Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ

Практикум

Владимир 2022

УДК 004
ББК 32.973
И73

Авторы-составители: М. И. Озерова, И. Е. Жигалов

Рецензенты:
Доктор технических наук, профессор
профессор кафедры вычислительной техники и систем управления
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
В. Н. Ланцов

Кандидат физико-математических наук, доцент
зав. кафедрой информационных технологий
Российской академии народного хозяйства и государственной службы
при Президенте Российской Федерации
(Владимирский филиал)
И. В. Сидорова

Издается по решению редакционно-издательского совета ВлГУ

Интеллектуальные технологии : практикум / авт.-сост.:
И73 М. И. Озерова, И. Е. Жигалов ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2022. – 128 с.
ISBN 978-5-9984-1518-0.

Знакомит магистрантов со средой выполнения MatLab, организацией линейной регрессии, классификацией данных, методами обучения без учителя. Рассмотрены основные подходы, методы и алгоритмы описания классов, нахождения решающих функций, выбора информативной системы признаков.

Предназначен для студентов магистратуры направлений подготовки 09.04.02 «Информационные системы и технологии» и 09.04.04 «Программная инженерия».

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Табл. 3. Ил. 44. Библиогр.: 38 назв.

УДК 004
ББК 32.973

ISBN 978-5-9984-1518-0

ВВЕДЕНИЕ

Искусственный интеллект (ИИ) (от англ. *Artificial intelligence*, *AI*) – раздел компьютерной лингвистики и информатики, занимающийся формализацией проблем и задач, которые напоминают задачи, выполняемые человеком. При этом в большинстве случаев алгоритм решения задачи неизвестен заранее. Точного определения этой науки нет, поскольку в философии не решен вопрос о природе и статусе человеческого интеллекта. Нет и точного критерия достижения компьютером «разумности». Для оценки уровня интеллектуальности машин был предложен ряд гипотез, например Тест Тьюринга и гипотеза Ньюэлла – Саймона. Сейчас существует множество подходов как к пониманию задач ИИ, так и к созданию интеллектуальных систем.

ИИ связан с психологией, нейрофизиологией, трансгуманизмом и другими науками. Как и все компьютерные науки, ИИ использует математический аппарат. Особое значение для него имеют философия и робототехника.

Термин ***интеллект*** (лат. *intellectus*) означает ум, рассудок, способность мышления и рационального познания. Обычно под интеллектом подразумевают способность приобретать, запоминать, применять и преобразовывать знания для решения каких-то задач. Благодаря этим качествам человеческий мозг способен решать разнообразные задачи, в том числе те, для которых нет заранее известных методов решения.

Термин «искусственный интеллект» возник сравнительно недавно, однако уже сейчас практически невозможно представить себе мир без него. Его исторический путь напоминает синусоиду, каждый «взлет» которой инициируется некоторой новой идеей. Чаще всего люди не замечают присутствия ИИ, но если бы вдруг его не стало, то это коренным образом отразилось бы на нашей жизни. Сферы, в которых используются технологии ИИ, постоянно пополняются: когда-то это были программы для игры в шахматы, потом роботы-пылесосы, сейчас алгоритмы способны сами проводить торги на биржах.

Само понятие «искусственный интеллект» образовалось на базе утверждения, что интеллект человека может быть детально описан и впоследствии успешно имитироваться машиной. ИИ был причиной огромного оптимизма ученых и исследователей, потому что под влиянием первых успехов казалось, что появление такой технологии решит множество задач. Однако вскоре ИИ показал ошеломляющую сложность реализации.

Основные направления развития ИИ включают в себя рассуждения, знания, планирование, обучение, языковую коммуникацию, восприятие и способность двигать объекты и манипулировать ими. Универсальный ИИ (или «сильный ИИ») все еще в планах на будущее. В настоящее время популярны подходы к развитию ИИ, которые включают в себя статистические методы вычислительного интеллекта и традиционной символической технологии ИИ. Существует огромное количество инструментов, использующих ИИ: разные версии поисковых алгоритмов; алгоритмы математической оптимизации, логики; методы, основанные на вероятности, и многие другие.

Цель дисциплины «Интеллектуальные технологии» – дать систематический обзор существующих интеллектуальных методов обработки информации, математических методов ИИ, а также способствовать освоению теоретических и практических знаний и навыков использования технологий для обработки информации.

Глава 1. СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

В настоящее время ИИ используется в самых разных областях человеческой деятельности, начиная от универсальных задач вроде обучения и восприятия и заканчивая конкретными задачами, например, в игре в шахматы, доказательстве математических теорем, написании стихотворений, а также диагностике заболеваний.

Что же такое ИИ? За 50 лет развития ИИ как научной области появились различные определения этой технологии, зачастую явно противоречащие друг другу. Эта путаница возникла в силу того, что под основное понятие можно подвести совершенно разные виды систем. Легче всего выделить четыре специальных класса, для которых применимо понятие ИИ:

- системы, которые рассуждают как люди;
- системы, которые рассуждают рационально;
- системы, которые действуют как люди;
- системы, которые действуют рационально.

Первые два типа систем концентрируются на вопросах мышления и рассуждения, тогда как другие два касаются поведения. За всю историю развития ИИ как науки ученые применяли все четыре подхода к определению ИИ. Особо стоит отметить слово «рационально» – оно подразумевает, что существует какая-то идеальная концепция разумности, которую мы и считаем рациональной. Рассмотрим по порядку все подходы с точки зрения формирования различных направлений ИИ.

1.1. Системы, действующие как человек, – тест Тюринга

В 1950 году английский математик Алан Тюринг в статье под названием «Может ли машина мыслить?» описал процедуру, позволяющую определить момент, когда машина сравняется в плане разумности с человеком. Позже этот тест получил название теста Тьюринга. Чтобы вести диалог, согласно этому тесту система должна обладать следующими способностями:

1) обработкой текстов на естественном языке для успешного ведения диалога;

2) представлением знаний для хранения текста, а также дополнительных сведений о контексте диалога;

3) автоматизированным механизмом рассуждений, который на основе имеющейся информации позволяет делать умозаключения, а также задавать наводящие вопросы;

4) машинным обучением для адаптации к новым обстоятельствам, а также обнаружению и экстраполяции шаблонов.

Оригинальный тест Тюринга специально исключил физическое взаимодействие человека и машины, поскольку физическая симуляция не является необходимым признаком разумности. Однако существует также полный тест Тюринга, включающий в себя видеосигналы и способность физического взаимодействия с системой;

5) компьютерным зрением для восприятия объектов;

6) манипулированием объектами как человек, что может позволить только развитие робототехники.

Эти шесть способностей по большей части и являются проблемами ИИ, и надо отдать должное Тюрингу за создание теста, который сформулировал все эти вопросы, причем на заре развития ИИ. Тем не менее этот тест на протяжении всей своей истории сталкивался с весьма серьезной критикой. Главный аргумент критиков заключается в том, что история науки и техники свидетельствует о неэффективности попыток имитировать живую природу при попытке построения технической системы. Веками люди пытались создать систему крыльев, которая позволила бы им летать подобно птицам. Все эти эксперименты потерпели неудачу. Летать человек стал после того, как отказался от попыток подражать природе и пошел своим путем. Сначала через конструирование воздушных шаров и дирижаблей, а затем – летательных аппаратов тяжелее воздуха и с искусственными двигателями. Ни в одном тексте, посвященном самолетостроению и вопросам аэродинамики, вы не встретите задачи: «Сделать самолет, который бы летал, как голуби, и делал это так искусно, что ввел бы в заблуждение других голубей». Сама постановка задачи в таком стиле вызвала бы у авиаконструкторов удивление.

1.2. Системы, мыслящие как человек, – когнитивные науки

Чтобы реализовать такого рода системы, надо разобраться с тем, как мыслит сам человек. Этого можно достичь либо интроспекцией, либо с помощью психологических экспериментов. Как только появится достаточно точная теория мыслительных процессов, ее можно будет формализовать и выразить в виде программы.

Например, Г. Саймон и А. Ньюэлл, создавшие универсальный решатель задач, не были удовлетворены тем, что он просто решал задачи. Их цель заключалась в сравнении процесса рассуждения этой программы с процессом рассуждения человека при решении задачи. Их универсальный решатель задач *General problem solver (GPS)* был ценен не столько своими практическими результатами в области доказательства теорем, сколько новаторским подходом, оказавшимся весьма плодотворным. На заре существования ИИ часто смешивались два подхода: модели ИИ для компьютеров и экспериментальные методы психологии. Автор в статье тех времен мог заявлять, что алгоритм, который хорошо себя показывает при решении определенной задачи, является адекватной моделью для рассуждения человека над этой задачей, и наоборот. В настоящий момент принято разделять эти две области, что позволяет им развиваться куда быстрее, чем во времена смешения. Правда в области компьютерного зрения и естественных языков эти две области по-прежнему серьезно взаимодействуют. Здесь основная проблема та же, что и для теста Тюринга: мы пытаемся построить систему, мыслящую так же, как и человек. Опыт науки и техники подсказывает, что с куда большей вероятностью удастся добиться успеха при построении разумных систем, думающих и принимающих решение отлично от человека.

1.3. Системы, рассуждающие рационально, – законы логики

Еще Аристотель предпринял попытку зафиксировать правила «правильных рассуждений». Его силлогизмы представляли собой шаблоны для рассуждений, которые всегда приводили к истиным заключениям, если их посылки были верны. Эти законы мышления и послужили основанием для такой науки, как логика. Один из известнейших силлогизмов звучит следующим образом.

Большая посылка. Все люди смертны.

Малая посылка. Сократ – человек.

Заключение. Сократ смертен.

В XIX веке логики выработали нотации для обозначения практически всех вещей и взаимоотношений между ними. К 1965 году уже были разработаны программы, которые в теории могли решить любую разрешимую задачу, описанную в логической нотации. Однако на этом пути встретилось два препятствия. Во-первых, не всегда легко из неформального описания задачи получить формальное математическое представление (возьмите поваренные рецепты), особенно если знание не является на 100 % достоверным. Во-вторых, существует огромная пропасть между «можно решить в принципе» и «можно решить на практике за приемлемые время и цену». С. А. Кук в своей работе *The Compexity of Theorem Proving Procedures*, заложившей основные понятия *NP*-полных задач, заострил этот вопрос до предела. Кук доказал, что задача выполнимости булевых формул является *NP*-полной. Тем самым он поднял вопрос о равенстве классов сложности *P* и *NP* – один из сложнейших вопросов теории вычислительных систем, на который до сих пор нет ответа. И хотя эти препятствия существуют в любой системе ИИ, наиболее драматично и ярко они проявились в системах на основе логических правил.

1.4. Системы, ведущие себя рационально, – интеллектуальные агенты

Агент – это что-то, что действует (от лат. *agere* – делать что-то). Агент имеет ряд существенных отличий от обычных программ:

- способен действовать автономно;
- способен воспринимать внешнее окружение;
- способен адаптироваться к изменениям;
- способен следовать поставленной цели.

В предыдущем подходе упор делался на корректных (с точки зрения логики) рассуждениях. Иногда такие рассуждения являются частью агента. С другой стороны, логических рассуждений недостаточно, чтобы действовать рационально. Есть рациональные действия,

которые вообще не используют логических рассуждений (например, рефлексы). Зачастую такой подход оказывается эффективнее с точки зрения рациональности поведения. Это легко показать на следующем примере. Предположим, что вы дотронулись до раскаленного чайника рукой. Измерения показали, что если бы сигнал от кожи руки прошел по нервной системе до мозга, мозг проанализировал эту информацию и отдал приказ отдернуть руку, то итоговое время составило бы примерно 1 с – вполне достаточно, чтобы получить серьезный ожог. Однако большинство людей успевают отдернуть руку от чайника где-то за 0,3 с, используя выработанные в ходе жизни рефлексы.

Глава 2. ОСНОВАНИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

2.1. Философия

Философия пытается ответить на следующие вопросы.

- Можно ли использовать формальные правила, для того чтобы прийти к корректным заключениям?
- Откуда берется разум в человеческом мозге?
- Каков источник наших знаний?
- Как знание приводит к действию?

Как упоминалось выше, Аристотель был первым, кто описал законы рационального поведения. Он создал неформальную систему силлогизмов. Позднее христианский философ Раймунд Луллий высказал идею, что полезные для нас рассуждения могут возникать чисто механически. Он построил специальную машину для доказательства бытия Божия. Автоматизация вычислений существовала уже в Средние века: Леонардо да Винчи предложил конструкцию механического калькулятора. Хотя он его так и не построил, современные реконструкторы подтвердили верность его чертежей, сконструировав по ним действующий калькулятор.

Рене Декарт был сторонником дуализма. Он считал, что часть нашего разума (или души, или духа) внеприродного происхождения и не зависит от чисто физических особенностей мозга. Он считал, что у животных такого нет. Альтернатива дуализму – материализм (отсюда

дискуссии о свободе воли). Что касается источника знания, то здесь определяющим для науки является движение эмпиризма, основоположник которого – Фрэнсис Бэкон.

Английский философ Дэвид Юм предложил принцип индукции: общие правила могут быть выведены из наблюдения над повторяющимися связями между элементами. Юм утверждал, что индуктивное умозаключение и вера в причинность не могут быть обоснованы рационально; вместо этого они являются результатом обычаев и умственных привычек. Он не был согласен с представлением о существовании врожденных идей. В начале XX века на основе учения Юма возникло движение логического позитивизма: все наши знания могут быть охарактеризованы логическими теориями, связанными с наблюдениями и фактами, которые мы воспринимаем с помощью сенсоров.

Наконец, уже Аристотель задумывался над вопросами, как знание приводит к действию, различая концепцию целей и средств (цель – подцели – действия). Эта концепция была с успехом воплощена в одной из ранних систем ИИ – *General Purpose Solver*.

2.2. Математика

Применительно к ИИ математика пытается ответить на следующие вопросы.

- Каковы формальные правила для вывода корректных умозаключений?
- Что можно вычислить?
- Что делать с недостоверной информацией?

Математическая логика связана с работами английского математика Д. Буля. В 1879 году немецкий логик Г. Фреге расширил алгебру логики, описанную в работах Буля, включив в нее объекты и отношения и тем самым создав логику первого порядка, которая и лежит в основе представлений знаний большинства современных систем. Фреге и Пирс ввели в язык алгебры логики предикаты, предметные переменные индустрии и кванторы, что позволило строить системы логики в виде логического исчисления.

Развитие компьютерной индустрии привело к появлению огромного количества алгоритмов. Однако ряд алгоритмов возник задолго до появления компьютеров. Самый древний из известных – это алгоритм

Евклида, позволяющий найти наибольший общий делитель для двух чисел. В начале XX века немецкий математик Давид Гильберт поставил вопрос о том, является ли система аксиом арифметики логически непротиворечивой. Б. Рассел и А. Н. Уайтхед в своей работе «Основания математики» стремились показать с помощью набора аксиом и нескольких основных понятий, что вся математика сводится к логике. Если бы удалось выявить такой набор аксиом, то, используя стандартные правила, можно было бы получать новые знания чисто механическим способом. Эти попытки перечеркнул австрийский математик Курт Гёдель своей знаменитой теоремой о неполноте. Теорема представляет собой формулировку и доказательства двух теорем математической логики о принципиальных ограничениях формальной арифметики и, как следствие, всякой формальной системы. Теорема известна в нескольких формулировках. В контексте ИИ представляют интерес следующие формулировки его теорем.

1. Если формальная логика непротиворечива, то в ней существует невыводимая и неопровержимая формула.

2. Если формальная логика непротиворечива, то в ней невыводима некоторая формула, содержательно утверждающая непротиворечивость этой арифметики.

3. Всякая достаточно сильная рекурсивно аксиоматизируемая непротиворечивая теории первого порядка неполна (обобщенная формулировка).

Гёдель показал, что в логике первого порядка (которая чаще всего используется в интеллектуальных системах) невозможно описать принцип индукции – один из наиболее распространенных способов доказательства утверждений в математике. Более того, существуют функции над целыми числами, которые могут быть однозначно описаны (т. е. для каждого входного значения они выдают вполне конкретное выходное значение), но которые при этом невозможно описать в виде алгоритма.

Американский ученый Стивен Кук в своей работе *The Complexity of Theorem* в 1971 году описал класс *NP*-полных задач и показал, что существуют задачи, решение которых по времени либо по памяти будет выражаться экспоненциальной зависимостью от размера задачи.

Он же поставил вопрос о том, можно ли любую задачу подобного рода свести к задаче, решаемой за полиномиальное время (знаменитая формула $P \vDash NP$). На этот вопрос ответа пока что найти не удалось, однако большинство специалистов в данной области склоняются к тому, что $P \vDash NP$, т. е. всегда будут существовать задачи, решение которых быстро найти не удастся (в связи с комбинаторным взрывом). В статье *Reasons to believe*[1] один из ведущих специалистов в данной области Скотт Ааронсон привел ряд серьезных аргументов, почему на практике наверняка $P \vDash NP$. Это показывает ограниченность как человеческого, так и всех искусственных реализаций интеллекта. Всегда будут существовать сложные задачи, решение которых будет представлять проблему даже для самого совершенного интеллекта.

2.3. Экономика

Применительно к ИИ экономика пытается ответить на следующие вопросы:

- Как принимать решения, чтобы максимизировать прибыль?
- Как это делать, когда другие мешают?
- Что делать, если прибыль можно будет получить лишь в отдаленном будущем?

Уже Адам Смит, заложивший основы современной политэкономии, рассматривал понятие предпочтительного результата и полезности действий экономических агентов. Дж. фон Нейман и О. Моргенштерн в труде «Теория игр и экономическое поведение» (1944) рассматривали интеллектуальных агентов для нахождения такой стратегии поведения, которая позволит получить максимальный выигрыш или добиться минимального ущерба, если выигрыш невозможен. Чаще всего в реальном мире агенты действуют, не имея точной информации, и им нужно выбрать на очередном шаге не то действие, которое приносит максимальный локальный выигрыш (на такой основе действуют так называемые жадные алгоритмы), агент должен подобрать последо-

[1] Reasons to believe [Электронный ресурс]. URL: https://www.scottaaronson.com/blog/?p=122 (дата обращения: 15.09.2021).

вательность операций, которые приведут в конечном итоге к максимальному выигрышу, с учетом того, что каждый отдельный шаг может вполне вести к проигрышу. Особый интерес в рамках таких исследований представляют марковские процессы.

2.4. Нейробиология

Применительно к ИИ нейробиология пытается ответить на следующий вопрос.

- Как мозг обрабатывает информацию?

На вопрос о роли мозга в рассуждениях и при принятии решений значительно влияла полемика представителей двух философских направлений – дуалистов и материалистов. Дуалисты считали, что в живом человеке есть некоторая часть (возможно, душа) сознания и подсознания, которая существует независимо от материи и не исчезает после смерти. Материалисты отвергали это предположение, считая, что все когнитивные процессы должны иметь какое-то материальное воплощение.

Что точно было известно насчет мозга, это его влияние на характер. Это было ясно из того факта, что у людей, получивших травмы головы в той или иной области, зачастую кардинально менялся характер поведения, а также уровень интеллекта. Окончательно роль мозга стала ясна, когда французский антрополог П. Брока́ открыл в мозге специальную область, ответственную за речевой аппарат (так называемая зона Брока́). Это положило начало бурным исследованиям мозговой деятельности. Немецкий психиатр Г. Бергер в 1929 году разработал новый метод исследования – ЭЭГ мозга. Работы японского ученого-биофизика С. Огавы привели в 1990 году к появлению нового метода исследования мозга – МРТ. В результате этих исследований стало ясно, что большое количество клеток в мозге, объединенных сложными взаимосвязями, позволяет создавать весьма сложные когнитивные процессы. Эта точка зрения подкрепляется таким направлением ИИ, как нейронные сети. Интересно сравнить развитие элементной базы у компьютеров и людей на данный момент (рис. 1.1).

	Computer	Human Brain
Comp. units	1 CPU, 10^8 gates	10^{11} neurons
Stor. units	10^{10} bits RAM	10^{11} neurons
	10^{11} bits disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-3} sec
Bandwidth	10^{10} bits/sec	10^{14} bits/sec
Memory updates/sec	10^9	10^{14}

Рис. 1.1. Сравнение людей и компьютеров

Следует учитывать, что согласно закону Мура каждые несколько лет производительность вычислительных систем растет. По количеству элементарных ячеек компьютеры вскоре превзойдут человеческий мозг (уже почти превзошли по ряду показателей). Однако у мозга есть одно неоспоримое преимущество в том плане, что множество нейронов могут менять свое состояние одновременно, передавая сигналы изменения практически всем элементам сети мозга. У компьютеров с таким параллелизмом пока что серьезные проблемы (чаще всего в каждый момент времени компьютер может изменить состояние одной или нескольких ячеек памяти, а отнюдь не все сразу).

2.5. Психология

Применительно к ИИ психология пытается ответить на следующие вопросы.

- Как люди и животные мыслят и действуют?
- Всегда ли мыслительный процесс предшествует действию?

Начало экспериментальной психологии (опыты ставились на крысах и птицах) положил немецкий физик, врач Г. фон Гельмгольц со своей лабораторией в Лейпциге. К середине XX века в психологии стало доминировать такое направление, как *бихевиоризм*, которое рассматривало человека как сложную машину. Это направление, по сути, отвергало мыслительные конструкции, рассматривая человека как систему «стимул – ответ». Этот подход оказался весьма плодотворным при описании обычных млекопитающих (например, крыс), однако к человеку и нашим собратьям из семейства африканских человекообразных обезьян он оказался малоприменим в силу наличия у них способности к мыслительным процессам.

2.6. Компьютерная инженерия

Существование ИИ без компьютеров хотя и возможно в теории, однако на практике встречается крайне редко.

Применительно к ИИ компьютерная инженерия пытается ответить на следующие вопросы.

- Как построить эффективный компьютер?
- Какой язык программирования будет наиболее эффективен при построении ИИ?

В ИИ есть множество концепций, изначально появившихся в других областях компьютерной инженерии.

Перечислим кратко некоторые из них:

- концепция разделения времени, позволяющая множеству пользователей одновременно общаться с одним приложением;
- интерактивные интепретаторы – *LISP* задал тон в этом направлении на несколько десятилетий вперед;
- списочные типы – опять же *LISP*;
- управление памятью в автоматическом режиме (сборщики мусора) – снова *LISP*;
- символьное программирование;
- функциональное программирование;
- объектно-ориентированное программирование.

2.7. Теория контроля и кибернетика

Применительно к ИИ теория контроля пытается ответить на следующий вопрос.

- Как могут технические устройства сами себя контролировать и менять свое поведение?

Древнегреческий изобретатель Кесибий Александрийский создал водяные часы с регулятором, который позволял часам держать поток воды на стабильном уровне, необходимом для их исправного функционирования. Часы могли понижать или повышать скорость водного потока. Фактически это первое известное устройство, кото-

рое могло само контролировать свое поведение и изменять его произвольно, без контроля со стороны человека. В начале XVII века нидерландский изобретатель К. Дреббель создал термостат. В 1784 году шотландский инженер Джеймс Уатт создал паровой двигатель с системами саморегуляции. В XIX веке бурно развивалась теория стабильных систем с обратной связью. Эта область оказалась достаточно важной для последующего развития ИИ. Из этой теории благодаря трудам американского математика Норберта Винера выросла теория кибернетики.

Следует заметить, что между теорией контроля и ИИ есть много общего, однако гораздо больше различий. Таким образом, теория контроля позволяет создавать устройства со сложным поведением, но это поведение заложено на стадии проектирования, т. е. определено заранее. Устройство не может изменить свое поведение самостоятельно, если возникла некоторая непредвиденная ситуация. Оно неспособно в большинстве случаев объяснить свое поведение. Наконец, в такого рода системах отсутствуют какие-либо мыслительные процессы (или их имитация).

2.8. Лингвистика

Применительно к ИИ лингвистика пытается ответить на следующий вопрос.

- Как связаны язык и мышление?

