Министерство образования Российской Федерации

Томский государственный университет систем
управления и радиоэлектроники

С. А. Добрынин А. Е. Катков

СИСТЕМЫ ИСКУСТВЕННОГО ИНТЕЛЛЕКТА

Лабораторный практикум

Томск 2005

Рецензенты:

Кафедра Автоматизированных систем управления (АСУ) Томского государственного университета систем управления и радиоэлектроники, канд. техн. наук, доц.

Павлов С. Н.

Приводится описание цикла лабораторных работ по дисциплине «системы искусственного интеллекта»: искусственные нейронные сети – построение многослойной сети, обучение с учителем, алгоритм обратного распространения ошибки.

Каждый раздел снабжен необходимой для понимания теорией, вариантами задач и тестовыми примерами.

Издание представляет интерес не только для студентов, но и для аспирантов, инженеров, научных сотрудников, занимающихся вопросами искусственного интеллекта и решением прикладных задач.

Содержание

Искусственные нейронные сети представляют собой попытку использования процессов, происходящих в нервных системах живых существ, для выработки новых технологических решений.

Станислав Осовский

Предисловие

Целью лабораторных работ является закрепление теоретических знаний и практических навыков работы на ЭВМ по решения задач искусственного интеллекта.

Предполагается, что студенты на практических занятиях уже усвоили базовые понятия систем искусственного интеллекта и имеют опыт в разработке компьютерных программ.

В учебном пособии представлен цикл из одной работы. В первой работе описывается базовые методы построения и обучения многослойной искусственной нейронной сети с сигмоидальной функцией активации и обучение ее с учителем по алгоритму обратного распространения ошибки. Главы 1, 1.1, 1.2, 1.3, 1.4, 1.5 и 1.6 описывают общую теорию НС [2]. В главах 1.8 и 1.9 описывается алгоритм обратного распространения ошибки и библиотека классов для работы с нейронными сетями [3-6].

На лабораторных занятиях студенты составляют алгоритмы, пишут программы и проводят расчеты на ЭВМ. В ходе численных расчетов студенты выполняют сравнительный анализ различных методов вычислений на тестовых задачах. По каждой лабораторной работе готовится отчет, в котором приводится теория, алгоритмы, результаты вычислений, тексты программ.

Лабораторная работа №1

Обучение с учителем по алгоритму обратного распространения ошибки

Цель работы: закрепление на практике и сравнительный анализ построения многослойной, полно-связной искусственной нейронной сети обучаемой с учителем по алгоритму обратного распространения ошибки.

1. Теоретические положения

Развитие искусственных нейронных сетей вдохновляется биологией. То есть, рассматривая сетевые конфигурации и алгоритмы, исследователи мыслят их в терминах организации мозговой деятельности. Но на этом аналогия может и закончиться. Наши знания о работе мозга столь ограничены, что мало бы нашлось руководящих ориентиров для тех, кто стал бы ему подражать. Поэтому разработчикам сетей приходится выходить за пределы современных биологических знаний в поисках структур, способных выполнять полезные функции. Во многих случаях это приводит к необходимости отказа от биологического правдоподобия, мозг становится просто метафорой, и создаются сети, невозможные в живой материи или требующие неправдоподобно больших допущений об анатомии и функционировании мозга.

Нервная система человека, построенная из элементов, называемых нейронами, имеет ошеломляющую сложность. Около 10^{11} нейронов участвуют в примерно 10^{15} передающих связях, имеющих длину метр и более. Каждый нейрон обладает многими качествами, общими с другими элементами тела, но его уникальной способностью является прием, обработка и передача электрохимических сигналов по нервным путям, которые образуют коммуникационную систему мозга.

На рис. 1.1 показана структура пары типичных биологических нейронов. Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых синапсами. Принятые синапсом входные сигналы подводятся к телу нейрона. Здесь они суммируются, причем одни входы стремятся возбудить нейрон, другие – воспрепятствовать его возбуждению. Когда суммарное возбуждение в теле нейрона превышает некоторый порог, нейрон возбуждается, посылая по аксону сигнал другим нейронам. У этой функциональной основной схемы много усложнений и исключений, тем не менее, большинство искусственных нейронных сетей моделируют лишь эти простые свойства.

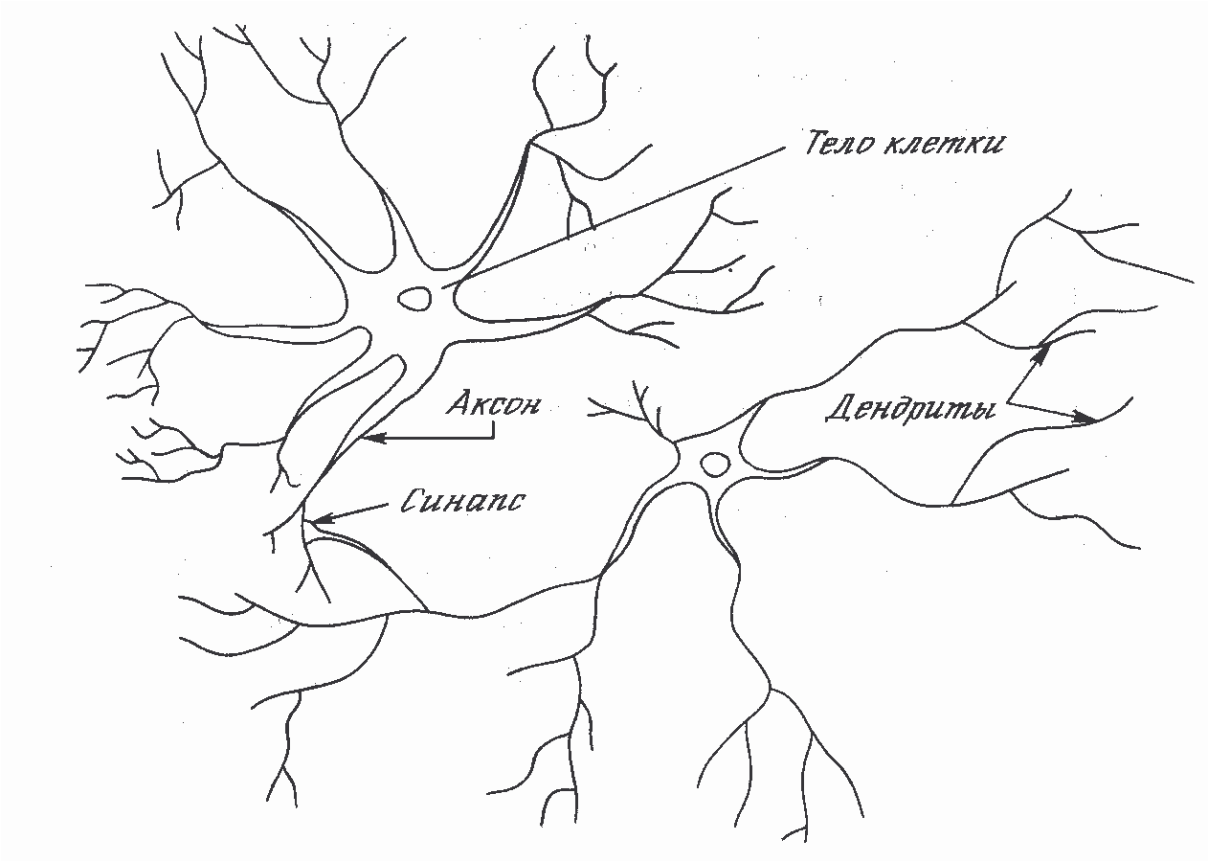


Рис. 1.1 - Биологический нейрон

1.1 Искусственный нейрон

Искусственный нейрон имитирует в первом приближении свойства биологического нейрона. На вход искусственного нейрона поступает некоторое множество сигналов, каждый из которых является выходом другого нейрона. Каждый вход умножается на соответствующий вес, аналогичный синоптической силе, и все произведения суммируются, определяя уровень активации нейрона. На рис. 1.2 представлена модель, реализующая эту идею. Хотя сетевые парадигмы весьма разнообразны, в основе почти всех их лежит эта конфигурация. Здесь множество входных сигналов, обозначенных x_1, x_2,…, x_n, поступает на искусственный нейрон. Эти входные сигналы, в совокупности, обозначаемые вектором **X**, соответствуют сигналам, приходящим в синапсы биологического нейрона. Каждый сигнал умножается на соответствующий вес w_1, w_2,…, w_n, и поступает на суммирующий блок, обозначенный Σ. Каждый вес соответствует «силе» одной биологической синоптической связи. (Множество весов в совокупности обозначается вектором **W**.) Суммирующий блок, соответствующий телу биологического элемента, складывает взвешенные входы алгебраически, создавая выход, который мы будем называть NET. В векторных обозначениях это может быть компактно записано следующим образом:

$$\text{NET} = \mathbf{XW}.$$

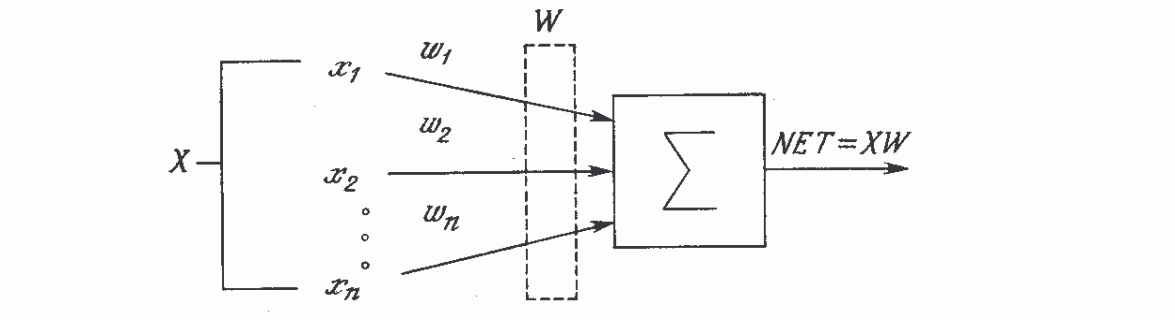


Рис. 1.2 - Искусственный нейрон

1.2. Активационные функции

Сигнал NET далее, как правило, преобразуется активационной функцией *F* и дает выходной нейронный сигнал OUT. Активационная функция может быть обычной линейной функцией

OUT = *K*(NET),

где *K* – постоянная, пороговой функции

OUT = 1, если NET > T,

OUT = 0 в остальных случаях,

где *T* – некоторая постоянная пороговая величина, или же функцией, более точно моделирующей нелинейную передаточную характеристику биологического нейрона и представляющей нейронной сети большие возможности.

