МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО

ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА БИОФИЗИКИ И БИОКИБЕРНЕТИКИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему:**Сравнительный анализ** **нейросетевых реализаций**

**алгоритмов распознавания образов.**

Студент И.Е.Шепелев

Научный руководитель А.И.Самарин

К ЗАЩИТЕ ДОПУСКАЮ

Зав. кафедрой д.б.н,профессор

Б.М.Владимирский

г.Ростов-на-Дону

1998 г.

**ОГЛАВЛЕНИЕ**

Стр.

Введение 2

1. Общее описание исследуемых нейронных сетей.
2. Однослойные сети. Персептрон.
3. Многослойные сети.
4. Сети Хопфилда.
5. Описание программных моделей и алгоритмов их обучения.
6. Методика и результаты экспериментальных исследований.
7. Проблема функции ИСКЛЮЧАЮЩЕЕ ИЛИ.

3.2 Исследование представляемости однослойной и двухслойной нейронной сети.

1. Релаксация стимула.
2. Возникновение ложного образа. Выработка прототипа.
3. Бистабильность восприятия.

Заключение.

Литература.

Приложение 1.

Приложение 2.

**Введение.**

 Основным назначением системы распознавания образов является отыскание решения о принадлежности предъявляемых ей образов некоторому классу. Один из важных подходов к задаче предполагает использование разделяющих функций. В условиях, когда мы обладаем лишь немногочисленными априорными сведениями о распознаваемых образах, при построении распознающей системы лучше всего использовать обучающую процедуру. На первом этапе выбираются произвольные разделяющие функции и затем в процессе выполнения итеративных шагов обучения эти разделяющие функции доводятся до оптимального либо приемлемого вида.

 К определению разделяющих функций в настоящее время существует несколько подходов. В данной работе рассматриваются два из них, которые имеют соответствующие нейросетевые реализации. Первый из них - персептронный подход.

 В начале 60х годов Минский, Розенблатт, Уиндроу и другие разработали сети, состоящие из одного слоя искусственных нейронов. Часто называемые персептронами, они были использованы для широкого класса задач, в том числе и искусственного зрения. Однако Минский и Пайперт [2] доказали, что используемые в то время однослойные сети теоретически неспособны решить многие простые задачи.

 Сети обратного распространения - наиболее успешный, по-видимому, из современных алгоритмов, преодолевает ограничения, указанные Минским. Обратное распространение является систематическим методом для обучения многослойных сетей.

Второй подход при определении разделяющих функций использует понятие потенциальной энергии. Так Хопфилдом [7] в задаче о релаксации некоторого начального состояния к одному из устойчивых состояний нейронной сети было введено понятие потенциальной энергии, которая уменьшалась в процессе релаксации. Соответствующие нейронные сети известны в литературе как сети Хопфилда. Это сети с обратными связями, которые в связи с этим обладают некоторыми свойствами, заслуживающими внимания.

На этапе бакалаврской работы цель состояла в создании компьютерных моделей указанных выше нейронных сетей и анализе некоторых свойств, связанных с их обучением и распознаванием ими образов.

**1.Общее описание исследуемых нейронных сетей.**

1. *Однослойные сети. Персептрон.*

 В соответствии с [3] персептрон представляет собой модель обучаемой распознающей системы. Он содержит матрицу светочувствительных элементов (S-элементы), ассоциативные элементы (А-элементы) и реагирующие элементы (R-элементы). По сути персептрон состоит из одного слоя искусственных нейронов, соединенных с помощью весовых коэффициентов с множеством входов.

 В 60е годы персептроны вызвали большой интерес. Розенблатт [3] доказал теорему об обучении персептрона и тем самым показал, что персептрон способен научиться всему, что он способен представлять. Уидроу [12-15] дал ряд убедительных демонстраций систем персептронного типа. Исследования возможности этих систем показали, что персептроны не способны обучиться ряду простых задач. Минский [2] строго проанализировал эту проблему и показал, что имеются жесткие ограничения на то, что могут выполнять однослойные персептроны, и, следовательно, на то, чему они могут обучаться.

 Один из самых пессимистических результатов Минского показывает, что однослойный персептрон не может воспроизвести такую простую функцию как  **ИСКЛЮЧАЮЩЕЕ ИЛИ**. Это - функция от двух аргументов, каждый из которых может быть нулем или единицей. Она принимает значение единицы, когда один из аргументов равен единице (но не оба). Если проблему представить с помощью однослойной однонейронной системы, то легко видеть, что при любых значениях весов и порогов невозможно расположить прямую линию, разделяющую плоскость (пространство образов) так, чтобы реализовывалась функция **ИСКЛЮЧАЮЩЕЕ ИЛИ.** Имеется обширный класс функций (наряду с функцией **ИСКЛЮЧАЮЩЕЕ ИЛИ**), не реализуемых однослойной сетью. Об этих функциях говорят, что они являются линейно неразделимыми, и они накладывают определенные ограничения на возможности однослойных сетей. Линейная разделимость ограничивает однослойные сети задачами классификации, в которых множества точек (соответствующих входным значениям) могут быть разделены геометрически. В случае двух входов разделитель является прямой линией. В случае трех входов разделение осуществляется плоскостью, рассекающей трехмерное пространство. Длячетырехили более входов визуализация невозможна и необходимо мысленно представить n-мерное пространство, рассекаемое ‘‘гиперплоскостью’’ - геометрическим объектом, который рассекает пространство четырех или большего числа измерений. Как показано в [11], вероятность того, что случайно выбранная функция окажется линейно разделимой, весьма мала. Так как линейная разделимость ограничивает возможности персептронного представления, то однослойные персептроны на практике ограничены простыми задачами.

 Чтобы сеть представляла практическую ценность, нужен систематический метод (алгоритм) для вычисления значений весов и порогов. Процедуру подстройки весов обычно называют обучением. Цель обучения состоит в том, чтобы для некоторого множества входов давать желаемое множество выходов. Алгоритм обучения персептрона был предложен в [3] и имеет множество модификаций. В настоящей работе реализованна модель одного из вариантов.

1. *Многослойные сети.*

Серьезное ограничение представляемости однослойными сетями можно преодолеть, добавив дополнительные слои. Многослойные сети можно получить каскадным соединением однослойных сетей, где выход одного слоя является входом для последующего слоя, причем такая сеть может привести к увеличению вычислительной мощности лишь в том случае, если активационная функция между слоями будет нелинейной.

Многослойные сети способны выполнять общие классификации, отделяя те точки, которые содержаться в выпуклых ограниченных или неограниченных областях. Если рассмотреть простую двухслойную сеть с двумя нейронами в первом слое, соединенными с единственным нейроном во втором слое, то каждый нейрон первого слоя разбивает плоскостьна две полуплоскости, образуя в пространстве образов V-образную область, а нейрон второго слоя реализует различные функции при подходящем выборе весов и порога. Аналогично во втором слое может быть использовано три нейрона с дальнейшим разбиением плоскости и созданием области треугольной формы. Включением достаточного числа нейронов во входной слой может быть образован выпуклый многоугольник любой желаемой формы. Точки, не составляющие выпуклой области, не могут быть отделены о других точек плоскости двухслойной сетью.