Основы вербального бихевиоризма заложил американский психолог Б. Скиннер. Интересны в этом плане не столько сами исследования Скиннера, сколько рецензия на них за авторством Ноама Хомского «Синтактические структуры». Хомский показал, как ребенок может понимать и создавать предложения, которые он никогда раньше не слышал. Эта теория была достаточно формальна для того, чтобы ее можно было записать в виде программы. *Ключевая проблема лингвистики в контексте ИИ – гипотеза о том, что любой интеллект должен интерпретировать данные в символьной форме.* Решение данной проблемы пока не найдено.

Глава 3. ИСТОРИЯ РАЗВИТИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

3.1. Зачатки искусственного интеллекта (1943 – 1955)

Первый общепризнанный труд в области ИИ – работа У. С. Мак-Каллока и В. Питтса под названием «Логическое исчисление идей, относящихся к нервной активности» (1943). Ученые опирались в своей работе на три основных источника:

- базовые сведения о физиологии и функционировании нейронов в человеческом мозге;
- формальный анализ пропозициональной логики, проведенный Б. Расселом и А. Н. Уайтхедом;
- теория вычислений, предложенная А. Тюрингом.

В своей работе У. С. Мак-Каллок и В. Питтс предложили модель искусственного нейрона, где каждый нейрон мог быть в состояниях *on* и *off*. Переключение в режим *on* происходило в результате воздействия достаточного количества стимулов от соседних нейронов. Авторы работы также показали, что любая вычисляемая функция может быть описана с помощью некоторой сети взаимосвязанных нейронов, и что все логические операции (И, ИЛИ, НЕ и т. д.) могут быть представлены в виде стандартных нейронных структур. Авторы также высказали предположение, что достаточно сложная и правильно сконструированная нейронная сеть способна к обучению. Канадский физиолог и нейропсихолог Дональд Хебб в своей работе *The Organization of Behavior: A Neuropsychological theory* (1949) предложил простое правило для обновления весов связей между нейронами.

- **Первое правило Хебба:** если сигнал персептрона неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.
- **Второе правило Хебба:** если сигнал персептрона неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.

Правило Хебба по-прежнему остается актуальным.

Два выпускника Принстона – М. Минский и Д. Эдмондс – сумели построить первую нейронную сеть в 1951 году. Эта сеть использовала

3 тыс. ламп и специальный механизм автопилота бомбардировщика B-24 для моделирования сети, состоящей из 40 нейронов. Однако аттестационная комиссия, которой Минский представил свою магистерскую диссертацию, скептично отнеслась к работе как к исследованию в области математики. Положение спас Джон фон Нейман, который заявил: «Если сейчас она не является таковой, то станет потом». Впоследствии Минский доказал ряд важных теорем, которые свидетельствовали об определенном наборе ограничений, присущих нейронным сетям.

Самым важным прорывом в теории в этот период следует считать статью «Вычислительные машины и разум» Тюринга (1950), в которой он предложил знаменитый тест, названный позже его именем. Помимо этого, в работе были предложены такие концепции, как машинное обучение, генетические алгоритмы и обучение с подкреплением.

3.2. Рождение искусственного интеллекта

Год рождения ИИ – 1956-й. Не отрицая важности предшествующего периода, именно 1956 год следует считать рубежом, выделившим ИИ в отдельную область. В этом году американский информатик Джон Маккарти (одна из наиболее влиятельных фигур в этой области и автор языка *LISP*) убедил М. Минского, К. Шеннона (создателя первой программы для игры в шахматы) и Н. Рочестера собрать наиболее авторитетных исследователей США в области ИИ вместе. Летом 1956 года был организован двухмесячный семинар в Дартмуте.

Именно на этом семинаре два исследователя из Карнеги – А. Ньюэлл и Г. Саймон – представили действительно работающую программу, способную доказывать математические теоремы. Вскоре после семинара эта программа сумела найти доказательства большинства теорем, упомянутых во второй главе «Принципов математики» Б. Рассела и Н. Уайтхеда. Особое впечатление на самого Рассела произвела демонстрация доказательства одной из теорем, найденного программой, которое оказалось короче доказательства, найденного людьми. Однако редакторы одного из самых авторитетных американских журналов *Journal of symbolic logic* не были так впечатлены. Они отказали в публикации статьи, где в качестве авторов были указаны Ньюэлл, Саймон

и их программа (программа была указана в качестве основного автора). Хотя сам по себе семинар в Дартмуте не привел к научным прорывам, он был чрезвычайно важен для организации научной деятельности.

В последующие 20 лет прогресс в области ИИ определялся в основном работами участников семинара и их научных школ. Именно на этом семинаре для этой научной области было принято предложенное Д. Маккарти определение «искусственный интеллект».

3.3. Время больших ожиданий (1956 – 1969)

Успех Ньюэлла и Саймона получил свое развитие в программе, названной *General Problem Solver* (*GPS*). В отличие от программы, представленной в Дартмуте, *GPS* с самого начала пыталась имитировать способы рассуждения, которые использует человек. Для решения задач (*problem-solving system*), в которых общие методы решения были отделены от знаний, определяющих конкретную задачу, подобный подход оказался крайне плодотворным. Успех *GPS* и других программ того периода привел к тому, что Ньюэлл и Саймон сформулировали знаменитую символьную гипотезу, которая гласила, что «физическая система представления символов является необходимой и достаточной для разумных действий в общем случае». Иначе говоря, они имели в виду, что любая система (машина или живое существо), чтобы действовать разумно, должна манипулировать последовательностями символов.

Тем временем в *IBM* был создан ряд важных программ в области ИИ. Дэвид Гелернтер (1959) создал программу *Geometry Theorem Prover,* которая сумела доказать ряд теорем, представляющих сложность для большинства студентов физико-математических факультетов. Начиная с 1952 года Артур Сэмюэл написал ряд программ для игры в шашки, постепенно доведя их до уровня профессионалов. В этой разработке он сумел опровергнуть устойчивое убеждение, что программы могут делать только то, что заложит в них создатель, поскольку его программы достаточно быстро научились играть сильнее, чем он сам.

В 1958 году Маккарти перебрался из Дартмута в Массачусетский технологический институт (*MIT*) и сделал три важных вклада в сфере ИИ.

• Он разработал высокоуровневый язык *LISP*, который стал доминирующим в ИИ на несколько десятилетий. Это был второй высокоуровневый язык (первым был *FORTRAN*).

• Столкнувшись с жесткой нехваткой вычислительных ресурсов в институте, Маккарти и его коллеги, страдавшие от той же проблемы, придумали концепцию разделения времени, которая оказалась исключительно важной для компьютерной индустрии в целом.

• Наконец в том же году он опубликовал статью, в которой была описана *Advice Taker* – первая полноценная программа в области ИИ. Подобно предшественникам, она должна была находить решения определенных задач. Однако ключевая особенность программы в том, что она должна была содержать лишь самые общие знания о мире. К примеру, программа могла составить маршрут поездки из дома до аэропорта путем использования нескольких простейших аксиом. Программа также поддерживала добавление новых аксиом для решения требуемой задачи, и поэтому могла быть использована для новых задач без переписывания исходного текста программ.

В 1958 году Марвин Минский также перебрался в *MIT*. Между ним и Маккарти возникли серьезные разногласия. Если Маккарти постоянно делал упор на символьных структурах и формальной логике, то Минского скорее интересовало, как создать работоспособную программу, способную решить действительно важную задачу. Поэтому Минский со своими учениками сосредоточился на создании программ, способных проявлять рациональность для задач из узких областей; эти области были названы *микромирами*. В решении задач из таких областей удалось добиться впечатляющих успехов. Программа *Saint* (1963) могла решать задачи интегрирования, которые студенты решают на первом курсе. Программа *Analogy* (1968) решала различные геометрические задачи из тестов на *IQ*. Программа *Student* (1967) за авторством Дэниела Боброва решала алгебраические задачи, записанные в следующей форме.

If the number of customers Tom gets twice the square of 20 percent of the number of advertisements he runs, and the number of advertisements he runs is 45, what is the number of customers Tom gets?

Здесь специально приводится оригинал задачи на английском, поскольку русский вариант представляет дополнительные сложности в силу специфики нашего языка (наличие окончаний в глаголах и т. д.).

Не стоит забывать и про нейронные сети. Работа С. Винограда и Дж. Коуэна «Надежные вычисления при наличии шумов» (1963) продемонстрировала, что большое количество базовых блоков сети могут представлять отдельные концепции. Правило Хебба, которое называют Дельта-правилом, описывает метод обучения персептрона по принципу градиентного спуска по поверхности ошибки. В 1960 году профессор Стэнфордского университета Бернард Уидроу вместе со своим аспирантом Тедом Хоффом на основе правила Хебба разработал новое правило обучения нейронных сетей. В 1962 году Ф. Розенблатт представил концепцию персептронов. Также Розенблатт доказал теорему о сходимости персептронов. Суть теоремы состоит в том, что алгоритм обучения (предложенный им в этой же работе) может настроить веса связей между нейронами для любых входных данных, если подходящий набор весов для этого набора вообще существует.

3.4. Охлаждение и возврат к реальности (1966 – 1973)

Предыдущий период был временем, когда исследователи не стеснялись делать очень смелые прогнозы в области ИИ. Например, Г. Саймон в 1957 году предсказал, что в течение 10 лет компьютеры смогут выиграть чемпионат мира по шахматам, а также доказать серьезные математические теоремы, с которыми не удалось справиться людям. Вместо 10 лет на это ушло примерно 40 лет.

Первые тревожные признаки того, что ИИ не может справиться со всеми поставленными задачами, возникли при попытке построить программы, способные переводить текст. Изначально работы в этой области спонсировали военные, которые хотели ускорить перевод русских научных работ после запуска спутника в СССР в 1957 году. Предполагалось, что простые синтаксические преобразования между русской и английской грамматикой, а также замена слов с помощью элементарных словарей, имеющихся в памяти программы, будут вполне

достаточными инструментами для полноценного перевода предложений с сохранением исходного смысла. Однако на поверку оказалось, что для такого перевода система должна располагать хотя бы минимальными знаниями о той конкретной профессиональной области, к которой относится статья, для разрешения неоднозначностей и обработки контекста статьи. Знаменитый перевод машинами библейской фразы на английском *The spirit is willing but the flesh is weak* на русский язык («Водка хорошая, а вот закуска протухла») наглядно демонстрировал всю сложность поставленной задачи. В 1966 году специальный отчет военного ведомства США констатировал, что нет ни одной системы, способной переводить научные статьи в целом, и нет даже ни одного перспективного прототипа. Практически все разработки в данном направлении были немедленно свернуты министерством обороны США.

Вторая проблема, с которой столкнулись ранние программы ИИ, – комбинаторный взрыв. Большинство программ тех времен работали путем перебора различных комбинаций в надежде найти решение. Это работало для микромиров из-за достаточно ограниченного пространства возможных комбинаций. Однако развитие теории вычислительной сложности показало, что существуют задачи, где нельзя найти решение путем перебора, даже если наращивать память и скорость оборудования (*NP*-полные задачи), поскольку время для решения растет экспоненциально размеру задачи.

Третья проблема, с которой столкнулись ранние программы ИИ, заключалась в фундаментальных ограничениях, присущих тем структурам, которыми пытались оперировать данные программы. Например, М. Минский и С. Пейперт в книге «Персептроны» (1969), проанализировав принципиальные возможности персептронов, пришли к выводу, что представить с их помощью можно очень немногое. Более конкретно: персептрон с двумя входами нельзя обучить распознавать ситуацию, когда значения входов не совпадают. Однако это не распространялось на более сложные многослойные сети, хотя прогресс в данном направлении остановился до открытия алгоритмов обучения сетей с помощью обратной связи.

Глава 4. ОСНОВНЫЕ ПОНЯТИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Можно выделить следующие наиболее важные направления развития интеллектуальных систем (т. е. систем, решающих задачи, традиционно относимые к интеллектуальной сфере), в которых широко используются методы распознавания образов:

- распознавание символов (печатного и рукописного текстов, банковских чеков и денежных купюр и т. д.);
- распознавание изображений, полученных в различных частотных диапазонах (оптическом, инфракрасном, радиочастотном, звуковом) и анализ сцен;
- распознавание речи;
- медицинская диагностика;
- использование систем безопасности;
- классификация, кластеризация и поиск в базах данных и знаний (в том числе и в интернет-ресурсах).

Для системы обработки информации *образ* – это совокупность данных об объекте или явлении, включающая параметры и связи этих данных.

Образ – это описание объекта или процесса, позволяющее выделять его из окружающей среды и группировать с другими объектами или процессами для принятия необходимых решений.

Цель процедуры распознавания (классификации) – получить ответ на вопрос: относится ли объект, описанный заданными характеристиками, к интересующим нас категориям (классам) и если относится, то к какой именно.

Классы – это категории объектов, которые должны быть выделены или на которые необходимо разделить все множество образов в процессе распознавания.

При распознавании изображений существенное влияние на точность оказывают:

- *масштаб.* Изображения имеют разный масштаб. Предметы, которые мы воспринимаем как одинаковые, на самом деле занимают разную площадь на разных изображениях;

• *место*. Интересующий нас объект может находиться в разных местах изображения;

• *фон и помехи*. Предмет, который мы воспринимаем как что-то отдельное, на изображении никак не выделен и находится на фоне других предметов. Кроме того, изображение не идеально и может быть подвержено всякого рода искажениям и помехам;

• *проекция, вращение и угол обзора*. Изображение является лишь двумерной проекцией нашего трехмерного мира. Поэтому поворот объекта и изменение угла обзора кардинальным образом влияют на его двумерную проекцию – изображение. Один и тот же объект может давать совершенно разную картинку в зависимости от поворота или расстояния до него;

• *анализ образа* – отсутствует качественное продвижение в решении таких задач, как анализ трехмерных сцен и перевод с одного языка на другой.

При сравнении зрительного образа с эталоном в зависимости от эталона можно выделить следующие методы.

1. *Растровые* – в виде матрицы точек; каждая точка обладает яркостью разной силы (0,1). Важно, чтобы входное изображение и эталон были приведены к одинаковым размерам и т. д.

Недостатки:

– необходимость выделять объекты из общего изображения;

– чувствительность к размерам изображения.

2. *Признаковый*. На изображении выделяются признаки: размер, цвет, объем и т. д. Выделяется множество признаков, которые можно представить как вектор в пространстве.

Недостатки:

– необходимость выделять объекты из образа;

– необходимость выделять признаки априори, что приводит к потере информации из-за неполного представления объекта;

– кластеризация – разбиение множества объектов на прикластеры по признакам.

3. *Структурный метод*. Структурные элементы образа (отрезки, дуги, точки) выделяются и распознаются.

Недостатки:

– необходимость сегментации;

– необходимость выделять признаки априори;

– реальное восприятие (человеческое мышление);

– целостное восприятие среды (картины).

В мозге строится целостная модель образа, работают и признаковое, и структурное восприятие образа. Сведения об объекте поступают в мозг по конечному числу сенсоров (анализируемых каналов), и с каждым сенсором можно сопоставить соответствующую характеристику объекта.

Человеку не нужно много времени, чтобы распознать лицо в толпе; человек легко воспринимает огромное количество информации, которую содержат в том числе и изображения, и точно определяет на них объекты.

Помимо признаков, соответствующих человеческому восприятию объекта, существует также выделенный признак либо группа признаков, которые мы называем классифицирующими; выяснение их значений при заданном векторе X – задача, которую выполняют естественные и искусственные распознающие системы.

Сложная задача распознавания образов требует разработки гибкой распознающей системы, которая могла бы классифицировать любой образ, имеющийся на изображении.

В целом процесс распознавания образов состоит из двух частей:

- обучение (система должна приобрести способность реагировать одинаково на все объекты одного образа);
- распознавание новых объектов.

Основные принципы разработки распознающих систем:

1) заложить в компьютер как можно больше известных образов-шаблонов и сравнивать их с поступающими для распознавания неизвестными образами;

2) обработать изображение и выделить его характерные признаки;

3) осуществить процесс обучения.

Любой алгоритм распознавания в общем виде можно представить как абстрактную функциональную систему R, состоящую из трех компонент [1]:

$R = \{A, S, P\}$, где

$A = \{Ak\}$, $k = 1 \ldots K$ – алфавит классов (множество категорий, по которым должны распределить образы);

$S = \{Sj\}, j = 1... n$ – словарь признаков (множество характеристик, из которых составляется описание образа);

$P = \{Pl\}, l = 1... L$ – множество правил принятия решения.

Схема общего алгоритма распознавания образов представлена на рис. 1.2.

Рис. 1.2. Схема общего алгоритма системы распознавания

На вход подается образ – некоторая конфигурация из элементов множества *S*, к ней применяют определенную последовательность правил из *P*, и в результате конфигурации присваивают индекс, соответствующий одному из элементов множества *A*.

Качество функционирования системы определяется тем, насколько часто присвоенный образу индекс совпадает с ожидаемым результатом.

Компоненты *A*, *S* представляют собой информационную часть системы, а *P* – методологическую.

Система распознавания включает в себя процессы как ***синтеза образов***, т. е. формирования описаний объектов распознавания и их классов, так и ***анализа образов***, т. е. сам процесс принятия решений. Общая схема построения системы распознавания представлена на рис. 1.3.

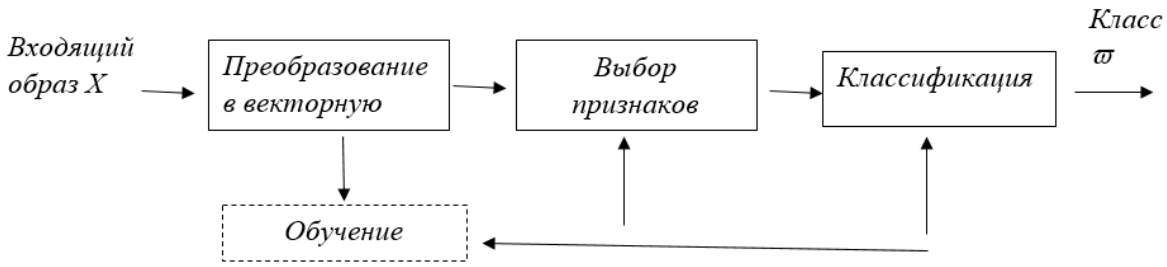

Рис. 1.3. Общая схема построения системы распознавания

Математическая постановка задачи

Пусть U – множество образов в данной задаче распознавания. Отдельный образ из этого множества будем обозначать символом x. Каждый образ $x \in U$ может характеризоваться бесконечным (и даже несчетным) числом признаков. На этапе формирования алфавита признаков мы должны выбрать некоторое подмножество признаков (как правило, конечное), которое называют *пространством признаков*. Это множество будем обозначать через X. Предположим, что во множестве образов U в данной задаче распознавания нас интересуют некоторые подмножества – классы. Множество классов $\Omega = \{\omega_1 \dots \omega_m\}$ является конечным, и классы образуют полную группу подмножеств из U (рис. 1.4).

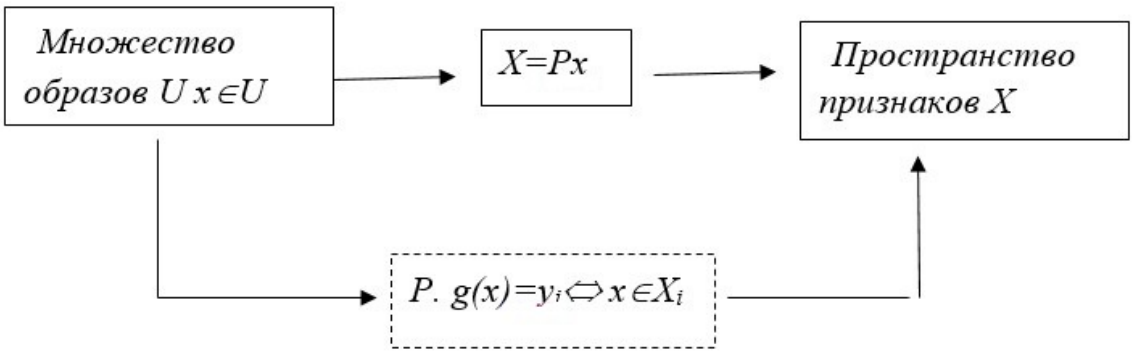

Рис. 1.4. Постановка задачи

Будем считать, что входной образ-символ X может принадлежать некоторому классу из множества всех классов $\Omega = \{\omega_1 \dots \omega_m\}$; каждый класс соответствует некоторому символу (букве, цифре и т. д.). Предполагается, что классы являются непересекающимися. Классифицировать образ x $\in U$ по классам $\omega_1 \dots \omega_m$ – значит найти функцию $g(x_i) = y$, если $x_i \in \varpi$.

Такая функция $U \rightarrow Y$, $Y = \{y_1 \dots y_m\}$ ставит в соответствие образу $x \in U$ метку $y_i \in Y$ того класса ϖ_i, которому он принадлежит, т. е. $g(x_i) = y$, если $x_i \in \varpi$. Задача распознавания образов состоит в соотнесении исходного образа x_i одному из классов ϖ_i.

Правила соотнесения образа одному из классов называют *классификатором*; реализуются эти правила в блоке классификации. Если образам соответствуют *векторы* – элементы метрического пространства, то соотнесение образа классу можно осуществить, например, с помощью вычисления расстояния между вектором и классом. На выходе классификатора должны получить тот класс (номер класса), которому принадлежит входной образ, с указанием степени достоверности классификации или получить информацию о том, что входной образ не принадлежит ни одному из классов.

В некоторых задачах (например, при кластеризации данных) множество меток Y неизвестно. В этом случае система распознавания сама должна разбить обучающую выборку на классы, исходя из некоторых критериев оптимальности.

1. Выбор наиболее информативных признаков, описывающих данный образ. Это одна из основных и важных задач в теории распознавания образов – найти минимальное количество признаков, наиболее информативно описывающих образы в данной системе (или задаче) распознавания. Полный набор выбранных для распознавания признаков называют *алфавитом признаков*. Минимальный же набор признаков, достаточный для решения данного класса задач распознавания, называют *словарем признаков*.

2. Описание классов распознаваемых образов. Эта задача сводится к определению границ классов. Границы классов могут быть заданы явно на этапе разработки системы распознавания или система сама должна их найти в процессе своей работы.

3. Нахождение оптимальных решающих процедур (методов классификации), т. е. методов соотнесения вектора признаков образа некоторому классу.

4. Оценка достоверности классификации образов. Это необходимо для оценки величины потерь, связанных с неправильной классификацией.

Глава 5. ПРЕДСТАВЛЕНИЕ ИЗОБРАЖЕНИЙ В ВЕКТОРНОЙ ФОРМЕ

Существует большое число различных форм представления изображений в распознающих устройствах или программах. Одна из наиболее простых и понятных – форма, использующая представление изображений в виде точек в некотором n-мерном пространстве. Каждая ось такого пространства естественным образом соотносится с одним из n входов или с одним из n рецепторов распознающей системы. Каждый из рецепторов может находиться в одном из m состояний, если они дискретны, или иметь бесконечно большое число состояний, если рецепторы непрерывны. В зависимости от вида используемых рецепторов может порождаться непрерывное, дискретное или непрерывно-дискретное n-мерное пространство.

Как правило, в пространство изображений вводится метрика – функция, которая каждой упорядоченной паре точек x и y пространства ставит в соответствие действительное число $d(x, y)$. При этом функция $d(x, y)$ обладает следующими свойствами:

1) $d(x, y) > 0$; $d(x, y) = 0$ тогда и только тогда, когда $x = y$;
2) $d(x, y) = d(y, x)$;
3) $d(x, y) < d(x, z) + d(z, y)$.

Введение метрики $d(x, y)$ в пространстве изображений позволяет говорить о близости или удаленности точек в этом пространстве или о мере сходства или различия анализируемых изображений. Понятие меры сходства изображений широко используется в теории распознавания образов. Однако формализация этого понятия при решении конкретных задач распознавания, как правило, не является тривиальной задачей. Более того, это одна из основных задач теории распознавания образов. Рассмотрим общие требования к мере сходства изображений.