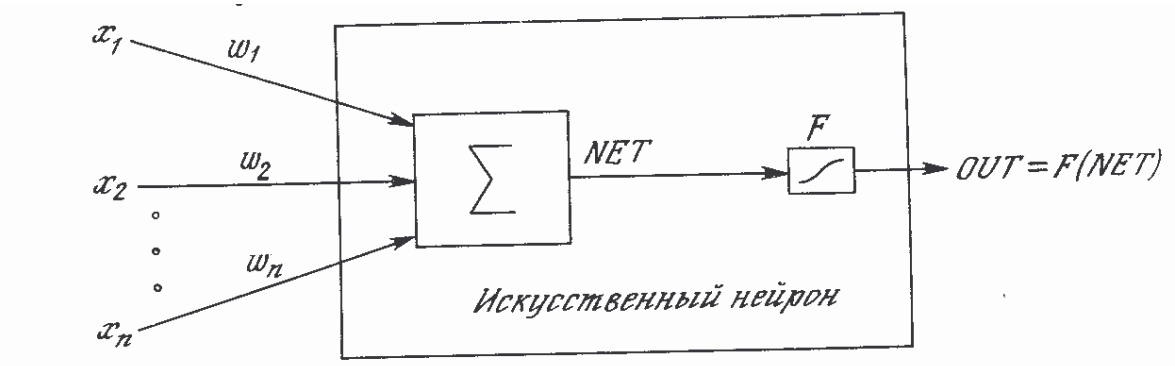


Рис. 1.3 - Искусственный нейрон с активационной функцией

На рис. 1.3 блок, обозначенный *F,* принимает сигнал NET и выдает сигнал OUT. Если блок *F* сужает диапазон изменения величины NET так, что при любых значениях NET значения OUT принадлежат некоторому конечному интервалу, то *F* называется *«сжимающей» функцией.* В качестве «сжимающей» функции часто используется логистическая или «сигмоидальная» (S-образная) функция, показанная на рис. 1.4а. Эта функция математически выражается как $F(x) = 1/(1 + e^{-x})$. Таким образом,

$$\mathrm{OUT} = \frac{1}{1 + e^{-\mathrm{NET}}}.$$

По аналогии с электронными системами активационную функцию можно считать нелинейной усилительной характеристикой искусственного нейрона. Коэффициент усиления вычисляется как отношение приращения величины OUT к вызвавшему его небольшому приращению величины NET.

Он выражается наклоном кривой при определенном уровне возбуждения и изменяется от малых значений при больших отрицательных возбуждениях (кривая почти горизонтальна) до максимального значения при нулевом возбуждении и снова уменьшается, когда возбуждение становится большим положительным. Гроссберг (1973) обнаружил, что подобная нелинейная характеристика решает поставленную им дилемму шумового насыщения. Каким образом одна и та же сеть может обрабатывать как слабые, так и сильные сигналы? Слабые сигналы нуждаются в большом сетевом усилении, чтобы дать пригодный к использованию выходной сигнал. Однако усилительные каскады с большими коэффициентами усиления могут привести к насыщению выхода шумами усилителей (случайными флуктуациями), которые присутствуют в любой физически реализованной сети. Сильные входные сигналы в свою очередь также будут приводить к насыщению усилительных каскадов, исключая возможность полезного использования выхода. Центральная область логистической функции, имеющая большой коэффициент усиления, решает проблему обработки слабых сигналов, в то время как области с падающим усилением на положительном и отрицательном концах подходят для больших возбуждений. Таким образом, нейрон функционирует с большим усилением в широком диапазоне уровня входного сигнала.

$$\mathrm{OUT} = \frac{1}{1 + \mathrm{e}^{-\mathrm{NET}}} = F(\mathrm{NET})$$

Рис. 1.4а - Сигмоидальная логистическая функция

Другой широко используемой активационной функцией является гиперболический тангенс. По форме она сходна с логистической функцией и часто используется биологами в качестве математической модели активации нервной клетки. В качестве активационной функции искусственной нейронной сети она записывается следующим образом:

OUT = th(x).

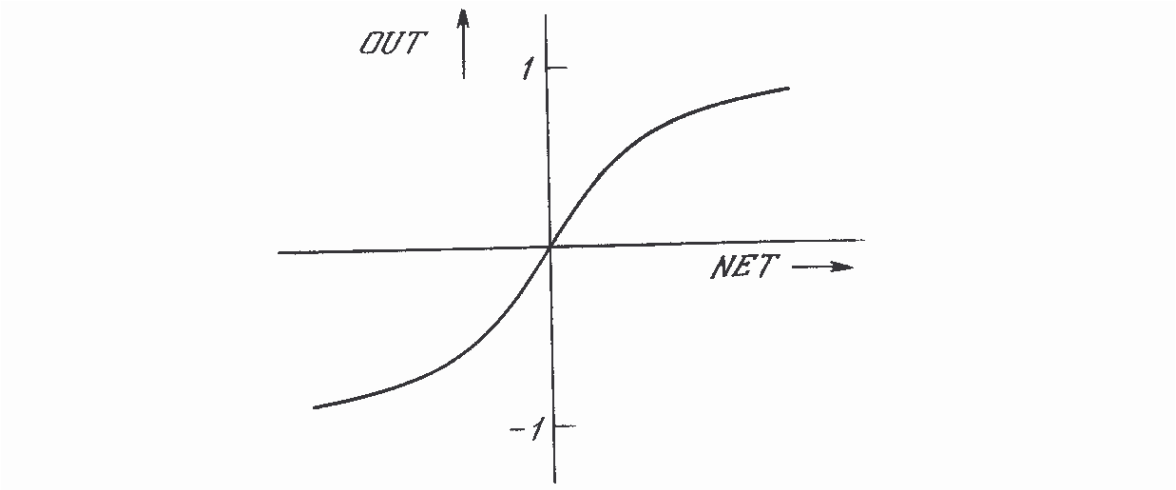


Рис. 1.4б - Функция гиперболического тангенса

Подобно логистической функции гиперболический тангенс является S-образной функцией, но он симметричен относительно начала координат, и в точке NET = 0 значение выходного сигнала OUT равно нулю (см. рис. 1.4б). В отличие от логистической функции гиперболический тангенс принимает значения различных знаков, что оказывается выгодным для ряда сетей.

Рассмотренная простая модель искусственного нейрона игнорирует многие свойства своего биологического двойника. Например, она не принимает во внимание задержки во времени, которые воздействуют на динамику системы. Входные сигналы сразу же порождают выходной сигнал. И, что более важно, она не учитывает воздействий функции частотной модуляции или синхронизирующей функции биологического нейрона, которые ряд исследователей считают решающими.

Несмотря на эти ограничения, сети, построенные из этих нейронов, обнаруживают свойства, сильно напоминающие биологическую систему. Только время и исследования смогут ответить на вопрос, являются ли подобные совпадения случайными или следствием того, что в модели верно схвачены важнейшие черты биологического нейрона.

1.3. Однослойные искусственные нейронные сети

Хотя один нейрон и способен выполнять простейшие процедуры распознавания, сила нейронных вычислений проистекает от соединений нейронов в сетях. Простейшая сеть состоит из группы нейронов, образующих слой, как показано в правой части рис. 1.5. Отметим, что вершины-круги слева служат лишь для распределения входных сигналов. Они не выполняют каких- либо вычислений, и поэтому не будут считаться слоем. По этой причине они обозначены кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами. Каждый элемент из множества входов **X** отдельным весом соединен с каждым искусственным нейроном. А каждый нейрон выдает взвешенную сумму входов в сеть. В искусственных и биологических сетях многие соединения могут отсутствовать, все соединения показаны в целях общности. Могут иметь место также соединения между выходами и входами элементов в слое.