 Трехслойная сеть является более общей. Ее классифицирующие возможности ограничены лишь числом искусственных нейронов и весов. Ограничения на выпуклость отсутствуют. Теперь нейрон третьего слоя принимает в качестве входа набор выпуклых многоугольников, и их логическая комбинация может быть невыпуклой. При добавлении нейронов и весов число сторон многоугольника может неограниченно возрастать. Это позволяет аппроксимировать область любой формы с любой точностью. В добавок не все выходные области второго слоя должны пересекаться. Возможно, следовательно, объединять различные области, выпуклые и невыпуклые, выдавая на выходе единицу всякий раз, когда входной вектор принадлежит одной из них.

 Для обучения искусственных нейронных сетей широко применяется процедура обратного распространения. Обратное распространение было независимо предложено в трех различных работах [8, 9, 10]. В работе программно реализованна двухслойная сеть обратного распространения.

 *1.3 Сети Хопфилда.*

Сети, рассмотренные выше, не имели обратных связей, т.е. связей, идущих от выходов сети к их входам. Отсутствие обратных связей гарантирует безусловную устойчивость сетей. Так как сети с обратными связями имеют пути от выходов к входам, то отклик таких сетей является динамическим, т.е. после приложения нового входа вычисляется выход и, передаваясь по сети обратной связи, модифицирует вход. Затем выход повторно вычисляется и процесс повторяется снова и снова. Для устойчивой сети последовательные итерации приводят к все меньшим изменениям выхода, пока в конце концов выход не становится постоянным. Для многих сетей процесс никогда не заканчивается, такие сети называются неустойчивыми. Проблема устойчивости ставила в тупик первых исследователей. Никто не был в состоянии предсказать, какие из сетей будут устойчивыми, а какие будут находится в постоянном изменении. К счастью, в работе [5] была получена теорема, описавшая подмножество сетей с обратными связями, выходы которых в конце концов достигают устойчивого состояния. Это замечательное достижение открыло дорогу дальнейшим исследованиям.

Дж. Хопфилд сделал важный вклад как в теорию, так и в применение систем с обратными связями. В его работе [7] при имитации поведения ансамбля нейронов использовались переменные, описывающие состояния нейронов (вектор состояния σ), и переменные, описывающие связи между нейронами (оператор памяти W),а также два уравнения , определяющие изменение σ со временем. Одно из этих уравнений представляет изменение σ под действием оператора W (выработка реакции на стимул), а второе - изменение матрицы W, квадратичное по σ (запоминание).При этом вектор состояния ансамбля нейронов представляет собой вектор в фазовом пространстве динамической системы , а ‘‘память’’ реализована как система аттракторов. Запоминание новой информации осуществляется путем усложнения по определенному алгоритму структуры аттракторов. Такой подход допускает простую механическую аналогию, если представить себе вектор состояния как положение частицы, движущейся под действием силы тяжести и трения по некоторому рельефу. При скатывании с ‘‘горы’’ в одну из ‘‘низин’’ потенциальная энергия системы уменьшается, и в конце концов материальная точка останавливается из-за трения. Положение частицы в конечном состоянии (т.е. та из низин, в которой она останавливается) зависит как от формы рельефа, так и от начального состояния, с которого началось скатывание. Функционирование сети легко визуализируется геометрически. В случае двух бинарных нейронов в выходном слое каждой вершине квадрата соответствует одно из четырех состояний системы (00,01,10,11). В случае трехнейронной системы пространство образов представлено кубом (в трехмерном пространстве), имеющим 8 вершин, каждая из которых помечена трехбитовым бинарным числом. В общем случае система с n нейронами имеет 2n различных сосояний и представляется n - мерным гиперкубом. Когда подается новый входной вектор, сеть переходит из вершины в вершину, пока не стабилизируется. Устойчивая вершина определяется сетевыми весами, текущими входами и величиной порога. Если входной вектор частично неправилен или неполон, то сеть стабилизируется к вершине, ближайшей к желаемой.

В данной работе представлена программная реализация сети Хопфилда и проиллюстрираваны ряд ее свойств.

**2. Описание программных моделей и алгоритмов их обучения.**

 В работе моделируется персептрон с последовательными связями, который представляет собой передающую сеть состоящую из генераторов сигнала трех типов: сенсорных элементов, ассоциативных элементов и реагирующих элементов. В качестве сенсорных элементов используются ячейки измерительной сетчатки. Сетчатка состоит из 35 элементов. S-элемент выдает выходной сигнал Si=+1,если соответствующая ему ячейка сетчатки закрашена; Si=-1 в противном случае. Каждый S-элемент связан с соответствующим А-элементом и выходной сигнал А-элемента равен выходному сигналу S-элемента. Топологическая структура персептрона с двумя R-элементами такова, что каждый А-элемент связан со всеми элементами. Эти связи обладают весовыми коэффициентами, которые модифицируются методом коррекции ошибок. Такая топологическая структура удобна с точки зрения ее программной реализации. Выход сети представлен двумя R-элементами, то целевые вектора имеют вид (-1,+1) и (1,-1) в зависимости от предъявляемой обучающей пары.

Обучение персептрона происходит по следующей схеме:

1. Все веса рандомизируются в величины из диапазона(-0.5,+0.5).
2. На вход сети подается обучающий вектор и вычисляется выход каждого нейрона как алгебраическая сумма его взвешенных входов .
3. В модели использован метод обучения Уидроу - Хоффа [15], где используется непрерывный сигнал, чтобы вычислить ошибку для каждого нейрона, то есть вычитается выходной сигнал, как алгебраическая сумма его взвешенных входов, из требуемого выхода.

 4.Корректируется веса сети так, чтобы минимизировать ошибку. (см. **Приложение** **2**. *Программа, моделирующая однослойную сеть.* )

 5.Повторяются шаги со второго по четвертый до тех пор пока общая ошибка сети не станет меньше допустимой, заданной в данной модели. Величина допустимой ошибки изменяется в целях эксперимента.

Далее работе моделировалась двухслойная сеть обратного распространения. Для подсчета количества слоев использованы следующие соображения: слой состоит из множества весов со следующими за ними нейронами, суммирующими взвешенные сигналы. Первый слой состоит из двадцати нейронов, второй (выходной) - из двух нейронов.

В качестве активационной функции нейронов первого и второго слоя использована сигмоидальная функция. Так как многослойные нейронные сети обладают большей представляющей мощностью, чем однослойные, только в случае присутствия нелинейности, то сигмоид обеспечивает это требование. Данная функция всюду дифференцируема, что используется в алгоритме обратного распространения. Ее дополнительные преимущество состоит в автоматическом контроле усиления. Для слабых сигналов кривая вход - выход имеет сильный наклон, дающий большое усиление, а когда величина сигнала становится больше, усиление падает. Таким образом большие сигналы воспринимаются сетью без насыщения, а слабые сигналы проходят по сети без чрезмерного ослабления.

