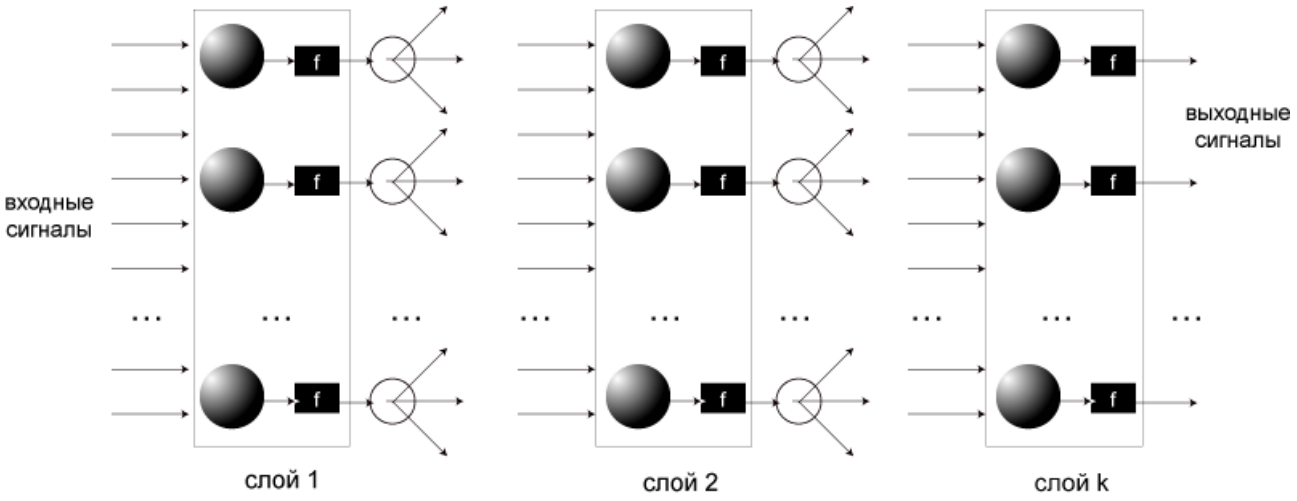


Рис. 8. Слоистая сеть

В нейроинформатике важный класс архитектур ИНС составляют слоистые нейронные сети [2][3]. Нейроны расположены в нескольких слоях (рис. 8). Нейроны первого слоя получают входные сигналы, преобразуют их и через точки ветвления передают нейронам второго слоя. Далее срабатывает второй слой и т.д. до k-го слоя, который выдает выходные сигналы. Если не оговорено противное, то каждый выходной сигнал i-го слоя подается на вход всех нейронов i+1-го. Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. Общепринятый способ подачи входных сигналов заключается в том, что все нейроны первого слоя получают каждый входной сигнал, причем количество нейронов первого слоя равно количеству элементов вектора входа, кодирующего входной сигнал.

Состояние нейрона определяется по формуле

$$S = \sum_{i=1}^{n} x_i w_i ,$$

где n – число входов нейрона, x_i – значение i-го входа нейрона, w_i – вес i-го синапса. Затем определяется значение аксона нейрона по формуле

$$Y = f(S),$$

где f –функция, которая называется активационной.

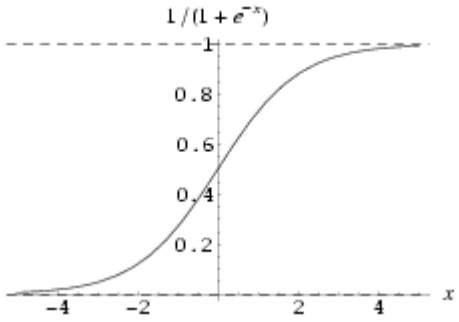
Описание активационных характеристик нейрона библиотеки Neural Metwork Library приведены в приложении 1. Наиболее часто в качестве активационной функции используется так называемый *сигмоид*, который имеет следующий вид:

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

Основное достоинство этой функции в том, что она дифференцируема на всей оси абсцисс и имеет очень простую производную:

$$f'(x) = \alpha f(x)(1 - f(x))$$

При уменьшении параметра a сигмоид становится более пологим, вырождаясь в горизонтальную линию на уровне 0,5 при a=0. При увеличении a сигмоид все больше приближается к функции единичного скачка.



***

Задание:

Создать простейшие модели нейронов: модель линейного нейрона, нейронов с пороговой и с логистической функциями активации. Дополнительно можно создать нейроны с остальными активационными функциями из приложения 1. Ознакомиться с функциями активации нейронов. Ознакомиться с библиотекой функций, предоставляемых средой MatLab NNT и характеристиками каждой из них. Каждый студент в индивидуальном порядке назначает случайным образом веса и смещения нейрона net.IW{1,1}, net.b{1}.

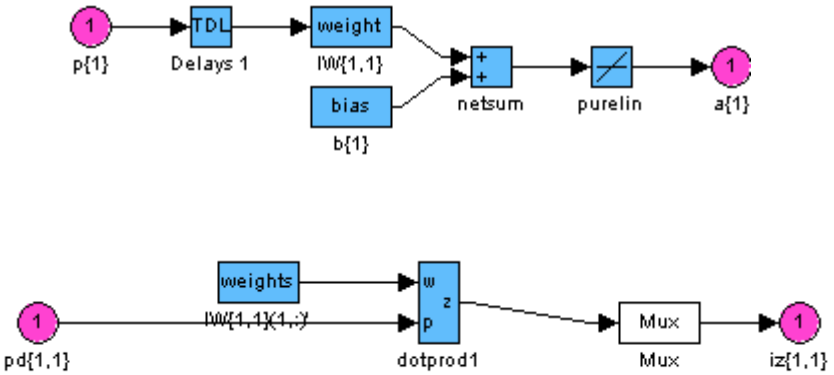
Отчет должен содержать: исходные данные, текст программы, ручной расчет нейронов, описания активационных функций среды Matlab NNT, представление нейронов в simulink.

Порядок выполнения работы (примеры):

1. Команда для моделирования нейронной сети с линейной функцией активации: **newlin**

net=newlin([-1 1;-1 1], 1); % *задаем 1 нейрон с линейной функцией активации с двумя синапсами и диапазоном входов* [-1 1;-1 1],
net.IW{1,1}=[3 7]; % *команда присвоения весов*
net.b{1}=[-5]; % *присваиваем вес вектору смещения*
p=[5;6]; % *задаем вектор входа p*
a=sim(net,p); % *моделирование сети*

gensim(net); % *построение архитектуры сети в среде simulink*
В среде *simulink* получаем иерархическую модель нейронной сети:



- p{1} - вектор входа, размерность которого указана в поле Dimentions (в данном случае - двухэлементный);
- (Delays1) - линия задержки;
- IW{1,1} - веса;
- b{1} - вектор смещения, вес которого указывается в поле Constant value;
- netsum - функция, которая определяет сумму взвешенных входов;
- purelin - блок линейной функции активации;
- a{1} - вектор выхода.

После просмотра слоев нейтронной сети, перейдем в окно Workspace - рабочее пространство программы (область памяти). Вид рабочего пространства для данной программы будет выглядеть следующим образом:

a	1x1	8	double array
net	1x1	18943	network object
p	2x1	16	double array

Рассчитаем значение выхода нейрона a по формуле: a = f(n)
$n = p_1 \cdot w_1 + p_2 \cdot w_2 + \ldots + p_n \cdot w_n + b$
$n = 3 \cdot 5 + 7 \cdot 6 - 5 = 52$
для линейной функции активации a = f(n)=52

2. Команда для моделирования нейронной сети со ступенчатой функцией активации (перцептрон): **newp**

net=newp([-1 1;-1 1], 1); % *задаем нейросеть с соответствующим диапазоном, с одним нейроном в единственном слое и ступенчатой функцией активации*
net.IW{1,1}=[3 7]; % *команда присвоения весов*
net.b{1}=[1]; % *присваиваем вес вектору смещения*
p=[5;6]; % *задаем вектор входа p*
a=sim(net,p); % *результат*
gensim(net); % *построение архитектуры сети*
Здесь элемент (hardlim) - блок пороговой функции активации.

Рассчитаем значение a по формуле:
$n = p_1 \cdot w_1 + p_2 \cdot w_2 + \ldots + p_n \cdot w_n + b$
$n = 3 \cdot 5 + 7 \cdot 6 - 5 = 52$
Для пороговой функции активации: $n \geq 0 \Rightarrow a = 1$

3. Команда для моделирования нейронной сети с сигмоидной функцией активации в виде гиперболического тангенса:
net=newff([-1 1;-1 1], 1); % *Создаем однонаправленную сеть с одним нейроном*
net.IW{1,1}= [0 0]; % *команда присвоения весов*
net.b{1}=[1]; % *присваиваем вес вектору смещения*
p=[5;6]; % *задаем вектор входа p*
a=sim(net,p); % *реакция сети на вектор входа p*
gensim(net); % *построение архитектуры сети*
Рассчитаем значение функции:
n = 1
Для сигмоидной функции в виде гиперболического тангенса
tansig(n) = (2/(1+exp(-2*1)))-1 = 0.7616

Контрольные вопросы к защите:

1. Определение нейрона. Основные свойства биологического и искусственного нейрона.
2. Общая формула искусственного нейрона.
3. Активационные характеристики и их формулы.
4. Коннекционизм. Эмерджентность нейронных структур.

Лабораторная работа № 2

Процедуры настройки параметров нейрона с пороговой функцией активации

Цель лабораторной работы:

Знакомство с правилом настройки параметров нейрона с пороговой функцией активации и сопутствующими функциями в Matlab.

Теоретическая часть *

Обучение нейронной сети - процедура настройки весов и смещений с целью уменьшить разность между желаемым (целевым) и истинным сигналами на выходе, используя некоторое правило настройки (обучения). Такой же подход справедлив и для одиночного нейрона. Процедуры обучения делятся на 2 класса: обучение с учителем и обучение без учителя. При обучении без учителя веса и смещения изменяются только в связи с изменениями входов сети. В этом случае целевые выходы в явном виде не задаются [1]. При обучении с учителем задается множество примеров, которое называется обучающим множеством. Обозначим через p вектор входов персептрона, а через t - вектор соответствующих желаемых выходов. Под перцептроном будем понимать нейросеть прямого распространения [1] со ступенчатой функцией активации. Цель обучения - уменьшить погрешность e = a - t, которая равна разности между реакцией нейрона a и вектором цели. Для успешной реализации алгоритмов обучения с учителем необходимы эксперты, которые должны предварительно сформировать обучающие множества. Разработка таких алгоритмов рассматривается как первый шаг в создании систем искусственного интеллекта [1].

Правило настройки (обучения) персептрона можно записать, связав изменение вектора весов Δw с погрешностью ***e = t - a***:

$$\Delta \mathbf{w} = (t - a)\mathbf{p} = e\mathbf{p}.$$

Учитывая, что смещение можно рассматривать как вес для единичного входа, получим аналогичное выражение для изменения смещения:

$$\Delta b = (t - a)\,1 = e.$$

В случае нескольких нейронов эти соотношения обобщаются следующим образом:

---

* Здесь и далее используются тексты из [1].

$$\begin{cases} \Delta \mathbf{W} = (\mathbf{t} - \mathbf{a})\mathbf{p}^{\mathbf{T}} = \mathbf{e}\mathbf{p}^{\mathbf{T}}; \\ \Delta \mathbf{b} = (\mathbf{t} - \mathbf{a}) = \mathbf{e}. \end{cases}$$

Тогда правило настройки (обучения) персептрона можно записать в следующей форме:

$$\begin{cases} \mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^{\mathbf{T}}; \\ \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e}. \end{cases}$$

Правило настройки персептрона гарантирует сходимость за конечное число шагов для всех задач классификации векторов, которые относятся к классу линейно отделимых, т.е. когда все пространство входов можно разделить на две области некоторой прямой линией, а в многомерном случае - гиперплоскостью. Доказано, что если решение задачи существует, то процесс обучения персептрона сходится за конечное число итераций. Каждая реализация процесса настройки с использованием всего обучающего множества называется проходом или циклом [Matlab]. Такой цикл может быть выполнен с помощью специальной функции адаптации adapt. **При каждом проходе функция adapt использует обучающее множество, вычисляет выход, погрешность и выполняет подстройку параметров персептрона**.

Для усвоения изложенного материала можно обратиться к демонстрационным программам:

- nnd4pr - позволяет выполнить многочисленные эксперименты по настройке (обучению) персептрона для решения задачи классификации входных векторов.
- demopl - решает задачу классификации с помощью простого персептрона.
- demop6 - иллюстрирует тщетность попытки классифицировать векторы входа, которые линейно неотделимы.

***

Задание:

Создать модель персептрона с одним нейроном, ступенчатой функцией активации, двумя или тремя синапсами; обучить этот нейрон при помощи функции adapt на обучающей выборке, состоящей из 3-4 пар вход-цель. Количество эпох адаптации должно быть не менее двух. Затем провести все расчеты вручную. Каждый студент в индивидуальном порядке подбирает обучающее множество. Отчет должен содержать ручные расчеты и код программы.