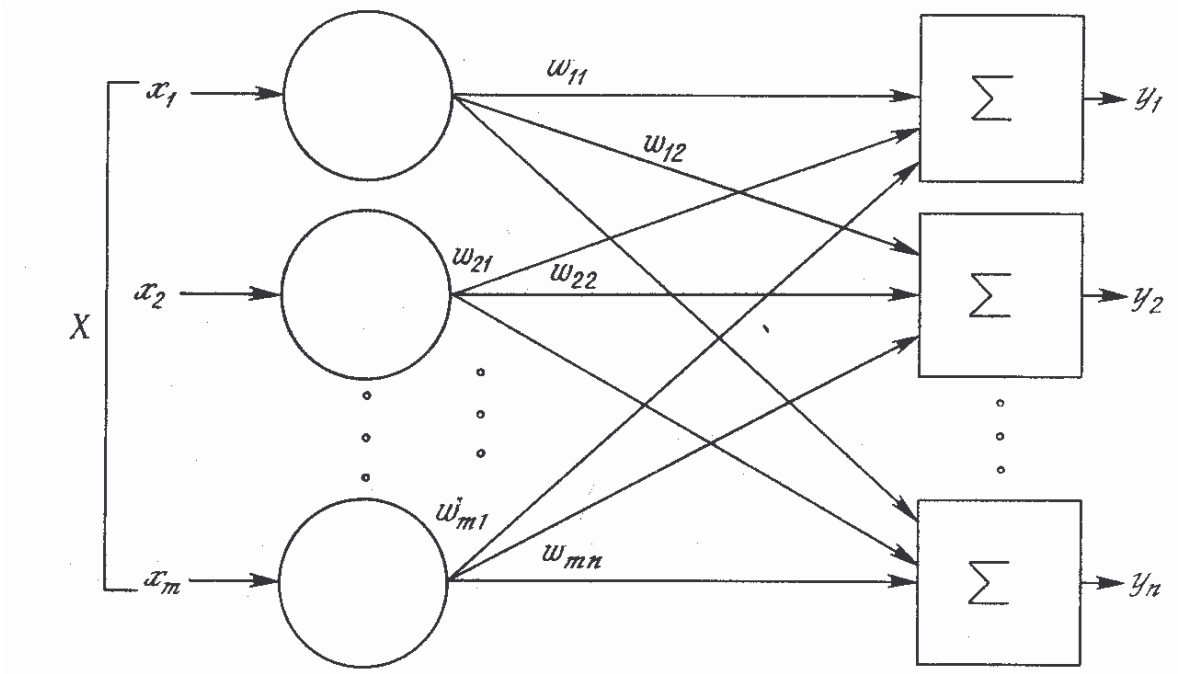


Рис. 1.5 - Однослойная нейронная сеть

Удобно считать веса элементами матрицы **W**. Матрица имеет *m* строк и *n* столбцов, где m – число входов, а n – число нейронов. Например, $w_{2,3}$ – это вес, связывающий третий вход со вторым нейроном. Таким образом, вычисление выходного вектора **N**, компонентами которого являются выходы OUT нейронов, сводится к матричному умножению **N = XW,** где **N** и **X** – векторы-строки.

1.4. Многослойные искусственные нейронные сети

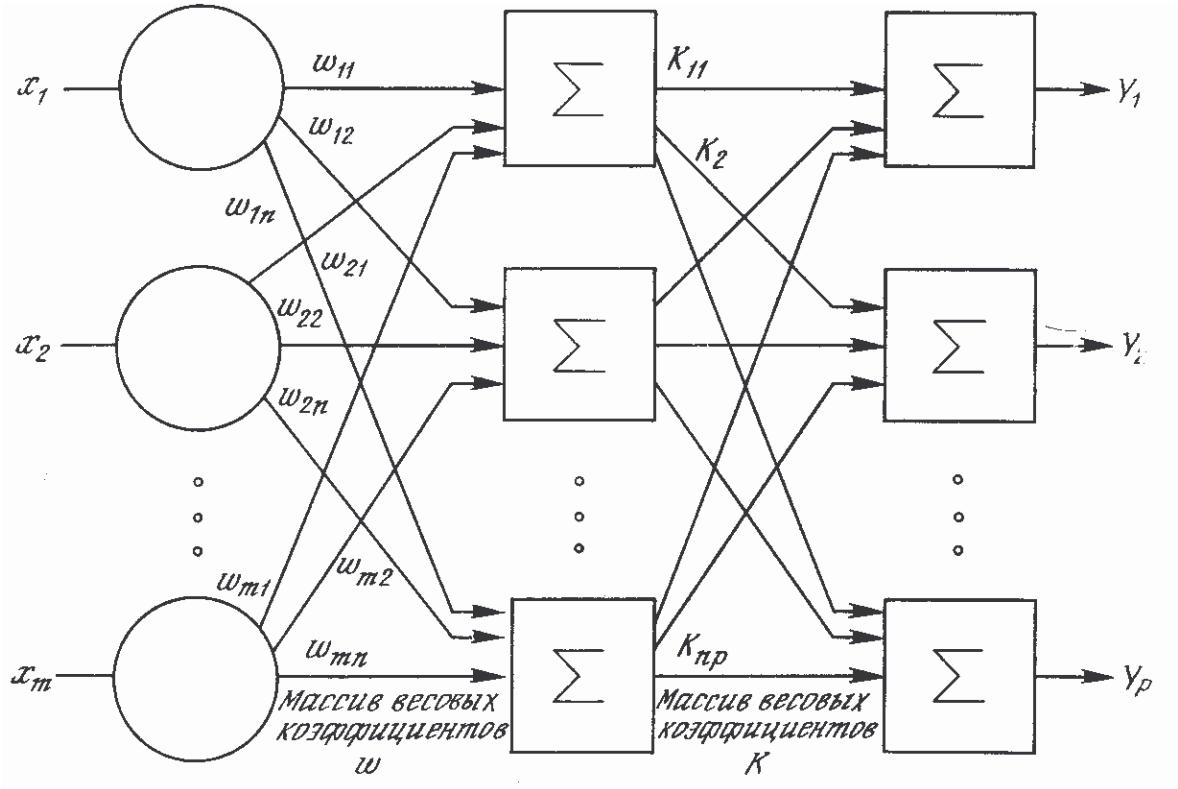


Рис. 1.6 - Двухслойная нейронная сеть

Более крупные и сложные нейронные сети обладают, как правило, и большими вычислительными возможностями. Хотя созданы сети всех конфигураций, какие только можно себе представить, послойная организация нейронов копирует слоистые структуры определенных отделов мозга. Оказалось, что такие многослойные сети обладают большими возможностями, чем однослойные, и в последние годы были разработаны алгоритмы для их обучения.

Многослойные сети могут образовываться каскадами слоев. Выход одного слоя является входом для последующего слоя. Подобная сеть показана на рис. 1.6 и снова изображена со всеми соединениями.

1.5. Нелинейная активационная функция

Многослойные сети не могут привести к увеличению вычислительной мощности по сравнению с однослойной сетью лишь в том случае, если активационная функция между слоями будет нелинейной. Вычисление выхода слоя заключается в умножении входного вектора на первую весовую матрицу с последующим умножением (если отсутствует нелинейная активационная функция) результирующего вектора на вторую весовую матрицу.

$$(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2$$

Так как умножение матриц ассоциативно, то

$$\mathbf{X}(\mathbf{W}_1\mathbf{W}_2).$$

Это показывает, что двухслойная линейная сеть эквивалентна одному слою с весовой матрицей, равной произведению двух весовых матриц. Следовательно, любая многослойная линейная сеть может быть заменена эквивалентной однослойной сетью. В гл. 2 показано, что однослойные сети весьма ограниченны по своим вычислительным возможностям. Таким образом, для расширения возможностей сетей по сравнению с однослойной сетью необходима нелинейная активационная функция.

1.6. Обучение сети

Сеть обучается, чтобы для некоторого множества входов давать желаемое (или, по крайней мере, сообразное с ним) множество выходов. Каждое такое входное (или выходное) множество рассматривается как вектор. Обучение осуществляется путем последовательного предъявления входных векторов с одновременной подстройкой весов в соответствии с определенной процедурой. В процессе обучения веса сети постепенно становятся такими, чтобы каждый входной вектор вырабатывал выходной вектор.

Различают алгоритмы обучения с учителем и без учителя. Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором числе таких обучающих пар. Предъявляется выходной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором, разность

(ошибка) с помощью обратной связи подается в сеть, и веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

Несмотря на многочисленные прикладные достижения, обучение с учителем критиковалось за свою биологическую неправдоподобность. Трудно вообразить обучающий механизм в мозге, который бы сравнивал желаемые и действительные значения выходов, выполняя коррекцию с помощью обратной связи. Если допустить подобный механизм в мозге, то откуда тогда возникают желаемые выходы? Обучение без учителя является намного более правдоподобной моделью обучения в биологической системе. Развитая Кохоненом и многими другими, она не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, т. е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процесс обучения, следовательно, выделяет статистические свойства обучающего множества и группирует сходные векторы в классы. Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения невозможно предсказать, какой выход будет производиться данным классом входных векторов. Следовательно, выходы подобной сети должны трансформироваться в некоторую понятную форму, обусловленную процессом обучения. Это не является серьезной проблемой. Обычно не сложно идентифицировать связь между входом и выходом, установленную сетью.

1.7. Распознавания образов с использованием НС

НС используются для решается большого круга задач. Области применения НС:

- обработка и анализ изображений;
- распознавание и классификация образов;
- сжатие данных;
- интерполирование функций многих переменных;
- распознавание речи независимо от диктора, перевод;
- обработка высокоскоростных цифровых потоков;
- автоматизированная система быстрого поиска информации (организация ассоциативной памяти);
- планирование применения сил и средств в больших масштабах;
- решение трудоемких задач оптимизации;
- адаптивное управление и предсказание.

Рассмотрим пример распознавания образов с использованием НС. В качестве математического представления образа будем понимать матрицу

чисел, в которой элементы могут принимать только два значения: -0.5 и 0.5. Где -0.5 будет условно соответствовать черному пикселю изображения, а 0.5 – белому. Например, для образов представляющих собой буквы “А”, “Б” и “В” матрица и графическое черно-белое изображение приведены в таблице 1.