Пусть задано некоторое конечное множество $S=\{S_1, S_2, \dots S_n\}$ входных изображений, каждое из которых является точкой в *n*-мерном пространстве изображений. Меру сходства изображений можно ввести как функцию двух аргументов $L(S_k, S_i), где S_k, S_i » S$. При этом функция $L(S_k, S_i)$ обладает следующими свойствами:

- свойством симметрии, т. е. $L(S_k, S_i)=L(S_i, S_k)$;
- областью значений функции является множество неотрицательных чисел, т. е. $L(S_k, S_i)\geq 0, k, i=1,2\dots, n$;
- мера сходства изображения с самим собой принимает экстремальное значение по сравнению с любым другим изображением, т. е. в зависимости от способа введения меры сходства выполняется одно из двух соотношений:

$L(S_k, S_k) = \max(L(S_k, S_i))$,

$L(S_k, S_k) = \min(L(S_k, S_i))$;

- в случае компактных образов функция $L(S_k, S_k)$ является монотонной функцией удаления точек S_k и S_i друг от друга в *n*-мерном пространстве.

Анализ свойств метрики и меры сходства изображений показывает, что требования к функции $L(S_k, S_i)$ нетрудно выполнить в метрических пространствах. В частности, если в метрическом пространстве введено расстояние, то оно может быть использовано в виде меры сходства изображений.

Имеется восемь различных способов расчета расстояний между изображениями

$$L(S_i, X_j) = \sqrt{\sum_{k=1}^{n} |S_{ik} - X_{jk}|^2}, \quad (1)$$

$$L(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} |S_{ik} - X_{jk}|^{\lambda}}, \quad (2)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} |S_{ik} - X_{jk}|, \quad (3)$$

$$L(S_i, X_j) = \sqrt{\sum_{k=1}^{n} \eta_k |S_{ik} - X_{jk}|^2}, \quad (4)$$

$$L(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} \eta_k \left|S_{ik} - X_{jk}\right|^{\lambda}}, \quad (5)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} \eta_k \left|S_{ik} - X_{jk}\right|, \quad (6)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} \left|\frac{S_{ik} - X_{jk}}{S_{ik} + X_{jk}}\right|, \quad (7)$$

$$L(S_i, X_j) = 1 - \frac{2}{n(n-1)} \sum_{q=1}^{n-1} \sum_{\substack{k=2 \\ k>q}}^{n} \Delta_{qk}^{i} \Delta_{qk}^{j}. \quad (8)$$

Однако большинство из них копируют друг друга с незначительными изменениями. Расстояния по Камберру (7) и по Кендалу (8) уникальны. А вот формулы 1 – 6 можно свести к одной, что облегчит программирование. Итоговая формула, которую будут использовать для описания расстояний 1 – 6, выглядит так:

$$(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} \eta_k \left|S_{ik} - X_{jk}\right|^{\lambda}}. \quad (9).$$

Формулы 1 – 6 получаются из 9-й путем изменения параметров η_k и λ.

Глава 6. ИСПОЛЬЗОВАНИЕ МЕТОДА ГЛАВНЫХ КОМПОНЕНТ В АНАЛИЗЕ ДАННЫХ. РЕАЛИЗАЦИЯ НА *PYTHON*

В задачах машинного обучения и анализа данных приходится сталкиваться с огромными наборами информации, обладающими сотнями или тысячами классов и переменных, что затрудняет процесс их исследования. Для решения этой проблемы удобно сокращать число переменных таким образом, чтобы при меньшем их числе можно было бы охватить бóльшую часть информации, необходимой для анализа.

Простой способ уменьшения размерности пространства переменных – применение методов матричной факторизации, в частности метода главных компонент.

Метод главных компонент – это статистическая операция, использующая ортогональное преобразование для трансформации

набора коррелированных переменных в линейно некоррелированные, называемые главными компонентами. При этом первая главная компонента имеет наибольшую возможную дисперсию, и каждая последующая, в свою очередь, тоже, но возможную при условии, что она ортогональна предыдущим компонентам. Результирующие векторы образуют некоррелированный ортогональный базис.

В качестве набора данных для реализации метода главных компонент использовались ирисы Фишера. Они содержат информацию о 150 экземплярах ириса, по 50 экземпляров из трех видов: ирис щетинистый, ирис виргинский и ирис разноцветный. Каждый экземпляр обладает четырьмя характеристиками: длиной наружной доли околоцветника; шириной наружной доли околоцветника; длиной внутренней доли околоцветника; шириной внутренней доли околоцветника.

Для начала необходимо загрузить набор ирисов (листинг 1). К слову, для работы с данными в экосистеме *Python* существует большое количество библиотек и инструментов, в данном случае используется *Pandas*.

Листинг 1. Загрузка набора данных

```
import pandas as pd
df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
    header=None,
    sep=',')
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.dropna(how="all", inplace=True)
```

Теперь набор данных хранится в виде матрицы, где столбцы являются характеристиками, а каждая строка представляет собой класс цветка.

Поскольку метод главных компонент дает подпространство признаков, которое максимизирует дисперсию вдоль осей, имеет смысл стандартизировать данные. Хотя все объекты в наборе ирисов были измерены в сантиметрах, выполним преобразование данных в единичный масштаб (это является требованием для оптимальной производительности многих алгоритмов машинного обучения), как показано в листинге 2.

Листинг 2. Масштабирование набора данных

```
x = df.loc[:, features].values
x_std = StandardScaler().fit_transform(x)
```

На следующем шаге необходимо вычислить собственные векторы и значения корреляционной матрицы. Они являются «ядром» метода главных компонент: собственные векторы, или главные компоненты, определяют направления нового пространства признаков, а собственные значения определяют их величину. Классический пример реализации данного метода – выполнение разложения по собственным значениям для матрицы ковариации $\sum$, являющейся матрицей $d \times d$. Ковариация между двумя объектами вычисляется следующим образом:

$$\sigma_{ijk} = \frac{1}{n-1}\sum_{i=1}^{N}(x_{ij} - \bar{x})(x_{ik} - \bar{x}_k).$$

Мы можем суммировать расчет ковариационной матрицы с помощью следующего матричного уравнения:

$$\Sigma = \frac{1}{n-1}((X - \bar{x})^T(X - \bar{x})).$$

С помощью *Python* (листинг 3).

Листинг 3. Расчет ковариационной матрицы

```
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
```

Затем необходимо выполнить разложение по собственным значениям для получившейся ковариационной матрицы (листинг 4).

Листинг 4. Получение собственных векторов и значений

```
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Типичная цель метода главных компонент – уменьшить размерность исходного пространства характеристик, проецируя его на меньшее подпространство, где собственные векторы образуют оси. Однако собственные векторы определяют только направления новой оси.

Чтобы решить, какой собственный вектор(ы) может быть отброшен без потери слишком большого количества информации для построения подпространства более низкой размерности, нужно проверить соответствующие собственные значения: собственные векторы с наименьшими собственными значениями несут наименьшую информацию о распределении данных и могут быть отброшены. Общий подход в методе главных компонент состоит в том, чтобы ранжировать собственные значения от самого высокого к самому низкому, чтобы выбрать вершину k собственных векторов, как реализовано в листинге 5.

Листинг 5. Сортировка значений собственных векторов

```
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
eig_pairs.sort(key=lambda x: x[0], reverse=True)
print('Отсортированные значения:')
for i in eig_pairs:
    print(i[0])
```

```
Отсортированные значения:
2.9303537755893174
0.9274036215173421
0.14834222648163944
0.020746013995595943
```

В результате получается, что бóльшая часть дисперсии (~ 72,77 %) может быть объяснена только первой главной составляющей. Второй компонент, в свою очередь, содержит примерно 23,03 %, в то время как третий и четвертый компоненты могут быть безопасно удалены.

Следующий шаг – построение проекционной матрицы, которая будет использоваться для преобразования ирисов в новое подпространство путем уменьшения четырехмерного пространства признаков до двухмерного подпространства, выбора «верхних двух» собственных векторов с самыми высокими собственными значениями, чтобы построить $(d \times k)$-размерную матрицу собственных векторов W (листинг 6).

Листинг 6. Построение проекционной матрицы

```
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1), eig_pairs[1][1].reshape(4,1)))
```

Далее используется (4×2)-размерная проекционная матрица W для преобразования данных в новое подпространство через уравнение $Y = X \times W$, где Y – матрица ранее преобразованных ирисов

(листинг 7). На рис. 1.5 продемонстрирован результирующий график, который отображает каждый класс данных.

Листинг 7. Преобразование данных в новое подпространство

```
Y = X_std.dot(matrix_w)

with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y[y==lab, 0],
                    Y[y==lab, 1],
                    label=lab,
                    c=col)
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()
```

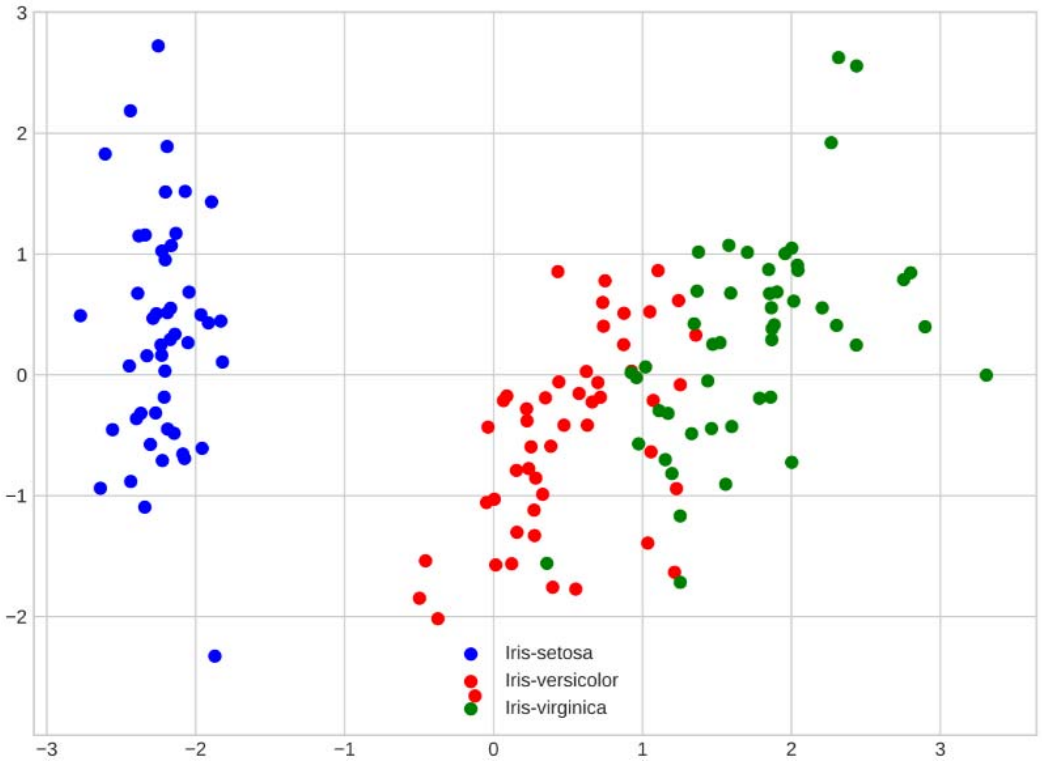

Рис. 1.5. График классифицированных классов данных

Таким образом, метод главных компонент позволяет идентифицировать многомерную структуру данных с уменьшением их размерности. В данном примере были рассмотрены основные этапы применения главных компонент на таком наборе данных, как ирисы Фишера, реализованного на языке программирования *Python*.

Глава 7. ОБЗОР СУЩЕСТВУЮЩИХ АЛГОРИТМОВ ВЫДЕЛЕНИЯ ГРАНИЦ НА ИЗОБРАЖЕНИЯХ

Роль сегментации – выделение на изображении областей с определенными свойствами. Сегментация [1] – сложный процесс в обработке и анализе изображений, так как зачастую на изображениях присутствуют шумы, искажения, текстурные области, схожие с областями, принадлежащими исследуемому объекту. Все это затрудняет процесс распознавания объектов, поэтому при сегментировании используют алгоритмы обнаружения границ. Рассмотрим наиболее популярные алгоритмы выделения границ на изображениях.

1. *Детектор границ Кэнни*

Алгоритм Кэнни [2] был разработан в 1986 году, когда компьютерное зрение только формировалось как отдельное направление компьютерной графики. Однако этот алгоритм и в наши дни остается одним из лучших детекторов границ. Алгоритм базируется на трех основных критериях:

- хорошее обнаружение, т. е. отношение сигнал/шум повышается;
- хорошая локализация;
- единственный отклик на одну границу.

Далее по полученным результатам определяют целевую функцию стоимости ошибок, минимизацию которой находят через линейный оператор для свёртки с изображением.

Во многих методах обработки изображений, в том числе и в данном алгоритме, для уменьшения вычислительных затрат изображение преобразовывают в оттенки серого [3].

В начале алгоритма применяется первая производная Гауссиана для уменьшения чувствительности алгоритма к шуму. После сглаживания границ изображения на контуре остаются только точки максимума градиента. Детектор Кэнни использует четыре фильтра для выявления горизонтальных, вертикальных и диагональных границ, чтобы удалять точки именно рядом с контуром и не разрывать область определения (линию) вблизи локальных максимумов градиента. Далее с помощью двух порогов происходит удаление слабых границ. В результате работы алгоритма по определению контура изображения происходит подавление краев, которые не связаны с сильными границами.

2. *Оператор Собеля*

Оператор [3] базируется на приближении значений градиента яркости изображения. Он вычисляет градиент яркости в каждой точке изображения и тем самым находит величину изменения яркости и ее направление. Результат показывает изменения яркости изображения в каждой точке, т. е. вероятность нахождения точки на контуре изображения искомого элемента, а также ориентацию вектора яркости.

Можно сказать, что результат оператора в точке, которая находится в постоянной яркости, – это нулевой вектор. А в точке, находящейся на границе с различной яркостью, – вектор, который пересекает границу в направлении возрастания яркости.

Оператор Собеля использует фильтрацию изображения на основе свертки по горизонтали и вертикали, поэтому он легко вычисляется. Оператор использует матрицу 3 × 3, благодаря которой свертывают исходное изображение для дальнейшего вычисления приближенных производных по горизонтальным и вертикальным направлениям.

3. *Векторный оператор Лапласа*

Алгоритм Лапласа [4] основан на поиске нулей. Он использует производные второго порядка. В отличие от алгоритмов, использующих градиентный подход для выделения границ, лапласиан является скалярной функцией. Он находит применение для выделения границ следующим образом: лапласиан принимает максимальное значение на участках «перегибов» функции яркости.

К недостаткам алгоритма можно отнести раздваивание границ. Если граница недостаточно резкая, т. е. имеются участки постоянного наклона, то может помочь специальная обработка для устранения раздваивания линий.

4. *Оператор Прюитт*

Алгоритм работает по принципу определения максимального отклика на множестве ядер свёртки для нахождения локальной ориентации границы в каждом пикселе [5].

Для этого используют различные матрицы. Из одной матрицы можно получить восемь, переставляя ее коэффициенты.

Максимальное значение каждого пикселя – это пиксель в полученном изображении. Его значения могут быть в границах от 1 до 8 в зависимости от того, какая матрица дала наибольший результат.

Оператор Прюитт еще называют подстановкой шаблонов границ. Изображению сопоставляют набор шаблонов, каждый из которых показывает расположение границ. Тогда расположение границы в пикселе формируется шаблоном, который больше всех соответствует близлежащему в окрестности пикселю.

5. *Перекрестный оператор Робертса*

Один из первых алгоритмов выделения границ. В этом методе используется суммарный вектор из двух диагональных векторов [6], который показывает наибольшую разницу градиентов между охваченными точками. А направление вектора будет указывать на наибольшую величину перепада между точками.

Оператор Робертса чаще всего используют благодаря его быстрым вычислениям, но по качеству он проигрывает альтернативным алгоритмам из-за чувствительности к шуму.

Сравнение алгоритмов

Из представленного обзора существующих алгоритмов сегментации можно сделать вывод, что они работают по-разному для различных задач. Требуется определить критерии сравнения работы данных алгоритмов. В задачах распознавания изображений важными критериями являются: точность, скорость работы алгоритма, наличие готовых решений и простота использования.

По выбранным критериям был произведен сравнительный анализ рассмотренных ранее алгоритмов. Оценка производилась с использованием шкалы от 1 до 5. Сравнение представлено в табл. 1.

Таблица 1

Сравнение алгоритмов выделения границ на изображениях

Критерий	**Оператор Кэнни**	**Оператор Собеля**	**Оператор Лапласа**	**Оператор Прюитт**	**Оператор Робертса**
Скорость	5	2	3	2	5
Точность	3	4	3	3	2
Шумоподавление	5	5	4	5	1
Простота применения	4	3	1	3	2
Доступность (количество готовых решений)	4	4	4	1	3
Итого	21	18	15	14	13

Глава 8. ОБЗОР ИНТЕЛЛЕКТУАЛЬНЫХ МЕТОДОВ МАШИННОГО ПЕРЕВОДА

Перевод текстов с одного естественного языка на другой рассматривается как процесс создания на другом языке некоторого текста, эквивалентного по содержанию и способам языкового выражения исходному. При этом человек, выполняя перевод, истолковывая и стилистически преобразуя текст, опирается на свое видение мира, проецирует текст сквозь призму своей личности, что приводит к проявлению индивидуальности в переводе, к неизбежному отклонению от текста оригинала. Использование технологии машинного перевода позволяет эффективно решать проблему субъективных трактовок, так как машина, содержащая в базе данных множество возможных вариантов перевода, не истолковывает, а лишь передает обнаруженные текстовые соответствия [10, с. 10]. Машинный перевод, будучи одной из наиболее важных областей компьютерной лингвистики, включает в себя все проблемы обработки речи на всех языковых уровнях. Среди преимуществ машинного перевода отмечают возможность обработки большого объема данных и высокую скорость перевода при общей «нейтральности» выходных текстов [12, с. 117 – 118].

Необходимо различать системы автоматизированного перевода (*computer aided software*) и машинный (автоматический) перевод. Автоматизированный перевод, к классу которого относится программное обеспечение класса *translation memory*, – это те программные средства, которые используются человеком в процессе перевода для повышения производительности труда. К машинному переводу (*machine translation*) относят технологии, позволяющие осуществлять перевод с одного языка на другой с помощью компьютерной программы без участия человека. При использовании программ автоматизированного перевода сокращается время, затрачиваемое на перевод, и увеличивается его качество, однако основная часть работы лежит на человеке-переводчике.

Средства автоматизации перевода обеспечивают более высокое качество перевода за счет единообразия терминологии и стиля, позволяют сохранить оригинальное форматирование, формировать базу данных перевода на основе уже переведенных текстов и их оригиналов. Машинный перевод осуществляет свои функции без участия человека и потому требует редактирования переведенного текста, так как неизбежно возникают ошибки и неточности, связанные с самой природой естественных языков: многозначностью, контекстуальностью, синонимичностью [13].

История развития машинного перевода берет начало в 1954 году, со времени первой публичной демонстрации перевода с помощью вычислительной техники. *Первое поколение технологий* не обладало возможностями решения проблем многозначности, не проводило лингвистический анализ, а сам перевод был приближен к пословному. *Второе поколение технологий* (вплоть до 1970-х годов) характеризовалось усилением роли языковых правил. В процессе перевода осуществлялось построение синтаксической структуры каждого предложения, основанное на правилах грамматики входного языка. Затем происходили преобразование в синтаксическую структуру выходного языка, подстановка слов из словаря и синтез предложения на выходном языке. Т*ретье поколение технологий* (до 1980-х) ознаменовалось появлением систем семантического типа, к которым относят системы машинного перевода с теорией «Смысл и текст» в своей основе. Суть теории заключалась в использовании метаязыка, который позволил бы наиболее точно передавать не только форму, но и содержание языковых знаков. Использование таких уровней, как морфологический, фонологический, синтаксический и семантический, как предполагалось, должно было существенно повысить точность и качество перевода. Однако исследователи, основываясь на данной теории, столкнулись с определенными трудностями при осуществлении перевода. В частности, не удалось разрешить проблему нахождения универсального для всех естественных языков смыслового представления. Немногим позже возникли интерактивные системы машинного перевода с привлечением участия че-

ловека на разных этапах перевода: пред- и постредактирование, частично автоматизированный перевод, смешанные системы. Доминирующей технологией для конца XX века стало обучение машины посредством предоставления достаточно большого количества параллельных текстов на разных языках. Такой подход в дальнейшем получил название статистического [14]. В настоящее время различают следующие технологии машинного перевода: аналитический, статистический и нейронный машинный перевод. ***Аналитический машинный перевод, или машинный перевод, основанный на правилах*** (*rule-based machine translation*), стал первой технологией машинного перевода. Он основывается на создании связей текста на исходном языке с текстом на требуемом языке, сохраняя при этом оригинальное значение. Различают три типа систем машинного перевода, основанного на правилах:

- *прямые системы (машинный перевод, основанный на словаре, Dictionary Based Machine Translation*) – в их основе лежит словарный пословный перевод, т. е. слова представлены в той же форме, как и в словаре, содержащемся в базе данных системы;

- *трансферные системы машинного перевода, основанного на правилах* (*Transfer Based Machine Translation*), – это один из наиболее широко используемых методов машинного перевода. В отличие от прямой модели машинного перевода, трансферный метод предполагает следование трем этапам в переводе: анализ исходного языка с целью определения грамматической структуры; перенос (трансфер) результирующей структуры на структуру целевого языка; генерация текста;

- *интерлингвальные системы* (*Interlingual RBMT Systems*) – при использовании данного метода текст исходного языка представляется в виде «нейтральной» структуры, независимой от каких-либо естественных языков. Текст на целевом языке формируется на основе этого нейтрального варианта. Одно из преимуществ данного метода – воз-

растание значения языка-посредника, что позволяет увеличить количество языков перевода [15]. Этапы процесса перевода: морфологический анализ, объединение отдельных слов в группы, синтаксический анализ и определение посредством алгоритма каждого слова как члена предложения, синтез предложений. Рассмотрим перевод предложения *A girl eats an apple* с исходного английского языка на немецкий. На первом этапе необходимо получить информацию о каждом из исходных слов: *a* – неопределенный артикль; *girl* – существительное; *eats* – глагол; *an* – неопределенный артикль; *apple* – существительное. Далее необходимо получить синтаксическую информацию о глаголе *to eat*: *NP-eat-NP*; *eat* – простое настоящее время, третье лицо, единственное число, активный залог. На третьем этапе происходит парсинг исходного предложения: (*NP an apple*) = прямое дополнение, зависимость от глагола *to eat*. Следует отметить, что возможны варианты, когда достаточно и частичного парсинга структуры предложения для создания карты структуры выходного текста. На четвертом этапе непосредственно происходит перевод английских слов на немецкий язык: *a* (категория: неопределенный артикль) = *ein* (категория: неопределенный артикль); *girl* (категория: существительное) = *Mädchen* (категория: существительное); *eat* (категория: глагол) = *essen* (категория: глагол); *an* (категория: неопределенный артикль) = *ein* (категория: неопределенный артикль); *apple* (категория: существительное) = *Apfel* (категория: существительное). На пятом этапе происходит генерация предложения на требуемом языке: изменяются словарные формы слов (применяются другое склонение, спряжение, число). Результатом становится перевод предложения на немецкий язык: *A girl eats an apple => Ein Mädchen isst einen Apfel*. Несмотря на такие положительные моменты, как синтаксическая и морфологическая точность, а также возможность настройки на предметную область, аналитический перевод требует постоянного поддержания и актуализации баз данных, длителен в разработке и игно-

рирует контекст [16]. Однако наиболее существенным преимуществом данного метода является отсутствие необходимости использования двуязычных текстов. Это позволяет создать систему перевода для таких языков, у которых нет общих текстов или какой-либо оцифрованной информации. Кроме того, созданную единожды систему машинного перевода, основанного на правилах, можно впоследствии использовать для перечисления всех известных системе слов и фраз, вариантов их перевода и вероятностей этих переводов. Знания системы о языке представлены в виде вероятностной модели языка. Ее использование обусловлено необходимостью выбора тех или иных вариантов и связей в зависимости от контекста. Задача декодера заключается в подборе вариантов перевода для исходного текста путем сочетания между собой фраз из модели перевода и их сортировки по убыванию вероятности. Далее происходит оценивание получившихся вариантов с помощью модели языка [16].

В случае использования технологии ***статистического перевода*** возможно включение дополнительной лингвистической информации для языков с богатым словоизменением. К положительным сторонам данного метода относят быструю настройку, экономию вычислительных ресурсов за счет исключения глубокого анализа текста, а также легкость, с которой системы справляются с переводом сложных и редких слов и терминов. Тем не менее у этого подхода есть и существенные недостатки, вызванные, прежде всего, особенностями естественных языков. Низкое качество перевода отмечается у языков, принадлежащих к разным языковым семьям, – в таком случае необходимо использование сложных моделей типа *tree-to-tree/tree-to-string* (как в случае с английским и японским языками). Также на точности выбора лексических единиц сказывается и дефицит параллельных корпусов текстов. Помимо указанных, при использовании метода статистического машинного перевода возникают также следующие проблемы:

• статистические аномалии. Часто при внесении информации о реальном мире используются имена собственные, которые в дальнейшем могут быть ошибочно использованы при переводе, к примеру, предложение *I took the train to Paris* может быть ошибочно переведено как «Я сел на поезд в Берлин» из-за обилия вариантов *train to Berlin* в тренировочном наборе;

• порядок слов в разных языках может существенно отличаться. Несомненно, можно синтаксически классифицировать слова в предложениях и получить обобщенную модель типа *SVO* (подлежащее – сказуемое – дополнение), но при этом сложно учитывать положение служебных частей речи и изменение порядка слов при смене типа предложения (например, на вопросительное);

• слова, не вошедшие в словарь. Эта проблема возникает ввиду недостатка данных при обучении системы, различиях в морфологии. Системы статистического перевода обычно хранят такие слова как отдельные символы без создания связи с другими словоформами или фразами [18]. Две указанные выше технологии развиваются и в настоящее время и сохраняют свою актуальность. Их переплетение и слияние породило отдельный метод машинного перевода, получивший название *гибридного*. Утверждается, что гибридный метод способен использовать особенности как машинного перевода, основанного на правилах, так и статистического машинного перевода. Перечислим некоторые подходы:

– *правила, доработанные статистикой,* – в данном случае словарный перевод улучшается и корректируется за счет статистических правил;

– *статистика, управляемая правилами,* – правила используются для предварительной обработки информации, а также на этапе статистической коррекции результирующего перевода.

Вместе с этим на пике популярности находятся методы, использующие ***искусственные нейронные сети*** (*neural machine translation*)

для перевода, которые обучаются совместно от начала до конца, чтобы максимизировать эффективность перевода [20]. Нейросети сегодня научились сами согласовывать род и падежи в разных языках. Рассматривая проблемы машинного перевода, уместно отметить и явление культурологической непереводимости отдельных единиц текста. Предполагается, что в случае нахождения возможности идентификации таких единиц в тексте перевода станет возможен их анализ и внесение в базы данных программ перевода. В связи с этим британский лингвист Дж. К. Кэтфорд предлагал разработать алгоритмы, базирующиеся на правилах, позволяющих обращаться к контекстуальному значению. Для целей машинного перевода эти правила могут иметь вид операционных команд для текстуального поиска элементов, маркированных в машинном словаре специальными диакритиками с предписанием вывести в каждом конкретном случае определенный, заложенный в систему эквивалент перевода отдельных единиц. Точное выполнение таких алгоритмов может существенно повысить качество, корректность перевода, но заменить человека они не смогут даже в долгосрочной перспективе – пока не будет создан полноценный ИИ [21].

ПРАКТИКУМ

Лабораторная работа № 1
ЛИНЕЙНАЯ РЕГРЕССИЯ

Цель работы: познакомиться со средой выполнения *MatLab* и организацией линейной регрессии.

Ход работы

1. Постановка задачи

Дана обучающая выборка $(x^{(1)}, y^{(1)}) \dots (x^m, y^m)$. Необходимо найти оптимальную функцию вида

$$h(x) = ax + b,$$

которая дает наименьшую ошибку на обучающей выборке.

2. Считывание данных

Для начала следует запустить *MatLab* (или *Octave*) и в нем выполнить следующую команду:

```
cd('/home/user/ai/01');
```

Естественно, путь следует указать собственный, а именно каталог, в котором будут располагаться программные файлы, а также файлы с входными данными. Создайте там файл *input.txt*, который будет содержать 20 строк следующего вида:

```
1, 1.5
2, 2
3, 2.5
…
20, 11
```

Иными словами, речь здесь идет о зависимости вида

$$h(x) = 1 + 0.5x.$$

Наша ключевая задача – считать эти данные из файла. Для этого в командной строке нужно выполнить следующие действия:

```
data = load('input.txt');
x = data( : , 1);
y = data( : , 2);
m = length(y);
```

Первая команда считывает данные из файла и создает на их основе матрицу размером 20×2. Вторая команда создает переменную *x* и записывает в нее первый столбец матрицы. Третья команда записывает в *y* второй столбец матрицы. Последняя строка вычитает размер вектора *y*.

По итогам выполнения команды в окне переменных должны отобразиться четыре новые переменные.

3. Приведение к матричному виду

Чтобы упростить порядок вычислений, полезно ввести параметр x_0, который для всех строк будет равен единице

X = [ones(m,1), data(: , 1)];

В результате получаем следующие строки в табл. 2.1.

Таблица 2.1

Строки обучающей выборки после введения базового элемента

№	x_0	x_1	y
1 2 3	1 1 1	1 2 3	1.5 2 2.5
20	1	20	11

4. Формализация задачи

Сформулируем задачу следующим образом. Пусть у нас имеется вектор θ из двух чисел. Тогда для любой строки $x^{(i)}$ (которая теперь содержит два элемента) $\theta x^{(i)}$ дает число (речь идет об операции произведения матрицы на вектор в том смысле, в котором эта операция определяется в линейной алгебре). Задача заключается в том, чтобы минимизировать функцию стоимости

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2, \qquad (1)$$

где

$$h_\theta(x) = \theta^x x = \theta_0 + \theta_1 x_1.$$

Стоит напомнить, что для всех строк

$x_0 = 1.$

Добиться этого можно с помощью постепенного спуска по итерациям, используя следующую формулу:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}.$$

5. Реализация функции стоимости

Создаем в каталоге файл *computeCost.m* следующего содержания:

```
function J = compute Cost(X, y, theta)
m = length(y); % number of traning examples
J = 0;
for i = 1:m
        delta = y(i)-X(i,:)*theta;
        J = J + delta*delta;
end
J = J/(2*m)
end
```

Данный код реализует напрямую цикл выражение 1. Стоит обратить внимание, что здесь используется матричное произведение.

6. Реализация функции градиентного спуска

Создаем в каталоге файл *gradientDescent.m* следующего содержания:

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
        m = length(y); % number of training examples
        J_history = zeros(num_iters, 1);
        n = length(theta);
        for iter = 1:num_iters
                theta = theta - (alpha/m)*X'*(X*theta-y);
                J_history(iter) = computeCost(X, y, theta);
        end
end
```

7. Проверка градиентного спуска

В командной строке выполните следующие инструкции:

```
theta = zeros(2, 1);
theta = gradientDescent(X, y, theta, 0.01, 500);
```

Данный код будет выполнять градиентный спуск с шагом 0.01 и за 500 итераций. Проверьте, каков будет итоговый *theta* в результате выполнения данного кода.

8. Введение дополнительных столбцов

Предположим, что мы хотим провести линейную регрессию от одной переменной x, однако предполагаем, что на самом деле зависимость выражается полиномом более высокого порядка, например:

$$f(x) = x^2 + 2x + 1.$$

Для того чтобы разрешить такую проблему, достаточно ввести новый столбец x_2, который будет содержать квадраты x_1 (то же самое можно проделать для кубов и более высоких степеней). Конкретно для нашей задачи это можно сделать так:

```
X = [ones(m,1), data(:,1), data(:,1).^2];
```

Выражение

```
x.^2
```

в *MatLab* означает не то, что матрица умножается сама на себя, а то, что каждый элемент матрицы возводится в квадрат.

Варианты задания

В каждом варианте заданий необходимо выполнить то, что описано в предыдущем разделе, а также реализовать линейную регрессию дополнительного выражения

1) $z = x^2 + y^2$;
2) $z = x^2 + xy + 1$;
3) $z = x^2 + y^2 + 2xy$;
4) $z = x^3 + y^2$;
5) $z = x + y/x$;
6) $z = x + \log(y^2)$;
7) $z = \sin(y) + x$;

8) $z = \cos(x) + \sin(y)$;
9) $z = \cos(x)^2 + \sin(y)$;
10) $z = x^2 + 2x + y$.

Для каждого варианта необходимо создать файл из ста строк, содержащий значения входных переменных и выходной переменной. Следует провести регрессию как с введением дополнительных значений, так и без них.

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Как именно влияет параметр α на характеристики алгоритма линейной регрессии?

2. Что будет, если сделать этот параметр слишком большим?

3. Что будет, если сделать этот параметр слишком маленьким?

4. Почему мы стремимся свести количество циклов к минимуму в *MatLab*, и вместо этого использовать матричные произведения и обработку векторов?

Лабораторная работа № 2
КЛАСТЕРИЗАЦИЯ

Цель работы: познакомиться с методами обучения без учителя.

Ход работы

1. Постановка задачи

В предыдущих работах мы рассматривали обучающие выборки, где каждому входному значению сопоставлялось выходное значение. На основе этой выборки мы пытались построить систему, пытающуюся предсказать выходные значения для комбинаций, которые не встречались в обучающей выборке.

В лабораторной работе № 2 мы рассмотрим ситуацию, когда выходных значений нет, т. е. нам просто дан набор строк размера n, которые можно рассматривать как точки в n-мерном пространстве; и нам интересно, не существует ли каких-то скрытых закономерностей среди этих точек. Наиболее очевидная закономерность в данном случае – не принадлежат ли эти точки отдельным группам, которые четко отличаются друг от друга. Эти группы назовем *кластерами*, а сам процесс определения того, каким группам принадлежат точки, – *кластеризацией*.

В методе k-средних алгоритм действует следующим образом. Ему дан набор из m точек в n-мерном пространстве, а также число k – количество кластеров, на которое надо разбить эти точки. Алгоритм должен найти k точек в пространстве, которые мы обозначим как μ_i – это будут центры кластеров. Кроме того, алгоритм должен сопоставить каждой точке из исходных индекс от 1 до k, т. е. определить, какому кластеру принадлежит эта точка. Обозначим индекс точки x_i как $c(x_i)$. Задача – найти центры кластеров и назначить точки таким образом, чтобы их сумма квадратов расстояний от точек до их центров была минимальной

$$\sum_{i=1}^{m} || x_i - \mu_{c(x_i)} ||^2.$$

Кластеризация может понадобиться в ряде задач. Например, для сегментации рынка, выявления различных групп пользователей в социальных сетях. Также кластеризация может использоваться для сжатия изображений. Например, у нас есть изображение, где каждый цвет описывается с помощью представления *RGB*. Мы можем сжать его, скажем, до 256 цветов, попробовав найти 256 оптимальных кластеров, и их центры использовать как цвета для представления исходного изображения. Это особенно хорошо работает в тех случаях, когда в исходном изображении есть регионы похожего цвета (например, тел или предметов), и в силу особенностей нервной системы и оптического восприятия человек даже не заметит разницы после сжатия.

2. Подготовка выборки

Для примера создадим набор точек, где точки будут принадлежать трем группам:

- кругу с радиусом 1 и центром в точке {0, 0};

- кругу с радиусом 1 и центром в точке {4, 4};
- кругу с радиусом 1 и центром в точке {2, 2}.

Сделаем это с помощью следующего кода:

```
X=zeros(150,2);
for i = 1:50
X(i,1) = rand()-0.5; X(i,2) =
rand()-0.5;
end
for i = 51:100
X(i,1) = rand()+3.5; X(i,2)
= rand()+3.5; end
for i = 101:150
X(i,1) = rand()+1.5;
X(i,2) = rand()+1.5;
end

plot(X(:,1), X(:,2), 'ko', 'MarkerFaceColor', 'y',
'MarkerSize', 7);
```

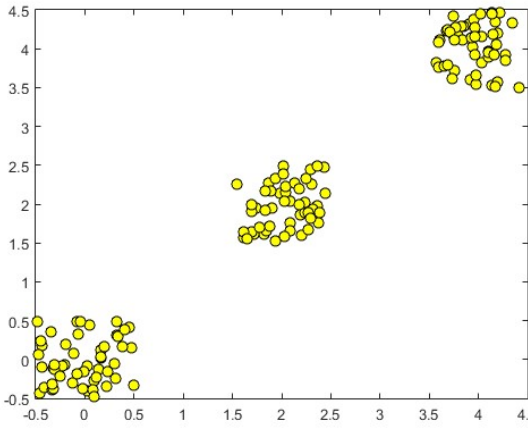

Рис. 2.2.1. Исходная выборка

Результат будет выглядеть так, как показано на рис. 2.2.1 ниже. Для человека очевидно, что эти точки разделяются на три различные группы. Вопрос в том, как заставить программу это определить.

3. Реализация алгоритма

Необходимо реализовать три основные функции. Первая, располагая точками *x* и уже имеющимися центрами кластеров, определяет, к какому центру ближе всего расположена эта точка

```
function idx = findClosestCentroids(X, centroids)
K = size(centroids, 1);
idx = zeros(size(X,1), 1);
for i=1:size(X,1)
        idx(i) = 1;
        value = sum((X(i,:)-centroids(1,:)).^2);
        for j = 2:K
                value1 = sum((X(i,:)-centroids(j,:)).^2);
                if value1 < value
                        idx(i) = j; value = value1;
                end
        end
end
end
```

В принципе здесь можно было бы избежать части циклов, но в данном случае код написан в таком стиле, чтобы был понятен процесс поиска наименьшего расстояния от точки до центра кластера. Далее, вторая функция высчитывает новые центры на основе выборки *x*

```
function centroids = computeCentroids(X, idx, K)
[m n] = size(X);
centroids = zeros(K, n);
for i = 1:K idx1 =
        (idx==i); s
        = idx1'*X;
        centroids(i,:) = s(:)/sum(idx1);
end
end
```

Здесь используется формула

$$\mu_k = \frac{1}{|C_k|}\sum_{i \in C_k} x_i,$$

где C_k – индексы точек, которые принадлежат k-му кластеру. Иными словами, новый центр вычисляется как геометрическое среднее между

точками, близкими к старому центру. Ну и основной алгоритм здесь выглядит так:

```
function [centroids, idx] = runkMeans(X, initial_centroids, ... max_iters)
% Initialize values
[m n] = size(X);
K = size(initial_centroids, 1); centroids = initial_centroids;
idx = zeros(m, 1);
% Run K-Means for i=1:max_iters
        % For each example in X, assign it to the closest
        centroid idx = findClosestCentroids(X, centroids);
        % Given the memberships, compute new centroids
        centroids = computeCentroids(X, idx, K);
end
end
```

Как получить начальные центры? Их можно задать вручную, но можно сделать это с помощью следующей функции, выбравшей случайным образом *k* точек из уже имеющихся

```
function centroids = kMeansInitCentroids(X, K)
randidx = randperm(size(X,1));
centroids = X(randidx(1:K), :);
```

Теперь попробуем вычислить центры кластеров, вызвав основной метод

```
centroids = runkMeans(X, kMeansInitCentroids(X, 3), 400);
```

Результат варьируется в зависимости от того, как были выбраны центры, но должно получиться что-то вроде этого:

```
        2.0319;1.9701
        3.9521;4.0336
        -0.0258;-0.0175
```

4. Визуализация кластеров

Теперь напишем функцию, которая, зная точки *x* и индексы кластеров, продемонстрирует это визуально

```
function plotDataPoints(X, idx, K)
% Create palette
palette = hsv(K + 1);
colors = palette(idx, :);
% Plot the data
scatter(X(:,1), X(:,2), 15, colors);
end
```

И вызовем функцию следующим образом:

```
plotDataPoints(X, findClosestCentroids(X, centroids), 3)
```

Результат представлен на рис. 2.2.2

Можно убедиться, что алгоритм действительно правильно разбил точки на группы.

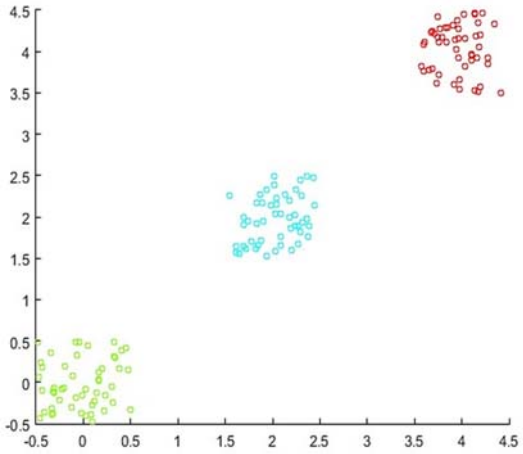

Рис. 2.2.2. Результат разбиения на кластеры

Варианты задания

Необходимо выполнить задание и убедиться, что кластеризация действительно работает. Также необходимо выполнить собственное задание.

1. Точки должны быть в трехмерном пространстве и располагаться внутри двух кубов с ребрами, равными единице, но в различных центрах.

2. Точки должны быть в трехмерном пространстве и располагаться внутри четырех кубов с ребрами, равными единице, но в различных центрах.

3. Точки должны быть в трехмерном пространстве и располагаться внутри трех шаров с радиусами, равными единице, но в различных центрах.

4. Точки должны быть в трехмерном пространстве и располагаться внутри пяти шаров с радиусами, равными единице, но в различных центрах.

5. Точки должны быть в трехмерном пространстве и располагаться внутри семи шаров с радиусами, равными единице, но в различных центрах.

6. Точки должны быть в двухмерном пространстве и располагаться внутри двух квадратов с ребрами, равными единице, но в различных центрах.

7. Точки должны быть в двухмерном пространстве и располагаться внутри четырех квадратов с ребрами, равными единице, но в различных центрах.

8. Точки должны быть в двухмерном пространстве и располагаться внутри трех кругов с радиусами, равными единице, но в различных центрах.

9. Точки должны быть в двухмерном пространстве и располагаться внутри пяти кругов с радиусами, равными единице, но в различных центрах.

10. Точки должны быть в двухмерном пространстве и располагаться внутри семи кругов с радиусами, равными единице, но в различных центрах.

Необходимо проверить, какие результаты будут показаны для различного количества кластеров.

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Какова сигнатура функции *scatter*? Что означает 15 при ее вызове?

2. Как правильно определить, какое именно количество кластеров имеется в выборке?

3. Как получить начальные центры кластеров?

4. Как работает алгоритм *k*-средних?

Лабораторная работа № 3
КЛАССИФИКАЦИЯ

Цель работы: познакомиться с методами классификации (логистической регрессии).

Ход работы

1. Постановка задачи

В предыдущей работе мы рассматривали обучающие выборки, где нам давались выходные значения y, и нам было интересно предсказать, каково будет значение y для вариантов, не встречающихся в выборке. Задача логистической регрессии возникает в том случае, когда y имеет фиксированный набор значений, и не может принимать каких-то значений не из этого набора. В предельном случае y может принимать только два значения: $\{0,1\}$. Таким образом, перед нами стоит задача отнести входящие данные изображения к одному из двух классов. Рассмотрим различные примеры.

У нас есть набор рисунков, на одной части которых изображены котята, а на другой – нет. Мы можем привести все рисунки к одинаковой размерности (например, 256 × 256) и монохромному изображению (т. е. каждая точка определяется яркостью по шкале от 0 до 255). Тогда каждый рисунок будет примером, содержащим 256^2 столбцов, где каждый столбец будет принимать значения от 0 до 255. Каждому из рисунков мы придадим выходное значение 1, если на нем изображены котята, и 0 – в противном случае. Задача состоит в том, чтобы обучить систему распознавать наличие котят на рисунках не из выборки.

У нас имеется набор писем, одна часть которого – спам, другая – нет. Каждое письмо можно преобразовать в набор элементов. Например, мы можем взять словарь и подсчитать, сколько раз встречается каждое слово в письме. Тогда каждое письмо можно представить как вектор, где каждый элемент будет соответствовать определенному слову. Задача – обучить систему классифицировать новые письма, являются ли они спамом или нет.

Для начала создадим обучающую выборку. В качестве таковой мы будем использовать точки на двумерной плоскости (координаты x_1 и x_2), а выходное значение определим следующим образом:

$$y = \begin{cases} 1, x_2 \geq x_1 \\ 0, x_2 < x_1. \end{cases}$$

Несложно видеть, что фактически плоскость разделена на две полуплоскости прямой, проходящей через центр координат. Для начала создадим обучающую выборку на сто строк

```
X = zeros(100, 2);
Y = zeros(100, 1);
```

Далее заполним эту выборку

```
for i = 1:100
   X(i,1) = rand();
   X(i,2) = rand();
   Y(i) = X(i,2)>=X(i,1);
end
```

Теперь попробуем нарисовать график

```
hold on;
pos = find(Y==1);
neg = find(Y==0);
plot(X(pos,1), X(pos,2), 'k+', 'LineWidth', 2, '
MarkerSize', 7);
plot(X(neg,1), X(neg,2), 'ko', 'MarkerFaceColor', 'y', '
MarkerSize', 7);
hold off;
```

Обратите внимание на несколько интересных особенностей. Мы находим индексы *pos* и *neg* с помощью специальной команды. Далее мы указываем функции *plot*, какие именно строки обучающей

выборки следует нарисовать разными символами. Если все проделано корректно, то результат будет как на рис. 2.3.1 (он может несколько отличаться из-за случайности исходного набора данных).

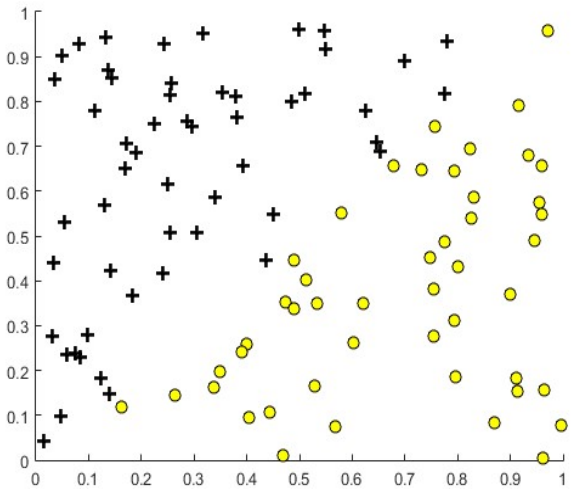

Рис. 2.3.1. Пример обучающей выборки

Варианты задания

1. Регрессия сигмоидной функции

Итак, что же делать с этой обучающей выборкой? Мы рассмотрим здесь новую функцию

$$h_\theta(x) = g(\theta^T x),$$

где g – это сигмоид

$$g(z) = \frac{1}{1+e^{-z}}. \quad (1)$$

Схематично сигмоид представлен на рис. 2.3.2.