Порядок выполнения работы (пример):

Пусть требуется с помощью персептрона решить задачу классификации векторов, если задано следующее обучающее множество:

$$\left\{\mathbf{p}_1=\begin{bmatrix}2\\2\end{bmatrix},t_1=0\right\}\quad\left\{\mathbf{p}_2=\begin{bmatrix}1\\-2\end{bmatrix},t_2=1\right\}\quad\left\{\mathbf{p}_3=\begin{bmatrix}-2\\2\end{bmatrix},t_3=0\right\}\quad\left\{\mathbf{p}_4=\begin{bmatrix}-1\\1\end{bmatrix},t_4=1\right\}.$$

Выберем персептрон с одним нейроном и двухэлементным вектором входа (рис. 9). Используем нулевые веса и смещение. Для обозначения переменных на каждом шаге используем индекс в круглых скобках. Таким образом, начальные значения вектора весов $w^T(0)$ и смещения b(0) равны соответственно $w^T(0) = [0\ 0]$ и $b(0) = 0$.

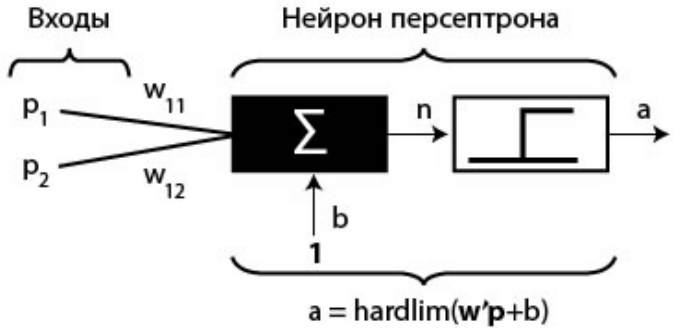


Рис. 9. Схема нейрона с пороговой функцией активации

Вычислим выход персептрона для первого вектора входа p1, используя начальные веса и смещение:

$$a=\text{hardlim}\left(\mathbf{w}^T(0)\,\mathbf{p}_1+b(0)\right)=\text{hardlim}\left(\begin{bmatrix}0&0\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}+0\right)=\text{hardlim}\,(0)=1.$$

Выход не совпадает с целевым значением t1, и необходимо применить правило настройки (обучения) персептрона, чтобы вычислить требуемые изменения весов и смещений:

$$\begin{cases}e=t_1-a=0-1=-1;\\ \Delta\mathbf{w}^T=e\mathbf{p}_1^T=(-1)[2\ \ 2]=[-2\ -2];\\ \Delta b=e=(-1)=-1.\end{cases}$$

Вычислим новые веса и смещение, используя правила обучения персептрона:

$$\begin{cases}\mathbf{w}^{T\,new}=\mathbf{w}^{T\,old}+\Delta\mathbf{w}^T=\begin{bmatrix}0&0\end{bmatrix}+\begin{bmatrix}-2&-2\end{bmatrix}=\begin{bmatrix}-2&-2\end{bmatrix}=\mathbf{w}^T(1);\\ b^{new}=b^{old}+\Delta b=0+(-1)=-1=b(1).\end{cases}$$

Обратимся к новому вектору входа *p2,* тогда, учитывая новые веса, получим:

$$a = \text{hardlim}(\mathbf{w}^{\mathrm{T}}(1)\mathbf{p}_2 + b(1)) = \text{hardlim}\left(\begin{bmatrix}-2 & -2\end{bmatrix}\begin{bmatrix}1\\-2\end{bmatrix} + (-1)\right) = \text{hardlim}(1) = 1.$$

В этом случае выход персептрона совпадает с целевым выходом, так что погрешность равна 0 и не требуется изменений в весах или смещении. Таким образом,

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(2) = \mathbf{w}^{\mathrm{T}}(1) = \begin{bmatrix}-2 & -2\end{bmatrix};\\ b(2) = b(1) = -1.\end{cases}$$

Продолжим этот процесс и убедимся, что после третьего шага настройки не изменились:

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(3) = \mathbf{w}^{\mathrm{T}}(2) = \begin{bmatrix}-2 & -2\end{bmatrix};\\ b(3) = b(2) = -1,\end{cases}$$

а после четвертого приняли значение

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(4) = \begin{bmatrix}-3 & -1\end{bmatrix};\\ b(4) = 0.\end{cases}$$

Чтобы определить, получено ли удовлетворительное решение, требуется сделать один проход через все векторы входа, чтобы проверить, соответствуют ли решения обучающему множеству. Вновь используем первый член обучающей последовательности и получаем:

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(5) = \mathbf{w}^{\mathrm{T}}(4) = \begin{bmatrix}-3 & -1\end{bmatrix};\\ b(5) = b(4) = 0.\end{cases}$$

Переходя ко второму члену, получим следующий результат:

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(6) = \begin{bmatrix}-2 & -3\end{bmatrix},\\ b(6) = 1.\end{cases}$$

Этим заканчиваются ручные вычисления. Теперь выполним аналогичные расчеты, используя функцию adapt. Вновь сформируем модель персептрона, изображенного на рис. 9:

```
net = newp([-2 2;-2 2],1);
```

Введем первый элемент обучающего множества

```
p = {[2; 2]};
t = {0};
```

Установим параметр passes (число проходов) равным 1 и выполним 1 шаг настройки:

```
net.adaptParam.passes = 1;
[net,a,e] = adapt(net,p,t);
a
a = [1]
e
e = [-1]
```

Скорректированные вектор весов и смещение равны:

```
twts = net.IW{1,1}
twts = -2 -2
tbiase = net.b{1}
tbiase = -1
```

Это совпадает с результатами, полученными при ручном расчете. Теперь можно ввести второй элемент обучающего множества и т. д., т. е. повторить всю процедуру ручного счета и получить те же результаты.

Можно эту работу выполнить автоматически, задав сразу все обучающее множество и выполнив 1 проход:

```
net = newp([-2 2;-2 2],1);
net.trainParam.passes = 1;
p = {[2;2] [1;-2] [-2;2] [-1;1]};
t = {0 1 0 1};
```

Теперь обучим сеть:

```
[net,a,e] = adapt(net,p,t);
```

Возвращаются выход и ошибка:

```
a
a = [1] [1] [0] [0]
e
e = [-1] [0] [0] [1]
```

Скорректированные вектор весов и смещение равны

```
twts = net.IW{1,1}
twts = -3    -1
tbiase = net.b{1}
tbiase = 0
```

Моделируя полученную сеть по каждому входу, получим:

```
a1 = sim(net,p)
a1 = [0] [0] [1] [1]
```

Можно убедиться, что не все выходы равны целевым значениям обучающего множества. Это означает, что следует продолжить настройку персептрона. Выполним еще 1 цикл настройки:

```
[net,a,e] = adapt(net,p,t);
a
a = [0]    [0]    [0]    [1]
e
e = [0]    [1]    [0]    [0]
twts = net.IW{1,1}
twts = 2    -3
tbiase = net.b{1}
tbiase = 1
a1 = sim(net,p)
a1 = [0]    [1]    [0]    [1]
```

Теперь решение совпадает с целевыми выходами обучающего множества и все входы классифицированы правильно.

Если бы рассчитанные выходы персептрона не совпали с целевыми значениями, то необходимо было бы выполнить еще несколько циклов настройки, применяя М-функцию adapt и проверяя правильность получаемых результатов.

Контрольные вопросы к защите:

1. Алгоритм настройки нейрона с пороговой функцией активации.
2. Что такое эпоха обучения?
3. На схеме нейрона покажите входы и цели, в векторной форме представьте сигналы.
4. Что такое обучающая выборка, тестовое множество.

Лабораторная работа № 3

Распознавание образов

Цель лабораторной работы:

Знакомство с обобщающими свойствами нейронной сети в задачах классификации, обучение нейронной сети, подбор ее параметров и формирование обучающего множества.

Теоретическая часть

Нейронные сети обратного распространения – это мощный инструмент анализа, поиска закономерностей, прогнозирования и управления. Такое название – сети обратного распространения (back propagation) - они получили из-за алгоритма обучения, в котором ошибка распространяется от выходного слоя к входному, т. е. в направлении, противоположном направлению распространения сигнала. При обучении ставится задача минимизации целевой функции ошибки нейросети:

$$E(w) = \frac{1}{2}\sum_{j=1}^{p}(y_j - d_j)^2$$

где y_j – значение j-го выхода нейросети, d_j – целевое значение j-го выхода, p – число нейронов в выходном слое.

Существует множество алгоритмов и их разновидностей для обучения нейронных сетей обратного распространения, рассмотрим один из простейших. Обучение нейросети производится методом градиентного спуска, т. е. на каждой итерации изменение веса производится по формуле

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}} \qquad (*)$$

где η – параметр, определяющий скорость обучения.

Справедливо равенство:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{dS_j} \cdot \frac{\partial S_j}{\partial w_{ij}} \qquad (**)$$

где y_j – значение выхода j-го нейрона, $S = \sum_{i=1}^{n} x_i w_i$ взвешенная сумма входных сигналов.

При этом $\frac{\partial S_j}{\partial w_{ij}} \equiv x_i$, где x_i – значение i-го входа нейрона.

Рассмотрим определение первого множителя формулы (**)

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{dS_k} \cdot \frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{dS_k} \cdot w_{jk}^{(n+1)}$$

где k – число нейронов в слое $n+1$.

Введем вспомогательную переменную

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{dS_j}$$

Тогда мы сможем определить рекурсивную формулу для определения $\delta_j^{(n)}$ n-го слоя, если нам известно $\delta_k^{(n+1)}$ следующего ***(n+1)***-го слоя:

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \right] \cdot \frac{dy_j}{dS_j}$$

Нахождение $\delta_j^{(n)}$ для последнего слоя НС не представляет трудности, так как нам известен целевой вектор, т. е. вектор тех значений, которые должна выдавать НС при данном наборе входных значений.

$$\delta_j^{(N)} = \left(y_i^{(N)} - d_i\right) \cdot \frac{dy_i}{dS_i}$$

Теперь запишем формулу (*) в раскрытом виде

$$\Delta w_{ij}^{(n)} = -\ \eta \cdot \delta_j^{(n)} \cdot x_i^n$$

Рассмотрим полный алгоритм обучения нейросети:

1. Подать на вход НС один из требуемых образов и определить значения выходов нейронов сети.
2. Рассчитать $\delta^{(N)}$ для выходного слоя сети по формуле

$$\delta_j^{(N)} = \left(y_i^{(N)} - d_i\right) \cdot \frac{dy_i}{dS_i}$$

и изменения весов $\Delta w_{ij}^{(N)}$ выходного слоя N по формуле

$$\Delta w_{ij}^{(n)} = -\ \eta \cdot \delta_j^{(n)} \cdot x_i^n$$

3. По формулам

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)}\right] \cdot \frac{dy_j}{dS_j}$$

и

$$\Delta w_{ij}^{(n)} = -\ \eta \cdot \delta_j^{(n)} \cdot x_i^n$$

рассчитать $\delta^{(N)}$ и $\Delta w_{ij}^{(N)}$ для остальных слоев НС, *n = N-1..1*.

4. Скорректировать все веса НС

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t)$$

5. Если ошибка существенна, то перейти на шаг 1.

На этапе 2 сети поочередно в случайном порядке предъявляются вектора из обучающей последовательности.

Простейший метод градиентного спуска очень неэффективен в случае, когда производные по различным весам сильно отличаются. В этом случае для

плавного уменьшения ошибки надо выбирать очень маленькую скорость обучения, но при этом обучение может занять непозволительно много времени. Простейшим методом усовершенствования градиентного спуска является введение момента μ, когда влияние градиента на изменение весов изменяется со временем. Тогда

$$\Delta w_{ij}^{(n)}(t) = -\eta \cdot \delta_j^{(n)} \cdot x_i^n + \mu \Delta w_{ij}^{(n)}(t-1)$$

Дополнительным преимуществом от введения момента является способность алгоритма преодолевать мелкие локальные минимумы.

Встроенные функции обучения ППП Neural Network Toolbox:

- 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

Подробная информация об этих и других функциях Matlab NNT представлена на сайте matlab.ru и в [1]. Подробное описание функции тренировки traincgf приведено в приложении 2. Демонстрационная программа сценарий appcr1 иллюстрирует, как распознавание символов может быть выполнено в сети с обратным распознаванием.

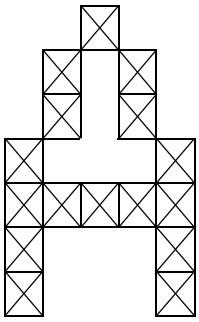
***

Задание

Создать нейронную сеть для распознавания символов латинского алфавита. В качестве датчика предполагается использовать систему распознавания, которая выполняет оцифровку каждого символа, находящегося в поле зрения. Обучить сеть на незашумленных символах, затем дообучить на зашумленных символах. Оценить минимальное и максимальное количество нейронов в скрытых слоях, минимальное и максимальное количество слоев, минимальный и максимальный уровень шума, оценить различные алгоритмы обучения при распознавании зашумленных букв латинского алфавита. Проектируемая нейронная сеть должна точно распознавать идеальные векторы входа и с максимальной точностью воспроизводить зашумленные векторы.

Порядок выполнения работы (пример)

М-функция prorob определяет 26 векторов входа, каждый из которых содержит 35 элементов, этот массив называется алфавитом. М-функция формирует входные переменные alphabet и targets, которые определяют массивы алфавита и целевых векторов. В результате каждый символ будет представлен шаблоном размера 5x7. Например символ **A** может быть представлен, как это показано на рисунке:



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

Для работы сети требуется 35 нейронов во входном слое – по количеству элементов матрицы, кодирующей символы алфавита и 26 в выходном – по количеству букв алфавита.