Обучение сети обратного распространения в модели происходило по следующей схеме:

1. Всем весам присвоены начальные значения, выбранные случайным образом из диапазона (-0.5,+0.5) .
2. Выбирается очередная обучающая пара из обучающего множества и подается входной вектор на вход сети.
3. *Проход вперед* . Вычисляется выход сети. Вычисления выполняются послойно. Сначала вычисляются выходы первого нейронного слоя, затем они используются в качестве входов второго слоя; вычисляются выходы нейронов второго слоя, которые и образуют выходной вектор сети.
4. Вычисляется разность между выходом сети и требуемым выходом (целевым вектором обучающей пары), то есть каждый из выходов сети вычитается из соответствующей компоненты целевого вектора, чтобы получить ошибку.
5. *Обратный проход* . Корректируется веса сети так, чтобы минимизировать ошибку. Так как для каждого нейрона выходного слоя задано целевое значение, то ошибка находится с использованием модифицированного дельта-правила для персептрона. Выходы нейронов скрытого слоя не имеют целевых значений для сравнения, то для поиска ошибки (и коррекции весов) делается ряд шагов, составляющие алгоритмы подстройки весов выходного и скрытого слоев (см. **Приложение** **2**. *Программа, моделирующая сеть обратного распространения* ).
6. Повторяются шаги со второго по пятый для каждого вектора обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемого уровня. Величина предельной ошибки изменяется в целях эксперимента.

 В модели сети используется метод ускорения обучения для алгоритма обратного распространения, предложенный в [9], увеличивающий также устойчивость процесса. Этот метод, названный импульсом, заключается в добавлении к коррекции веса члена, пропорционального величине предыдущего изменения веса. Коэффициент импульса выбран равным 0.5.

 В доказательстве алгоритмов обучения персептрона [3] и процедуры обратного распространения [9] ничего не говорится о том, сколько шагов требуется для обучения сети. В данной работе для контроля остановки процесса обучения этих сетей введена величина допустимой ошибки сети, которая изменяется в экспериментах с моделями, чтобы определить условие разделимости для конкретного обучающего множества.

Для исследования сети с обратными связями в работе моделируется предложенный Хопфилдом в [7] рекуррентный алгоритм анализа динамики ансамбля нейронов.

Состояние сети описывается множеством текущих значений сигналов от всех нейронов. На каждом шаге алгоритма меняется состояние лишь одного нейрона. Номер нейрона выбирается случайным образом. Он суммирует с весами Wij сигналы, поступающие от выходов нейронов сети по обратным связям и изменяет свое состояние в соответствии со значением пороговой функции. Состояние же остальных нейронов совпадает с состоянием на предыдущем шаге. Пороги возбуждения нейронов выбраны равными нулю. Состояние активного нейрона на текущем шаге алгоритма равно 1, неактивного - -1. Алгоритм использует модель обучения, предложенную Хэббом [6] (см. **Приложение** **2**. *Программа, моделирующая сеть Хопфилда.*). Это - модель обучения без учителя, в которой вес возрастает, если активированы оба нейрона, источник и приемник. Таким образом, часто используемые пути в сети усиливаются, что объясняет феномен обучения сети.

 В процессе релаксации к одному из устойчивых состояний, сети с обратными связями иллюстрируют ряд свойств, которые могут быть интерпретированы как релаксация стимула, выработка прототипа, бистабильность восприятия , которые показаны в данной работе на реализованной модели.

Все программы созданы в DELPHI и работают в операционной среде Windows.

**3. Методика и результаты экспериментальных исследований.**

1. *Проблема функции ИСКЛЮЧАЮЩЕЕ ИЛИ.*

 Как было сказано, один из самых пессимистических результатов Минского показывает, что однослойный персептрон не может воспроизвести такую простую функцию как *ИСКЛЮЧАЮЩЕЕ**ИЛИ*.

Для того, чтобы проиллюстрировать эту проблему, персептрону предоставляли четыре обучающих пары. К одному классу образов относили, подаваемое на измерительную сетчатку 5х7, шахматное поле и такое же поле, но с инвертируемым цветом. К другому классу - буквенные символы E и F (см. Рис.1 **Приложение** **1**.). В результате распознавания персептрон классифицировал первый из указанных образов как относящийся к первому классу, а второй, третий и четвертый - ко второму. Т. е. персептрон ошибся на втором образе.

Эти же обучающие пары предоставили для обучения и распознавания модели двухслойной сети. В результате сеть все образы классифицировала правильно. Эти результаты согласуются с теорией представляемости персептрона и двухслойных сетей.

1. *Исследование представляемости однослойной и двухслойной нейронной сети.*

 Исследование представляемости однослойной и двухслойной нейронных сетей производились на соответствующих моделях.

Результаты эксперимента с моделью персептрона показали, что персептрон способен моделировать функцию, разделяющую изображения цифр 1, 2, 3, 4, 5, 6, 7, 8 и 9, подаваемых на сетчатку, на два класса - четные и нечетные (см. Рис. 2. **Приложение** **1**.). Величина допустимой ошибки сети была равна 0.001.

Следующий эксперимент был поставлен для выяснения способности персептрона и двухслойной сети обратного распространения классифицировать буквенно-цифровые изображения (десять цифр и двадцать шесть букв латинского алфавита) на два класса - цифры и буквы (см. Рис. 3. **Приложение** **1**.). При уровне допустимой ошибки 0.001 персептрон путал классами цифры 1, 3, 7 и букву А. Сеть обратного распространения при том же уровне допустимой ошибки 0.001 путала классами цифры 3, 4, 5 и буквы S, I. Далее величина ошибки уменьшалась на порядок. Персептрон путал класс цифры 1, сеть обратного распространения - классами цифру 3 и буквы S, I. При уровне допустимой ошибки 0.00001 персептрон путал классами цифру 1 . Сеть обратного распространения при том же уровне допустимой ошибки 0.00001 путала класс буквы S. При уровне допустимой ошибки 0.000001 и персептрон, и сеть обратного распространения ошибок в классификации не допускали.

1. *Релаксация стимула.*

Как говорилось состояние сети Хопфилда описывается множеством текущих значений сигналов от всех нейронов, причем на каждом шаге алгоритма меняется состояние лишь одного нейрона, номер которого выбирается случайным образом. Он суммирует с весами Wij сигналы, поступающие от выходов нейронов сети по обратным связям и изменяет свое состояние в соответствии со значением пороговой функции. Пример последовательности состояний, которые проходит система нейронов при распознавании образов в модели Хопфилда, приведен на рис. 4. **Приложение** **1**, где показаны промежуточные состояния. Левая картинка представляет собой исходный стимул, а крайняя правая - образ, записанный в памяти системы. При такой релаксации энергия системы уменьшается, достигая с течением времени одного из локальных минимумов.