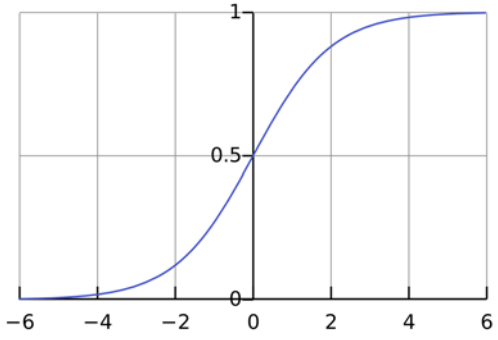

Рис. 2.3.2. Сигмоидная функция

Из рисунка видно, что для $x = 0$ эта функция равна 0.5, для положительных значений x она стремится к единице, а для отрицательных значений она стремится к нулю. Эта функция на выходе всегда дает значения в интервале [0; 1].

Фактически значение $g(x)$ можно трактовать как вероятность того, что y для данного x будет равен 1. Мы введем пороговое значение и будем считать, что если $h_\theta(x) \geq 0.5$, то система классифицирует данный пример как положительный ($y = 1$).

Тогда функция стоимости может быть вычислена следующим образом:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i \log\left(h_\theta(x^i)\right) - (1-y^i)\log(1-h_0(x^i))].$$

Нетрудно заметить что эта сумма обладает несколькими характерными особенностями. По сути она состоит из двух частей. Первая – где $y^i = 1$. Эта часть равна

$$-\frac{1}{m}\sum_{i=0}^{m}[\log(h_\theta(x^i))].$$

Из особенностей логарифма понятно, что чем ближе $h_\theta(x)$ к единице, тем ближе к нулю логарифм от этого значения и тем меньше ошибка. Идеальным является случай, когда для всех положительных примеров обучающей выборки $h_\theta(x)$ будет давать значение, равное единице.

Аналогично рассуждая и в случае с отрицательными примерами, мы приходим к выводам, что $J(\theta)$будет равна нулю только в том случае, когда $h_\theta(x)$ будет выдавать единицу для положительных примеров и ноль – для отрицательных примеров. Таким образом, попытка минимизировать эту функцию приведет нас к более-менее стабильному классификатору, т. е. к такому набору θ, который можно будет использовать на практике.

Частные градиенты для этой функции вычисляются следующим образом:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i) - y^i)x_j^i].$$

В этом несложно убедиться, если применить стандартные правила дифференцирования, что и позволяет нам произвести градиентный спуск. Для этого мы напишем специальную функцию, которая возвращает как стоимость для текущей точки, так и значения градиентов

```
function [J, grad] = costFunction(theta, X, Y)
% Initialize some useful values
m = length(Y); % number of training examples
% Calculate the sigmoids
sigmtable = sigmoid(X*theta);
% Calculate the cost for all examples
J = -(Y'*log(sigmtable)+(1-Y)'*log(1-sigmtable))/m;
% Calculate the gradients for all columns at the same time
grad = X'*(sigmtable-Y)/m;
end
```

Эта функция использует технику векторизации, т. е. она избегает циклов. Функцию *sigmoid* следует реализовать самостоятельно согласно формуле 1.

Теперь можно найти оптимальное значение θ с помощью следующего кода:

```
options = optimset('GradObj', 'on', 'MaxIter', 400);
[theta, cost] = fminunc(@(t)(costFunction(t,X,Y)), rand(3,1), options);
```

Данный код использует встроенную функцию *minunc*, которая позволяет найти локальный минимум. Но как удостовериться, что это действительно хороший классификатор? Для этого напишем функцию, которая будет классифицировать входные значения с помощью θ

```
function p = predict(theta, X)
m = size(X, 1); % Number of training examples
p = zeros(m, 1);
prediction = sigmoid(X*theta);
positive = find(prediction>=0.5);
negative = find(prediction<0.5); p(positive,1)=1; p(negative,1)=0;
end
```

Имея этот код, можно вычислить количество совпадений между выходными значениями обучающей выборки и классификатором

```
sum(predict(theta, X) == Y)
```

Если все сделано правильно, то классификатор выдаст значение 100.

2. Регуляризация

Предыдущий раздел не накладывал никаких ограничений на θ. Это может привести к тому, что классификатор будет либо недообучен (т. е. не для всех примеров обучающей выборки он выдаст правильный ответ), либо переобучен (т. е. он даст правильный ответ для всех примеров обучающей выборки, но не более того). Значение 100 указывает на то, что недообученности не наблюдается. Однако давайте попробуем использовать классификатор на значениях не из обучающей выборки и сгенерируем сто строк случайно

```
Xcheck = zeros(100, 3);
Ycheck = zeros(100, 1);
for i = 1:100
X1 = rand()*100;
X2 = rand()*100;
Xcheck(i,1) = 1;
Xcheck(i,2) = X1;
Xcheck(i,3) = X2;
Ycheck(i) = X2>=X1;
end
```

Если теперь запустить код,

```
sum(predict(theta, Xcheck) == Ycheck)
```

то можно увидеть число меньше ста, что является следствием переобученности. Чтобы этого избежать, введем регуляризацию и специальный параметр λ. Тогда стоимость ошибки будет вычисляться следующим образом:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i \log\left(h_\theta(x^i)\right) - (1-y^i)\log(1-h_0(x^i))] +$$
$$+\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$$

Обратите внимание, что здесь не учитывается θ_0, и что чем больше значение всех остальных элементов θ, тем больше и ошибка, т. е. теперь мы пытаемся уменьшить значение θ по возможности. Градиент для x_0 не изменился, а вот для всех остальных он вычисляется как

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left[(h_\theta(x^i) - y^i)x_j^i\right] + \frac{\lambda}{m}\theta_j.$$

Реализуем эту функцию следующим образом:

```
function [J, grad] = costFunctionReg(theta, X, y, lambda)
% Initialize some useful values m = length(y); % number of training ex-
amples
% You need to return the following variables correctly
J = 0;
grad = zeros(size(theta));
n = size(theta);
sigmtable = sigmoid(X*theta);
J = -(y'*log(sigmtable)+(1-y)'*log(1-sigmtable))/m +
(lambda/(2*m))*sum(theta(2:n).^2);
thetared = theta;
thetared(1,1) = 0;
grad = X'*(sigmtable-y)/m + (lambda/m)*thetared;
end
```

Чтобы вычислить теперь значение θ, можно запустить следующий код:

```
[theta, cost] = fminunc(@(t)(costFunctionReg(t,X,Y,0.01)), rand(3,1),
options);
```

Как вообще λ влияет на результат? Для столь простого примера это не будет особо заметно. Однако в случае, когда классификатором

выступает не прямая, а полином более высокого порядка, границей, разделяющей классы, будет более сложная кривая, которая не всегда верно классифицирует входные данные. На рис. 2.3.3, 2.3.4 и 2.3.5 показано это отличие

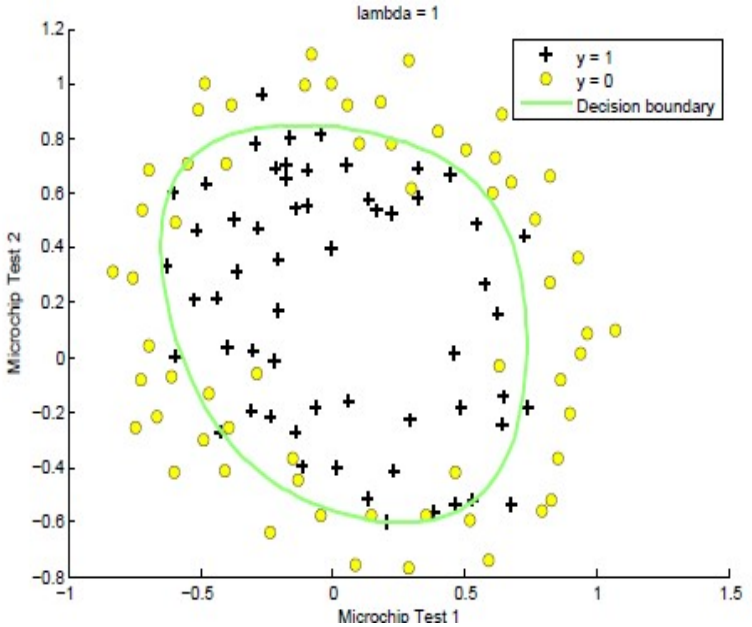

Рис. 2.3.3. Нормальный классификатор (λ = 1)

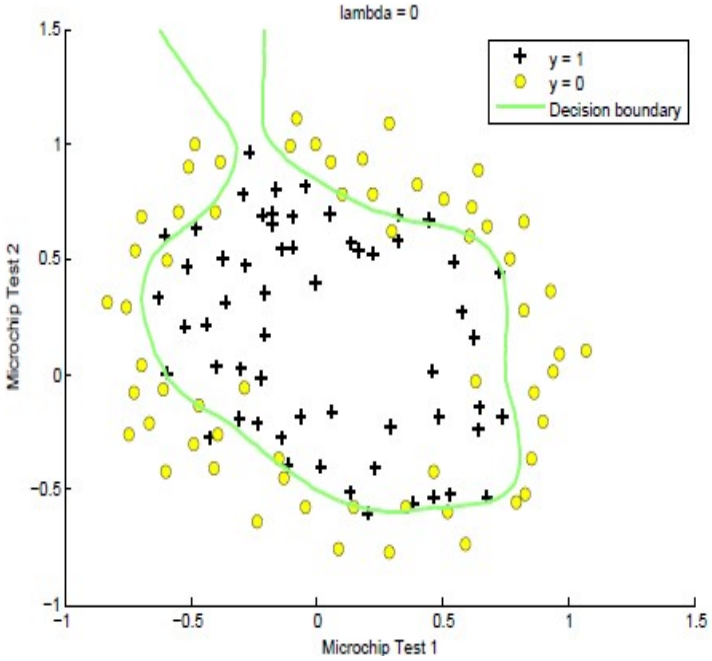

Рис. 2.3.4. Переобученный классификатор (λ = 0)

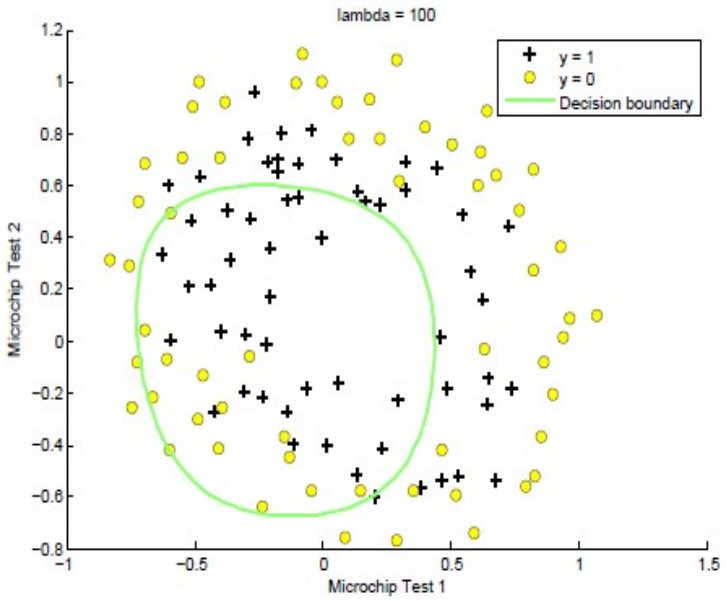

Рис. 2.3.5. Недообученный классификатор ($\lambda = 100$)

Варианты задания

В каждом варианте необходимо выполнить все примеры, приведенные выше. Кроме того, надо построить классификатор для собственного варианта

1) $x^2_1 + x^2_2 - 4 \geq 0$;
2) $x^3_1 + x_2 + 1 \geq 0$;
3) $x^3_1 + x^2_2 + 1 \geq 0$;
4) $x^2_1 + x_2 + x_1 \geq 0$;
5) $\sin(x_1) + \cos(x_2) \geq 0$;
6) $\sin^2(x_1) + \cos^2(x_2) - 1 \geq 0$;
7) $x^4_1 + x^2_2 + x_1 \geq 0$;
8) $x^4_1 + x^2_2 - 3 \geq 0$;
9) $\log(x^2_1 + x^2_2) \geq 0$;
10) $\log(x^2_1 + x^4_2) \geq 0$.

Необходимо проверить, насколько точно полученный классификатор распознает обучающую выборку и насколько хорошо он ведет

себя на примерах не из обучающей выборки. Сделать это следует с разными параметрами регуляризации. Также нужно показать, как именно выглядит график для обучающей выборки.

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Объясните смысл функции *minunc*. Что именно мы ей передаем, что означают @(*t*) в примерах выше и все остальные аргументы?
2. Объясните смысл функции *find*. Что именно она принимает на вход и что возвращает?
3. Какие вы знаете параметры регуляризации?
4. При каких условиях применяют задачу логистической регрессии?

Лабораторная работа № 4
РЕГРЕССИЯ ДЛЯ МНОЖЕСТВА КЛАССОВ

Цель работы: познакомиться с методами многокритериальной классификации.

Ход работы

1. Постановка задачи

В одной из предыдущих лабораторных работ мы рассматривали задачи классификации, постановка которых была сознательно упрощена для ознакомления с предметом. Вот два ключевых недостатка логистической регрессии из предыдущей лабораторной работы:

1) регрессия способна ответить лишь «ДА/НЕТ», чаще же всего нас интересуют случаи, когда классов для выходных переменных может быть несколько (например, пять различных видов изображений);

2) классификатор является линейным и не может классифицировать сложные нелинейные случаи, поэтому для улучшения качества классификации можно добавлять дополнительные данные за счет комбинаций переменных, но это довольно трудоемко для процесса обучения.

В лабораторной работе будет рассмотрена проблема классификации для множества возможных классов на примере такой задачи, как распознавание текста.

2. Входные данные

В данной лабораторной работе используется дополнительный файл *digits.mat*, который содержит часть данных из *MNIST*. Он содержит 500 примеров для обучения распознавания цифр (которые написаны с различными наклонами). Чтобы его загрузить, нужно в каталоге, где расположен этот файл, выполнить следующую команду:

```
> load('digits.mat');
```

В результате мы увидим две переменные *X* и *Y* в окне среды, где *X* имеет размерность 500 × 400, а *Y* – 500 × 1. *X* является представлением рисунка размером 20 × 20 пикселей в черно-белом формате, где у каждого пикселя задана яркость изображения. Что касается *Y*, то цифра 0 представлена значением 10 для упрощения работы *MatLab* (*Octave*), а цифры от 1 до 9 представлены напрямую.

3. Отображение входных данных

Интересный вопрос заключается в том, как заставить *MatLab* отображать эти наборы в виде реальных рисунков. Для этого напишем соответствующую функцию

```
function [retval] = displayRow (row)
% Compute the size of the matrix
n = floor(sqrt(size(row, 2)));
% Declare the matrix
matr = zeros(n,n);
% Fill the matrix from the linear row
for i = 1:n
    for j = 1:n
        matr(i,j) = row((i-1)*n+j);
```

```
    endfor
endfor

% Gray Image
colormap(gray);
% Draw the image itself
imagesc(matr);
% Do not show axis
axis image off

% Return just zero
retval = 0;
endfunction
```

Можно заметить, что функция состоит из двух частей. Первая часть преобразует строку из 400 элементов в матрицу размером 20 × 20. Вторая часть использует встроенную функцию *imagesc*, которая отображает набор пикселей, в данном случае в монохромном формате (где каждый пиксель обозначается его яркостью). Для примера попробуем отобразить одну из строк

```
> displayRow(X(1,:));
```

Результат должен получиться такой, как показано на рис. 2.4.

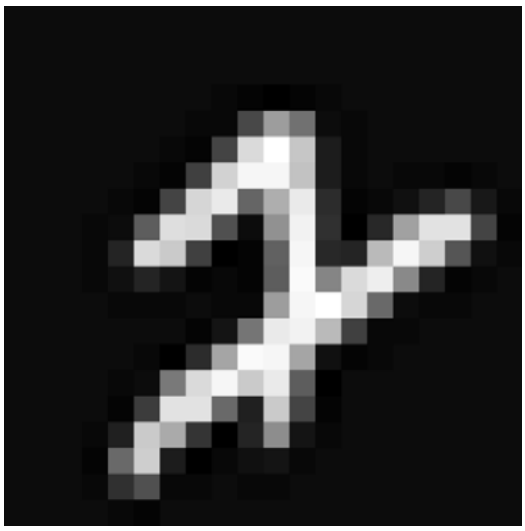

Рис. 2.4. Отображение цифры

4. Обучающий классификатор

Теперь напишем основную функцию, которая позволит нам получить сразу множество классификаторов для всех возможных классов значений *Y*

```
function [all_theta] = oneVsAll(X, y, num_labels, lambda)

% Some useful variables
m = size(X, 1);
n = size(X, 2);
% You need to return the following variables correctly
all_theta = zeros(num_labels, n + 1);
% Add ones to the X data matrix
X = [ones(m, 1) X];
options = optimset('GradObj', 'on', 'MaxIter', 50);
initial_theta = zeros(n+1,1);
for i = 1:num_labels
    all_theta(i,:) = fmincg (@(t)(costFunctionReg(t, X, (y == i), lambda)), ...
                initial_theta, options);
endfor
end
```

Логика очевидна: мы указываем количество возможных классов (от 1 до 10) и вычисляем для каждого класса его значение θ. Чтобы это запустить, нужно выполнить следующую команду:

```
> all_theta = oneVsAll(X,Y,10, 0.001);
```

5. Предсказание с помощью обучающего классификатора

Теперь напишем функцию, которая для набора строк для каждой строки вернет наиболее подходящий класс

```
function p = predictOneVsAll(all_theta, X)
m = size(X, 1);
num_labels = size(all_theta, 1);

% Add ones to the X data matrix
```

```
X = [ones(m, 1) X];
result = X*all_theta';
[M,I] = max(result, [], 2);
p = I;
end
```

Чтобы определить точность, нужно подсчитать количество совпадений между результатом этой функции и исходным *Y*. Это задание выполните самостоятельно.

Варианты задания

Для каждой задачи нужно составить скрипт, который подготовит тестовый набор данных и отобразит его в двух- или трехмерной плоскости.

1. Имеются две входные переменные *x* и *y*. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, x \le 0 \vee x \ge 2 \\ 1, y \le 0 \wedge x \in [0;2] \\ 2, otherwize. \end{cases}$$

2. Имеются две входные переменные *x* и *y*. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, x \le 0 \vee x \ge 2 \\ 1, y^2 \le 100 \wedge x \in [0;2] \\ 2, otherwize. \end{cases}$$

3. Имеются две входные переменные *x* и *y*. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, x^2 + y^2 \le 1 \\ 1, x^2 + y^2 \in [0;2] \\ 2, otherwize. \end{cases}$$

4. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, x \in [0;2] \\ 1, y \leq 0 \ \wedge x \notin [0;2] \\ 2, otherwize. \end{cases}$$

5. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом

$$class(x,y) = \begin{cases} 0, x \in [0;2] \\ 1, y^2 \leq 100 \ \wedge x \notin [0;2] \\ 2, otherwize. \end{cases}$$

6. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, x^2 + y^2 \leq 1 \\ 1, x^2 + y^2 \in [1;3] \\ 2, otherwize. \end{cases}$$

7. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, \sin(x) \leq \cos(y) \\ 1, \sin^2(x) + \cos^2(y) \leq 1 \\ 2, otherwize. \end{cases}$$

8. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, \sin^2(x) + \cos^2(y) \in [0;0.25] \\ 1, \sin^2(x) + \cos^2(y) \in [0.25;0.75] \\ 2, otherwize. \end{cases}$$

9. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, & x \le \ln y \\ 1, x \le 2 * \ln y \\ 2, otherwize. \end{cases}$$

10. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, & \ln(x * y) \le 5 \\ 1, \ln(x) + \ln(y) \le 5 \\ 2, otherwize. \end{cases}$$

Содержание работы

1. Цель работы
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Что возвращает функция *max* от трех аргументов в *predictOneV s All?*
2. Дайте определение многокритериальной классификации.
3. Как определить точность распознавания?
4. Что такое функция стоимости и для чего она нужна?

Лабораторная работа № 5
ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ

Цель работы: познакомиться с эволюционными алгоритмами.

Генетические алгоритмы в той или иной вариации были известны как минимум с 50-х годов XX века, однако по большей части они опирались на операцию одной лишь мутации, что не могло привести их к успеху в общем случае. В 1975 году американский ученый Д. Холланд показал, как одновременное использование операции кроссовера и мутации в ходе эволюции может привести к нахождению успешного решения множества задач. В своей работе Холланд представил математическое

обоснование того, почему эволюция успешна именно в том случае, когда использует именно мутации и скрещивание особей.

Перечислим основные этапы решения задачи с помощью генетического алгоритма.

- На первом шаге выбирают схему представления особи. Схема должна однозначно позволять восстанавливать из генетического описания особи то решение, которое мы ищем. К примеру, пусть мы хотим представить в виде набора генов формулу $f(x)=ax^2 + bx + c$. Одним из способов представления будет такое: пусть особь состоит из 192 бит. Первые 64 бита соответствуют представлению числа a в двоичном коде, следующие 64 бита соответствуют b, и последние 64 бита соответствуют c. Выбор формата в данном случае является произвольным и опирается на стандартную архитектуру *Intel* 64-разрядных платформ, и может отличаться от реальной задачи. Важно, что по представлению особи можно однозначно восстановить функцию $f(x)$, которой она соответствует с конкретными коэффициентами. Кроме того, можно избежать чрезвычайно больших и чрезвычайно малых значений коэффициентов, ограниченных каким-то диапазоном. Пусть заранее известно, что a принадлежит диапазону от 0 до 100. Тогда значение a (которое представлено числом a' от 0 до $2^{64} - 1$) можно вычислить как $100(a'/2^{64})$.
- На втором шаге генерируют начальную популяцию определенного размера N. Обычно N выбирают в диапазоне от 500 до нескольких тысяч. Начальную популяцию генерируют с помощью генератора случайных чисел. После начинается процесс эволюции. Какие генераторы следует использовать, зависит от конкретной задачи. Например, в качестве генератора можно использовать равномерное распределение в диапазоне от 0 до 100, если можно предположить заранее, что для решения все коэффициенты будут укладываться в этот диапазон.
- Для каждой популяции в особи вычисляется ее фитнес-функция, или численное представление приспособленности конкретной функции.
- На основе текущего поколения создается новое. Чем выше приспособленность особи, тем больше ее шансы попасть в следующие поколения.
- При операции простого клонирования особь переходит в новое поколение без изменений. Если ограничиться только этой операцией, то в ходе эволюции популяция придет к результату, когда все особи

являются копией, наиболее приспособленной из тех, которыми мы располагали изначально.

• При операции мутации особь переходит в новое поколение с незначительными изменениями, т. е. небольшое количество битов в представляемой особи поменяет знак на противоположный: с нуля на единицу или наоборот. В этом случае эволюция может находить оптимальные решения, но это будет происходить медленно, и поэтому нужно использовать третью операцию.

• При операции кроссовера из поколения выбирают две особи. Особи не должны быть одинаковыми, так как в этом случае кроссовер не даст эффекта. Затем выбирают точку разрыва. Начало первой особи склеивают с концом второй особи или наоборот. Полученные потомки переходят в новое поколение.

Виды фитнес-функций:

стандартная: $f_s(x) = \max - f_r(x)$;

смещенная: $f_a(x) = 1 / 1 + f_s(x)$;

нормализованная: $f_n(x) = f_a(x) / \sum f_a(x_i) \;\; i = 1 \ldots N.$

Ход работы

Для начала создадим изображение земли черного цвета

```
earth = zeros(100, 3);
imshow(earth);
```

Результат представлен на рис. 2.5.1.