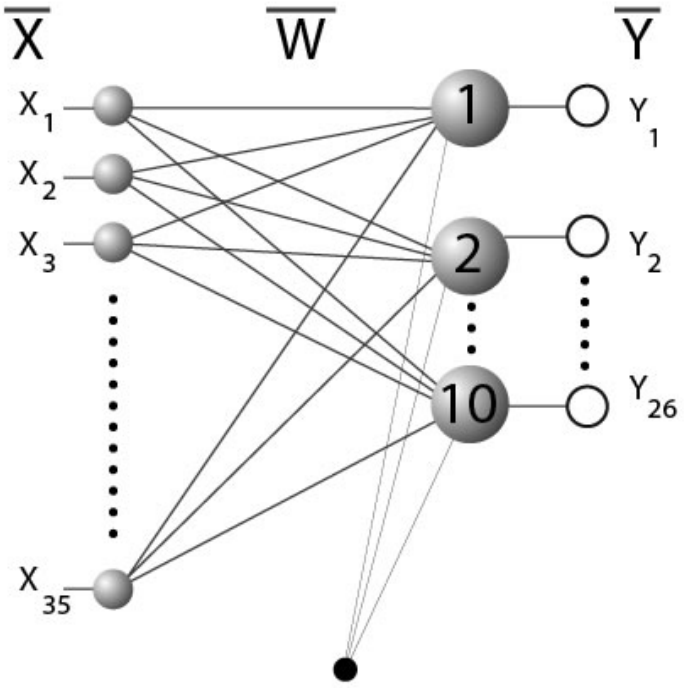


Рис. 10. Схема слоистой нейронной сети для задачи распознавания образов

Для решения задачи выберем, например, двухслойную нейронную сеть с логарифмическими сигмоидальными функциями активации в каждом слое. Такая функция активации выбрана потому, что диапазон выходных сигналов для этой функции определяется от 0 до 1, и этого достаточно, чтобы сформировать значения выходного вектора.

Скрытый слой имеет 10 нейронов. 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 вектор выхода, состоящий точно из нулей и единиц. Поэтому по завершении этапа обучения выходной сигнал обрабатывается М-функцией compet, которая присваивает значение 1 единственному элементу вектора выхода, а всем остальным – значение 0.

Инициализация сети:

clear; clc;

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

```
[alphabet, targets]=prprob;
[R,Q]=size(alphabet);
[S2,Q] = size(targets);
```

Двухслойная 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;
```

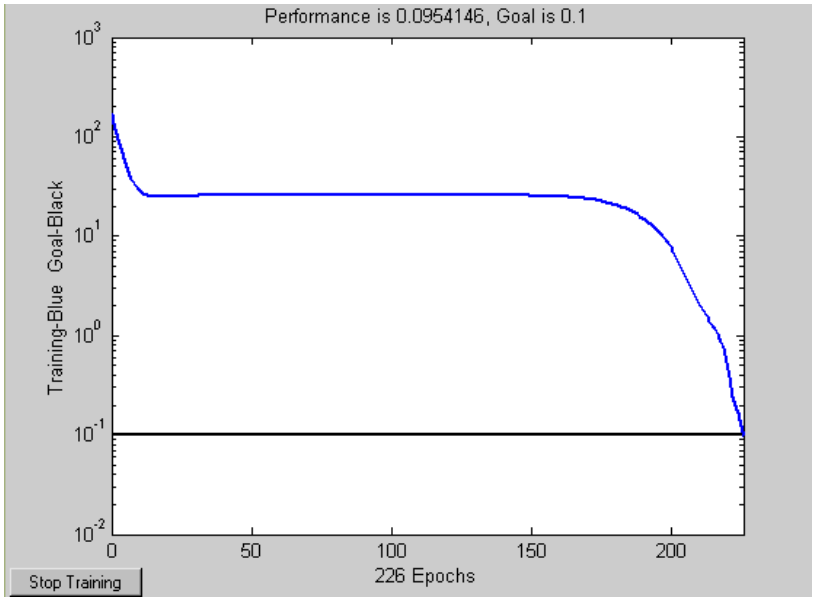
Обучение:

Чтобы создать нейронную сеть, которая может обрабатывать зашумленные векторы входа, следует выполнить обучение нейросети 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 trainbxp, которая реализует метод обратного распространения ошибки с возмущением и адаптацией параметра скорости настройки.

Обучение без шума:

Сеть первоначально обучается в отсутствие шума с максимальным числом циклов обучения 5000 либо до достижения допустимой средней квадратичной погрешности равной 0,1:

```
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);
```



Обучение в присутствии шума:

Чтобы спроектировать нейронную сеть, не чувствительную к воздействию шума, обучим ее с применением двух идеальных и двух зашумленных копий векторов алфавита. Целевые векторы состоят из четырех копий векторов. Зашумленные векторы имеют шум со средним значением 0,1 и 0,2. Это обучает нейрон правильно распознавать зашумленные символы и в то же время хорошо распознавать идеальные векторы.

При обучении с шумом максимальное число циклов обучения сократим до 300, а допустимую погрешность увеличим до 0,6:

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

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; %Максимальное количество циклов
обучения
net.trainParam.show = 5; % Частота выхода результатов на экран
[netn, tr] = train(netn, P, T);
```

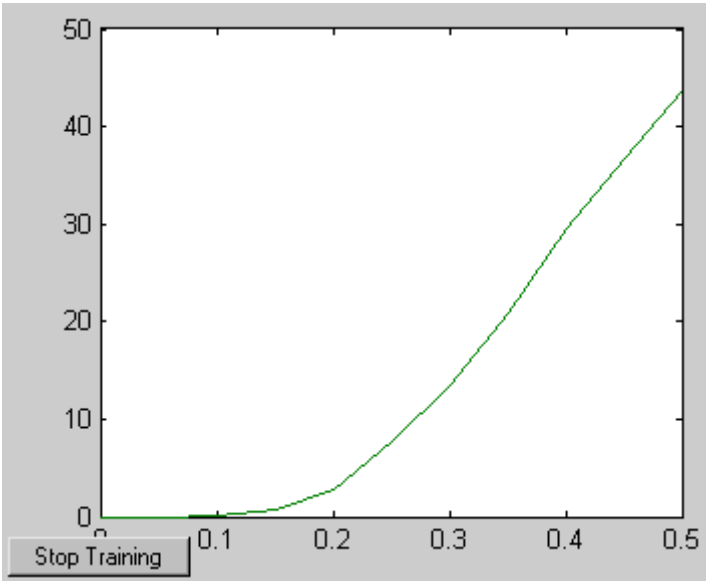
Эффективность функционирования системы:

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.

```
noise_range = 0:.05:.5;
max_test = 100;
network1 = [];
network2 = [];
T = targets;
%Выполнить тест
for noiselevel = noise_range
  errors1 = 0;
  errors2 = 0;

 for i=1:max_test
   P=alphabet + randn(35,26)*noiselevel;

   A = sim(net,P);
   AA = compet(A);
   errors1 = errors1 + sum(sum(abs(AA-T)))/2;

   An = sim(net,P);
   AAn = compet(An);
   errors2 = errors2 + sum(sum(abs(AAn-T)))/2;
   echo off
 end
 % Среднее значение ошибок (100 последовательностей из 26 векторов целей)
 network1 = [network1 errors1/26/100];
 network2 = [network2 errors2/26/100];
end
```

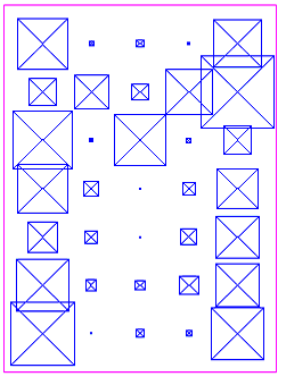
Тестирование реализуется следующим образом. Шум со средним значением 0 и стандартным отклонением от 0 до 0,5 с шагом 0,05 добавляется к векторам входа. Для каждого уровня шума формируется 100 зашумленных последовательностей для каждого символа и вычисляется выход сети. Выходной сигнал обрабатывается М-функцией compet с той целью, чтобы выбрать только один из 26 элементов вектора выхода. После этого оценивается количество ошибочных классификаций и вычисляется процент ошибки.

График погрешности сети от уровня входного шума:



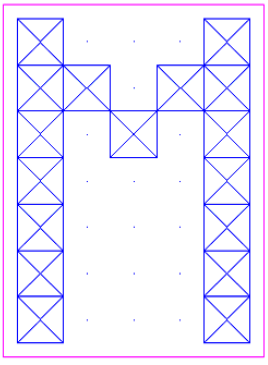
Проверим работу нейронной сети для распознавания символов. Сформулируем зашумленный вектор входа для символа M:

```
noisyM=alphabet(:,13)+randn(35,1)*0.2;
figure(1);plotchar(noisyM);%Зашумленный символ M;
```



```
A2 = sim(net,noisyM);
A2 = compet(A2);
answer = find(compet(A2) == 1)
plotchar(alphabet(:,answer)); %Распознанный символ
```

Нейронная сеть выделила 10 правильных элементов и восстановила символ M без ошибок:



Контрольные вопросы к защите:

1. Алгоритм обратного распространения ошибки.
2. Схема многослойной нейронной сети для распознавания образов.
3. Схема распознавания образов при прямом и обратном распространении сигнала в нейронной сети.
4. Приведите примеры распознавания образов из различных предметных областей (химия, кибернетика, металлургия, биология, медицина, промышленность и т.д.).

Лабораторная работа № 4

Аппроксимация нелинейных зависимостей

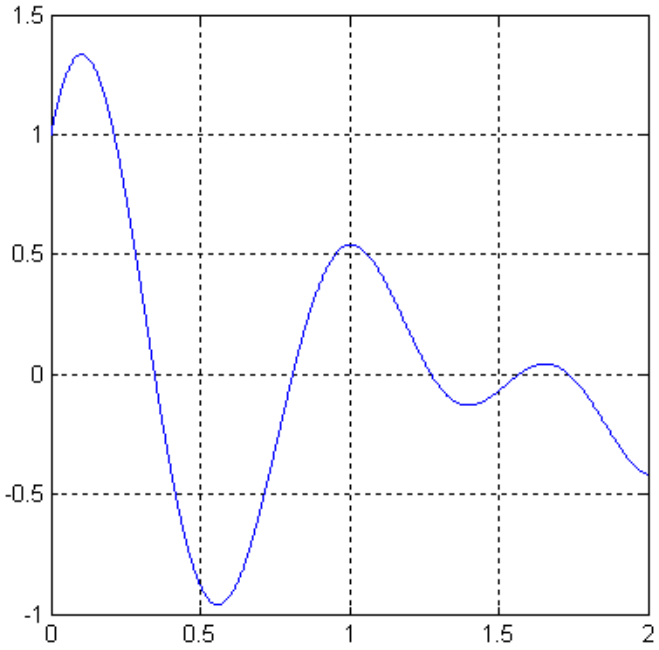
Цель работы:

Задать произвольную нелинейную (например, тригонометрическую) функцию, создать нейронную сеть и обучить ее по выборке из 20-30 % точек из области определения функции через равный интервал. Восстанавливать исходный сигнал, моделируя обученную сеть на всей области определения. Определить оптимальную архитектуру сети. Оценить влияние вида функции вычисления ошибки на качество аппроксимации.

***

Порядок выполнения работы (пример):

Зададим сигнал вида $y = \sin(2\pi x) + \cos(2\pi x + x)$ с областью определения [0; 2]. Дискретный сигнал получен в результате квантования исходного сигнала с тактом дискретности 0,0001. На рисунке показан график сигнала :

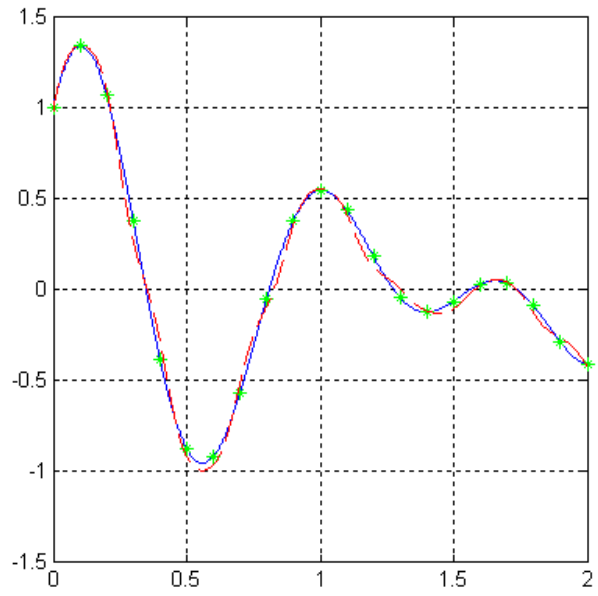


Создаем выборку из 30% точек и синтезируем нейронную сеть. Воспользуемся двухслойной моделью нейронной сети, количество нейронов в скрытом слое примем 10 (основываясь на результатах предыдущей лабораторной работы), в выходном – 1, по количеству аппроксимированных переменных. Функция активации в скрытом слое - гиперболическая тангенциальная, в выходном – линейная (это связано с областью определения аппроксимируемой функции).