1. *Возникновение ложного образа. Выработка прототипа.*

Устойчивыми состояниями сети Хопфилда могут быть также образы, не записанные ее память - ложные образы. На Рис. 5. **Приложение** **1.**  показан пример устойчивого ложного образа (), возникающий при распознавании стимулов сетью, в матрице связей которой записаны образы **a, b, c.**

При увеличении числа образов, записываемых в память системы, отвечающие им минимумы энергии могут сливаться. На рис. 6. **Приложение** **1.** показана группа образов (получающихся при небольших искажениях), записанные в память сети - a, b, c. При предъявлении сети этих образов для распознавания, сеть приходит к состоянию, соответствующего выработанному прототипу - d.

Модель демонстрирует процедуру разобучения, предназначенную для устранения из памяти ложных образов. Разобучение состоит в предъявлении сети образа, к которому релаксировал стимул. При этом из матрицы связи вычитается тот член, который при обучении бы прибавлялся. В применении к случаю прототипа с тремя записанными в памяти образами (см. Рис. 6. **Приложение** **1.**), разобучение приводит к исчезновению прототипа и к появлению вместо одной потенциальной ямы, соответствующей прототипу, трех потенциальных ям, соответствующих каждому образу, записанному в памяти сети.

1. *Бистабильность восприятия.*

Эксперименты с моделью сети Хопфилда показали (см. Рис. 7. **Приложение** **1.**), что средние стимулы последовательности могут восприниматься либо как искаженный левый, либо как искаженный правый образы, т. е. имеет место бистабильностьвосприятия. Смена устойчивого состояния происходит после предъявления четвертой картинки.

**Заключение**.

В работе созданы программные модели трех нейронных сетей: персептрон, сеть обратного распросранения и сеть Хопфилда. Модели позволяют проиллюстрироваь основные достоинства и недостатки, а также ряд специфических свойств реализованных моделей.

Во всех моделях для приложения внешнего стимула использовалась, специально разработанная программно, измерительная сетчатка.

 Результаты серии экспериментов, проведенных на моделях, показали, что:

* Способность персептрона и сети обратного распространения моделировать определенную функцию зависит от допустимой общей ошибки сети.
* Топологическая структура сети Хопфилда обуславливает ее свойства, которые можно интерпритировать как релаксация стимула, выработка прототипа, бистабильность восприятия.

 В дальнейшем планируется разработка программных моделей более сложных нейронных сетей и их комбинаций с целью получения наиболее эффективных алгоритмов для задачи распознавания образов.

**Литераура.**

1.Горбань А.Н.,Россиев Д.А..Нейронные сети на персональном компьюере.

 2. Минский М.Л.,Пайперт С..Персепроны.М.: Мир.1971

 3. Розенблатт Ф.Принципы нейродинамики.М.: Мир.1965

 4. Уоссермен Ф.Нейрокомпьютерная техника.М.: Мир.1992.237С

1. Cohen M.A.,Grossberg S.G.Absoiute stability of global pattern formation and parallel memory storage by compatitive neural networks.1983
2. Hebb D.O.Organization of behavior.New York:Science Edition
3. Hopfield J.J.Neural networks and physical systems with emergent collective computational abilities. Proseedings of the National Academy of Science 79.1982
4. Parker D.B. Learning-logic. Invention Report. 1982
5. Rumelhart D.E. Hinton G.E.,Williams R.J. Learning internal representations by error propagation. In Parallel distributed processing, vol.1986
6. Werbos P.G. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974
7. Wider R.O. Single-stage logic, Paper presented at the AIEE Fall General Meeting. 1960
8. Windrow B. The speed of adaptetion in adaptive control system. 1961
9. Windrow B. A statistical theory of adaptetion. Adaptive control systems. 1963
10. Windrow B., Angell J.B. Reliable, trainable networks for computing and control. 1962
11. Windrow B., Hoff N.E. Adaptive switching circuits. 1960

**Приложение 1.**

 

 

**Рис. 1.**

  

  

  

**Рис. 2.**

   

   

   

   

**Рис. 3.**

   

   

  



**Рис. 3.**

 

 



**Рис. 4.**









**Рис. 5.**









**Рис. 6.**

  

  

**Рис. 7.**

**Приложение 2.**

*Программа, моделирующая однослойную сеть.*

unit UPerc;

interface

uses

 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,

 Percept\_Field, StdCtrls, Buttons, ExtCtrls;

const InputLayerUnits=35;

 OutputLayerUnits=2;

 eta=0.05;

 epsilon=0.000001;

 numberpatterns=36;

type

 TFrmPerceptron = class(TForm)

 Percept\_FieldPerc: TPercept\_Field;

 GroupBoxTrain: TGroupBox;

 GroupBoxInit: TGroupBox;

 ComboBoxABC: TComboBox;

 ComboBoxDigits: TComboBox;

 BtnNext: TButton;

 BitBtnClose: TBitBtn;

 EditNumPat: TEdit;

 LabelNumPat: TLabel;

 GroupBoxRec: TGroupBox;

 LabelInput: TLabel;

 BtnOutput: TButton;

 LabelOdd: TLabel;

 RadioGroupTarget: TRadioGroup;

 RadioButtonOdd: TRadioButton;

 RadioButtonEven: TRadioButton;

 LabelOr: TLabel;

 LabelEven: TLabel;

 procedure ComboBoxABCChange(Sender: TObject);

 procedure ComboBoxDigitsChange(Sender: TObject);

 procedure Percept\_FieldPercMouseDown(Sender: TObject;

 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

 procedure BitBtnCloseClick(Sender: TObject);

 procedure FormCreate(Sender: TObject);

 procedure BtnNextClick(Sender: TObject);

 procedure BtnOutputClick(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 FrmPerceptron: TFrmPerceptron;

var

 w:array[1..OutputLayerUnits,1..InputLayerUnits] of real;

 indexBtnNextClick:byte;

 activation:array[1..OutputLayerUnits] of real;

 OutputLayerOutput:array[1..OutputLayerUnits] of shortint;

 target:array[1..numberpatterns,1..OutputLayerUnits] of shortint;

 v:array[1..numberpatterns,1..InputLayerUnits] of shortint;

implementation

{$R \*.DFM}

procedure TFrmPerceptron.Percept\_FieldPercMouseDown(Sender: TObject;

 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

 var m,k:BYTE;

 correctRect:shortint;

 L,T,H,V:INTEGER;

begin

 L:=0;

 T:=0;

 H:=Percept\_FieldPerc.UnitHorizontal;

 V:=Percept\_FieldPerc.UnitVertical;

 for m :=1 to Percept\_FieldPerc.UnitRectVert do

 begin

 for k :=1 to Percept\_FieldPerc.UnitRectHorz do

 begin

 if (X<H) and (X>L) and (Y<V) and (Y>T) then

 begin

 correctRect:=k+Percept\_FieldPerc.UnitRectHorz\*(m-1);

 if (Button=mbLeft) and

 (Percept\_FieldPerc.Brushes[correctRect]=Percept\_FieldPerc.BackGroundBrush) then

 begin

 Percept\_FieldPerc.Brushes[correctRect]:=Percept\_FieldPerc.RectBrush;

 end

 else

 if (Button=mbRight) and

 (Percept\_FieldPerc.Brushes[correctRect]=Percept\_FieldPerc.RectBrush)then

 begin

 Percept\_FieldPerc.Brushes[correctRect]:=Percept\_FieldPerc.BackGroundBrush;

 end;

 end;

 inc(L,Percept\_FieldPerc.UnitHorizontal);

 inc(H,Percept\_FieldPerc.UnitHorizontal);

 end;

 inc(T,Percept\_FieldPerc.UnitVertical);

 inc(V,Percept\_FieldPerc.UnitVertical);