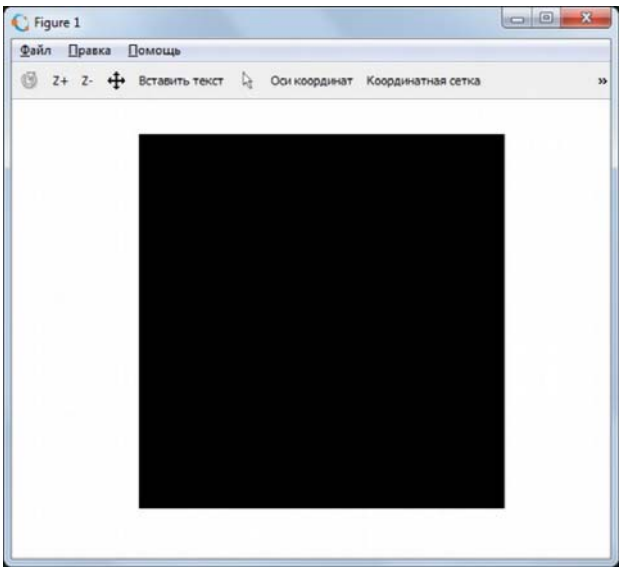

Рис. 2.5.1. Земля черного цвета

Сделаем землю более зеленой, причем с вариациями. Определим следующую функцию:

```
function variated = variate(matr, delta)
  m = size(matr,1);
  n = size(matr,2);
  variated = matr + rand(m,n)*2.0*delta-delta;
  variated = min(variated, ones(m,n));
```

Данная функция вносит небольшую вариацию в имеющуюся матрицу цветов. Применим ее к земле

```
earth(:, 2) = 1;
earth = variate(earth, 0.1);
imshow(earth);
```

Результат представлен на рис. 2.5.2.

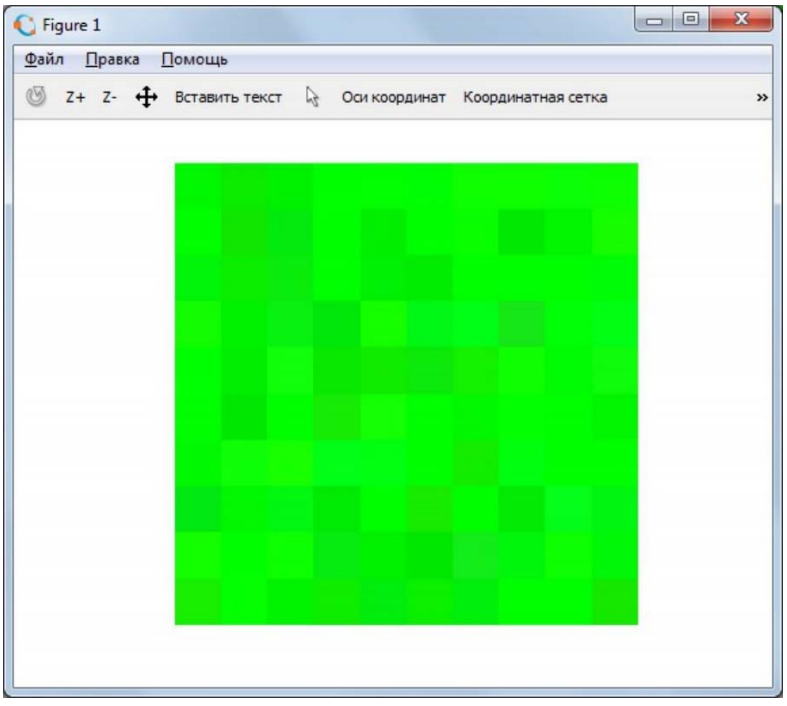

Рис. 2.5.2. Зеленая лужайка

Для интереса прогоним цикл и посмотрим, что будет, если применять эту мутацию цвета дальше

```
earth_random = earth;
for i = 1:10000
  earth_random = variate(earth_random, 0.01);
end
imshow(earth_random);
```

Результат представлен на рис. 2.5.3.

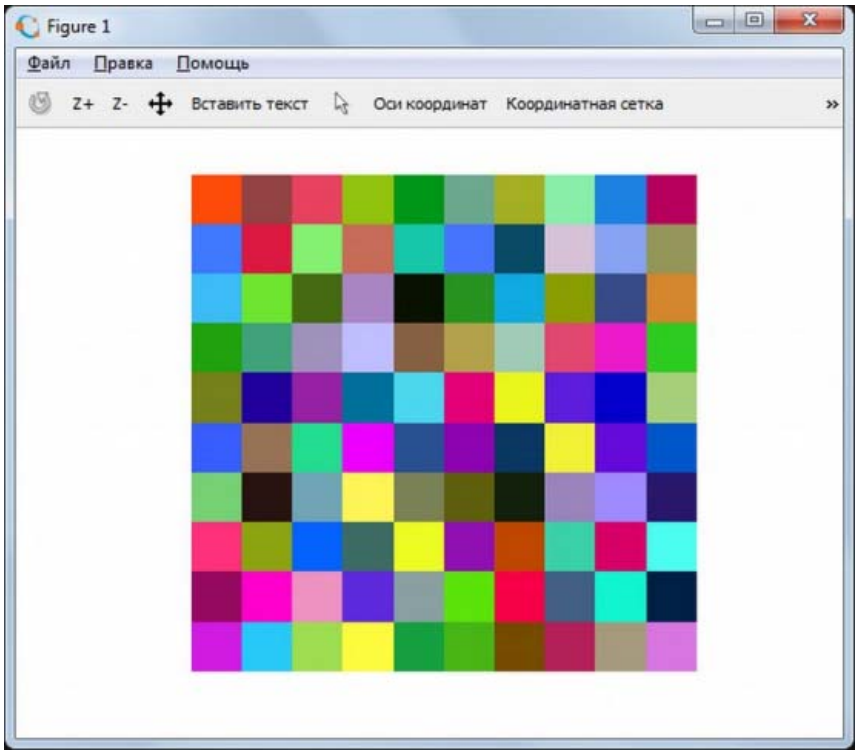

Рис. 2.5.3. Зеленая лужайка и хаос

Нетрудно заметить, что сама по себе мутация приводит изначально однородную структуру к полному дисбалансу. Поэтому поступим иначе: создадим изначально разнородный набор животных и посмотрим, что с ним произойдет, если к нему применять мутацию в связке с естественным отбором

```
animals = rand(400, 3);
imshow(animals);
```

Результат представлен на рис. 2.5.4.

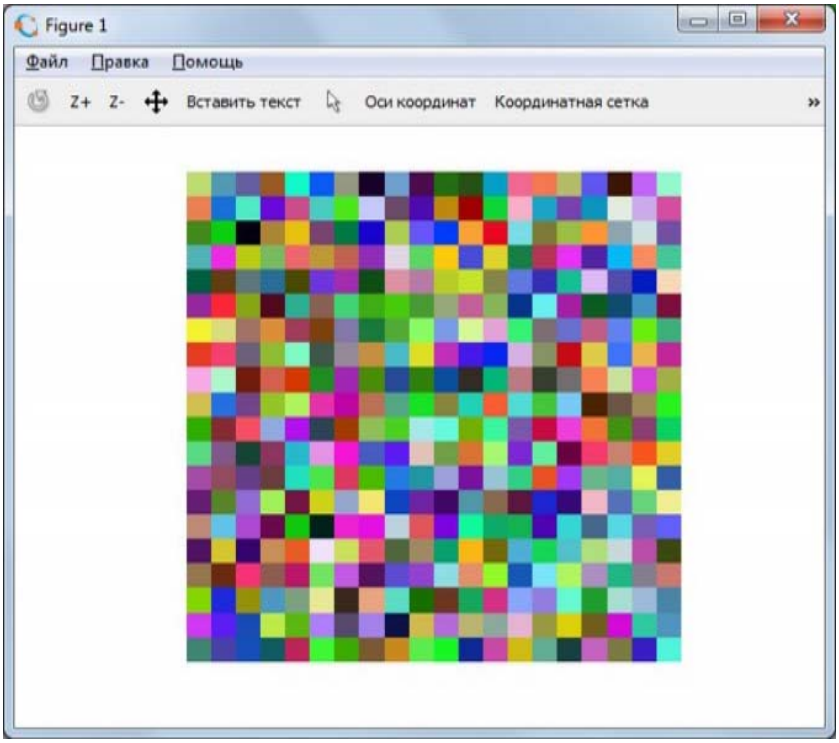

Рис. 2.5.4. Изначальная популяция животных

Теперь будем оценивать приспособленность окраса животного к зеленой траве в условиях выживания среди хищников как разницу между его цветом и зеленым цветом лужайки с помощью следующей функции:

```
function f = fitness(animals, colour)
  delta = animals - colour;
  delta = delta.^2;
  f = sum(delta')';
```

Чем меньше разница, тем более высокий процент выживания животного. А теперь включим естественный отбор. На каждом этапе эволюции специальный отряд хищников будет убивать десять животных ($\frac{1}{40}$ от всей популяции), причем выбирая именно наиболее выделяющихся окрасом

```
function evolved = evolve(animals, colour, kill)
   m = size(animals, 1);
   f = fitness(animals, colour);
   threshold = sort(f, 'descend')(kill+1);
   survivors = animals(find(f<=threshold), :);
   p = randperm(m-kill);
  evolved = [survivors;survivors(p(1:kill),:)];
```

А теперь устроим несколько итераций эволюции

```
for i = 1:20
  animals = evolve(animals, [0,1,0], 10);
  animals = variate(animals, 0.00001);
end
imshow(animals);
```

Результаты представлены на рис. 2.5.5.

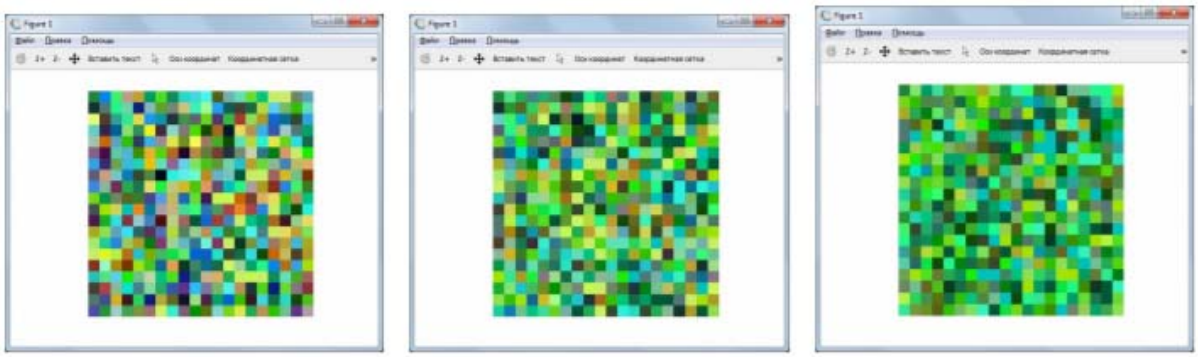

Рис. 2.5.5. Результат эволюции (20, 40, 60 поколений)

Как видно, цвета становятся все менее пестрыми с ходом эволюции, поскольку яркая окраска делает животных более заметными для хищников. Посмотрим, что будет по итогам нескольких сотен поколений (рис. 2.5.6).

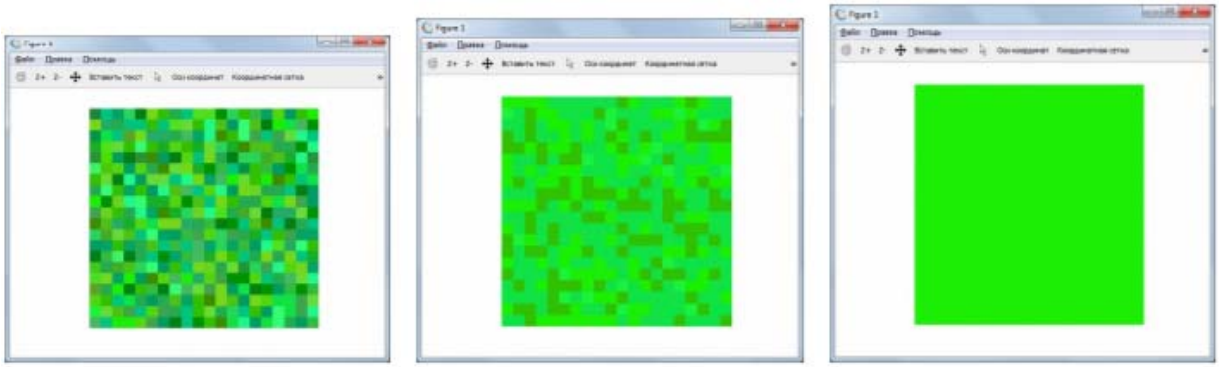

Рис. 2.5.6. Результат эволюции (100, 200, 400 поколений)

Что и требовалось доказать: естественный отбор действительно работает, и его можно использовать для решения более практических задач.

Варианты задания

Независимо от варианта задания необходимо разработать на основе приведенного алгоритма свой алгоритм, отвечающий заданным условиям. Для каждого поколения необходимо получить статистические показатели для фитнеса:

- наилучший показатель;
- наихудший показатель;
- средний показатель;
- среднеквадратичное отклонение.

Изменения всех показателей, кроме отклонения, надо показать на графике. Помимо этого, с помощью эволюции необходимо решить задачу предсказания, т. е. животное будет тем приспособленнее, чем меньше его суммарная ошибка на обучающей выборке

1) $z = x^2 + y^2$;
2) $z = x^2 + xy + 1$;
3) $z = x^2 + y^2 + 2xy$;
4) $z = x^3 + y^3$;
5) $z = x + \frac{y}{x}$;
6) $z = x + \log(y^2)$;
7) $z = \sin(y) + x$;
8) $z = \cos(x) + \sin(y)$;
9) $z = \cos(x)^2 + \sin(y)$;
10) $z = x^2 + 2x + y$.

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Дайте определение генетических алгоритмов.
2. Каковы основные операции генетических алгоритмов?
3. Каковы количественные характеристики успешности работы генетического алгоритма?
4. С помощью какой функции можно вносит вариацию в имеющуюся матрицу цветов?

Лабораторная работа № 6
НЕЙРОСЕТЕВОЕ РАСПОЗНАВАНИЕ ПЕЧАТНЫХ СИМВОЛОВ В СРЕДЕ *MATLAB*

Цель работы

1. Построение нейронных сетей в среде *MATLAB*.
2. Исследование возможностей распознавания печатных символов с помощью нейронных сетей.

Теория

Искусственные нейронные сети (НС) представляют собой математическую модель функционирования биологических НС – сетей нервных клеток живого организма. Как и в биологической НС, основным элементом искусственной НС является нейрон. Соединенные между собой нейроны образуют слои, количество которых может варьироваться в зависимости от сложности НС и решаемых ею задач. Теоретические основы программирования таких НС описываются во многих работах [7; 10; 11].

Одна из актуальных задач распознавания образов – распознавание визуальных образов. Машины, способные распознавать рукописный текст и рисунки на бумаге, символы на банковских карточках или чеках существенно облегчают человеческий труд и ускоряют рабочий

процесс, и при этом снижают риск ошибки за счет отсутствия человеческого фактора [6].

Цель лабораторной работы – создать НС, которая сможет распознавать визуальные образы букв русского алфавита. Программный код, который решает подобную задачу, присутствует в системе *Matlab* как демонстрационная программа с названием *appcr1*. Подробно этот код разобран и пояснен в книге «Нейронные сети» В. С. Медведева, В. Г. Потемкина [6], а также описан в работе И. С. Миронова, С. В. Скурлаева [7].

В системе *Matlab* также присутствует инструмент *NNtool*, имеющий графический интерфейс пользователя, который существенно облегчает задачу и может быть легко использован даже неопытным пользователем. Подробно этот инструмент описан в работах В. Иванникова, А. Ланнэ [5] и П. А. Сахнюка [8] и др. [12] В работе А. И. Шеремет, В. В. Перепелицы, А. М. Денисовой показан пример разработки НС для распознавания визуальных образов символов латинского алфавита с помощью *NNtool* [9]. Зарубежные ученые также применяют искусственные НС в своих исследованиях [13, 14].

Ход работы

1. Подготовка эталонных (обучающих) образов печатных символов в виде набора графических файлов.

Пример набора – последовательность из десяти цифр от 0 до 9. В этом примере число образов $M = 10$. В случае, когда каждый класс образов характеризуется лишь своим эталоном, мы имеем число классов, также равное M. Каждый образ формируется в виде графического файла в битовом формате. Тип файла (расширение) определяется используемыми в среде *MATLAB* типами графических файлов. Рекомендуется использовать расширение *tif*.

Для создания графических файлов образов удобно использовать среду *Adobe Photoshop*. В этом случае при создании каждого файла необходимо проделать следующую последовательность операций:

1) создать новый файл, задав его параметры: имя – *XXXX*; ширина $N1$ пикселей; высота $N2$ пикселей; цветовой режим – битовый формат. Значения $N1$, $N2 = 8\ldots 20$ задает преподаватель;

2) используя инструменты типа «Кисть», «Ластик» и другие, создать требуемый образ символа;

3) с помощью команды «Сохранить как» сохранить созданный образ в виде файла типа *tif*.

На рис. 2.6.1 приведены примеры графических символов цифр при $N1 = 10$, $N2 = 12$ пикселей.

Рис. 2.6.1. Примеры графических символов цифр

2. Создание и обучение НС в среде *MATLAB* и распознавание печатных символов с помощью обученной НС.

На данном этапе выполнение работы в среде *MATLAB* производится с помощью программы *sr_newff*, которая реализует следующие функции:

– формирование числовых массивов эталонных образов, используемых в качестве обучающих;

– подготовка данных, необходимых для создания НС;

– создание НС, задание параметров обучения НС и обучение НС.

Эталонный образ каждого символа представлен в виде вектора-столбца [N, 1], число элементов N которого равно числу признаков (иначе говоря, N – размерность пространства признаков). Такой вектор-столбец формируется из двумерного массива-изображения [$N1$, $N2$], который, в свою очередь, формируется при считывании графического файла образа с помощью команд

imread (*FILENAME*) – процедура чтения графического файла;

$X = reshape\ (A, [N, 1])$ – процедура преобразования двумерного массива A [$N1$, $N2$] в одномерный вектор-столбец X [N, 1], где $N = N1N2$.

Процедура умножения массива на единицу приводит к смене типа элементов массива с *logical* (для элементов битового формата) на *double*.

Для удовлетворительной работы НС недостаточно формирования лишь одного обучающего образа для каждого класса (типа символа) образов. Это связано с тем, что распознаваемые образы (на этапе работы НС – в режиме распознавания) всегда отличаются от обучающих по ряду причин:

– различие шрифтов и стилей печатных символов;

– погрешности сканирования и неточности совмещения символа и окна сканирования;

– низкое качество печати, дефекты бумаги и т. д.

В силу указанных причин для надежного распознавания образов НС следует обучать на достаточно представительном множестве образов, входящих в один и тот же класс. В программе *sr_newff* формирование дополнительных обучающих образов производится путем незначительного искажения эталонных образов, считываемых из графических файлов. Искажение образа-эталона каждого класса реализуется путем добавления к нему равномерного (по площади изображения) шума типа «Соль и перец», представляющего собой случайное искажение отдельных пикселей изображения. Степень искажения характеризуется числом p = [0; 1], определяющим долю искаженных пикселей. Такой подход при формировании образов позволяет, во-первых, быстро получать большое число обучающих образов и, во-вторых, регулировать (путем изменения значения p) степень разброса множества образов в пределах одного класса.

Подготовка данных, необходимых для создания НС, включает в себя:

1) формирование двумерного массива обучающих образов XR [N, K], каждый столбец которого представляет собой набор N признаков одного образа, а число столбцов K равно числу обучающих образов;

2) формирование двумерного массива желаемых откликов YR [NY, K], где NY – число выходов НС (т. е. число нейронов выходного слоя); K – число обучающих образов. Отклик YR[:, k] (в общем случае – вектор-столбец) соответствует k-му обучающему образу – вектору XR[:, k];

3) формирование двумерного массива $R[N, 2]$, определяющего минимальное $R(n, 1)$ и максимальное R(n, 2) значение n-го признака, $n = 1 \ldots N$. Создание НС. В общем случае НС *net* создается с помощью команды *net* = *nnnnn* (*P*1, *P*2 … *PL*), где *nnnnn* – тип НС; *P1*… *PL* – параметры НС.

Рассмотрим встроенную функцию *Matlab prprob*, которая представляет собой матрицу, содержащую набор признаков букв латинского языка. Каждая буква имеет размерность 7 × 5 пикселей.

Создадим подобную матрицу с буквами русского алфавита. Для этого разработаем в графическом редакторе шаблон каждого символа такой же размерности (рис. 2.6.2, 2.6.3).

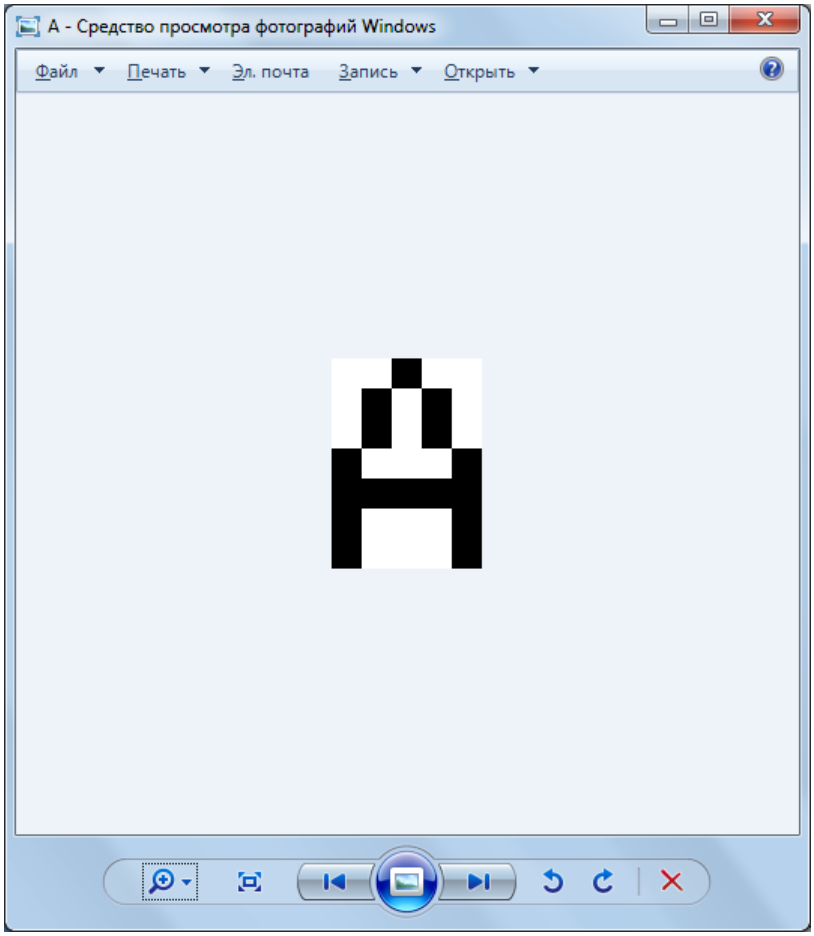

Рис. 2.6.2. Шаблон буквы А, созданный в графическом редакторе

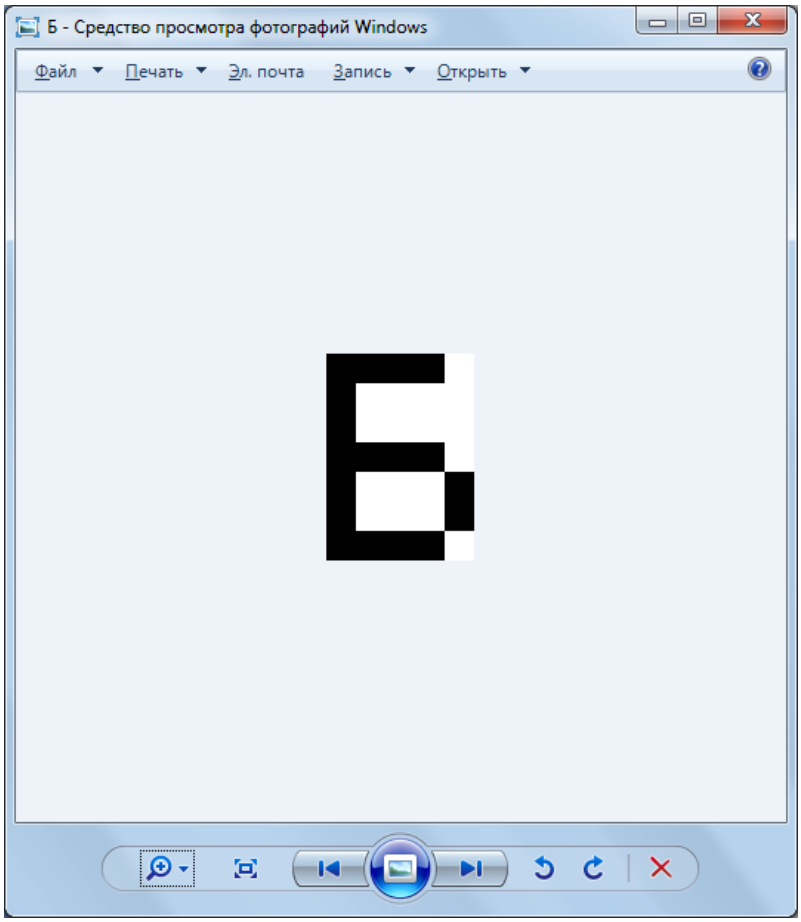

Рис. 2.6.3. Шаблон буквы Б, созданный в графическом редакторе

После того как созданы шаблоны для каждой буквы, необходимо написать функцию, которая будет считывать необходимые признаки символов с графического файла в нужном нам формате.