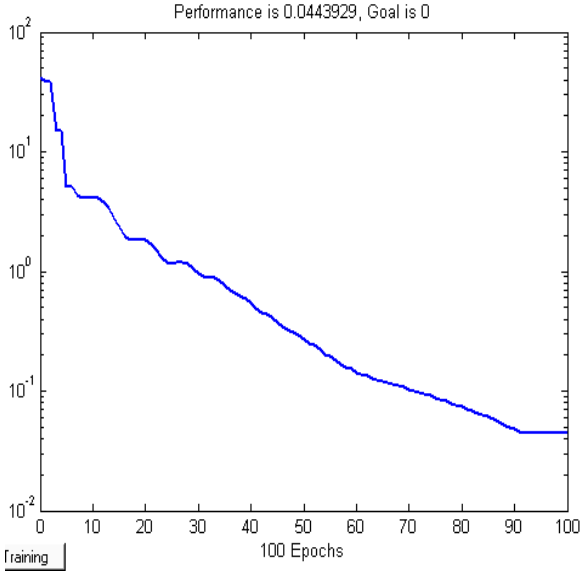
Функции оценки нейросети:
mae - средняя абсолютная ошибка
mse – среднеквадратичная ошибка
msereg - среднеквадратичная ошибка w/reg
sse - суммарная квадратичная ошибка

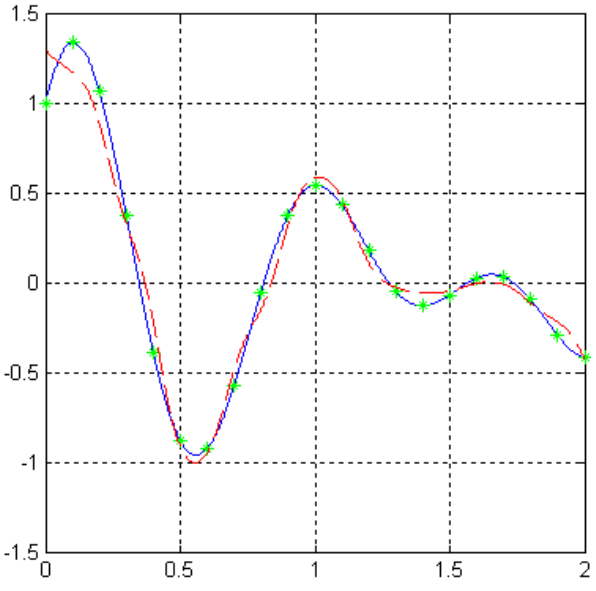
Подробнее об этих и других функциях см. в [1].

Составим таблицу, в первом столбце которой будет исходная и аппроксимированная функции, а во втором – функция ошибки. Это наглядно позволит увидеть влияние параметров ошибки на качество аппроксимации.

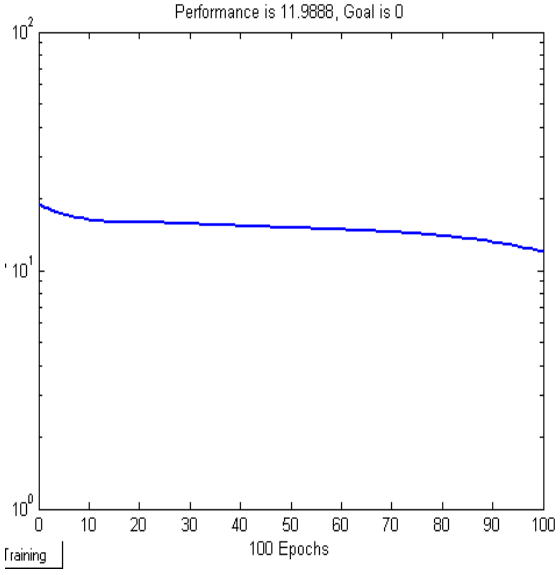


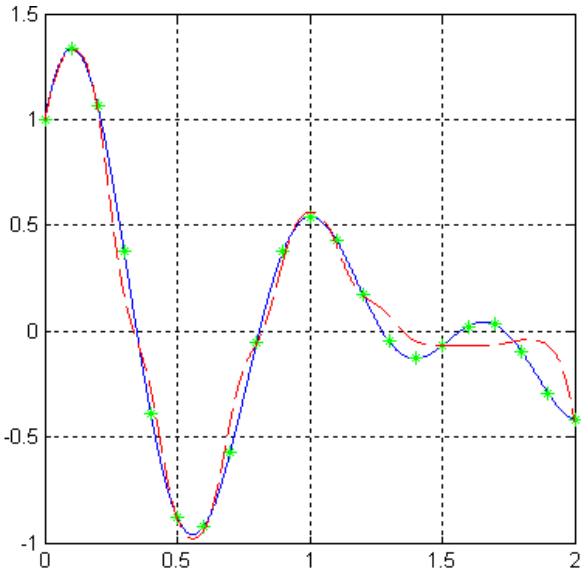
SSE – сумма квадратов ошибок



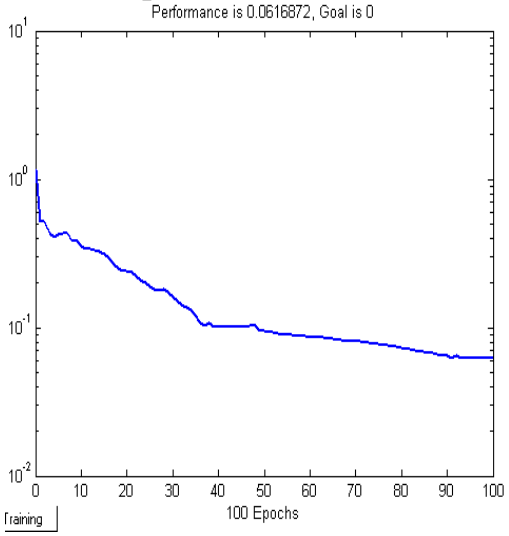


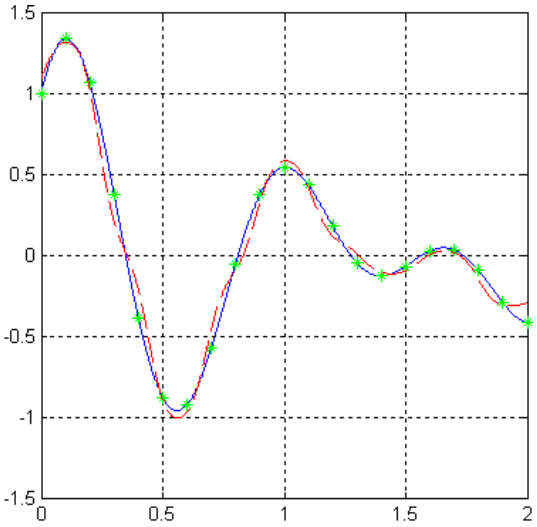
MSEREG – комбинированная ошибка



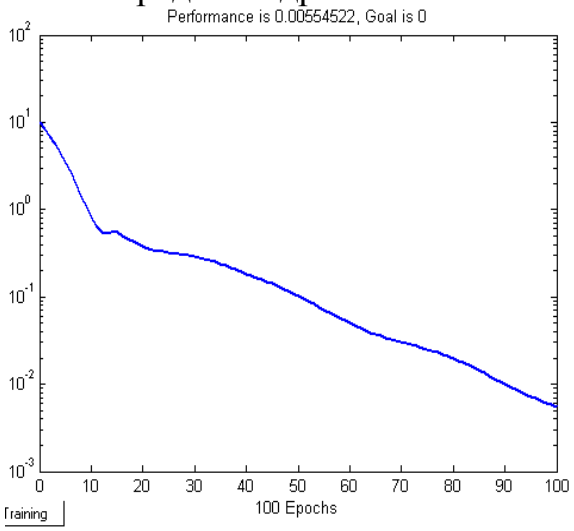


MAE – средняя абсолютная ошибка





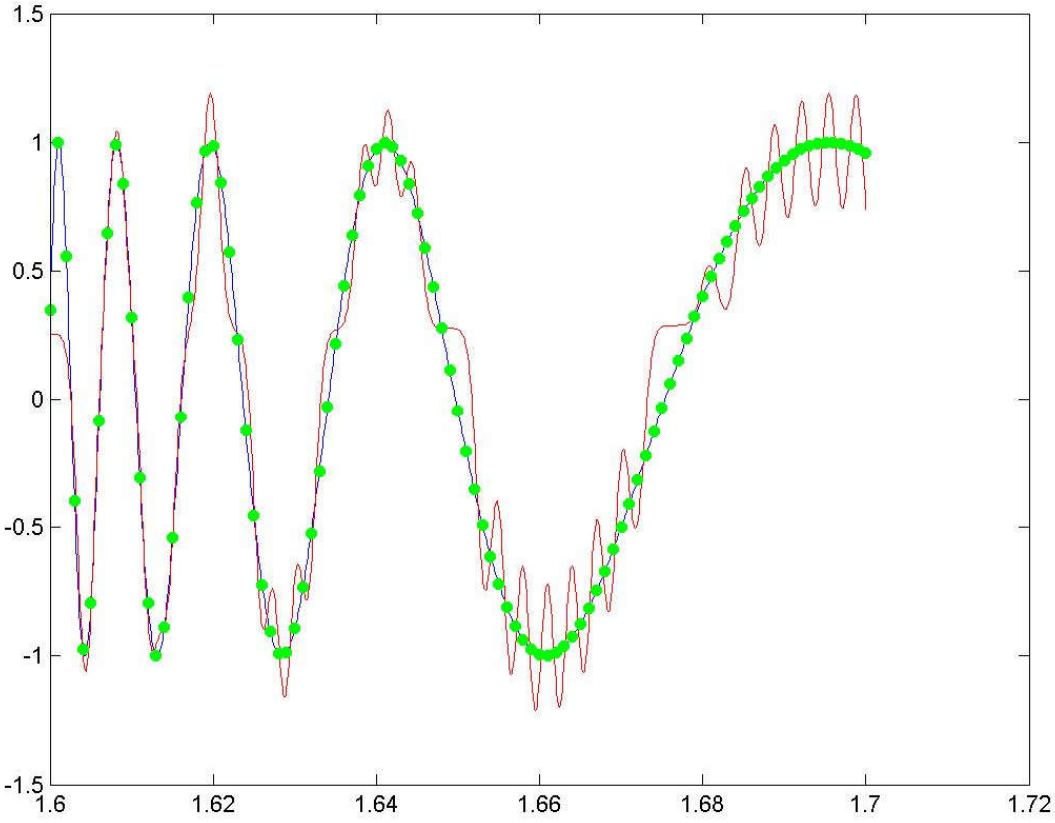
MSE – среднеквадратичная ошибка



Листинг программы:

```
clear; clc;
x=0:0.0001:2;
T=sin(x*2*pi)+cos(x*2*pi+x);
figure(2); plot(x,T);hold on; axis square; grid on;
p=0:0.1:2;
u=sin(p*2*pi)+cos(p*2*pi+p);
figure(2); plot(p,u,'*g'); hold on;
net = newff([minmax(p)], [10 1], {'tansig','purelin'}, 'traingdx');
% net.performFcn = 'msereg';
%net.performFcn = 'mae';
net = train(net,p,u)
res = sim(net,x)
figure(2); plot(x,res,'--r'); hold on;
gensim(net);
```

На рисунке ниже представлен пример явления переобучения нейронной сети. Более гладкая линия соответствует исходной функции, более волнистая синтезируется сетью по точкам. Видно, что в некоторых частях графика аппроксимация имеет значительную погрешность. В качестве задания для самостоятельной работы предлагается найти способ ухода от переобучения в зависимостях подобного вида. Следует отметить, что решение задач подобного вида может иметь большую практическую значимость.



Контрольные вопросы к защите:

1. Схема нейронной сети для аппроксимации нелинейных зависимостей. Ее входы и выходы, слои, обучающая выборка.
2. Почему обычно при аппроксимации в выходном слое функция активации линейна, а во всех остальных слоях - нелинейна?
3. Методы инициализации нейронной сети.
4. Приведите примеры нелинейных зависимостей в различных предметных областях (химия, кибернетика, металлургия, биология, медицина, промышленность и т.д.)

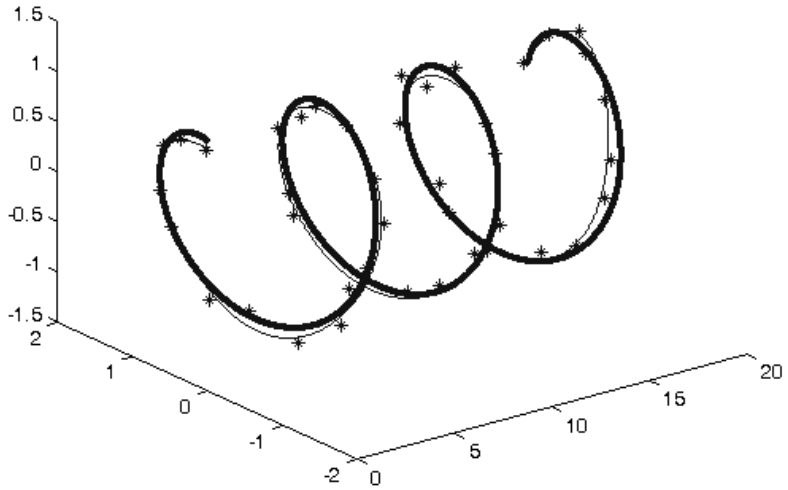
Лабораторная работа № 5

Аппроксимация нелинейных зависимостей в условиях шума

Эта лабораторная работа демонстрирует устойчивость нейросети к шумам в обучающей выборке. Задание аналогично предыдущей работе, только в обучающую выборку необходимо добавить шум. Уровень шума надо оценить. Для выполнения этой работы желательно использовать трехмерные зависимости.