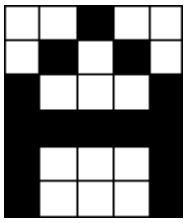
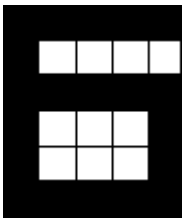
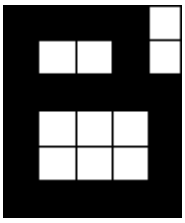
Название образа	“А”	“Б”	“В”
Черно-белое изображение			
Матрица 5x6	–0.5 –0.5 0.5 –0.5 –0.5 –0.5 0.5 –0.5 0.5 –0.5 0.5 –0.5 –0.5 –0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 –0.5 –0.5 –0.5 0.5 0.5 –0.5 –0.5 –0.5 0.5	0.5 0.5 0.5 0.5 0.5 0.5 –0.5 –0.5 –0.5 –0.5 0.5 0.5 0.5 0.5 0.5 0.5 –0.5 –0.5 –0.5 0.5 0.5 –0.5 –0.5 –0.5 0.5 0.5 0.5 0.5 0.5 0.5	0.5 0.5 0.5 0.5 –0.5 0.5 –0.5 –0.5 0.5 –0.5 0.5 0.5 0.5 0.5 0.5 0.5 –0.5 –0.5 –0.5 0.5 0.5 –0.5 –0.5 –0.5 0.5 0.5 0.5 0.5 0.5 0.5
Матрица 5x6 в которой для наглядности -0.5 заменено на “.” а 0.5 на “X”.	..X.. .X.X. X...X XXXXX X...X X...X	XXXXX X.... XXXXX X...X X...X XXXXX	XXXX. X..X. XXXXX X...X X...X XXXXX

Таблица 1 – Представление образов матрицей коэффициентов

Входными данными НС является вектор, поэтому наши исходные данные необходимо преобразовать к векторному виду. Матрицу nxm мы можем представить в виде вектора длиной n*m, путем выписывания построчно коэффициентов матрицы в вектор. Для матрицы 5x6 соответствующий вектор будет содержать 30 элементов.

Матрица	..X.. .X.X. X...X XXXXX X...X X...X
вектор	..X.. .X.X. X...X XXXXX X...X X...X

Таблица 2 – Представление матрицы в виде вектора

Выходными данными НС являются тоже вектор. Этот вектор мы должны интерпретировать в терминах решаемой задачи. Зададим размерность выходного вектора сети равным количеству распознаваемых образов. Для нашей задачи размерность равна – 3. Пусть образам “А”, “Б” и “В” соответствуют классы 1, 2 и 3. Тогда выходной вектор, соответствующий

классу n будет иметь значение 0.5 в n-ой позиции и -0.5 в остальных. Пример приведен в таблице 3.

Название класса	“А”	“Б”	“В”
Номер класса	1	2	3
Выходной вектор	X..	.X.	..X

Таблица 3 – соответствие выходных векторов классам.

Схема использования ИНС представлена на рис 1.7 На вход ИНС подается символ, который нужно распознать. ИНС производит подсчеты своих коэффициентов и выдает результат. Распознавать ИНС может только те символы, которым ее обучали или близкие (искаженные или зашумленные).

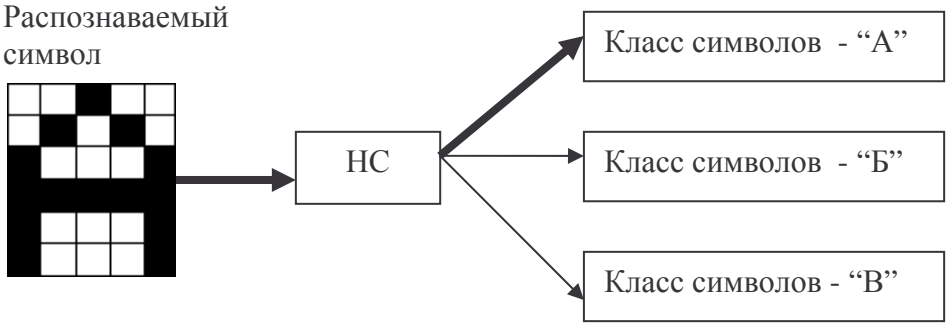


Рис. 1.7 – Схема использования НС

1.8. Алгоритм обратного распространения

Среди различных структур нейронных сетей (НС) одной из наиболее известных является многослойная структура, в которой каждый нейрон произвольного слоя связан со всеми аксонами нейронов предыдущего слоя или, в случае первого слоя, со всеми входами НС. Такие НС называются полносвязными. Когда в сети только один слой, алгоритм ее обучения с учителем довольно очевиден, так как правильные выходные состояния нейронов единственного слоя заведомо известны, и подстройка синаптических связей идет в направлении, минимизирующем ошибку на выходе сети. По этому принципу строится, например, алгоритм обучения однослойного персептрона. В многослойных же сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, не известны, и двух или более слойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходах НС. Один из вариантов решения этой проблемы – разработка наборов выходных сигналов, соответствующих входным, для каждого слоя НС, что, конечно, является очень трудоемкой операцией и не всегда осуществимо. Второй вариант – динамическая подстройка весовых коэффициентов синапсов, в ходе которой выбираются, как правило, наиболее слабые связи и изменяются на малую величину в ту или иную сторону, а сохраняются только те изменения, которые повлекли уменьшение ошибки на выходе всей сети. Очевидно, что

данный метод "тыка", несмотря на свою кажущуюся простоту, требует громоздких рутинных вычислений. И, наконец, третий, более приемлемый вариант – распространение сигналов ошибки от выходов НС к ее входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Этот алгоритм обучения НС получил название процедуры обратного распространения. Именно он будет рассмотрен в дальнейшем.

Согласно методу наименьших квадратов, минимизируемой целевой функцией ошибки НС является величина:

$$E(w) = \frac{1}{2}\sum_{j,p}(y_{j,p}^{(N)} - d_{j,p})^2 \quad (1)$$

где $y_{j,p}^{(N)}$– реальное выходное состояние нейрона j выходного слоя N нейронной сети при подаче на ее входы p-го образа; d_{jp} – идеальное (желаемое) выходное состояние этого нейрона.

Суммирование ведется по всем нейронам выходного слоя и по всем обрабатываемым сетью образам. Минимизация ведется методом градиентного спуска, что означает подстройку весовых коэффициентов следующим образом:

$$\Delta w_{ij}^{(n)} = -\eta \cdot \frac{\partial E}{\partial w_{ij}} \quad (2)$$

Здесь w_{ij} – весовой коэффициент синаптической связи, соединяющей i-ый нейрон слоя n-1 с j-ым нейроном слоя n, η – коэффициент скорости обучения, $0<\eta<1$.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j} \cdot \frac{\partial s_j}{\partial w_{ij}} \quad (3)$$

Здесь под y_j, как и раньше, подразумевается выход нейрона j, а под s_j – взвешенная сумма его входных сигналов, то есть аргумент активационной функции. Так как множитель dy_j/ds_j является производной этой функции по ее аргументу, из этого следует, что производная активационной функция должна быть определена на всей оси абсцисс. В связи с этим функция единичного скачка и прочие активационные функции с неоднородностями не подходят для рассматриваемых НС. В них применяются такие гладкие функции, как гиперболический тангенс или классический сигмоид с экспонентой. В случае гиперболического тангенса

$$\frac{dy}{ds} = 1 - s^2 \quad (4)$$

Третий множитель $\partial s_j/\partial w_{ij}$, очевидно, равен выходу нейрона предыдущего слоя $y_i^{(n-1)}$.

Что касается первого множителя в (3), он легко раскладывается следующим образом[2]:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot \frac{\partial s_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot w_{jk}^{(n+1)} \quad (5)$$

Здесь суммирование по k выполняется среди нейронов слоя n+1.

Введя новую переменную

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j} \quad (6)$$

мы получим рекурсивную формулу для расчетов величин $d_j^{(n)}$ слоя n из величин $d_k^{(n+1)}$ более старшего слоя n+1.

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \right] \cdot \frac{dy_j}{ds_j} \quad (7)$$

Для выходного же слоя

$$\delta_l^{(N)} = (y_l^{(N)} - d_l) \cdot \frac{dy_l}{ds_l} \quad (8)$$

Теперь мы можем записать (2) в раскрытом виде:

$$\Delta w_{ij}^{(n)} = -\eta \cdot \delta_j^{(n)} \cdot y_i^{(n-1)} \quad (9)$$

Иногда для придания процессу коррекции весов некоторой инерционности, сглаживающей резкие скачки при перемещении по поверхности целевой функции, (9) дополняется значением изменения веса на предыдущей итерации

$$\Delta w_{ij}^{(n)}(t) = -\eta \cdot (\mu \cdot \Delta w_{ij}^{(n)}(t-1) + (1-\mu) \cdot \delta_j^{(n)} \cdot y_i^{(n-1)}) \quad (10)$$

где μ – коэффициент инерционности, t – номер текущей итерации.

Таким образом, полный алгоритм обучения НС с помощью процедуры обратного распространения строится так:

1. Подать на входы сети один из возможных образов и в режиме обычного функционирования НС, когда сигналы распространяются от входов к выходам, рассчитать значения последних. Напомним, что

$$s_j^{(n)} = \sum_{i=0}^{M} y_i^{(n-1)} \cdot w_{ij}^{(n)} \quad (11)$$

где M – число нейронов в слое n-1 с учетом нейрона с постоянным выходным состоянием +1, задающего смещение; $y_i^{(n-1)} = x_{ij}^{(n)}$ – i-ый вход нейрона j слоя n.

$y_j^{(n)} = f(s_j^{(n)})$, где $f()$ – сигмоид (12)

$y_q^{(0)} = I_q$, (13)

где I_q – q-ая компонента вектора входного образа.