Для этого выберем в командном меню *FileàNewàFunction M-file*. Откроется графический редактор, в который необходимо вставить приведенный ниже код.

Код функции *ImgRead*

```
function y = Imgread(x)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
img=imread(x);
img1=img(:,:,1)';
```

```
img2=reshape (img1,[35, 1]);
for i=1:35
if (img2(i,1)==0)
img2(i,1)=1;
end
end
for i=1:35
if (img2(i,1)==255)
img2(i,1)=0;
end
end
y=img2;
end
```

Используя данную функцию, создадим матрицу признаков русского алфавита, введя в окно команд следующий код:

```
images1=Imgread('C:\alphabet\A.png');
…
images33=Imgread('C:\alphabet\Я.png');
RA=[images1,images2,images3,images4,images5,images6,images7,
images8,...
images9,images10,images11,images12,images13,images14,images15,
images16,...
images17,images18,images19,images20,images21,images22,images23,...
images24,images25,images26,images27,images28,images29,images30,
images31,...
images32,images33];
```

Заметим, что для того, чтобы код работал, шаблоны должны располагаться в каталоге *C:\alphabet*; в качестве имени должен быть сам символ, с разрешением *png* или данные об изменении расширения, например в формате *jpeg*, в код.

Теперь есть матрица *RA*, которая содержит в себе набор признаков русского алфавита, и будет использоваться в роли входных данных при создании НС.

В качестве матрицы целей создадим единичную матрицу размерностью 33 × 33.

Для этого введем код. Также объявим переменные, содержащие в себе количество строк и столбцов, как в таблице.

Создадим переменные для НС в *NNtool*

```
P=double(RA);
T=eye(33);
[R,Q] = size(P);
[S2,Q] = size(T);
```

Для обучения сети нам понадобятся данные с шумом. Создадим эти данные, введя следующий код в окно команд:

```
%Создаем переменные для обучения на зашумленных данных нейронной сети в
%NNtool
P1=P;
T1=T;
for i=1:100
P1=[P1,P+rand(R,Q)*0.1,P+rand(R,Q)*0.2];
T1=[T1,T,T];
end
```

Для симуляции сети нам понадобится переменная, содержащая в себе набор признаков одной буквы, например буквы И.

Введем соответствующую команду, заодно выведем на экран получившийся зашумленный образ (рис. 2.6.4).

```
noisy10 = P(:,10) + randn(35,1)*0.2;
plotchar(noisy10); % Зашумленный символ И
```

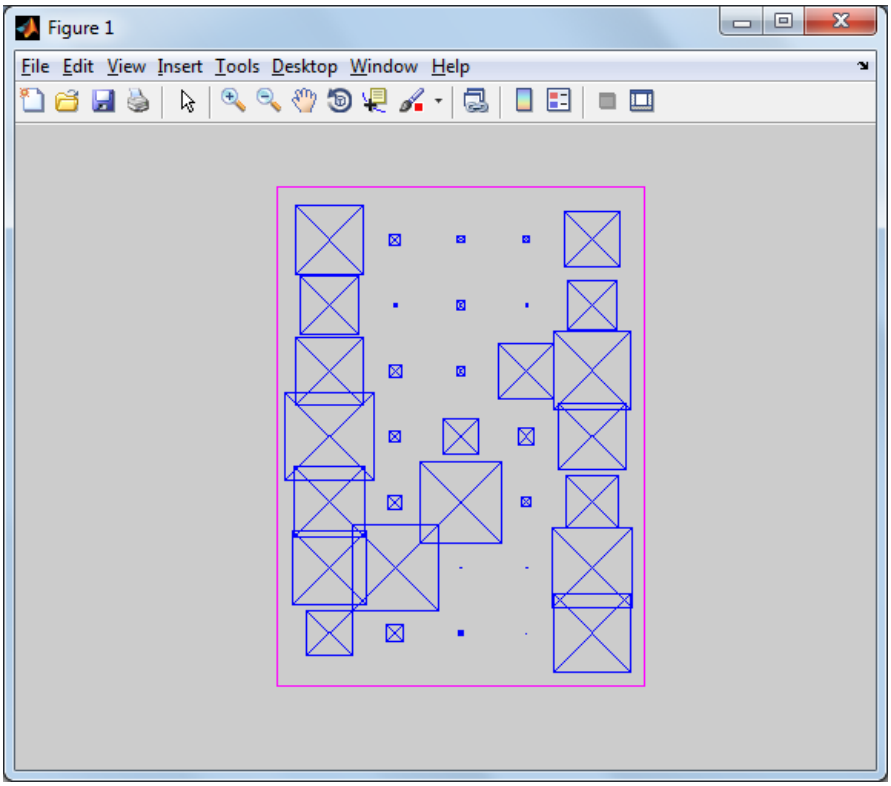

Рис. 2.6.4. Зашумленный образ буквы И

Теперь вызовем инструмент *NNtool*, где с помощью графического интерфейса создадим НС. Сделать это можно с помощью соответствующей команды *nntool* (рис. 2.6.5).

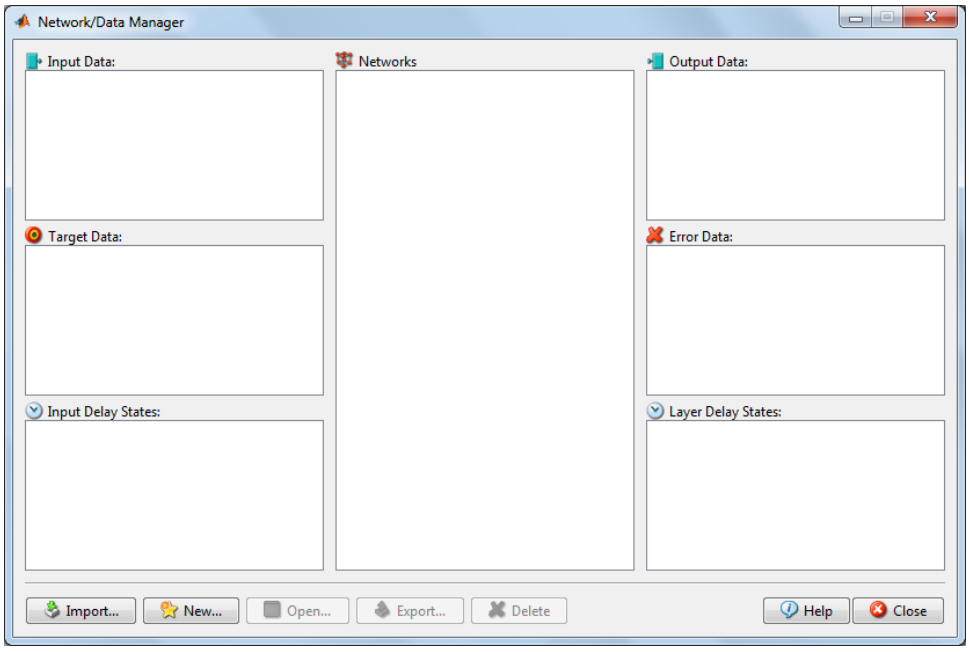

Рис. 2.6.5. Окно менеджера данных НС

С помощью кнопки *Import* добавляем необходимые нам переменные (рис. 2.6.6).

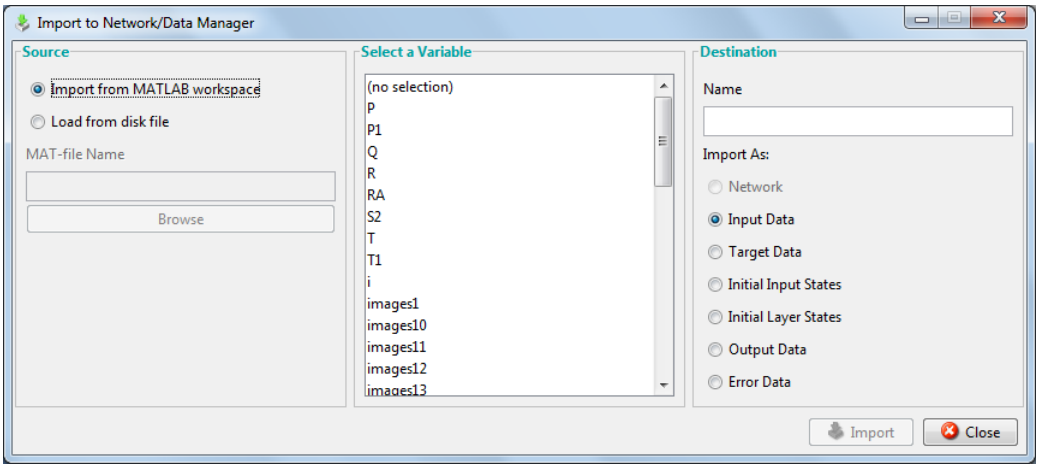

Рис. 2.6.6. Окно импорта данных

Переменные *P*, *P*1 и *noisy*10 добавляем как *input data*.

Переменные *T*, *T*1 добавляем как *Target data*.

После импорта всех переменных окно должно выглядеть как на рис. 2.6.7.

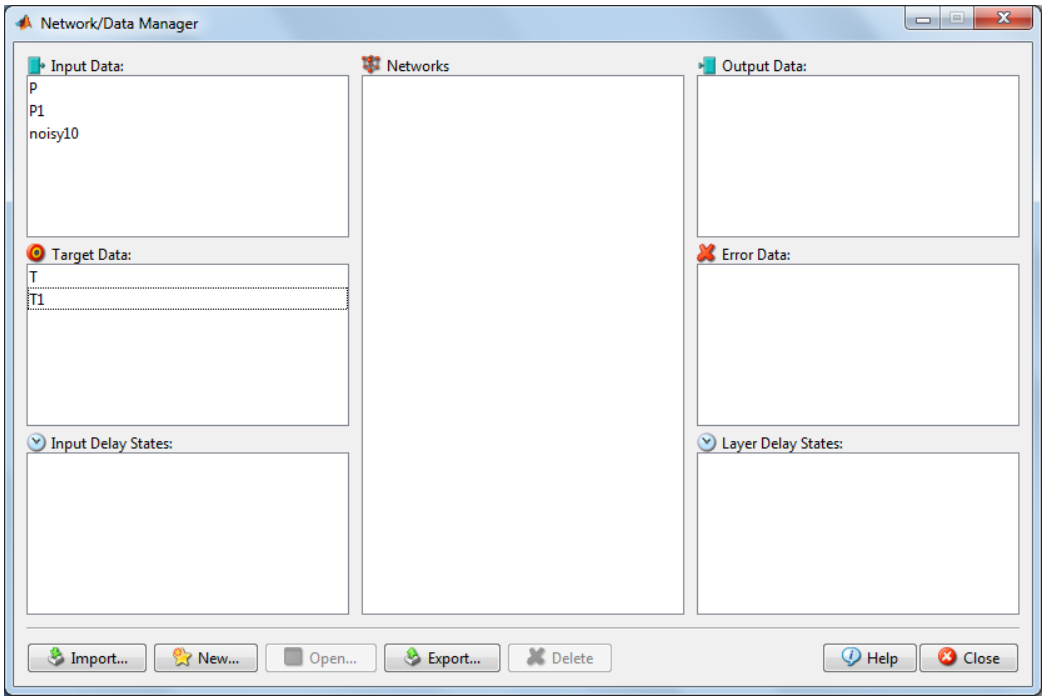

Рис. 2.6.7. Окно менеджера данных НС
после импорта переменных

Нажав кнопку *New*, приступим к созданию НС. В окне параметров НС введем настройки (рис. 2.6.8).

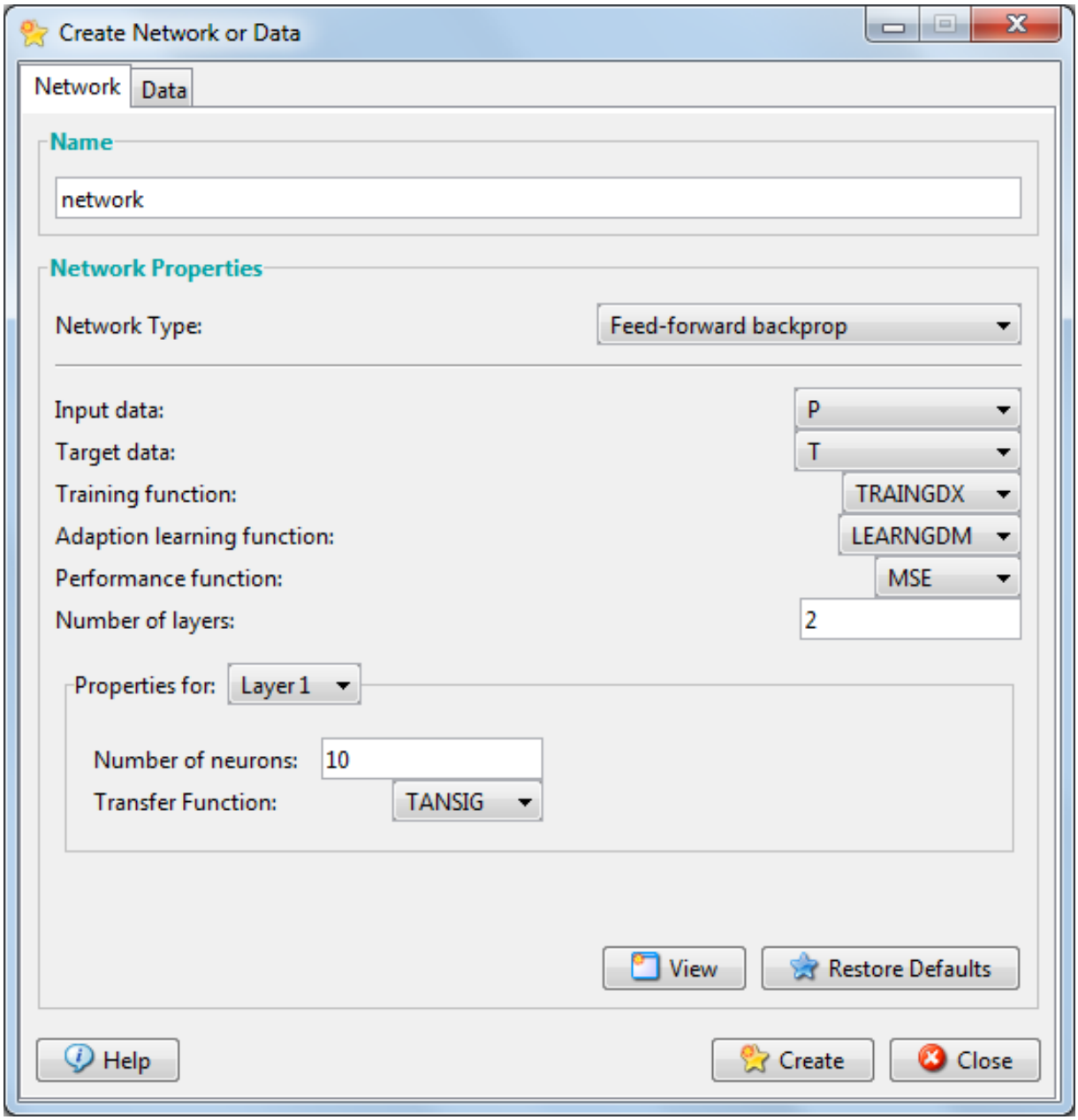

Рис. 2.6.8. Окно создания НС

После выбранных параметров создадим сеть, нажав кнопку *Create*.

После этого сеть должна появиться в окне менеджера данных НС в разделе *networks* (рис. 2.6.9).

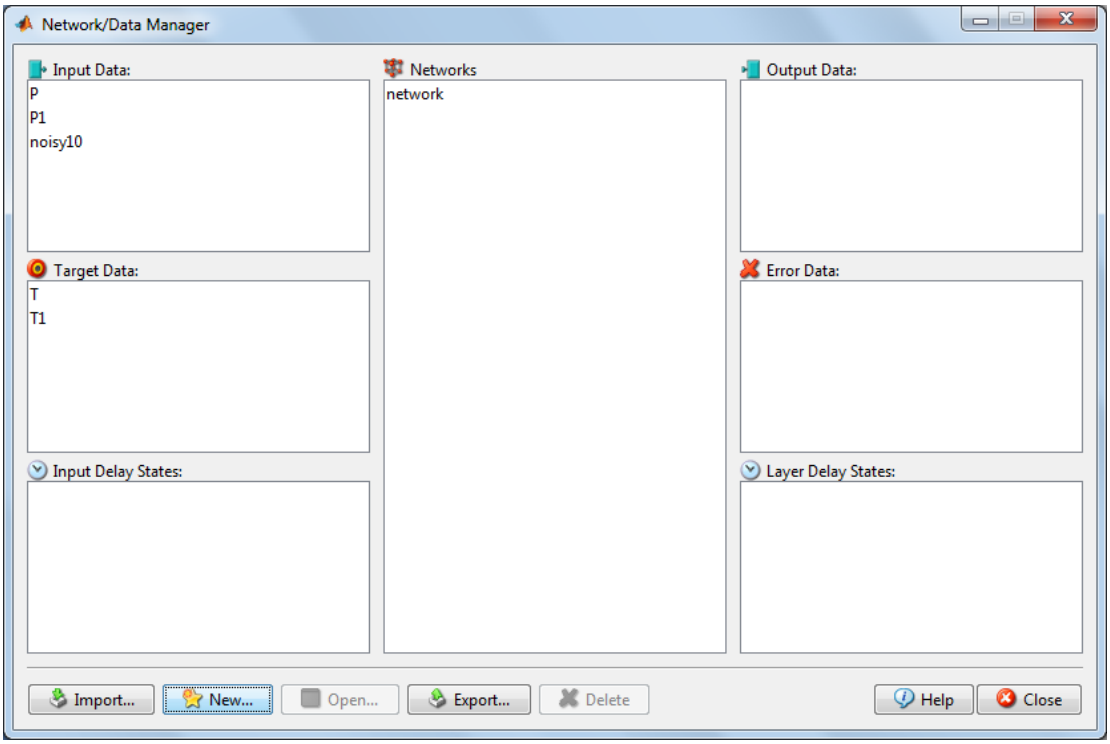

Рис. 2.6.9. Окно менеджера данных НС
после создания нейронной сети

Нажав дважды на созданную сеть, откроем НС. Появится окно с вкладками (рис. 2.6.10), в которых можно посмотреть структуру НС, обучить ее, провести симуляцию, изменять веса входных данных.

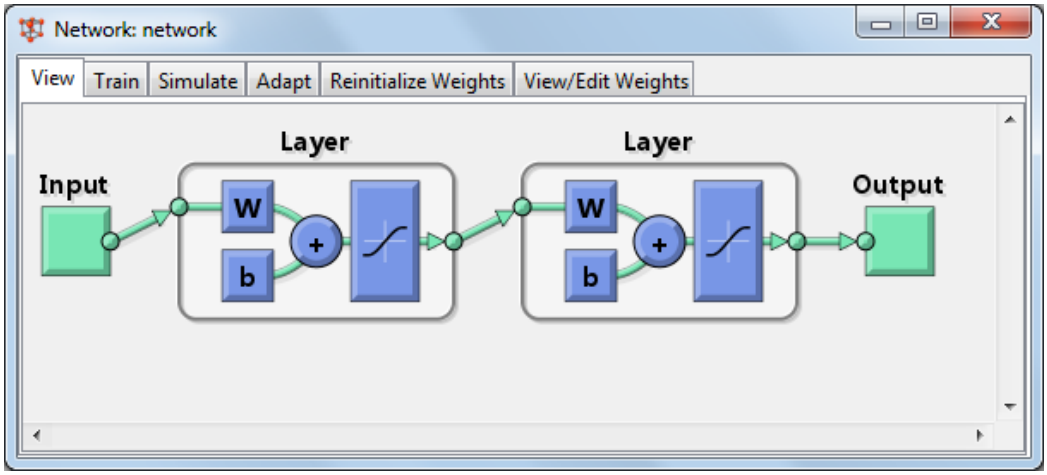

Рис. 2.6.10. Структура НС

Рассмотрим вкладку *Train*, где будет проводиться обучение НС. Вначале проведем обучение на идеальных данных. Введем соответствующие параметры (рис. 2.6.11).

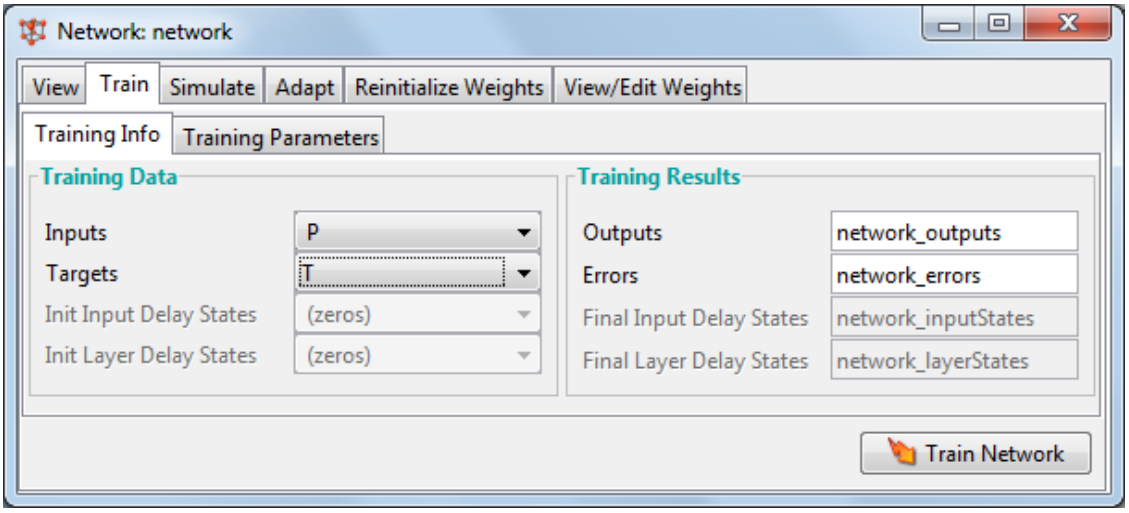

Рис. 2.6.11. Настройка данных, с помощью которых будет происходить обучение

После установки всех параметров запускаем обучение, нажав кнопку *Train Network* (рис. 2.6.12).