Листинг примера:

```
clc; clear;
x=0:0.5:20; % интервал x для обучения
xx=0:0.0025:20; % тестовое множество
y=sin(x); z=cos(x); % зависимость
y1=(y+randn(1,size(y,2))*0.1); % добавление  шума
z1=(z+randn(1,size(z,2))*0.1); % добавление  шума
yz1 = [y1; z1];
net=newff([minmax(x)],[10 2],{'logsig','purelin'}); % создание  сети
net=train(net,x, yz1); % обучнение сети
res=sim(net,xx); % моделирование сети на тестовом множестве
figure();
plot3(xx,sin(xx),cos(xx), '.'); hold on; %- синий - исходная зависимость
plot3(x,y1,z1,'*b'); hold on; %- звездочки - зашумленные примеры
plot3(xx, res(1,:),res(2,:)); % - тонкая линия -  восстановленная по примерам
зависимость
```



Контрольные вопросы к защите:

1. Схема нейронной сети для аппроксимации нелинейных зависимостей в условиях шума. Ее входы и выходы, слои, обучающая выборка.
2. Природа шума в физическом мире при распознавании образов.
3. Приведите примеры нелинейных зависимостей в условиях шума в различных предметных областях (химия, кибернетика, металлургия, биология, медицина, промышленность и т.д.)

Лабораторная работа № 6

Построение нечеткой экспертной системы

*Теоретическая часть. Нечеткие выводы **

Используемый в различного рода экспертных и управляющих системах механизм нечетких выводов в своей основе имеет базу знаний, формируемую специалистами предметной области в виде совокупности нечетких предикатных правил вида:

$П_1$: если x есть A_1, тогда y есть B_1 ;
$П_2$: если x есть A_2, тогда y есть B_2 ;
……
$П_N$: если x есть A_N, тогда y есть B_N,

где

x — входная переменная (имя для известных значений данных);

y — переменная вывода (имя для значения данных, которое будет вычислено);

A и B — функции принадлежности, определенные соответственно на x и y.

Пример подобного правила: если x — низко, то y — высоко.
Приведем более детальное пояснение. Знание эксперта A —> B отражает нечеткое причинное отношение предпосылки и заключения, поэтому его можно назвать нечетким отношением и обозначить через R:

$$R = A \rightarrow B,$$

где «→» называют нечеткой импликацией.

---

* Здесь и далее текст взят из [2].

Нечеткий логический вывод осуществляется за следующие четыре этапа:

1. Нечеткость (введение нечеткости, фазификация, fuzzification). Функции принадлежности, определенные на входных переменных применяются к их фактическим значениям для определения степени истинности каждой предпосылки каждого правила.

2. Логический вывод. Вычисленное значение истинности для предпосылок каждого правила применяется к заключениям каждого правила. Это приводит к одному нечеткому подмножеству, которое будет назначено каждой переменной вывода для каждого правила. В качестве правил логического вывода обычно используются только операции min (МИНИМУМ) или prod (УМНОЖЕНИЕ). В логическом выводе МИНИМУМА функция принадлежности вывода «отсекается» по высоте, соответствующей вычисленной степени истинности предпосылки правила (нечеткая логика «И»). В логическом выводе УМНОЖЕНИЯ функция принадлежности вывода масштабируется при помощи вычисленной степени истинности предпосылки правила.

3. Композиция. Все нечеткие подмножества, назначенные каждой переменной вывода (во всех правилах), объединяются вместе, чтобы формировать одно нечеткое подмножество для каждой переменной вывода. При подобном объединении обычно используются операции max (МАКСИМУМ) или sum (СУММА). При композиции МАКСИМУМА комбинированный вывод нечеткого подмножества конструируется как поточечный максимум по всем нечетким подмножествам (нечеткая логика «ИЛИ»). При композиции СУММЫ комбинированный вывод нечеткого подмножества конструируется как поточечная сумма по всем нечетким подмножествам, назначенным переменной вывода правилами логического вывода.

4. В заключение (дополнительно) - приведение к четкости (дефазификация, defuzzification), которое используется, когда полезно преобразовать нечеткий набор выводов в четкое число. Имеется большое количество методов приведения к четкости, некоторые из которых рассмотрены в приложении 3.

Пусть некоторая система описывается следующими нечеткими правилами:

$П_1$: если x есть A, тогда w есть D
$П_2$: если y есть B, тогда w есть E
$П_3$: если z есть C, тогда w есть F

где x, y и z — имена входных переменных, w — имя переменной вывода, A, B, C, D, E, F — заданные функции принадлежности (треугольной формы).

Процедура получения логического вывода иллюстрируется рис. 11. Предполагается, что входные переменные приняли некоторые конкретные (четкие) значения – x_0, y_0, z_0.

В соответствии с приведенными этапами на этапе 1 для данных значений и исходя из функций принадлежности A, B, C находятся степени истинности $\alpha(x_0)$, $\alpha(y_0)$ и $\alpha(z_0)$ для предпосылок каждого из трех приведенных правил (см. рис. 11).

На этапе 2 происходит «отсекание» функций принадлежности заключений правил (т.е. D, E, F) на уровнях $\alpha(x_0)$, $\alpha(y_0)$ и $\alpha(z_0)$.

На этапе 3 рассматриваются усеченные на втором этапе функции принадлежности и производится их объединение с использованием операции max, в результате чего получается комбинированное нечеткое подмножество, описываемое функцией принадлежности $\mu_{\Sigma}(w)$ и соответствующее логическому выводу для выходной переменной w.

Наконец, на 4-м этапе — при необходимости — находится четкое значение выходной переменной, например, с применением центроидного метода: четкое значение выходной переменной определяется как центр тяжести для кривой $\mu_{\Sigma}(w)$, т.е.

$$w_o = \frac{\int\limits_{\Omega} w\mu_{\Sigma}(w)dw}{\int\limits_{\Omega} \mu_{\Sigma}(w)dw}$$

Рассмотрим следующие наиболее часто используемые модификации алгоритма нечеткого вывода, полагая для простоты, что базу знаний организуют два нечетких правила вида:

$П_1$: если x есть A_1 и B_1, тогда z есть C_1 ;
$П_2$: если x есть A_2 и B_2, тогда z есть C_2 ,

где x и y – имена входных переменных, z – имя переменной вывода, A_1,A_2,B_1,B_2,C_1,C_2 – некоторые заданные функции принадлежности, при этом четкое значение z_0 необходимо определить на основе приведенной информации и четких значений x_0 и y_0.

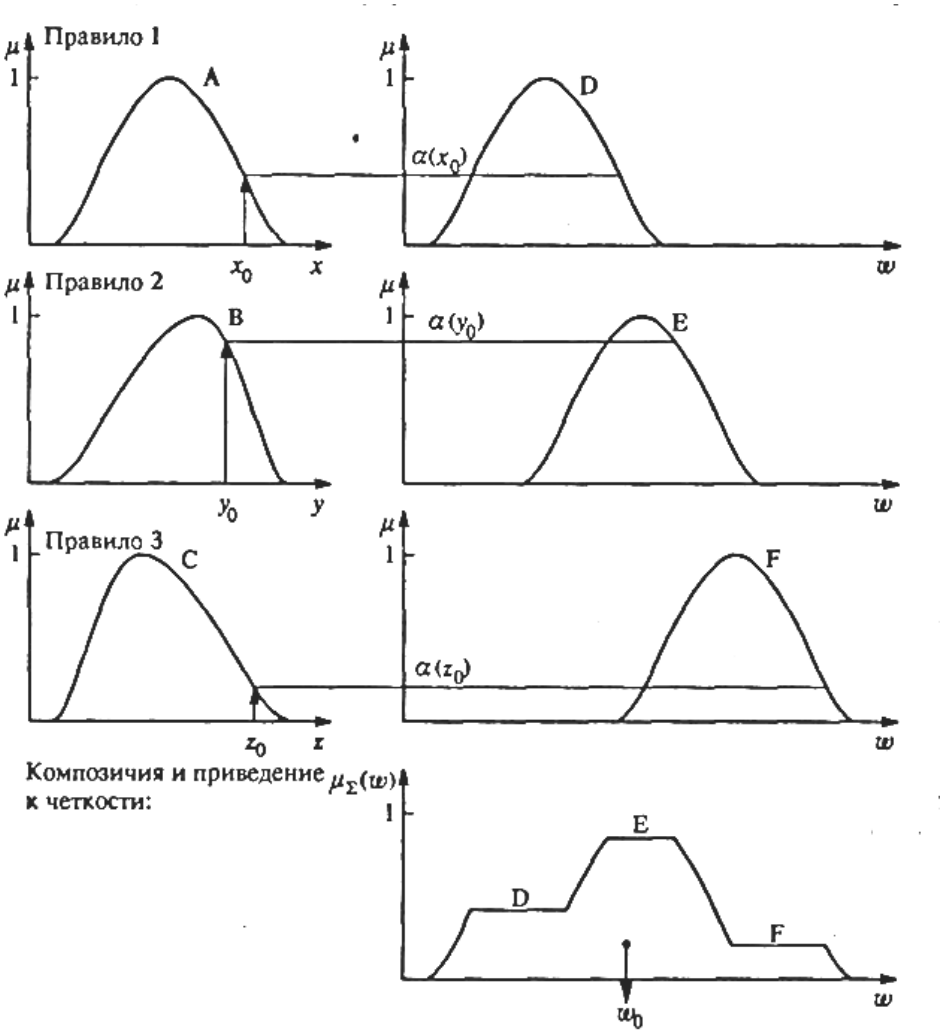


Рис. 11. Иллюстрация к процедуре логического вывода

Алгоритм Mamdani. Данный алгоритм соответствует рассмотренному примеру и рис. 11. В рассматриваемой ситуации он математически может быть описан следующим образом:

1. Нечеткость: находятся степени истинности для предпосылок каждого правила:

$A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_1(y_0)$.

2. Нечеткий вывод: находятся уровни «отсечения» для предпосылок каждого из правил (с использованием операции МИНИМУМ)

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0);$$
$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

где через «$\wedge$» обозначена операция логического минимума (min), затем находятся «усеченные» функции принадлежности:

$$C_1'(z) = (\alpha_1 \wedge C_1(z));$$
$$C_2'(z) = (\alpha_2 \wedge C_2(z)).$$

3. Композиция: с использованием операции МАКСИМУМ (max, далее обозначаемой как «V») производится объединение найденных усеченных функций, что приводит к получению итогового нечеткого подмножества для переменной выхода с функцией принадлежности:

$$\mu_{\Sigma}(z) = C(z) = C_1'(z) \vee C_2'(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)).$$

4. Наконец, приведение к четкости (для нахождения z_0 проводится, например, центроидным методом).

Пример нечеткой экспертной системы "сколько будет стоить туристическая поездка?"

Рассмотрим теперь методику построения нечеткой экспертной системы, которая должна помочь пользователю с ответом на вопрос: сколько будет стоить туристическая поездка?