 L:=0;

 H:=Percept\_FieldPerc.UnitHorizontal;

 end;

end;

procedure TFrmPerceptron.BitBtnCloseClick(Sender: TObject);

begin

 Close;

end;

procedure TFrmPerceptron.FormCreate(Sender: TObject);

var i,j:byte;

 rand:real;

begin

 //numberpatterns:=2;//10;

 EditNumPat.Text:=inttostr(numberpatterns);

 BtnNext.Font.Color:=clRed;

 indexBtnNextClick:=0;

 LabelInput.Visible:=False;

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Randomize;// случайные веса (-0.5,0.5)

 for i := 1 to OutputLayerUnits do

 begin

 for j := 1 to InputLayerUnits do

 begin

 rand:=Random-0.5;

 w[i,j]:=rand;

 end;

 end;

end;

procedure TFrmPerceptron.BtnNextClick(Sender: TObject);

var i,j,m:byte;

 sum:real;

 neterror,err:real;

 error:array[1..OutputLayerUnits] of real;

 stop:boolean;

 krandom:integer;

begin

 indexBtnNextClick:=indexBtnNextClick+1;

 for m:=1 to InputLayerUnits do begin

 if (Percept\_FieldPerc.Brushes[m]=Percept\_FieldPerc.RectBrush) then

 begin

 v[indexBtnNextClick,m]:=1;

 end

 else

 if (Percept\_FieldPerc.Brushes[m]=Percept\_FieldPerc.BackGroundBrush) then

 begin

 v[indexBtnNextClick,m]:=-1;

 end;

 end;

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ODD or EVEN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if RadioButtonOdd.Checked then

 begin

 target[indexBtnNextClick,1]:=1;

 target[indexBtnNextClick,2]:=-1;

 end

 else

 if RadioButtonEven.Checked then

 begin

 target[indexBtnNextClick,1]:=-1;

 target[indexBtnNextClick,2]:=1;

 end;

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if (indexBtnNextClick+1)=numberpatterns then

 begin

 BtnNext.Caption:='last';

 end

 else

 begin

 if (indexBtnNextClick)=numberpatterns then

 begin

 BtnNext.Font.Color:=clWindowText;

 BtnNext.Caption:='finished';

 LabelInput.Font.Color:=clRed;

 LabelInput.Visible:=True;

 end

 else

 begin

 BtnNext.Caption:='next';

 end;

 end;

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if (indexBtnNextClick)=numberpatterns then

 begin

 repeat

 stop:=false;

 for m := 1 to numberpatterns do

 begin

 for i := 1 to OutputLayerUnits do

 begin

 sum:=0;

 for j := 1 to InputLayerUnits do

 begin

 sum:=sum+w[i,j]\*v[m,j];

 end;

 activation[i]:=sum;

 if sum>=0 then

 begin

 OutputLayerOutput[i]:=1;

 end

 else

 begin

 OutputLayerOutput[i]:=-1;

 end;

 end;

 neterror:=0;

 for i := 1 to OutputLayerUnits do

 begin

 err:=target[m,i]-activation[i];

 error[i]:=err;

 neterror:=neterror+0.5\*sqr(err);

 end;

 if neterror<epsilon then

 begin

 stop:=true;

 end;

 end;

 if not stop then //обучение

 begin

 Randomize;

 for krandom := 1 to 10\*numberpatterns do

 begin

 m:=1+Round(Random(numberpatterns));

 for i := 1 to OutputLayerUnits do

 begin

 sum:=0;

 for j := 1 to InputLayerUnits do

 begin

 sum:=sum+w[i,j]\*v[m,j];

 end;

 activation[i]:=sum;

 if sum>=0 then

 begin

 OutputLayerOutput[i]:=1;

 end

 else

 begin

 OutputLayerOutput[i]:=-1;

 end;

 end;

 neterror:=0;

 for i := 1 to OutputLayerUnits do

 begin

 err:=target[m,i]-activation[i];

 error[i]:=err;

 neterror:=neterror+0.5\*sqr(err);

 end;

 for i := 1 to OutputLayerUnits do

 begin

 for j := 1 to InputLayerUnits do

 begin

 w[i,j]:=w[i,j]+eta\*error[i]\*v[m,j];

 end;

 end;

 end;

 end;//if

 until stop;//end;

 end; // if

end;

procedure TFrmPerceptron.BtnOutputClick(Sender: TObject);

var z:array[1..InputLayerUnits] of shortint;

 m,i,j:byte;

 Output:array[1..InputLayerUnits] of real;

 sum:real;

begin

 for m:=1 to InputLayerUnits do begin

 if (Percept\_FieldPerc.Brushes[m]=Percept\_FieldPerc.RectBrush) then

 begin

 z[m]:=1;

 end

 else

 if (Percept\_FieldPerc.Brushes[m]=Percept\_FieldPerc.BackGroundBrush) then

 begin

 z[m]:=-1;

 end;

 end;

 for i := 1 to OutputLayerUnits do

 begin

 sum:=0;

 for j := 1 to InputLayerUnits do

 begin

 sum:=sum+w[i,j]\*z[j];

 end;

 Output[i]:=sum;

 end;

 if (Output[1]>Output[2]) then

 begin

 LabelOdd.Font.Color:=clRed;

 LabelEven.Font.Color:=clWindowText;

 end

 else begin

 if (Output[2]>Output[1]) then

 begin

 LabelEven.Font.Color:=clRed;

 LabelOdd.Font.Color:=clWindowText;

 end;

 end;

end;

end.

*Программа, моделирующая сеть обратного распространения*

unit UBack;

interface

uses

 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,

 StdCtrls, Percept\_Field, Buttons, ExtCtrls;

const FirstLayerUnits=35;

 SecondLayerUnits=20;

 ThirdLayerUnits=2;

 numberpatterns=36;

 NumLayers=3;

 epsilon=0.000001;

 eta=0.05;

 alpha=0.5;

type

 TFrmBack = class(TForm)

 BitBtnClose: TBitBtn;

 Percept\_FieldBack: TPercept\_Field;

 GroupBoxTrain: TGroupBox;

 ComboBoxABC: TComboBox;

 ComboBoxDigits: TComboBox;

 GroupBoxInit: TGroupBox;

 EditNumPat: TEdit;

 LabelNumPat: TLabel;

 BtnNext: TButton;

 GroupBoxRec: TGroupBox;

 LabelInput: TLabel;

 RadioGroupTarget: TRadioGroup;

 RadioButtonLetter: TRadioButton;

 RadioButtonFigure: TRadioButton;

 ButtonOut: TButton;

 LabelFigure: TLabel;

 LabelOr: TLabel;