2. Рассчитать $\delta^{(N)}$ для выходного слоя по формуле (8).

Рассчитать по формуле (9) или (10) изменения весов $\Delta \boldsymbol{w}^{(N)}$ слоя N.

3. Рассчитать по формулам (7) и (9) (или (7) и (10)) соответственно $\delta^{(n)}$ и $\Delta \boldsymbol{w}^{(n)}$ для всех остальных слоев, n=N-1,...1.

4. Скорректировать все веса в НС

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t) \quad (14)$$

5. Если ошибка сети существенна, перейти на шаг 1. В противном случае – конец.

Сети на шаге 1 попеременно в случайном порядке предъявляются все тренировочные образы, чтобы сеть, образно говоря, не забывала одни по мере запоминания других. Алгоритм иллюстрируется рисунком 1.

Из выражения (9) следует, что когда выходное значение $y_i^{(n-1)}$ стремится к нулю, эффективность обучения заметно снижается. При двоичных входных векторах в среднем половина весовых коэффициентов не будет корректироваться[3], поэтому область возможных значений выходов нейронов [0,1] желательно сдвинуть в пределы [-0.5,+0.5], что достигается простыми модификациями логистических функций. Например, сигмоид с экспонентой преобразуется к виду

$$f(x) = -0.5 + \frac{1}{1+e^{-\alpha \cdot x}} \quad (15)$$

1.9. Описание шаблона программы и форматов входных данных

Как видно из формул, описывающих алгоритм функционирования и обучения НС, весь этот процесс может быть записан и затем запрограммирован в терминах и с применением операций матричной алгебры. Судя по всему, такой подход обеспечит более быструю и компактную реализацию НС, нежели ее воплощение на базе концепций объектно-ориентированного (ОО) программирования. Однако в последнее время преобладает именно ОО подход, причем зачастую разрабатываются специальные ОО языки для программирования НС, хотя, с моей точки зрения, универсальные ОО языки, например C++ и Pascal, были созданы как раз для того, чтобы исключить необходимость разработки каких-либо других ОО языков, в какой бы области их не собирались применять.

И все же программирование НС с применением ОО подхода имеет свои плюсы. Во-первых, оно позволяет создать гибкую, легко перестраиваемую иерархию моделей НС. Во-вторых, такая реализация наиболее прозрачна для программиста, и позволяет конструировать НС даже непрограммистам. В-третьих, уровень абстрактности программирования, присущий ОО языкам, в будущем будет, по-видимому, расти, и реализация НС с ОО подходом позволит расширить их возможности. Исходя из вышеизложенных соображений, приведенная в листингах библиотека классов и программ,

реализующая полносвязные НС с обучением по алгоритму обратного распространения, использует ОО подход. Вот основные моменты, требующие пояснений.

Прежде всего необходимо отметить, что библиотека была составлена и использовалась в целях распознавания изображений, однако применима и в других приложениях. В файле neuro.h в приведены описания двух базовых и пяти производных (рабочих) классов: Neuron, SomeNet и NeuronFF, NeuronBP, LayerFF, LayerBP, NetBP, а также описания нескольких общих функций вспомогательного назначения, содержащихся в файле subfun.cpp. Методы пяти вышеупомянутых рабочих классов внесены в файлы neuro_ff.cpp и neuro_bp.cpp, представленные в приложении. Такое, на первый взгляд искусственное, разбиение объясняется тем, что классы с суффиксом _ff, описывающие прямопоточные нейронные сети (feedforward), входят в состав не только сетей с обратным распространением – _bp (backpropagation), но и других, например таких, как с обучением без учителя, которые будут рассмотрены в дальнейшем.

В ущерб принципам ОО программирования, шесть основных параметров, характеризующих работу сети, вынесены на глобальный уровень, что облегчает операции с ними. Параметр SigmoidType определяет вид активационной функции. В методе NeuronFF::Sigmoid перечислены некоторые его значения, макроопределения которых сделаны в заголовочном файле. Пункты HARDLIMIT и THRESHOLD даны для общности, но не могут быть использованы в алгоритме обратного распространения, так как соответствующие им активационные функции имеют производные с особыми точками. Это отражено в методе расчета производной NeuronFF::D_Sigmoid, из которого эти два случая исключены. Переменная SigmoidAlfa задает крутизну α сигмоида ORIGINAL из (15). MiuParm и NiuParm – соответственно значения параметров μ и η из формулы (10). Величина Limit используется в методах IsConverged для определения момента, когда сеть обучится или попадет в паралич. В этих случаях изменения весов становятся меньше малой величины Limit. Параметр dSigma эмулирует плотность шума, добавляемого к образам во время обучения НС. Это позволяет из конечного набора "чистых" входных образов генерировать практически неограниченное число "зашумленных" образов. Дело в том, что для нахождения оптимальных значений весовых коэффициентов число степеней свободы НС – N_w должно быть намного меньше числа накладываемых ограничений – $N_y \cdot N_p$, где N_p – число образов, предъявляемых НС во время обучения. Фактически, параметр dSigma равен числу входов, которые будут инвертированы в случае двоичного образа. Если dSigma = 0, помеха не вводится.

Методы Randomize позволяют перед началом обучения установить весовые коэффициенты в случайные значения в диапазоне [-range,+range]. Методы Propagate выполняют вычисления по формулам (11) и (12). Метод NetBP::CalculateError на основе передаваемого в качестве аргумента массива верных (желаемых) выходных значений НС вычисляет величины δ. Метод

NetBP::Learn рассчитывает изменения весов по формуле (10), методы Update обновляют весовые коэффициенты. Метод NetBP::Cycle объединяет в себе все процедуры одного цикла обучения, включая установку входных сигналов NetBP::SetNetInputs. Различные методы PrintXXX и LayerBP::Show позволяют контролировать течение процессов в НС, но их реализация не имеет принципиального значения, и простые процедуры из приведенной библиотеки могут быть при желании переписаны, например, для графического режима. Это оправдано и тем, что в алфавитно-цифровом режиме уместить на экране информацию о сравнительно большой НС уже не удается.

Сети могут конструироваться посредством NetBP(unsigned), после чего их нужно заполнять сконструированными ранее слоями с помощью метода NetBP::SetLayer, либо посредством NetBP(unsigned, unsigned,...). В последнем случае конструкторы слоев вызываются автоматически. Для установления синаптических связей между слоями вызывается метод NetBP::FullConnect.

После того как сеть обучится, ее текущее состояние можно записать в файл (метод NetBP::SaveToFile), а затем восстановить с помощью метода NetBP::LoadFromFile, который применим лишь к только что сконструированной по NetBP(void) сети.

Для ввода в сеть входных образов, а на стадии обучения – и для задания выходных, написаны три метода: SomeNet::OpenPatternFile, SomeNet::ClosePatternFile и NetBP::LoadNextPattern. Если у файлов образов произвольное расширение, то входные и выходные вектора записываются чередуясь: строка с входным вектором, строка с соответствующим ему выходным вектором и т.д. Каждый вектор есть последовательность действительных чисел в диапазоне [-0.5,+0.5], разделенных произвольным числом пробелов. Если файл имеет расширение IMG, входной вектор представляется в виде матрицы символов размером dy*dx (величины dx и dy должны быть заблаговременно установлены с помощью LayerFF::SetShowDim для нулевого слоя), причем символ 'x' соответствует уровню 0.5, а точка – уровню -0.5, то есть файлы IMG, по крайней мере – в приведенной версии библиотеки, бинарны. Когда сеть работает в нормальном режиме, а не обучается, строки с выходными векторами могут быть пустыми. Метод SomeNet::SetLearnCycle задает число проходов по файлу образов, что в сочетании с добавлением шума позволяет получить набор из нескольких десятков и даже сотен тысяч различных образов.

Теперь коснемся вопроса емкости НС, то есть числа образов, предъявляемых на ее входы, которые она способна научиться распознавать. Для сетей с числом слоев больше двух, он остается открытым. Как показано в [4], для НС с двумя слоями, то есть выходным и одним скрытым слоем, детерминистская емкость сети C_d оценивается так:

$$N_w/N_y<C_d<N_w/N_y\cdot\log(N_w/N_y) \quad (16)$$

где N_w – число подстраиваемых весов, N_y – число нейронов в выходном слое.

Следует отметить, что данное выражение получено с учетом некоторых ограничений. Во-первых, число входов N_x и нейронов в скрытом слое N_h должно удовлетворять неравенству $N_x+N_h>N_y$. Во-вторых, $N_w/N_y>1000$. Однако вышеприведенная оценка выполнялась для сетей с активационными функциями нейронов в виде порога, а емкость сетей с гладкими активационными функциями, например – (15), обычно больше. Кроме того, фигурирующее в названии емкости прилагательное "детерминистский" означает, что полученная оценка емкости подходит абсолютно для всех возможных входных образов, которые могут быть представлены N_x входами. В действительности распределение входных образов, как правило, обладает некоторой регулярностью, что позволяет НС проводить обобщение и, таким образом, увеличивать реальную емкость. Так как распределение образов, в общем случае, заранее не известно, мы можем говорить о такой емкости только предположительно, но обычно она раза в два превышает емкость детерминистскую.

В продолжение разговора о емкости НС логично затронуть вопрос о требуемой мощности выходного слоя сети, выполняющего окончательную классификацию образов. Дело в том, что для разделения множества входных образов, например, по двум классам достаточно всего одного выхода. При этом каждый логический уровень – "1" и "0" – будет обозначать отдельный класс. На двух выходах можно закодировать уже 4 класса и так далее. Однако результаты работы сети, организованной таким образом, можно сказать – "под завязку", – не очень надежны. Для повышения достоверности классификации желательно ввести избыточность путем выделения каждому классу одного нейрона в выходном слое или, что еще лучше, нескольких, каждый из которых обучается определять принадлежность образа к классу со своей степенью достоверности, например: высокой, средней и низкой. Такие НС позволяют проводить классификацию входных образов, объединенных в нечеткие (размытые или пересекающиеся) множества. Это свойство приближает подобные НС к условиям реальной жизни.