Основываясь на каких-то устоявшихся обычаях и интуитивных представлениях, примем, что задача о чаевых может быть описана следующими предложениями:

Если едем в Европу или в плохую гостиницу, то цена будет низкой

Если едем в Азию или в среднюю гостиницу, то цена будет средней

Если едем в Америку или в гостиницу высшей категории, то цена будет высокой

Если поездка осуществляется поездом, то цена будет ниже

Если поездка осуществляется самолетом, то цена будет выше

Категорию гостиницы будем определять по классу «звездности»:

0-2 – низшая категория

2-4 – средняя категория

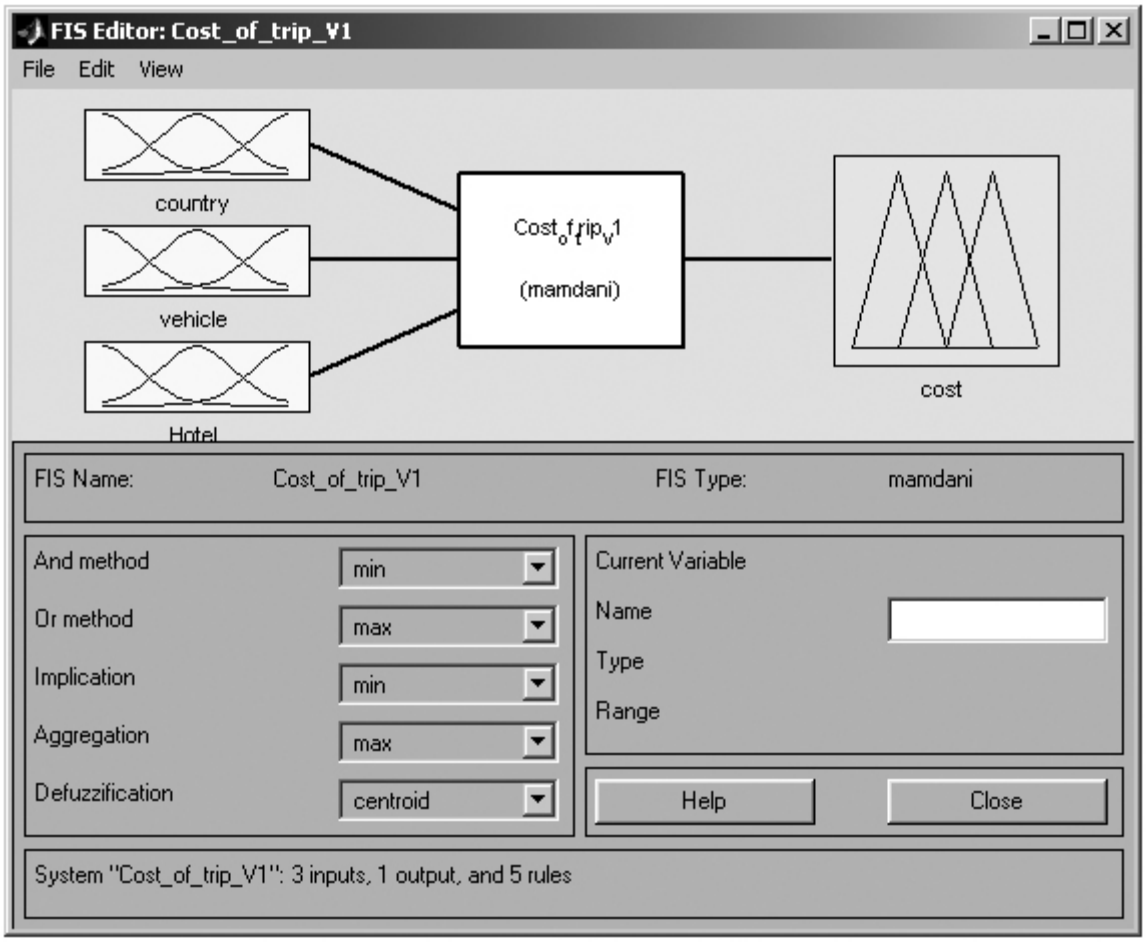
4-5 – высшая категория

Будем предполагать далее, что отправным пунктом будет Европа, а Америка будет самой дальней точкой маршрута. Представленной информации достаточно для проектирования простой нечеткой экспертной системы.

Система будет иметь:

- Три входа: «Страна\Транспорт\Гостиница»
- Один выход: «цена поездки»
- Пять правил «если…то»(в соответствии с приведенными условиями) для центров функций и принадлежности входов и выходов.

Построим данную систему, используя алгоритм вывода Mamdani:



Вид окна FIS-редактора после задания структуры системы

Командой fuzzy запускаем FIS-редактор. По умолчанию, исходный алгоритм вывода — типа Mamdani (о чем говорит надпись в центральном белом блоке), здесь никаких изменений не требуется, но в системе должно быть три входа, поэтому через пункт меню Edit/Add input добавляем в систему еще два входа (в окне редактора появляется второй и третий желтый блок с именем input2 и input3). Делая далее однократный щелчок левой кнопкой мыши по блоку input 1, меняем в поле имени его имя на страна(country), завершая ввод нового имени нажатием клавиши Enter. Аналогичным образом устанавливаем имя транспортное средство(vehicle) блоку input2, блоку input3 – «гостиница»(hotel) и «цена»(cost) — выходному блоку (справа вверху) outputl. Присвоим сразу же и имя всей системе, например, «cost of trip» (по-английски это – цена тур. поездки), выполнив это через пункт меню File/Save to workspace as... (Сохранить в рабочем пространстве как...). Вид окна редактора после указанных действий приведен на рисунке.

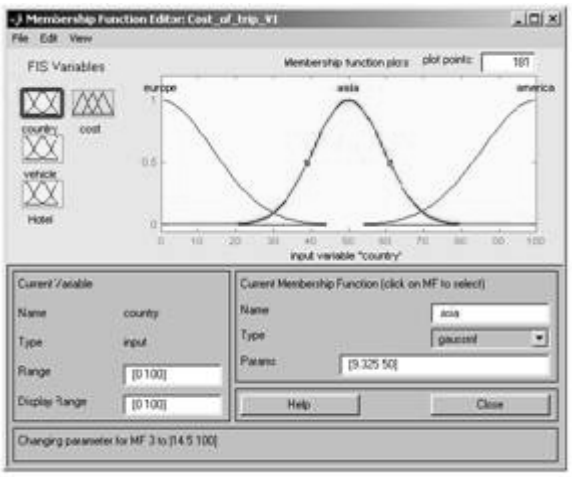
2. Зададим теперь функции принадлежности переменных. Программу-редактор функций принадлежности можно открыть тремя способами:

- через пункт меню View/Edit membership functions...,

• двойным щелчком левой кнопки мыши по иконке, отображающей соответствующую переменную,
• нажатием клавиш Ctrl+2.

Любым из приведенных способов перейдем к данной программе. Задание и редактирование функций принадлежности начнем с переменной «страна».

Будем считать точкой отправления всех маршрутов Москву. Тогда все конечные точки маршрутов можно разделить по удаленности на три основные группы: Европа, Азия и Америка. Введем условную шкалу от 0 до 100 процентов, обозначающую удаленность пункта назначения от пункта отправления. В полях Range и Display Range установим диапазон изменения и отображения этой переменной — от 0 до 100, подтверждая ввод нажатием клавиши Enter. Затем кликнув на надписи mf1(она должна выделиться красным цветом) в поле(*name*) пишем «Европа», в поле тип(*type*) меняем значение с trimf на gaussmf, а в поле *параметры* записываем [13.82 0], далее кликнем на значении mf2 и, повторяя действия, меняем имя на Азия, тип на gaussmf, а параметры на [9.325 50], после этого кликнем на значение mf3, изменим параметры: имя – Америка, тип – gaussmf, параметры - [14.5 100].
В итоге получим:



Затем изменим параметр «Hotel» (гостиница) аналогично предыдущему примеру:
Для параметра mf1 введем значения: имя – плохая, тип – trimf, параметры - [-5 0 5]
Для параметра mf2 введем значения: имя – средняя, тип – gaussmf, параметры - [1.5 5]
Для параметра mf3 введем значения: имя – хорошая, тип – trimf, параметры - [5 10 15]

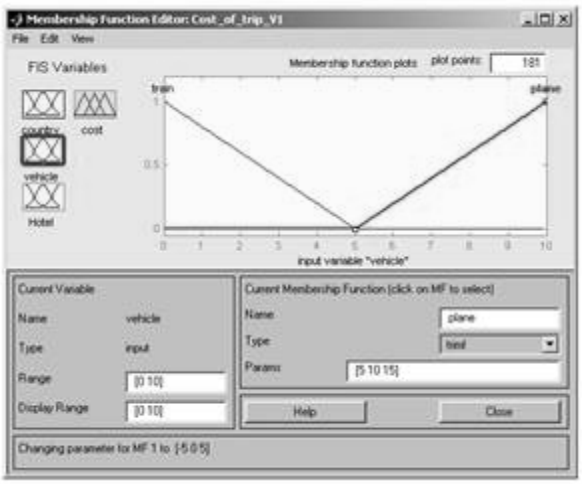
Так как по условию задачи, транспорт имеет только два типа переменных: самолет и поезд, то в блоке «vehicle»(транспорт) нужно удалить одну из переменных.

Делается это следующим образом:

- Выделяется одна из переменных(mf1,mf2,mf3) – любая
- Нажать клавишу Del

Можно также воспользоваться меню edit→remove selected mf. Далее вводим значения аналогично предыдущим примерам, для оставшихся переменных:
Для первой переменной введем значения: имя – поезд, тип – trimf, параметры - [-5 0 5]
Для второй переменной введем значения: имя – самолет, тип – trimf,параметры - [5 10 15]
Диапазон значений для данной переменной [0 10] – вводится в область Range.

В итоге должно получиться следующее:



3. Перейдем к конструированию правил. Для этого выберем пункт меню View/Edit rules... Далее ввод правил производится следующим образом:

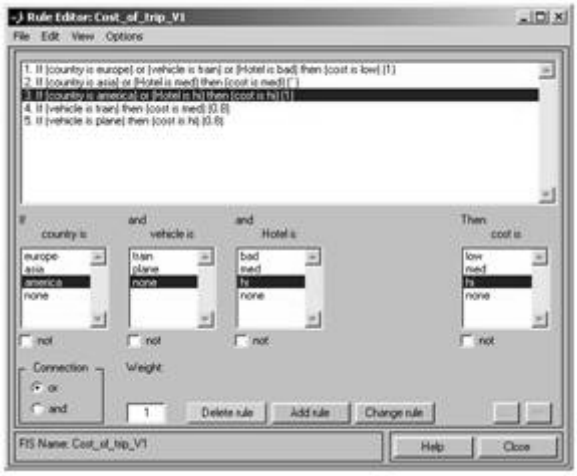
1. Открываем окно ввода правил, кликнув на центральном (белом) блоке или выполнив пункт меню Edit → Rules, или нажав комбинацию клавиш Ctrl+3.

2. Выбираем параметры будущего правила, кликая на ползунки, в конце нажимаем клавишу Add Rule.

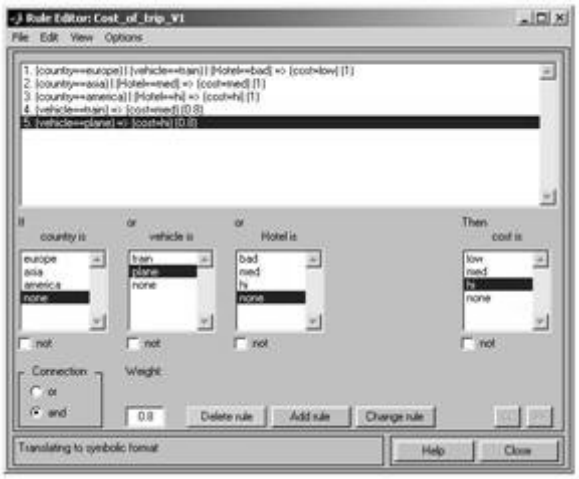
Правила можно удалять и изменять, для чего нужно нажать соответствующие кнопки: Change Rule, Delete Rule.

Правилам можно назначать веса, максимальный вес 1, минимальный 0, в соответствии с предложениями, описывающими задачу. Заметим, что в первом, втором, третьем правилах в качестве «связки» в предпосылках правила не-

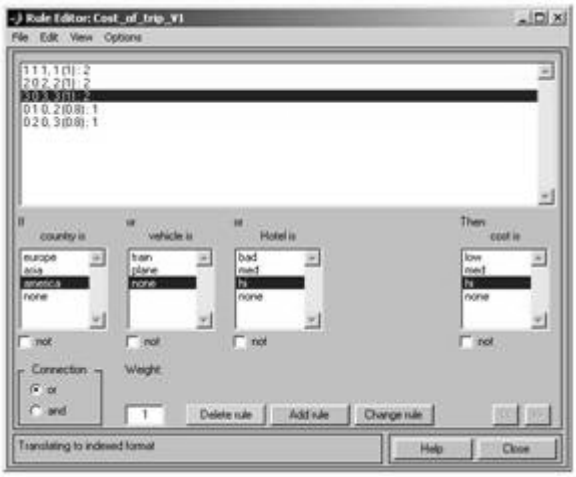
обходимо использовать не «И» (and), а «ИЛИ» (or); при вводе четвертого и пятого правила, где отсутствует переменная «транспорт», для нее выбирается опция none. Итоговый набор правил отображен на рисунке и выглядит следующим образом:



Такая (подробная, verbose) запись представляется достаточно понятной; единица в скобках после каждого правила указывает его «вес» (Weight), т.е. значимость правила. Данный вес можно менять, используя соответствующее поле в левой нижней части окна редактора правил. Правила представимы и в других формах: символической (symbolic) и индексной (indexed), при этом переход от одной формы к другой происходит через опции пункта меню редактора правил Options/Format. Вот как выглядят рассмотренные правила в символической форме:

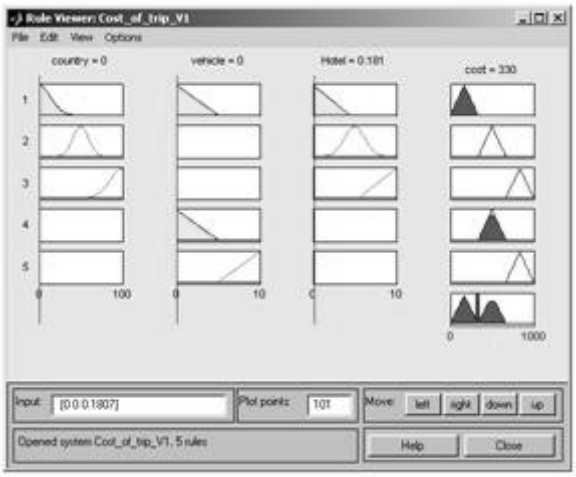


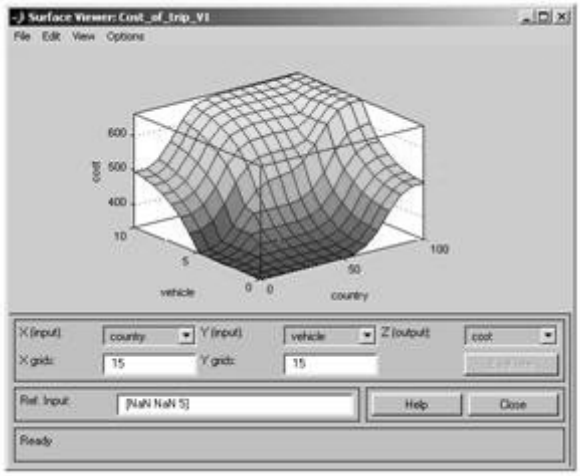
Наконец, самый сжатый формат представления правил — индексный — является тем форматом, который в действительности используется программой. В этом формате приведенные правила выглядят так:



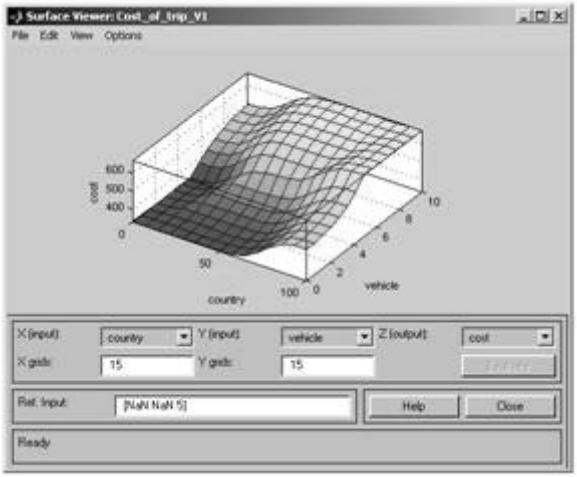
4. Проверим систему в действии. Откроем (через пункт меню View/View rules...) окно просмотра правил и установим значения переменных:

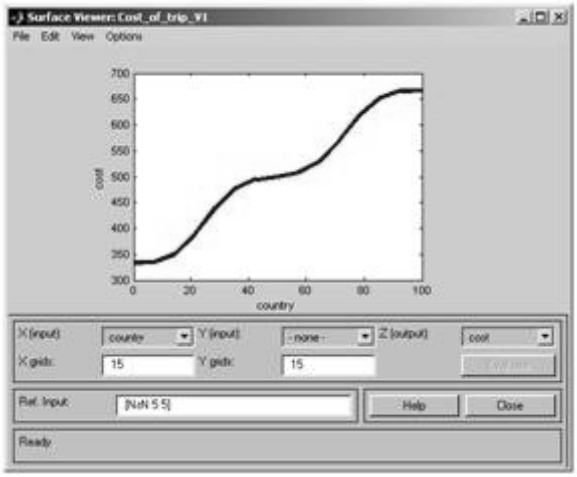
Страна=0 (т. е. Европа), Траспорт=0 (т.е. поезд) и Гостиниция=0 (т.е. низшая категория) . Увидим ответ: Цена поездки= 330 (т.е. низкая). Ну что ж, с системой не поспоришь, надо платить.





Подтверждением отмеченной зависимости выходной переменной от входных может служить вид поверхности отклика, который представляется при выборе пункта меню View/View surface.





С помощью мышки график можно поворачивать во все стороны.

В открывшемся окне, меняя имена переменных в полях ввода (X(input) и Y(input)), можно задать и просмотр одномерных зависимостей, например «страны» от «цены».

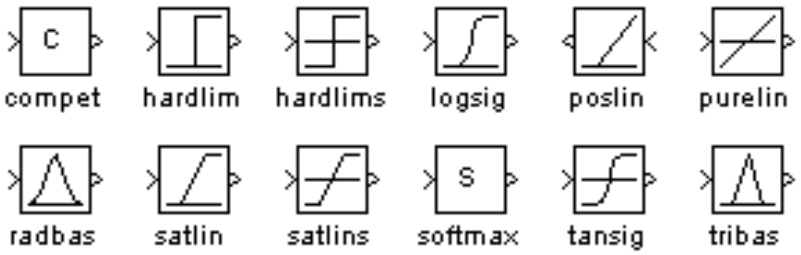
Контрольные вопросы к защите:

1. Лингвистическая переменная.
2. Нечеткое число.
3. Функции принадлежности. Виды задания функции принадлежности, формулы.
4. Логический нечеткий вывод. Алгоритмы Mamdani.
5. Нечеткая логика в различных предметных областях.

ПРИЛОЖЕНИЕ 1

Выписки из справочника [1]

Раздел Transfer Functions пакета MatLab Neural Network Toolbox включает в себя следующие функции активации:



- ***Compet -*** конкурентная функция активации. *Синтаксис:* A = compet(N). Compet(N) имеет один входной аргумент – N - SxQ матрицу входных векторов (столбцов) и возвращает выходные векторы с единицей в позиции, где входной вектор имеет максимальное значение, и нули в остальных позициях.

- ***Hardlim -*** ступенчатая (пороговая) функция активации. *Синтаксис:* A = hardlim(N). Hardlim(N) имеет один входной аргумент – N - SxQ матрицу входных векторов (столбцов) и возвращает выходные векторы с единицей в позиции, где элемент входного вектора имеет положительное значение и нули в остальных позициях. *Алгоритм:* hardlim(n) = 1, если n≥0 и 0, в остальных случаях.

- ***Hardlims -*** симметричная ступенчатая функция активации. *Синтаксис:* A=hardlims(N). Hardlims(N) имеет один входной аргумент – N - SxQ матрицу входных векторов (столбцов) и возвращает выходные векторы с +1 в позиции, где элемент входного вектора имеет положительное значение и -1 в остальных позициях. *Алгоритм:* hardlims(n) = 1, если n ≥0 и -1, в остальных случаях.

- ***Logsig -*** логарифмическая сигмоидальная функция активации. *Синтаксис:* A=logsig(N). Logsig (N) имеет один входной аргумент – N - SxQ матрицу входных векторов (столбцов) и возвращает выходные векторы со значениями в диапазоне от 0 до 1. *Алгоритм:* logsig(n)=1/(1+exp(-n)).

- ***Poslin -*** положительная линейная функция активации. *Синтаксис:* A = poslin(N). Poslin(N) имеет один входной аргумент – N - SxQ матрицу входных векторов (столбцов) и возвращает значения входных векторов в

позиции, где элемент входного вектора имеет положительное значение и 0 в остальных позициях. *Алгоритм:* poslin(n) = n, если n≥0 и 0, если n≤0.

• ***Purelin -*** линейная функция активации. *Синтаксис:* A = purelin(N). Purelin(N) имеет один входной аргумент – N - SxQ матрицу входных векторов (столбцов) и возвращает выходные векторы без изменений. *Алгоритм:* purelin(n)=n.

• ***Radbas -*** радиальная базисная функция активации. *Синтаксис:* A = radbas(N). Radbas (N) имеет один входной аргумент - N - SxQ матрицу входных векторов (столбцов) и возвращает векторы, элементы которых вычисляются с помощью радиальной базисной функции. *Алгоритм:* $a=\exp(-n^2)$.

• ***Satlin -*** линейная функция активации с насыщением. *Синтаксис:* A = satlin(N). Satlin(N) имеет один входной аргумент - N - SxQ матрицу входных векторов (столбцов) и возвращает векторы, элементы которых находятся в диапазоне от 0 до +1. *Алгоритм:* satlin(n)=0, если n≤0; satlin(n)=n, если 0≤n≤1; satlin(n)=1, если n≥1.

• ***Satlins -*** симметричная линейная функция с насыщением. *Синтаксис:* A = satlins(N). Satlins(N) имеет один входной аргумент - N - SxQ матрицу входных векторов (столбцов) и возвращает векторы, элементы которых находятся в диапазоне от -1 до +1. *Алгоритм:* satlin(n)=-1, если n≤0; satlin(n)=n, если -1≤n≤1; satlin(n)=1, если n≥1.

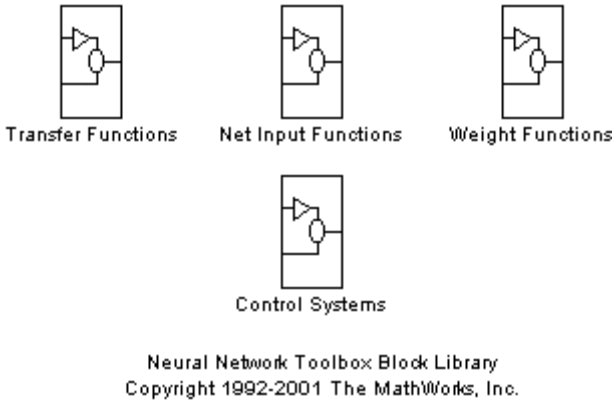
• ***Softmax -*** функция активации, уменьшающая диапазон значений. *Синтаксис:* A = softmax(N). Softmax(N) имеет один входной аргумент - N - SxQ матрицу входных векторов (столбцов) и возвращает векторы, элементы которых находятся в диапазоне от 0 до +1 при сохранении отношений между элементами.

• ***Tansig -*** сигмоидная функция в виде гиперболического тангенса. *Синтаксис:* A = tansig(N). Tansig(N) имеет один входной аргумент - N - SxQ матрицу входных векторов (столбцов) и возвращает выходные векторы со значениями в диапазоне от -1 до 1. *Алгоритм:* tansig(n)= (2/(1+exp(-2n)))-1. Эта функция математически эквивалентна Tanh(n), но она вычисляется в MATLAB намного быстрей, чем Tanh(n) и мало отличается от нее. Tansig больше подходит для использования в нейронных сетях, где важна высокая скорость вычислений.

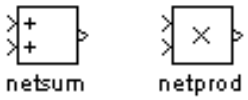
• ***Tribas -*** треугольная функция активации. *Синтаксис:* A = tribas(N). Tribas(N) имеет один входной аргумент - N - SxQ матрицу входных

векторов (столбцов) и возвращает выходные векторы в соответствии с функцией tribas. *Алгоритм:* tribas(n)=1-abs(n), если $-1 \le n \le 1$; tribas(n)=0, если $-1 > n > 1$.

Библиотека "Library: neural" выглядит следующим образом:



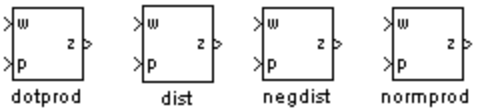
Раздел Net Input Functions включает в себя следующие функции:



- ***Netsum*** - сумма взвешенных входов. Функция N=netsum(z_1, z_2, …) вычисляет функцию накопления потенциала нейрона в виде суммы элементов взвешенных входов z_i размера S·Q, где S-число нейронов в слое, Q-число элементов вектора входа.

- ***Netprod -*** поэлементное произведение взвешенных входов. Функция N=netprod(z_1, z_2, …) вычисляет функцию накопления потенциала нейрона в виде поэлементного произведения массивов взвешенных входов z_i размера S·Q, где S-число нейронов в слое, Q-число элементов вектора входа. $N=\Pi z_i = z_1 \cdot z_2 \ldots \cdot z_n$

Раздел Weight Functions включает в себя следующие функции:



- ***Dotprod*** **-** скалярное произведение. Функция z=dotprod(w,p) вычисляет матрицу взвешенных входов в виде произведения w·p массива входов p размера R·Q и матрицы весов w размера S·R. Результатом является матрица z размера S·Q.

- ***Dist -*** евклидово расстояние. Функция взвешивания z=dist(w,p) вычисляет евклидово расстояние между каждой строкой w_i матрицы весов w и каждым столбцом p_j матрицы входов p в соответствии со следующим соотношением: z_{ij}=norm(w_i-p_j'). Функция z=dist(w,p) вычисляет матрицу взвешенных входов для матрицы входов p размера R·Q и матрицы весов w размера S·R.

- ***Negdist -*** отрицательное евклидово расстояние. Эта функция применяется в самоорганизующихся сетях как функция взвешивания. Функция взвешивания z=negdist(w,p) вычисляет отрицательное евклидово расстояние между каждой строкой w_i матрицы весов w и каждым столбцом p_j матрицы входов p в соответствии со следующим соотношением: z_{ij}=norm(w_i-p_j').

- ***Normprod -*** нормированное скалярное произведение. Функция взвешивания normprod отличается от функции dotprod тем, что она является нормированной функцией и не имеет производной. Функция z=normprod(w,p) вычисляет массив взвешенных входов в виде нормированного скалярного произведения w·p/sum(p,1), где w-матрица весов размера S·R, p-массив входов размера R·Q. Результатом является массив значений взвешенных входов z размера S·Q.

Команда **NEWFF** реализует однонаправленную сеть, обучаемую с применением алгоритма обратного распространения.

Синтаксис:
net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

Описание:
net = newff создает новую сеть с использованием диалогового окна.

NEWFF(PR, [S1 S2...SNl], {TF1 TF2...TFNl}, BTF, LF, PF) в качестве входных параметров использует:

PR - Rx2 матрица минимальных и максимальных значений строк входной матрицы с размерностью RxQ. Для получения матрицы PR можно использовать функцию minmax;

Si – количество нейронов в i –м слое, N1 – количество слоев;

TFi - функция активации i - го слоя, по умолчанию = 'tansig';

BTF – обучающая функция обратного распространения, по умолчанию='trainlm';

BLF – алгоритм подстройки весов и смещений (обучающий алгоритм), по умолчанию = 'learngdm';

PF - функция оценки функционирования сети, по умолчанию = 'mse';
и возвращает однонаправленную сеть, состоящую из N слоев.