 LabelLetter: TLabel;

 procedure BitBtnCloseClick(Sender: TObject);

 procedure ComboBoxABCChange(Sender: TObject);

 procedure ComboBoxDigitsChange(Sender: TObject);

 procedure Percept\_FieldBackMouseDown(Sender: TObject;

 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

 procedure FormCreate(Sender: TObject);

 procedure BtnNextClick(Sender: TObject);

 procedure ButtonOutClick(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 FrmBack: TFrmBack;

var

 wFirstSecond:array[1..SecondLayerUnits,1..FirstLayerUnits] of real;

 wSecondThird:array[1..ThirdLayerUnits,1..SecondLayerUnits] of real;

 indexBtnNextClick:byte;

 target:array[1..numberpatterns,1..ThirdLayerUnits] of real;

 v:array[1..numberpatterns,1..FirstLayerUnits] of real;

implementation

{$R \*.DFM}

procedure TFrmBack.BitBtnCloseClick(Sender: TObject);

begin

 Close;

end;

procedure TFrmBack.Percept\_FieldBackMouseDown(Sender: TObject;

 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

var m,k:BYTE;

 correctRect:shortint;

 L,T,H,V:INTEGER;

begin

 L:=0;

 T:=0;

 H:=Percept\_FieldBack.UnitHorizontal;

 V:=Percept\_FieldBack.UnitVertical;

 for m :=1 to Percept\_FieldBack.UnitRectVert do

 begin

 for k :=1 to Percept\_FieldBack.UnitRectHorz do

 begin

 if (X<H) and (X>L) and (Y<V) and (Y>T) then

 begin

 correctRect:=k+Percept\_FieldBack.UnitRectHorz\*(m-1);

 if (Button=mbLeft) and

 (Percept\_FieldBack.Brushes[correctRect]=Percept\_FieldBack.BackGroundBrush) then

 begin

 Percept\_FieldBack.Brushes[correctRect]:=Percept\_FieldBack.RectBrush;

 end

 else

 if (Button=mbRight) and

 (Percept\_FieldBack.Brushes[correctRect]=Percept\_FieldBack.RectBrush)then

 begin

 Percept\_FieldBack.Brushes[correctRect]:=Percept\_FieldBack.BackGroundBrush;

 end;

 end;

 inc(L,Percept\_FieldBack.UnitHorizontal);

 inc(H,Percept\_FieldBack.UnitHorizontal);

 end;

 inc(T,Percept\_FieldBack.UnitVertical);

 inc(V,Percept\_FieldBack.UnitVertical);

 L:=0;

 H:=Percept\_FieldBack.UnitHorizontal;

 end;

end;

procedure TFrmBack.FormCreate(Sender: TObject);

var i,j:byte;

 rand:real;

begin

 EditNumPat.Text:=inttostr(numberpatterns);

 BtnNext.Font.Color:=clRed;

 indexBtnNextClick:=0;

 LabelInput.Visible:=False;

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Randomize;// случайные веса (-0.5,0.5)

 for i := 1 to SecondLayerUnits do

 begin

 for j := 1 to FirstLayerUnits do

 begin

 rand:=Random-0.5;

 wFirstSecond[i,j]:=rand;

 end;

 end;

 for i := 1 to ThirdLayerUnits do

 begin

 for j := 1 to SecondLayerUnits do

 begin

 rand:=Random-0.5;

 wSecondThird[i,j]:=rand;

 end;

 end;

end;

procedure TFrmBack.BtnNextClick(Sender: TObject);

var i,j,m:byte;

 sumFirstSecond,

 sumSecondThird:real;

 stop:boolean;

 OutputSecond:array[1..SecondLayerUnits] of real;

 OutputThird:array[1..ThirdLayerUnits] of real;

 output,err,neterror:real;

 OutLayerError:array[1..ThirdLayerUnits] of real;

 SecondLayerError:array[1..SecondLayerUnits] of real;

 FirstLayerError:array[1..FirstLayerUnits] of real;

 dWeightSecondThird:array[1..ThirdLayerUnits,1..SecondLayerUnits] of real;

 dWeightFirstSecond:array[1..SecondLayerUnits,1..FirstLayerUnits] of real;

 dWeight:real;

 krandom:integer;

begin

 indexBtnNextClick:=indexBtnNextClick+1;

 for m:=1 to FirstLayerUnits do begin

 if (Percept\_FieldBack.Brushes[m]=Percept\_FieldBack.RectBrush) then

 begin

 v[indexBtnNextClick,m]:=1;

 end

 else

 if (Percept\_FieldBack.Brushes[m]=Percept\_FieldBack.BackGroundBrush) then

 begin

 v[indexBtnNextClick,m]:=-1;

 end;

 end;

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ODD or EVEN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if RadioButtonFigure.Checked then

 begin

 target[indexBtnNextClick,1]:=0.9;//1;

 target[indexBtnNextClick,2]:=0.1;//-1;

 end

 else

 if RadioButtonLetter.Checked then

 begin

 target[indexBtnNextClick,1]:=0.1;//-1;

 target[indexBtnNextClick,2]:=0.9;//1;

 end;

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if (indexBtnNextClick+1)=numberpatterns then

 begin

 BtnNext.Caption:='last';

 end

 else

 begin

 if (indexBtnNextClick)=numberpatterns then

 begin

 BtnNext.Font.Color:=clWindowText;

 BtnNext.Caption:='finished';

 LabelInput.Font.Color:=clRed;

 LabelInput.Visible:=True;

 end

 else

 begin

 BtnNext.Caption:='next';

 end;

 end;

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if (indexBtnNextClick)=numberpatterns then

 begin

 repeat

 stop:=false;

 for m := 1 to numberpatterns do

 begin

 for i := 1 to SecondLayerUnits do

 begin

 sumFirstSecond:=0;

 for j := 1 to FirstLayerUnits do

 begin

 sumFirstSecond:=sumFirstSecond+wFirstSecond[i,j]\*v[m,j];

 end;

 OutputSecond[i]:=1/(1+exp(-sumFirstSecond));

 end;

 for i := 1 to ThirdLayerUnits do

 begin

 sumSecondThird:=0;

 for j := 1 to SecondLayerUnits do

 begin

 sumSecondThird:=sumSecondThird+wSecondThird[i,j]\*OutputSecond[j];

 end;

 OutputThird[i]:=1/(1+exp(-sumSecondThird));

 end;

 neterror:=0;

 for i := 1 to ThirdLayerUnits do

 begin

 output:=OutputThird[i];

 err:=target[m,i]-output;

 OutLayerError[i]:=output\*(1-output)\*err;

 neterror:=neterror+0.5\*sqr(err);

 end;

 if neterror<epsilon then

 begin

 stop:=true;

 end;

 end;//\*\*\* for m:=..... \*\*\*\*\*\*

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*обучение\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 if not stop then

 begin

 Randomize;

 for krandom:=1 to 10\*numberpatterns do

 begin

 m:=1+Round(Random(numberpatterns));