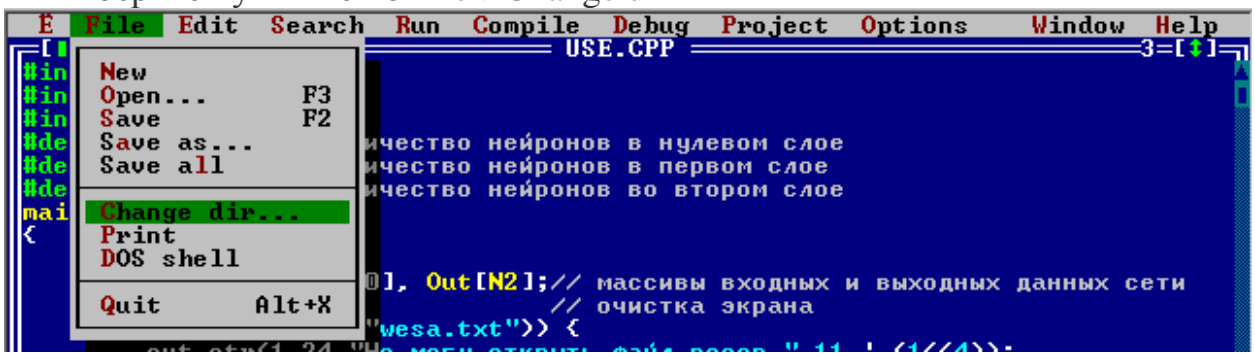
Рассматриваемая НС имеет несколько "узких мест". Во-первых, в процессе обучения может возникнуть ситуация, когда большие положительные или отрицательные значения весовых коэффициентов сместят рабочую точку на сигмоидах многих нейронов в область насыщения. Малые величины производной от логистической функции приведут в соответствие с (7) и (8) к остановке обучения, что парализует НС. Во-вторых, применение метода градиентного спуска не гарантирует, что будет найден глобальный, а не локальный минимум целевой функции. Эта проблема связана еще с одной, а именно – с выбором величины скорости обучения. Доказательство сходимости обучения в процессе обратного распространения основано на производных, то есть приращения весов и, следовательно, скорость обучения должны быть бесконечно малыми, однако в этом случае обучение будет происходить неприемлемо медленно. С другой стороны, слишком большие коррекции весов могут привести к постоянной неустойчивости процесса обучения. Поэтому в качестве η обычно

выбирается число меньше 1, но не очень маленькое, например, 0.1, и оно, вообще говоря, может постепенно уменьшаться в процессе обучения. Кроме того, для исключения случайных попаданий в локальные минимумы иногда, после того как значения весовых коэффициентов стабилизируются, η кратковременно сильно увеличивают, чтобы начать градиентный спуск из новой точки. Если повторение этой процедуры несколько раз приведет алгоритм в одно и то же состояние НС, можно более или менее уверенно сказать, что найден глобальный максимум, а не какой-то другой.

Существует и иной метод исключения локальных минимумов, а заодно и паралича НС, заключающийся в применении стохастических НС, но о них лучше поговорить отдельно.

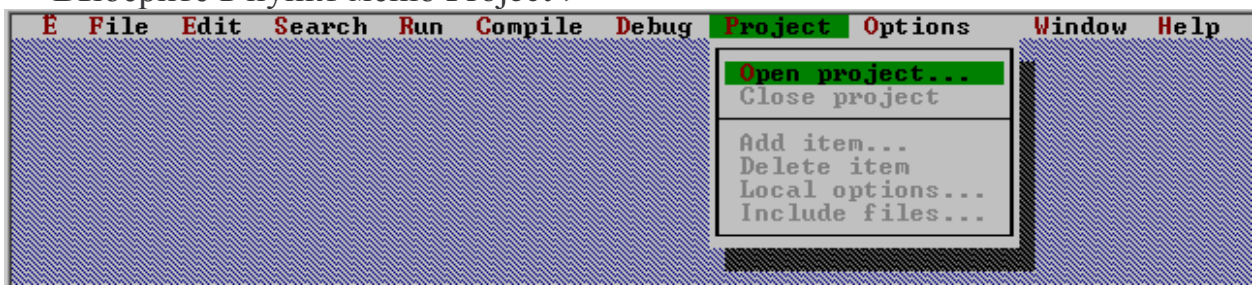
2. Задание на ЭВМ

2.1 Сделайте копию папки neuro и переименуйте ее например в “labrab”. Папку neuro оставьте не тронутой для образца, в случае затруднений с программой от туда всегда можно будет взять рабочий код. Откройте Borland C++3.1 и сделайте папку labrab рабочей директорией. Для этого выберите пункт меню File->Change dir

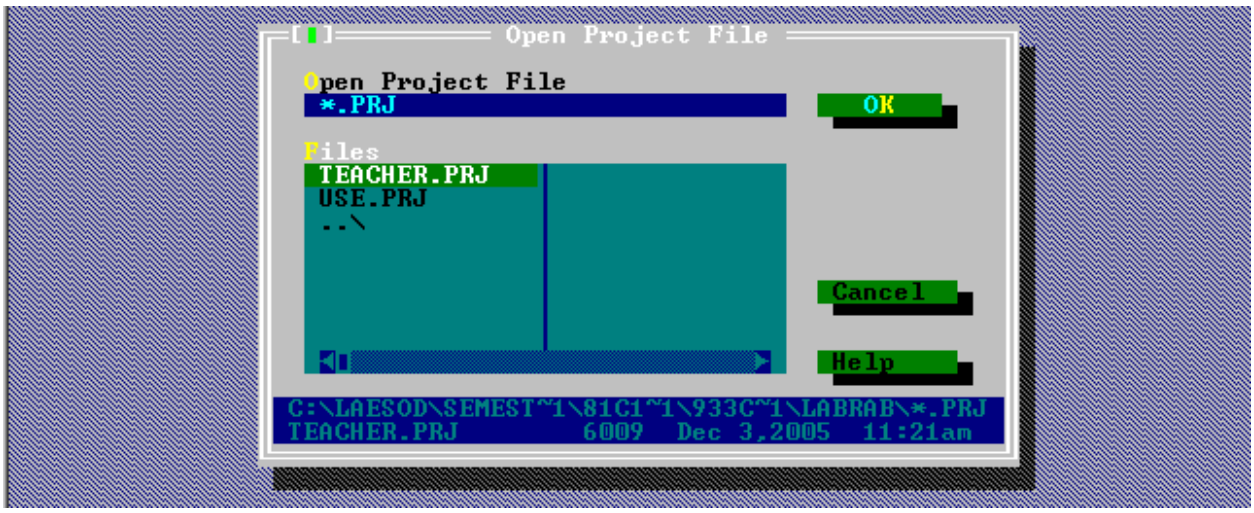


Выберите директорию labrab на вашем компьютере.

Выберите в пункт меню Project->

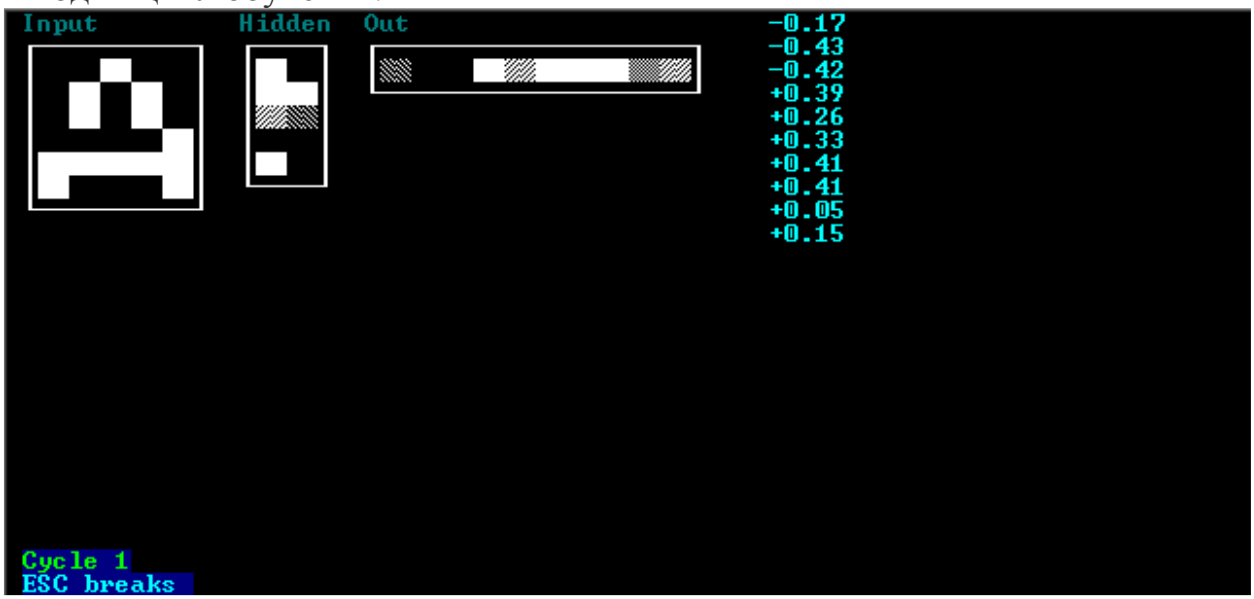


Откройте проект teacher.prj



Запустите программу – CTRL-F9

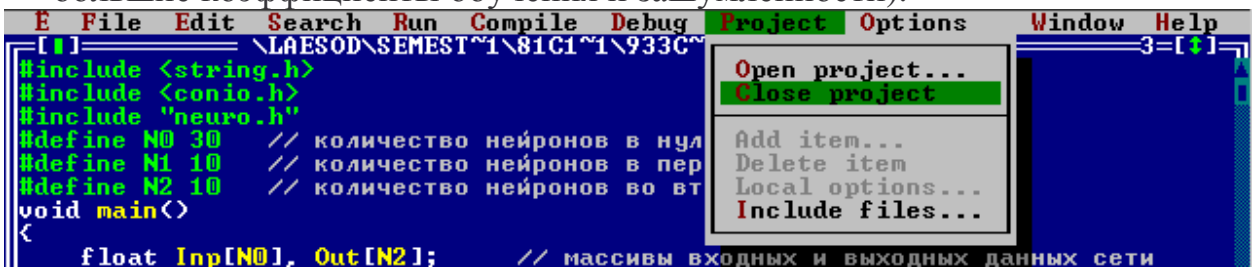
Программа ждет команды от пользователя. Если нажать ESC, то программа завершиться. Если нажать ENTER, то программа выполнит один цикл обучения.