В качестве функций активации TFi можно использовать любые дифференцируемые функции активации, например, TANSIG, LOGSIG, или PURELIN.

В качестве обучающей функции BTF можно использовать любые функции на основе алгоритма обратного распространения, например TRAINLM, TRAINBFG, TRAINRP, TRAINGD, и т.д

Следует обратить внимание, что функция TRAINLM используется по умолчанию, поскольку она обеспечивает наиболее быстрое обучение. Но она требует много памяти. Поэтому, если Вы получаете сообщение "out-of-memory error", необходимо попытаться выполнить одно из следующих действий:

(1) замедлить процедуру обучения TRAINLM, снизив требования к объему оперативной памяти путем установки параметра NET.trainParam.mem_reduc равным 2 или более (см. HELP TRAINLM);

(2) использовать функцию обучения TRAINBFG, которая работает медленнее, но требует меньше памяти, чем TRAINML;

(3) использовать функцию обучения TRAINRP, которая работает медленнее, но требует меньше памяти, чем TRAINBFG.

В качестве обучающей функции BLF можно использовать любые из алгоритмов обратного распространения. Например, LEARNGD или LEARNGDM.

В качестве функции оценки функционирования сети могут быть использованы любые дифференцируемые функции, например, MSE или MSEREG.

Пример:

Имеем набор входных значений P и соответствующих им выходных эталонных значений T. Задача - создать сеть для нахождения выходных значений по входным.

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4];
```

Создаем двухслойную однонаправленную сеть. Диапазон входных значений от 0 до 10, первый слой состоит из 5 нейронов с функциями активации TANSIG, второй слой содержит один нейрон с функцией активации PURELIN. Для обучения используем функцию TRAINLM:

```
net = newff([0 10],[5 1],{'tansig' 'purelin'});
```

Моделируем сеть. Подаем на вход сети входные значения P. Получаем выходные значения Y:

```
Y = sim(net,P);
```

Строим график, демонстрирующий отклонения выходных значений Y от целевых значений T для необученной сети:

```
plot(P,T,P,Y,'o')
```

Обучаем (тренируем) сеть. Количество эпох тренировки – 50:

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

Снова моделируем (теперь уже обученную сеть):

```
Y = sim(net,P);
```

Строим график для обученной сети:

```
plot(P,T,P,Y,'o')
```

Алгоритм:

Однонаправленные нейронные сети состоят из Nl слоев, использующих весовую функцию DOTPROD , входную функцию сети NETSUM и выбранную функцию активации. Первый слой получает веса, приходящие с входа. Каждый последующий слой получает веса от предыдущего слоя. Все слои обладают смещениями. Последний слой является выходом сети. Веса и смещения каждого слоя инициализируются функцией INITNW. Адаптация выполняется с помощью функции TRAINS, которая подстраивает веса в соответствии с выбранной обучающей функцией. Тренировка сети выполнена с помощью выбранной функции. Оценка функционирования сети выполняется с помощью выбранной функции оценки.

ПРИЛОЖЕНИЕ 2

Функция тренировки **TRAINCGF**

TRAINCGF - Тренировка по методу связанных градиентов

Синтаксис:

```
[net,tr,Ac,El] = traincgf(net,Pd,Tl,Ai,Q,TS,VV,TV)
info = traincgf(code)
```

Описание:
TRAINCGF - функция обучения сети, которая модифицирует веса и смещения в соответствии с методом обратного распространения на основе связанных градиентов Флетчера-Ривса (Fletcher-Reeves).

TRAINCGF(NET,Pd,Tl,Ai,Q,TS,VV,TV)
Входные параметры:
NET – нейронная сеть;
Pd - векторы входных задержек;
Tl - векторы эталонов слоя;
Ai - начальные условия входных задержек;
Q - размер пакета;
TS - временные шаги;
VV - пустая матрица или структура контрольных векторов;
TV - пустая матрица или структура тестовых векторов.
Выходные параметры:
NET – тренированная сеть;
TR - запись, включающая параметры тренировки;
TR.epoch – количество эпох тренировки;
TR.perf – параметр тренировки;
TR.vperf – параметр контрольной проверки;
TR.tperf – параметр тестирования;
Ac - суммарные выходы слоя для последней эпохи;
El - ошибки слоя для последней эпохи.

Тренировка реализуется в соответствии с параметрами функции обучения TRAINCGF. Эти параметры вместе с их значениями по умолчанию перечислены ниже:
net.trainParam.epochs – (100) - Максимальное количество эпох тренировки;
net.trainParam.show – (25) - Количество эпох между графиками;
net.trainParam.goal – (0) - Условие остановки по отклонению от эталона;
net.trainParam.time – (inf) - Максимальное время тренировки в секундах;
net.trainParam.min_grad – (1e-6) - Минимальный градиент;
net.trainParam.max_fail – (5) - Максимальное количество ошибок на контрольном массиве;
net.trainParam.searchFcn - ('srchcha') – Имя используемой процедуры линейного поиска.

Параметры, связанные с методами линейного поиска (для разных методов используется разное количество параметров):
net.trainParam.scal_tol – (20) - Деление на дельту для определения допуска при линейном поиске;
net.trainParam.alpha – (0.001) - Масштабный коэффициент, который определяет достаточное уменьшение функционирования;

net.trainParam.beta – (0.1) - Масштабный коэффициент, который определяет достаточно большое приращение;
net.trainParam.delta – (0.01) - Величина начального шага;
net.trainParam.gama – (0.1) - Параметр предназначенный для устранения малых уменьшений функционирования;
net.trainParam.low_lim – (0.1) - Нижний предел изменения размера шага;
net.trainParam.up_lim – (0.5) - Верхний предел изменения размера шага;
net.trainParam.maxstep – (100) - Максимальная длительность шага;
net.trainParam.minstep – (1.0e-6) - Минимальная длительность шага;
net.trainParam.bmax – (26) - Максимальный размер шага.

Размерности переменных:
Pd - NoxNixTS массив ячеек, каждый элемент которого P{i,j,ts} - матрица DijxQ,
Tl - NlxTS массив ячеек, каждый элемент которого P{i,ts} - матрица VixQ,
Ai - NlxLD массив ячеек, каждый элемент которого Ai{i,k} - матрица SixQ,

где
Ni = net.numInputs,
Nl = net.numLayers,
LD = net.numLayerDelays,
Ri = net.inputs{i}.size,
Si = net.layers{i}.size,
Vi = net.targets{i}.size,
Dij = Ri * length(net.inputWeights{i,j}.delays).

Если VV не [], то представляет собой структуру контрольных векторов:
VV.PD – Задержки контрольных входов;
VV.Tl – Контрольные эталоны слоя;
VV.Ai – Входные начальные условия контрольного массива;
VV.Q - Размер пакета контрольных массивов;
VV.TS – Временные шаги контрольного массива.

Контрольные векторы используются для досрочной остановки тренировки, если показатель функционирования сети на контрольном массиве векторов перестанет улучшаться или будет оставаться на одном уровне для MAX_FAIL эпох подряд.

Если TV не [],то представляет собой структуру тестовых векторов:
TV.PD - Задержки тестовых входов;
TV.Tl - Тестовые эталоны слоя;
TV.Ai - Входные начальные условия тестового массива;
TV.Q - Размер пакета тестовых массивов;
TV.TS - Временные шаги тестового массива.

Эти векторы используются для проверки обобщающей способности тренированной сети.

TRAINCGF(CODE) возвращает полезную информацию для каждой из строк CODE:
'pnames' - Имена параметров тренировки;
'pdefaults' – Параметры тренировки по умолчанию.

Использование.
Стандартную сеть, которая использует TRAINCGF, можно создать с помощью функций NEWFF, NEWCF или NEWELM.
Для того чтобы создать пользовательскую сеть, которую необходимо тренировать с помощью TRAINCGF:

1) Установить NET.trainFcn как 'traincgf'. Тем самым NET.trainParam будет по умолчанию TRAINCGF.
2) Установить требуемые значения для свойств NET.trainParam.

Вызов TRAIN с готовой сетью обеспечит тренировку сети с помощью TRAINCGF.

Примеры.

Заданы входы P и эталоны T. Создать сеть для решения этой задачи.

```
P = [0 1 2 3 4 5];
T = [0 0 0 1 1 1];
```

Создадим двухслойную однонаправленную сеть. Диапазон входных значений [0,10]. Первый слой содержит два TANSIG нейрона, второй слой состоит из одного LOGSIG нейрона. Для тренировки используем функцию TRAINCGF.

```
% Создаем и проверяем сеть
net = newff([0 5],[2 1],{'tansig','logsig'},'traincgf');
a = sim(net,p)
% Тренируем и тестируем сеть
net.trainParam.epochs = 50;
net.trainParam.show = 10;
net.trainParam.goal = 0.1;
net = train(net,p,t);
a = sim(net,p)
```

Алгоритм.
TRAINCGF может обучать любую сеть, если ее весовые, входные и активационные функции дифференцируемы.

Для вычисления производных эффективности функционирования PERF относительно весовых переменных X используется алгоритм обратного распространения. Каждая переменная модифицируется согласно уравнению:

$$X = X + a*dX;$$

где dX - направление поиска. Параметр a выбирается таким образом, чтобы минимизировать эффективность функционирования вдоль направления поиска. Функция линейного поиска searchFcn используется, чтобы локализовать точку минимума. Первое направление поиска – обратный градиент функционирования. По окончании итерации направление поиска вычисляется в соответствии со следующей формулой:

$$dX = -gX + dX\_old*Z;$$

где gX - градиент.

Параметр Z может быть вычислен несколькими путями. Для метода Флетчера-Ривса Z вычисляется по формуле:

$$Z=normnew\_sqr/norm\_sqr,$$

где norm_sqr - нормированный квадрат предыдущего градиента, normnew_sqr – нормированный квадрат текущего градиента.
Более детально с алгоритмом можно ознакомиться в "Scales, Introduction to Non-Linear Optimization 1985, p. 78".

Тренировка останавливается, если выполняется одно из следующих условий:
1) Достигнуто максимальное значение количества эпох “EPOCHS".
2) Превышено значение максимального времени тренировки “TIME".
3) Эффективность функционирования достигнет значения “GOAL".
4) Градиент эффективности функционирования уменьшится ниже “MINGRAD".
5) Контрольное значение функционирования увеличилось более чем “MAX_FAIL” раз подряд после того, как оно в последний раз уменьшилось (при использовании контрольного массива).

ПРИЛОЖЕНИЕ 3 Методы приведения к четкости

1. Центроидный.

Для непрерывного варианта:

$$z_0 = \frac{\int\limits_{\Omega} zC(z)\,dz}{\int\limits_{\Omega} C(z)\,dz};$$

Для дискретного варианта:

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i}.$$

2. Первый максимум (First-of-Maxima). Четкая величина переменной вывода находится как наименьшее значение, при котором достигается максимум итогового нечеткого множества, т. е. (см рис.ниже)

$$z_0 = \min\left(z | \mathrm{C}(z) = \max_u \mathrm{C}(u)\right).$$

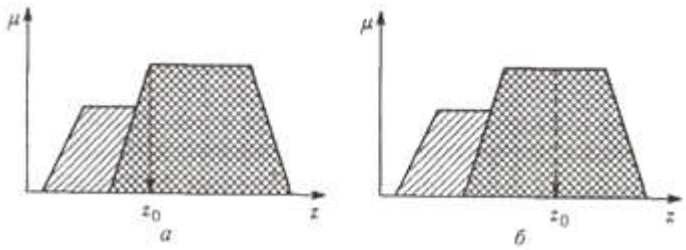


Иллюстрация к методам приведения к четкости: а — первый максимум; *б*— средний максимум

3. Средний максимум (Middle-of-Maxima). Четкое значения находится по формуле

$$z_0 = \frac{\int_{\mathrm{G}} z\, dz}{\int_{\mathrm{G}} dz},$$

$$z_0 = \frac{1}{N} \sum_{j=1}^{N} z_j.$$

4. Критерий максимума (Max-Criterion). Четкое значение выбирается произвольно среди множества элементов, доставляющих максимум C, т.е.

$$z_0 \in \left\{z | \mathrm{C}(z) = \max_u \mathrm{C}(u)\right\}.$$

5. Высотная дефазификация (Height defuzzification). Элементы области определения Ω,для которых значения функции принадлежности меньше, чем некоторый уровень a в расчет не принимаются, и четкое значение рассчитывается по формуле где $C\alpha$ - нечеткое множество α-уровня (см. выше).

Список литературы

1. Медведев В., Потемкин В. Нейронные сети. MATLAB 6. - М: Диалог-МИФИ, 2002 – 489 с.
2. Круглов В.В., Дли М.И, Р.Ю Голунов “Нечеткая логика и искусственные нейронные сети “- М: ФИЗМАТЛИТ 2001 – 224 с.
3. Каллан Р. Основные концепции нейронных сетей. – М.: Вильямс, 2001. – 288 с.
4. Хомич А.В., Степанян И.В., Карпишук А.В. Принцип блочности в эволюционной оптимизации структур нейронных сетей. - Нейрокомпьютеры: разработка и применение. - 2006. - №3. - С. 17-25.