 //\*\*\*\*\*\*\*\*\*\*\*PROPAGATION\*\*\*\*\*\*\*\*\*\*\*\*

 for i := 1 to SecondLayerUnits do

 begin

 sumFirstSecond:=0;

 for j := 1 to FirstLayerUnits do

 begin

 sumFirstSecond:=sumFirstSecond+wFirstSecond[i,j]\*v[m,j];

 end;

 OutputSecond[i]:=1/(1+exp(-sumFirstSecond));

 end;

 for i := 1 to ThirdLayerUnits do

 begin

 sumSecondThird:=0;

 for j := 1 to SecondLayerUnits do

 begin

 sumSecondThird:=sumSecondThird+wSecondThird[i,j]\*OutputSecond[j];

 end;

 OutputThird[i]:=1/(1+exp(-sumSecondThird));

 end;

 neterror:=0;

 for i := 1 to ThirdLayerUnits do

 begin

 output:=OutputThird[i];

 err:=target[m,i]-output;

 OutLayerError[i]:=output\*(1-output)\*err;

 neterror:=neterror+0.5\*sqr(err);

 end;

 //\*\*\*\*\*\*\*\*\*BACKPROPAGATION\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 for i := 1 to SecondLayerUnits do

 begin

 output:=OutputSecond[i];

 err:=0;

 for j := 1 to ThirdLayerUnits do

 begin

 err:=err+wSecondThird[j,i]\*OutLayerError[j];

 end;

 SecondLayerError[i]:=output\*(1-output)\*err;

 end;

 for i := 1 to FirstLayerUnits do

 begin

 output:=v[m,i];

 err:=0;

 for j := 1 to SecondLayerUnits do

 begin

 err:=err+wFirstSecond[j,i]\*SecondLayerError[j];

 end;

 FirstLayerError[i]:=output\*(1-output)\*err;

 end;

 //\*\*\*\*\*\*\*\*\*\*\*

 for i := 1 to SecondLayerUnits do

 begin

 for j := 1 to FirstLayerUnits do

 begin

 dWeightFirstSecond[i,j]:=0;

 end;

 end;

 for i := 1 to ThirdLayerUnits do

 begin

 for j := 1 to SecondLayerUnits do

 begin

 dWeightSecondThird[i,j]:=0;

 end;

 end;

 //\*\*\*\*\*\*\*\*\*\*\*

 dWeight:=0;

 for i := 1 to SecondLayerUnits do

 begin

 for j := 1 to FirstLayerUnits do

 begin

 output:=v[m,j];

 err:=SecondLayerError[i];

 dWeight:=dWeightFirstSecond[i,j];

 wFirstSecond[i,j]:=wFirstSecond[i,j]+eta\*err\*output+alpha\*dWeight;

 dWeightFirstSecond[i,j]:=eta\*err\*output;

 end;

 end;

 dWeight:=0;

 for i := 1 to ThirdLayerUnits do

 begin

 for j := 1 to SecondLayerUnits do

 begin

 output:=OutputSecond[j];

 err:=OutLayerError[i];

 dWeight:=dWeightSecondThird[i,j];

 wSecondThird[i,j]:=wSecondThird[i,j]+eta\*err\*output+alpha\*dWeight;

 dWeightSecondThird[i,j]:=eta\*err\*output;

 end;

 end;

 end;//\*\*\*\*for krandom:=.......\*\*\*\*\*\*\*\*\*\*\*

 end;

 until stop;

 end;//\*\*\* IF \*\*\*\*\*\*\*\*

end;

procedure TFrmBack.ButtonOutClick(Sender: TObject);

var m,i,j:byte;

 z:array[1..FirstLayerUnits] of shortint;

 sumFirstSecond,sumSecondThird:real;

 OutputSecond:array[1..SecondLayerUnits] of real;

 OutputThird:array[1..ThirdLayerUnits] of real;

begin

 for m:=1 to FirstLayerUnits do begin

 if (Percept\_FieldBack.Brushes[m]=Percept\_FieldBack.RectBrush) then

 begin

 z[m]:=1;

 end

 else

 if (Percept\_FieldBack.Brushes[m]=Percept\_FieldBack.BackGroundBrush) then

 begin

 z[m]:=-1;

 end;

 end;

 for i := 1 to SecondLayerUnits do

 begin

 sumFirstSecond:=0;

 for j := 1 to FirstLayerUnits do

 begin

 sumFirstSecond:=sumFirstSecond+wFirstSecond[i,j]\*z[j];

 end;

 OutputSecond[i]:=1/(1+exp(-sumFirstSecond));

 end;

 for i := 1 to ThirdLayerUnits do

 begin

 sumSecondThird:=0;

 for j := 1 to SecondLayerUnits do

 begin

 sumSecondThird:=sumSecondThird+wSecondThird[i,j]\*OutputSecond[j];

 end;

 OutputThird[i]:=1/(1+exp(-sumSecondThird));

 end;

 if (OutputThird[1]>OutputThird[2]) then

 begin

 LabelFigure.Font.Color:=clRed;

 LabelLetter.Font.Color:=clWindowText;

 end

 else begin

 if (OutputThird[2]>OutputThird[1]) then

 begin

 LabelLetter.Font.Color:=clRed;

 LabelFigure.Font.Color:=clWindowText;

 end;

 end;

end;

end.

*Программа, моделирующая сеть Хопфилда*

unit UHop;

interface

uses

 Windows, Messages, SysUtils,

 Classes, Graphics, Controls,

 Forms, Dialogs, StdCtrls, Buttons, Percept\_Field;

const numberneurons=35;

type

 TFrmHop = class(TForm)

 BitBtnClose: TBitBtn;

 GrpBoxTraining: TGroupBox;

 GrpBoxInitial: TGroupBox;

 EditThres: TEdit;

 EditNumPat: TEdit;

 LabelThres: TLabel;

 LabelNumPat: TLabel;

 BtnNext: TButton;

 GrpBoxRec: TGroupBox;

 LabelInput: TLabel;

 BtnOutput: TButton;

 BitBtnCancel: TBitBtn;

 ButtonDelay: TButton;

 ComboBoxABC: TComboBox;

 ComboBoxDigits: TComboBox;

 Percept\_FieldHop: TPercept\_Field;

 ButtonRetrain: TButton;

 procedure Percept\_FieldHopMouseDown(Sender: TObject;

 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

 procedure BitBtnCloseClick(Sender: TObject);

 procedure EditNumPatChange(Sender: TObject);

 procedure EditThresChange(Sender: TObject);

 procedure FormCreate(Sender: TObject);

 procedure BtnNextClick(Sender: TObject);

 procedure BtnOutputClick(Sender: TObject);

 procedure BitBtnCancelClick(Sender: TObject);

 procedure ButtonDelayClick(Sender: TObject);

 procedure ComboBoxABCChange(Sender: TObject);

 procedure ComboBoxDigitsChange(Sender: TObject);

 procedure ButtonRetrainClick(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 FrmHop: TFrmHop;

var numberpatterns,threshold:shortint;

 w:array[1..numberneurons,1..numberneurons] of shortint;

 iindex,jindex,indexBtnNextClick:byte;

 stop:boolean;

implementation

{$R \*.DFM}

procedure TFrmHop.Percept\_FieldHopMouseDown(Sender: TObject;