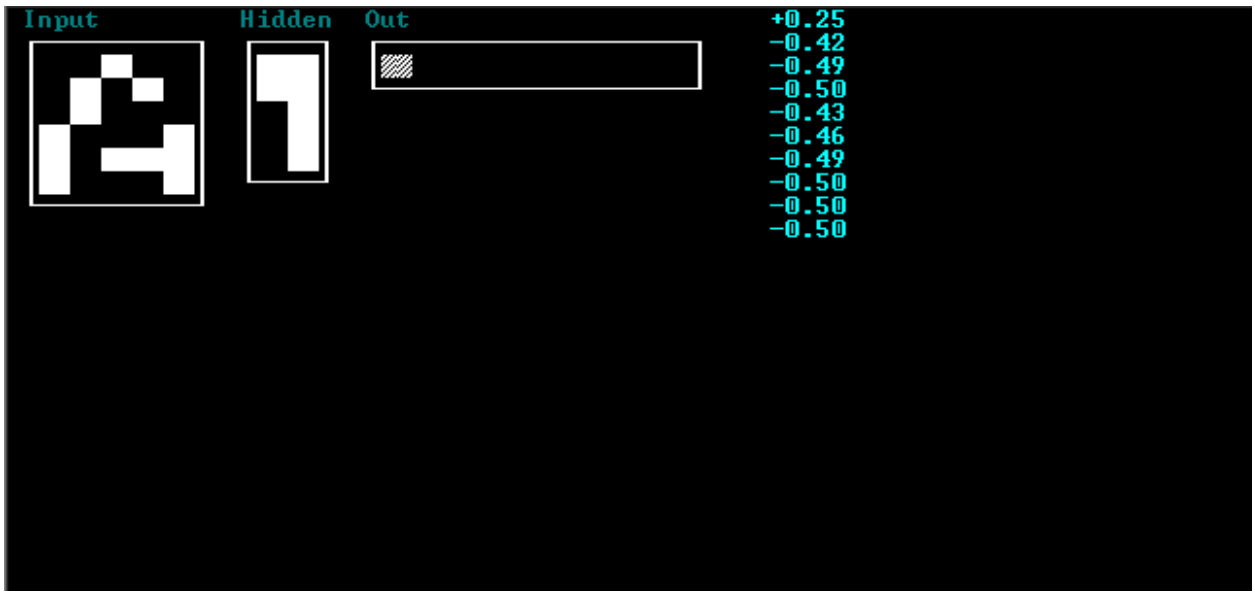
Результаты одного цикла обучения отобразятся на экране. Входной вектор (нулевой слой сети) имеет название “Input”, первый слой сети (скрытый) –

“Hidden” и второй слой сети (выходной) – “Out”. Для наглядности входной вектор отображается не как вектор а как матрица 5x6, промежуточный слой - матрица 2x5 и выходной – матрица 10x1. На экран отображается значение аксонов нейронов во всех слоях сети 0-2. В нулевом слое 30 нейронов, в промежуточном 10 и выходном 10. Значение каждого аксона нейрона лежит в диапазоне [-0.5;0.5]. Черный цвет соответствует -0.5, а белый 0.5. Также, в правом верхнем углу выведены на экран значения аксонов нейронов выходного слоя в числовом формате. Если нажать на любую клавишу, то процесс обучения пройдет автоматически и остановится, когда будет достигнута заданная точность.

После остановки обучения, программа предложит сохранить веса НС в файл. Если ввести пустую строку, то сохранение производится, не будет. Сохраните веса НС в файл wesa.txt. Просмотрите этот файл, например блокнотом. Теперь приступим к использованию обученной НС. Закройте проект. (Замечание: если параметры сети подобраны не удачно, то процесс обучения не когда не закончится. Это может произойти из за малого количества нейронов в скрытом слое. Этому могут способствовать большие коэффициенты обучения и зашумленности).



Откройте проект use.prj. Запустите программу. Отображение результатов аналогично описанию программы teacher. На вход сети подается тестируемый вектор. Значения промежуточного слоя не как не интерпретируются. Индекс максимального элемента в выходном массиве соответствует номеру класса к которому НС отнесла распознаваемый символ. При нажатии на любую клавишу будет распознан следующий вектор из файла. Процесс завершится, когда кончатся данные.



2.1 Используя обучающую программу (teacher), приведенную в разделе «тестовые задачи», построить двухслойную нейронную сеть и обучить ее распознавать графические образы из индивидуального задания.

2.2 Сохранить веса сети в файл. Затем загрузить веса из файла в программу и протестировать сеть на примерах, не предоставляемых ей во время обучения. Использовать обучающую программу (use).

2.3 Определить максимальное число нейронов в промежуточном слое, при котором процесс обучения не когда не закончиться.

2.4 Введите зашумление в обучающую выборку и повторите обучение, сохраните веса в файл. Сравните результаты распознавания сетью обучающейся с шумом и без него.

2.5 Подготовьте отчет, содержащий следующие пункты:

- Титульный лист
- Содержание
- Краткая теория
- Результаты работы. Таблица, содержащая следующие столбцы: SigmoidType, NiuParm, NiuParm, DSigma, количество циклов, потраченных на обучение. Графическое представление обучающих и тестирующих образов.
- Листинг программы
- Листинг обучающей выборки
- Листинг тестируемой выборки

3. Варианты заданий

3.1 цифры 0..9 матрица 6x6
3.2 десять букв латинского алфавита матрица 6x7
3.3 десять букв русского алфавита матрица 7x6
3.4 пять дорожных знаков матрица 8x8