 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

var m,k:BYTE;

 correctRect:shortint;

 L,T,H,V:INTEGER;

begin

 L:=0;

 T:=0;

 H:=Percept\_FieldHop.UnitHorizontal;

 V:=Percept\_FieldHop.UnitVertical;

 for m :=1 to Percept\_FieldHop.UnitRectVert do

 begin

 for k :=1 to Percept\_FieldHop.UnitRectHorz do

 begin

 if (X<H) and (X>L) and (Y<V) and (Y>T) then

 begin

 correctRect:=k+Percept\_FieldHop.UnitRectHorz\*(m-1);

 if (Button=mbLeft) and

 (Percept\_FieldHop.Brushes[correctRect]=Percept\_FieldHop.BackGroundBrush) then

 begin

 Percept\_FieldHop.Brushes[correctRect]:=Percept\_FieldHop.RectBrush;

 end

 else

 if (Button=mbRight) and

 (Percept\_FieldHop.Brushes[correctRect]=Percept\_FieldHop.RectBrush)then

 begin

 Percept\_FieldHop.Brushes[correctRect]:=Percept\_FieldHop.BackGroundBrush;

 end;

 end;

 inc(L,Percept\_FieldHop.UnitHorizontal);

 inc(H,Percept\_FieldHop.UnitHorizontal);

 end;

 inc(T,Percept\_FieldHop.UnitVertical);

 inc(V,Percept\_FieldHop.UnitVertical);

 L:=0;

 H:=Percept\_FieldHop.UnitHorizontal;

 end;

end;

procedure TFrmHop.BitBtnCloseClick(Sender: TObject);

begin

 Close;

end;

procedure TFrmHop.EditThresChange(Sender: TObject);

begin

 threshold:=strtoint(EditThres.Text);

end;

procedure TFrmHop.EditNumPatChange(Sender: TObject);

begin

 numberpatterns:=strtoint(EditNumPat.Text);

end;

procedure TFrmHop.FormCreate(Sender: TObject);

var i,j:byte;

begin

 threshold:=0;

 EditThres.Text:=inttostr(threshold);

 numberpatterns:=3;

 EditNumPat.Text:=inttostr(numberpatterns);

 BtnNext.Font.Color:=clRed;

 for i:=1 to numberneurons do begin

 for j:=1 to numberneurons do begin

 w[i,j]:=0;

 end;

 end;

 indexBtnNextClick:=0;

 LabelInput.Visible:=False;

end;

procedure TFrmHop.BtnNextClick(Sender: TObject);

var i,j,m:byte;

 v:array[1..numberneurons] of shortint;

begin

 indexBtnNextClick:=indexBtnNextClick+1;

 for m:=1 to numberneurons do begin

 if (Percept\_FieldHop.Brushes[m]=Percept\_FieldHop.RectBrush) then

 begin

 v[m]:=1;

 end

 else

 if (Percept\_FieldHop.Brushes[m]=Percept\_FieldHop.BackGroundBrush) then

 begin

 v[m]:=0;

 end;

 end;

 for i:=1 to numberneurons-1 do begin

 for j:=i+1 to numberneurons do begin

 w[i,j]:=w[i,j]+(2\*v[i]-1)\*(2\*v[j]-1);

 w[j,i]:=w[i,j];

 end;

 end;

 if (indexBtnNextClick+1)=numberpatterns then

 begin

 BtnNext.Caption:='last';

 end

 else

 begin

 if (indexBtnNextClick)=numberpatterns then

 begin

 BtnNext.Font.Color:=clWindowText;

 BtnNext.Caption:='finished';

 LabelInput.Font.Color:=clRed;

 LabelInput.Visible:=True;

 end

 else

 begin

 BtnNext.Caption:='next';

 end;

 end;

end;

procedure TFrmHop.BtnOutputClick(Sender: TObject);

var i,j,m,indicator:byte;

 y,z:array[1..numberneurons] of shortint;

 wij,wijthres:shortint;

 k:longint;

begin

 for m:=1 to numberneurons do begin

 if (Percept\_FieldHop.Brushes[m]=Percept\_FieldHop.RectBrush) then

 begin

 z[m]:=1;

 end

 else

 if (Percept\_FieldHop.Brushes[m]=Percept\_FieldHop.BackGroundBrush) then

 begin

 z[m]:=0;

 end;

 end;

 for m := 1 to numberneurons do

 begin

 y[m]:=z[m];

 end;

 indicator:=0;

 while indicator=0 do begin

 for i:=1 to numberneurons do begin

 wij:=0;

 for j:=1 to numberneurons do begin

 if i<>j then

 wij:=wij+w[i,j]\*z[j];

 end;

 wijthres:=wij-threshold;

 if wijthres>=0 then

 z[i]:=1

 else

 z[i]:=0;

 end;

 i:=1;

 while i<=numberneurons do begin

 if z[i]=y[i] then begin

 y[i]:=z[i];

 indicator:=1;

 i:=i+1;

 end

 else begin

 indicator:=0;

 repeat

 y[i]:=z[i];

 i:=i+1;

 until i>numberneurons;

 end;

 end;{while}

 end;{while}

 for m := 1 to numberneurons do

 begin

 if z[m]=1 then

 begin

 Percept\_FieldHop.Brushes[m]:=Percept\_FieldHop.RectBrush;

 end

 else

 if z[m]=0 then

 begin

 Percept\_FieldHop.Brushes[m]:=Percept\_FieldHop.BackGroundBrush;

 end;

 stop:=false;

 repeat

 Application.ProcessMessages;

 until stop;

 end;

end;

procedure TFrmHop.BitBtnCancelClick(Sender: TObject);

var i,j:byte;

begin

 BtnNext.Font.Color:=clRed;

 for i:=1 to numberneurons do begin

 for j:=1 to numberneurons do begin

 w[i,j]:=0;

 end;

 end;

 indexBtnNextClick:=0;

 LabelInput.Visible:=False;

 BtnNext.Caption:='first';

 for i := 1 to numberneurons do

 begin

 Percept\_FieldHop.Brushes[i]:=Percept\_FieldHop.BackGroundBrush;

 end;

end;

procedure TFrmHop.ButtonDelayClick(Sender: TObject);

begin

 stop:=true;

end;

procedure TFrmHop.ButtonRetrainClick(Sender: TObject);

var i,j,m:byte;

 v:array[1..numberneurons] of shortint;

begin

 for m:=1 to numberneurons do begin

 if (Percept\_FieldHop.Brushes[m]=Percept\_FieldHop.RectBrush) then

 begin

 v[m]:=1;

 end

 else

 if (Percept\_FieldHop.Brushes[m]=Percept\_FieldHop.BackGroundBrush) then

 begin

 v[m]:=0;

 end;

 end;

 for i:=1 to numberneurons-1 do begin

 for j:=i+1 to numberneurons do begin

 w[i,j]:=w[i,j]-(2\*v[i]-1)\*(2\*v[j]-1);

 w[j,i]:=w[i,j];

 end;

 end;

end;

end.