4. Тестовые задачи

4.1. Текст основной программы обучения (teacher.cpp)

```
#include <string.h>
#include <conio.h>
#include "neuro.h"
void main()
{
    int N0 = 30;      // количество нейронов в нулевом слое
    int N1 = 10;      // количество нейронов в первом слое
    int N2 = 10;      // количество нейронов во втором слое
    NetBP N(3,N0,N1,N2);     // конструктор создания сети из трех слоев
       // нулевой слой фиктивный и в вычислениях не участвует
       // он нужен для хранения вектора входных данных
       // поэтому сеть счинается двухслойной
    float *Inp, *Out;         // массивы входных и выходных данных сети
    Inp = new float [N0];       // массив входных данных сети
    Out = new float [N2];       // массив выходных данных сети
    unsigned count;           // количество циклов обучения сети
    unsigned char buf[256];   // буфер для обработки текста
    int i;                    // хранит код нажатой клавиши
    ClearScreen();    // очистка экрана
    N.FullConnect();  // связывание слоев сети друг с другом
    N.GetLayer(0)->SetName("Input"); // задание имени слоя
    N.GetLayer(0)->SetShowDim(1,1,5,6); // задание кординат
       // и формы вывода слоя на экран
    N.GetLayer(1)->SetName("Hidden");   // задание имени слоя
    N.GetLayer(1)->SetShowDim(15,1,2,5);// задание кординат
       // и формы вывода слоя на экран
    N.GetLayer(2)->SetName("Out");      // задание имени слоя
    N.GetLayer(2)->SetShowDim(23,1,10,1);// задание кординат
       // и формы вывода слоя на экран
    N.SaveToFileConfigShow("config.txt");  // сохраняем в файл параметры
       // отображения слоев на экране
    SetSigmoidType(HYPERTAN);// задание вида активационной функции
       // HYPERTAN - гиперболический тангенс (рекомедуется)
       // ORIGINAL - сигмоид
    SetNiuParm(0.1);  // коэфициент обучения сети лежит в интервале 0..1
       // рекомендуеться использовать 0.1
       // при малом значении обучение будет проходить очень долго
       // при большом - старые навыки сети будут полностью заменяться новыми
       // и процесс обучения не когда не закончиться
    SetLimit(0.001);  // Величина Limit используется в методах IsConverged
       // для определения момента, когда сеть обучится или попадет в
       // паралич. В этих случаях изменения весов становятся меньше
       // малой величины Limit (рекомендуется 0.001)
    SetDSigma(1);     // Параметр DSigma эмулирует плотность шумаб
       // добавляемого к образам во время обучения НС.
       // Фактически, параметр DSigma равен числу входовб которые будут
       // инвертированы в случае двоичного образа.
       // Если DSigma = 0, помеха не вводится. (рекомендуется 1, 2)
    N.Randomize(1);   // Методы Randomize позволяют перед началом
       // обучения установить весовые коэффициенты в случайные
       // значения в диапазоне [-range+range]. (рекомендуется 1)
    N.SetLearnCycle(64000U); // Метод SomeNet::SetLearnCycle задает
       // число проходов по файлу образов, что в сочетании с
       // добавлением шума позволяет получить набор из нескольких
       // десятков и даже сотен тысяч различных образов.
       // (рекомендуется 64000U)
    N.OpenPatternFile("teacher.img");        // имя файла с обучающей выборкой
       // Если у файлов образов произвольное расширение?
       // то входные и выходные вектора записываются чередуясь,
       // строка с входным вектором, строка с соответствующим ему
       // выходным вектором и т.д. Каждый вектор есть последовательность
       // действительных чисел в диапазоне [-0.5,+0.5], разделенных
       // произвольным числом пробелов. Если файл имеет расширение IMG,
       // входной вектор представляется в виде матрицы символов
       // размером dy*dx (величины dx и dy должны быть заблаговременно
       // установлены с помощью LayerFF::SetShowDim для нулевого слоя),
       // причем символ 'x' или любой другой символ кроме
       // символа точка соответствует уровню 0.5,
       // а точка - уровню -0.5, то есть файлы IMG,
       // по крайней мере - в приведенной версии библиотеки,
       // бинарны. Когда сеть работает в нормальном режиме,
       // а не обучается, строки с выходными векторами могут быть пустыми.
    i=13;      // код клавиши Enter - 13
    for(count=0;;count++)     // основной цикл обучения
    {
       if(N.LoadNextPattern(Inp,Out)) break;// загружаем из файла пару
              // входного и выходного вектора
       N.Cycle(Inp,Out);     // обучеем сеть на текущем примере.
       N.GetLayer(0)->Show();// вывод на экран нулевого слоя
       N.GetLayer(1)->Show();  // вывод на экран первого слоя
```

```
        N.GetLayer(2)->Show();  // вывод на экран второго слоя
        N.GetLayer(2)->PrintAxons(47,0); // вывод на экран коэфициентов
                // второго слоя
        sprintf(buf,"Cycle %u",count);       // формируем строку с информацией
                                        // о количестве циклов
        out_str(1,23,buf,10 | (1<<4));  // выводим количество циклов на экран
        out_str(1,24,"ESC breaks ",11 | (1<<4)); // выводи подсказку на экран
        if(kbhit() || i==13) i=getch();      // если нажата клавиша или раньше
                // был нажат Enter тогда считать код нажатой клавиши
        if(i==27) break;// если нажат ESC то завершение обучения и выход
        if(i=='s' || i=='S') goto save;// если нажата клавиша 's' то
                // переход на сохранение файла
        if(count && N.IsConverged()) // если сеть обучена тогда сохранение
        {
            save:
            out_str(40,24,"FileConf:",15 | (1<<4));// выводим сообщение
            gotoxy(50,25);    // позиционируем курсор
            gets(buf); // вводи имя файла для сохранения весов сети
            if(strlen(buf)) N.SaveToFile(buf); // если имя файла не пустое
                // тогда созранить веса в файле
            break;     // завершение цикла обучения
        }
    }
    N.ClosePatternFile();     // закрытие файла с обучающей выборкой
    delete [] Inp;              // массив входных данных сети
    delete [] Out;              // массив выходных данных сети
}
```

4.2. Текст входного файла с обучающей выборкой (teacher.img)

```
..x..
.x.x.
.x.x.
x...x
xxxxx
x...x
A.........
x...x
xx.xx
xx.xx
x.x.x
x.x.x
x...x
.M........
x...x
x...x
xxxxx
x...x
x...x
x...x
..H.......
x...x
x..xx
x.x.x
x.x.x
xx..x
x...x
...N......
x..xx
x.x..
xx...
x.x..
x..x.
x...x
....K.....
...x.
..x.x
.x..x
x...x
x...x
x...x
.....L....
.xxx.
x...x
x....
x....
x...x
.xxx
......C...
xxxxx
```

```
..x..
..x..
..x..
..x..
..x..
.......T..
xxxxx
x...x
x...x
x...x
x...x
x...x
........P.
xxxx.
x...x
xxxx
x...x
x...x
xxxx.
.........B
```

4.1. Текст основной программы тестирования (use.cpp)

```
#include <string.h>
#include <conio.h>
#include "neuro.h"
main()
{
    NetBP N;
    if(N.LoadFromFile("wesa.txt")) {
       out_str(1,24,"Не могу открыть файл весов.",11 | (1<<4));
       getch();
       return 1;};// построение сети на основе
       // файла весов сети полученного в результате обучения
    if(N.FullConnect()) {
       out_str(1,24,"Не соответствие файла весов и параметров сети.",11 | (1<<4));
       getch();
       return 1;}; // связывание слоев сети друг с другом
    ClearScreen();                // очистка экрана
    float *Inp, *Out;// массивы входных и выходных данных сети
    Inp = new float [N.GetLayer(0)->GetRang()]; // массив входных данных сети
    Out = new float [N.GetLayer(N.GetRang()-1)->GetRang()]; // массив выходных данных сети
    switch (N.LoadFromFileConfigShow("config.txt")){
       case 1:
           out_str(1,24,"Не могу открыть файл конфигурации отображения на экран.",11 | (1<<4));
            getch();
           return 1;
       case 2:
           out_str(1,24,"Не соответствие файла конфигурации и количества слоев сети.",11 |
(1<<4));
           getch();
           return 1;
    }; // открытие файла конфигурации отображения на экран
       // этот файл формируется при обучении сети и в нем хранятся
       // имена слоев сети, положение на экране и формат вывода.
    SetSigmoidType(HYPERTAN);// задание вида активационной функции
       // HYPERTAN - гиперболический тангенс (рекомедуется)
       // ORIGINAL - сигмоид
    if(N.OpenPatternFile("use.img")) {
       out_str(1,24,"Не могу открыть файл с тестируемой выборкой.",11 | (1<<4));
       getch();
       return 1;};// имя файла с тестируемой выборкой
       // Если у файлов образов произвольное расширение?
       // то входные и выходные вектора записываются чередуясь,
       // строка с входным вектором, строка с соответствующим ему
       // выходным вектором и т.д. Каждый вектор есть последовательность
       // действительных чисел в диапазоне [-0.5,+0.5], разделенных
       // произвольным числом пробелов. Если файл имеет расширение IMG,
       // входной вектор представляется в виде матрицы символов
       // размером dy*dx (величины dx и dy должны быть заблаговременно
       // установлены с помощью LayerFF::SetShowDim для нулевого слоя),
       // причем символ 'x' или любой другой символ кроме
       // символа точка соответствует уровню 0.5,
       // а точка - уровню -0.5, то есть файлы IMG,
       // по крайней мере - в приведенной версии библиотеки,
       // бинарны. Когда сеть работает в нормальном режиме,
       // а не обучается, строки с выходными векторами могут быть пустыми.
    int i=13;  // код клавиши Enter - 13
    for(;;)// основной цикл тестирования
    {
```

```
        if(N.LoadNextPattern(Inp,Out)) break;// загружаем из файла пару
                // входного и выходного вектора в данной программе
                // выходной вектор не нужен
                // когда образци кончатся то завершаем цикл тестирования
        N.SetNetInputs(Inp);  // заносим вектор входных данных в сеть
        N.Propagate();          // прощитываем значение сети
        N.GetLayer(0)->Show();// вывод на экран нулевого слоя
        N.GetLayer(1)->Show();  // вывод на экран первого слоя
        N.GetLayer(2)->Show();  // вывод на экран второго слоя
        N.GetLayer(2)->PrintAxons(47,0); // вывод на экран коэфициентов
                // второго слоя
        if(kbhit() || i==13) i=getch();      // если нажата клавиша или раньше
                // был нажат Enter тогда считать код нажатой клавиши
        if(i==27) break;// если нажат ESC то завершение обучения и выход
    }
    N.ClosePatternFile();// закрытие файла с тестирующей выборкой
    delete [] Inp;              // массив входных данных сети
    delete [] Out;              // массив выходных данных сети
    return 0;
}
```

4.4 Текст входного файла с тестируемой выборкой (use.img)

```
..x..
.x.x.
.x...
x...x
x.xxx
x...x
A.........
x...x
xx.xx
xx.xx
..x.x
x.x.x
x....
.M........
x...x
x....
xx.xx
x...x
x...x
x...x
..H.......
x...x
x..x.
x.x.x
..x.x
xx..x
x...x
...N......
...xx
x.x..
x....
x.x..
x..x.
x...x
....K.....
...x.
..x.x
.x…
x...x
....x
x...x
.....L....
.xx..
x...x
x....
.....
x...x
.xxx
......C...
x.xxx
..x..
..x..
..x..
…..
..x..
.......T..
x.xxx
```

```
x...x
x...x
x...x
x....
x...x
........P.
xxxx.
....x
xxxx
x..xx
x....
xxxx.
.........B
```

Литература

1. Осовский С. Нейронные сети для обработки информации / Пер. с польского И.Д. Рудинского. – М.: Финансы и статистика, 2002. – 344 с.: ил.
2. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика. — М.: Мир, 1992.
3. С. Короткий Нейронные сети: основные положения - Электронная публикация.
4. С. Короткий Нейронные сети: алгоритм обратного распространения - Электронная публикация.
5. С. Короткий Нейронные сети: Нейронные сети: обучение без учителя - Электронная публикация.
6. С. Короткий Нейронные сети Хопфилда и Хэмминга - Электронная публикация.