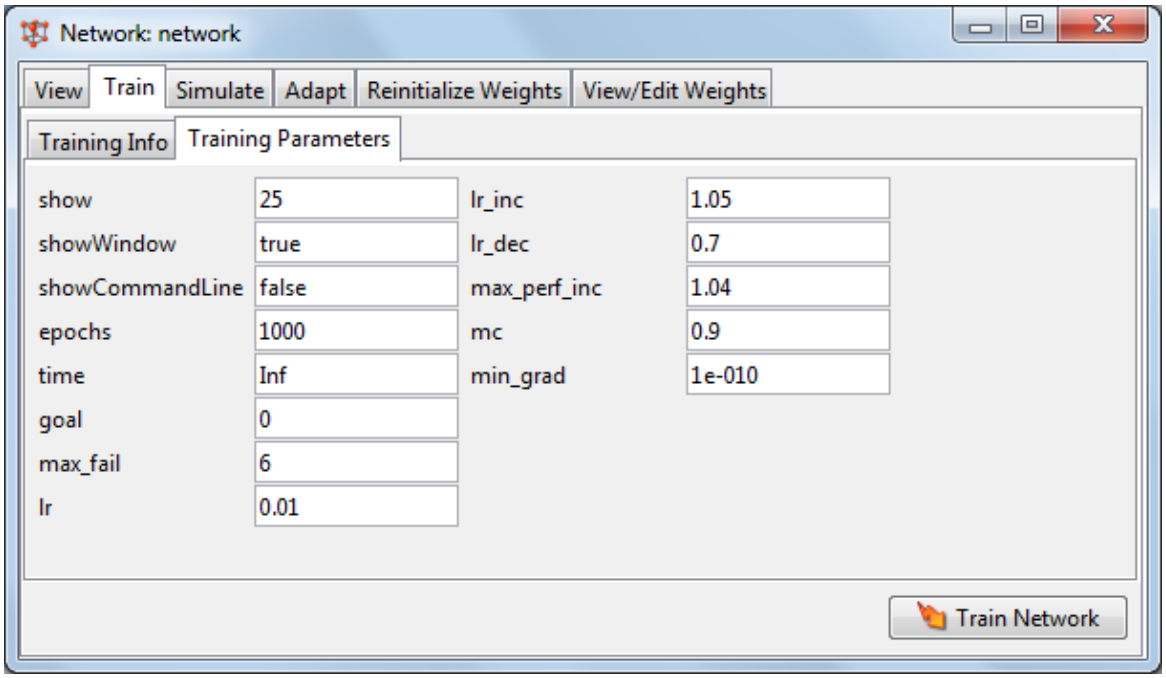

Рис. 2.6.12. Параметры обучения НС

В появившемся окне (рис. 2.6.13) можем наблюдать процесс обучения НС.

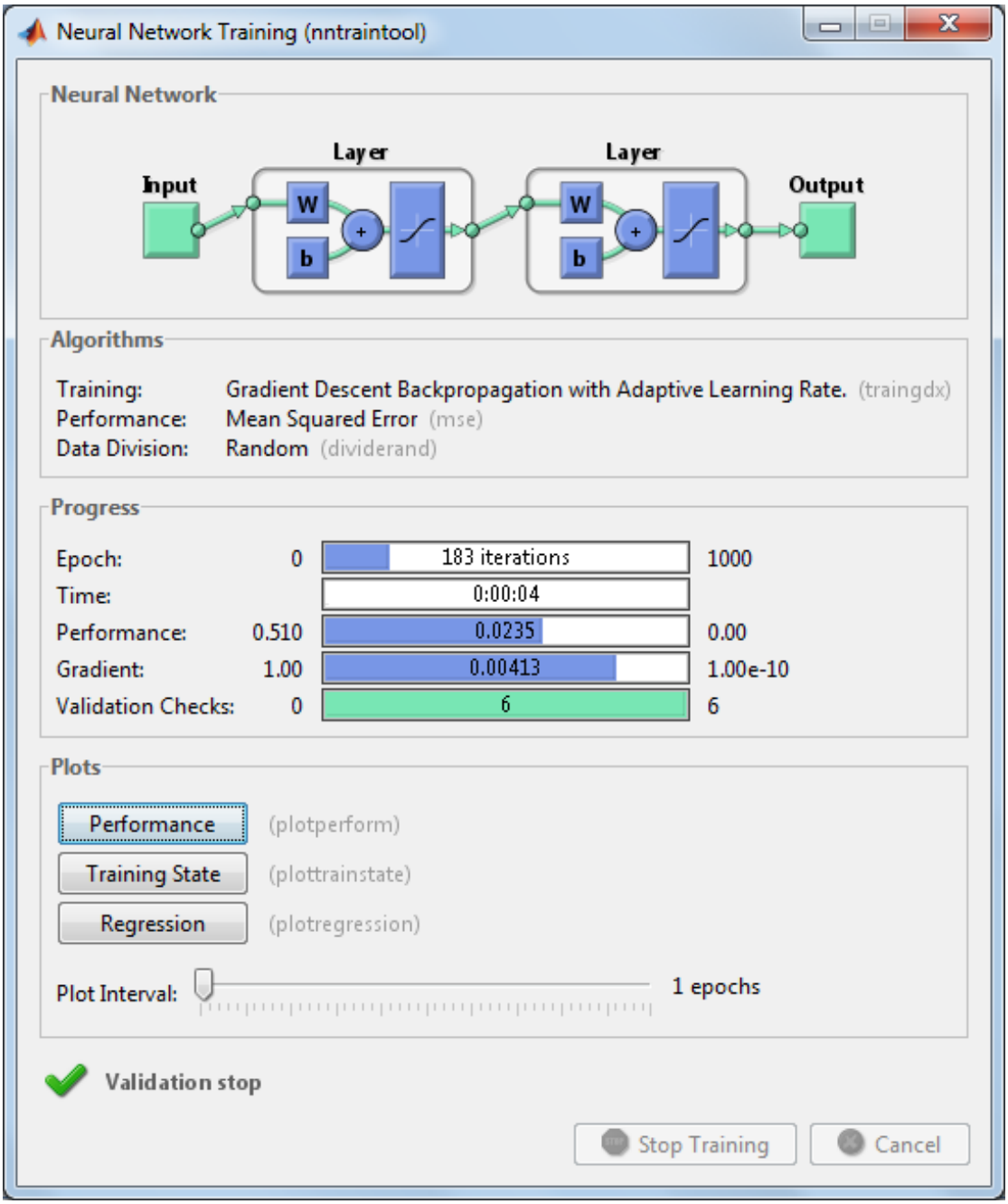

Рис. 2.6.13. Процесс обучения НС

С помощью кнопки *Performance* можно просмотреть процесс обучения с помощью графика (рис. 2.6.14).

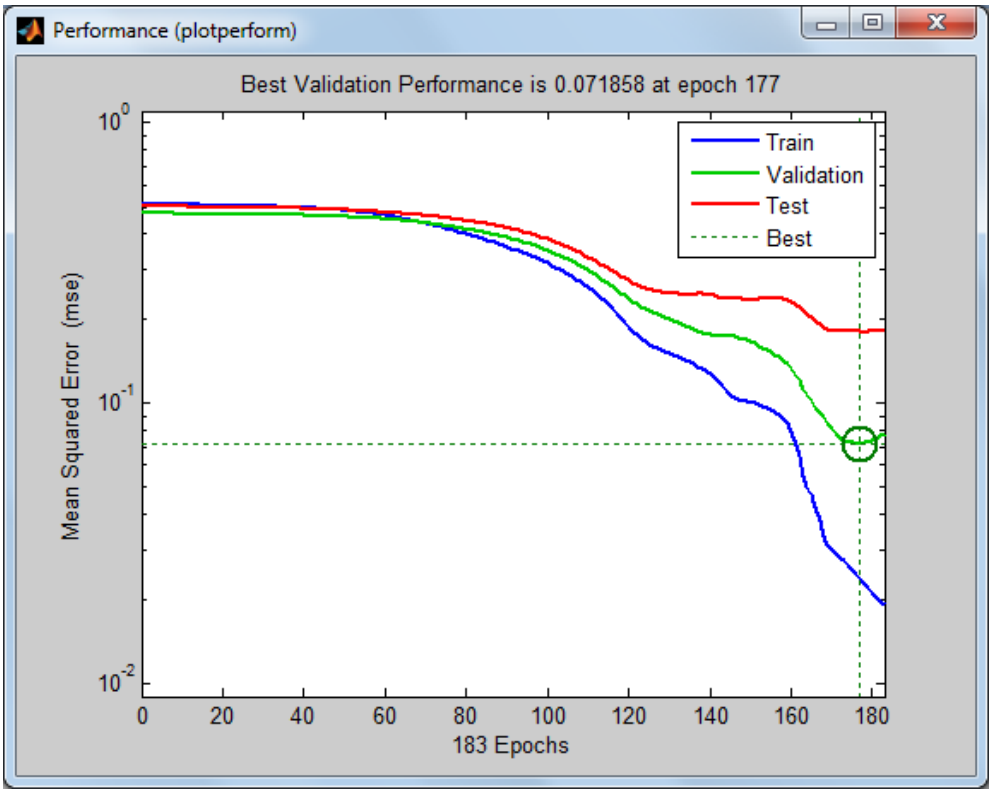

Рис. 2.6.14. График обучения НС

Теперь необходимо провести обучение на данных с шумом. Для этого изменим параметры во вкладке *train* (рис. 2.6.15, 2.6.16).

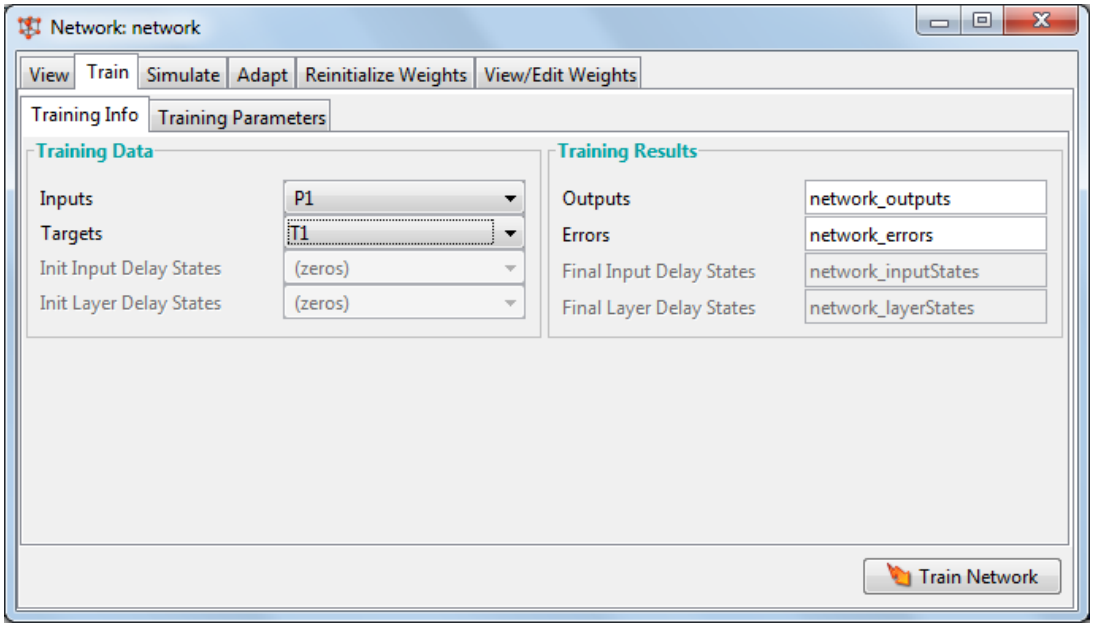

Рис. 2.6.15. Изменение данных обучения с шумом
на вкладке Train- Training Info

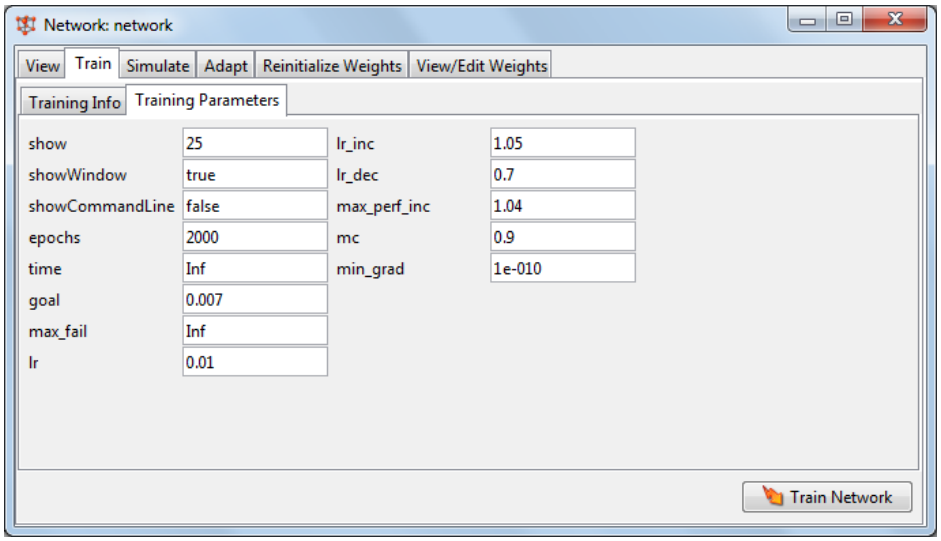

Рис. 2.6.16. Изменение параметров обучения с шумом НС на вкладке Train- Training Parameters

Окно процесса обучения будет выглядеть как рис. 2.6.17.

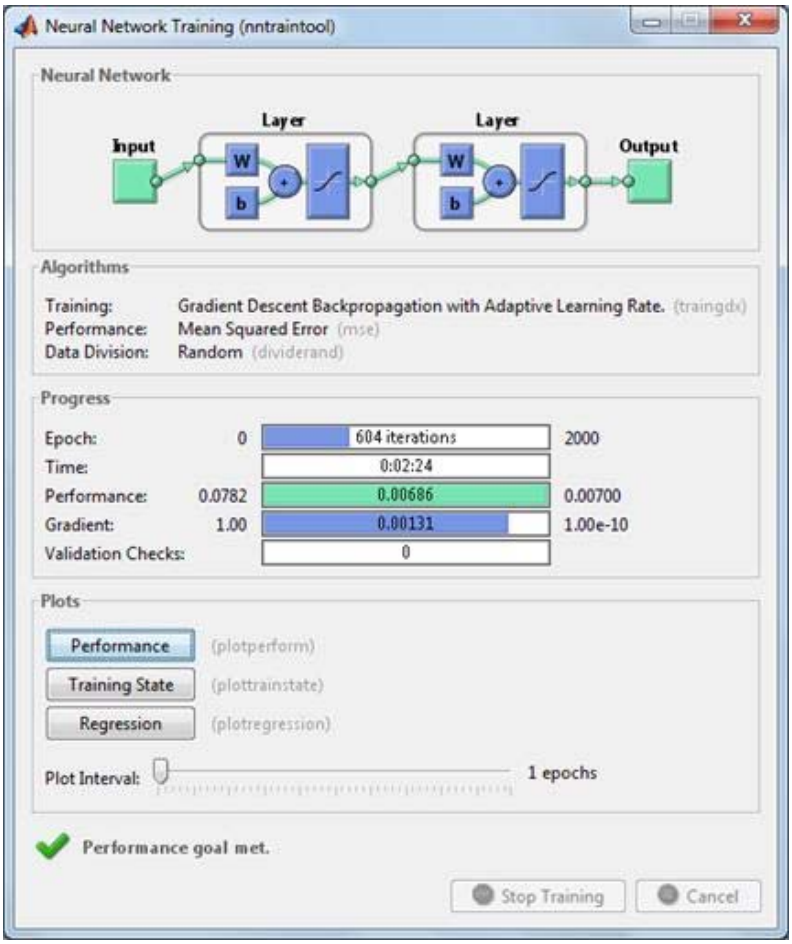

Рис. 2.6.17. Процесс обучения НС на данных с шумом

График обучения представлен на рис. 2.6.18.

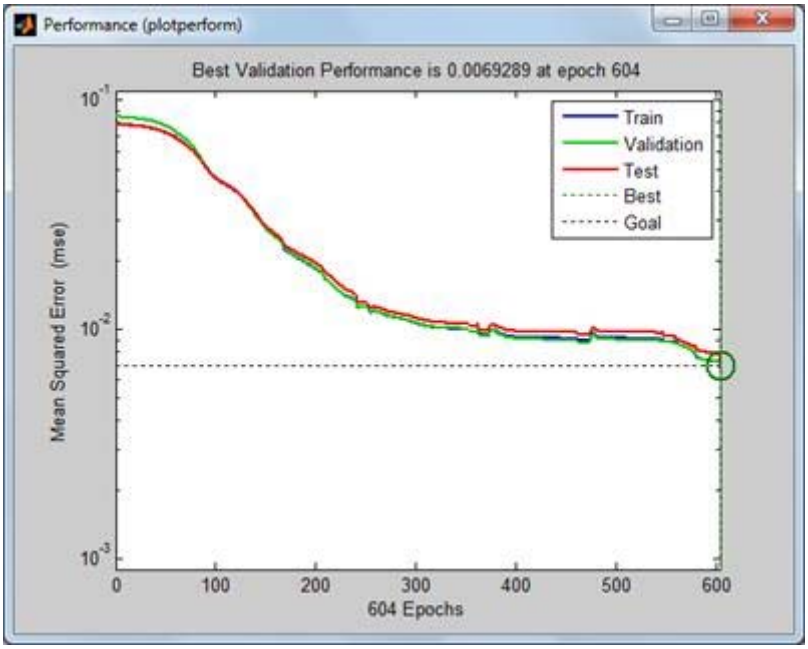

Рис. 2.6.18. График обучения НС на данных с шумом

В результате сеть обучена. Теперь необходимо проверить нейронную сеть. Для этого создана переменная *noisy*10, которая содержит в себе символ «И» с шумом.

Перейдем на вкладку *Simulate*, в окне созданной сети (рис. 2.6.19).

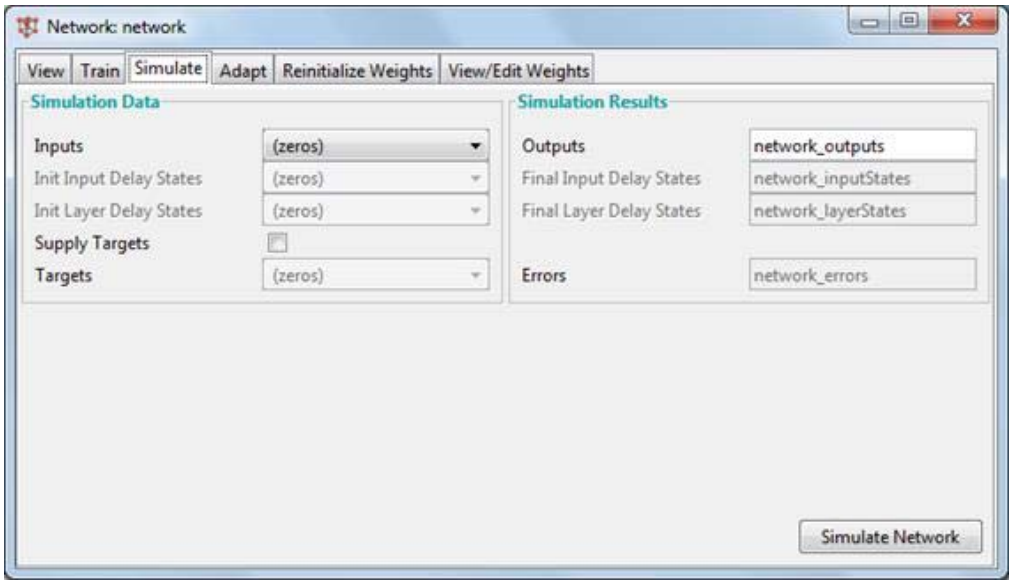

Рис. 2.6.19. Вкладка Simulate

Выберем в качестве входных данных переменную *noisy*10, а в качестве выходных данных напишем переменную *Ans* (рис. 2.6.20).

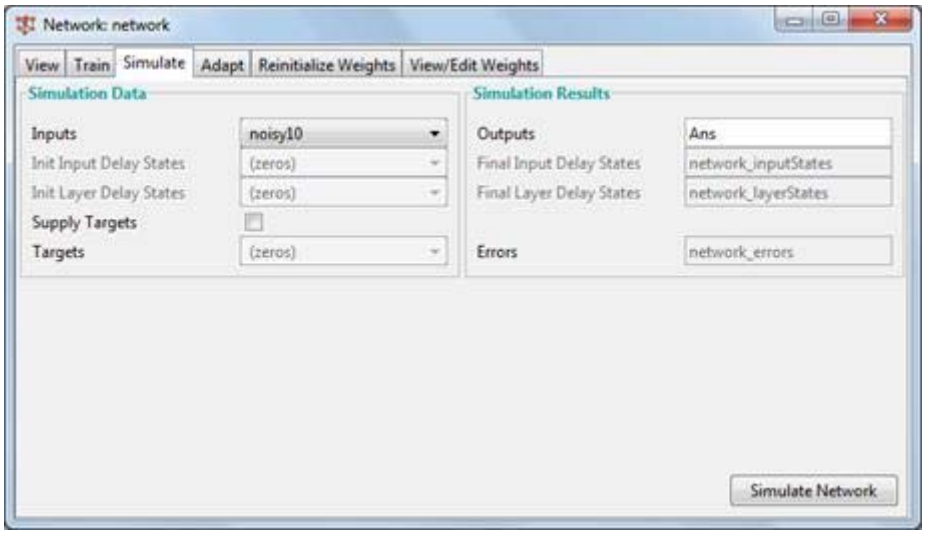

Рис. 2.6.20. Вкладка Simulate с выбранными параметрами

Для начала процесса симуляции необходимо нажать кнопку *Simulate Network*.

После этого в окне менеджера данных НС появится переменная *Ans* (рис. 2.6.21).

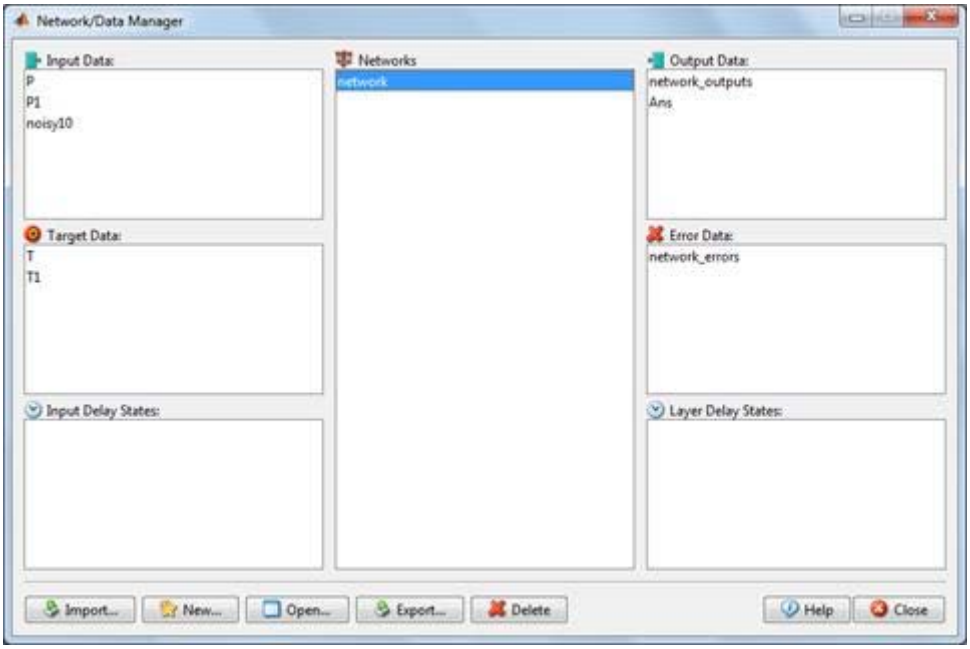

Рис. 2.6.21. Окно менеджера данных НС после проведенной симуляции

Экспортируем НС и переменную *Ans*. Для этого необходимо нажать кнопку *Export*, в появившемся окне выбрать необходимые нам переменные и еще раз нажать кнопку *Export* (рис. 2.6.22).

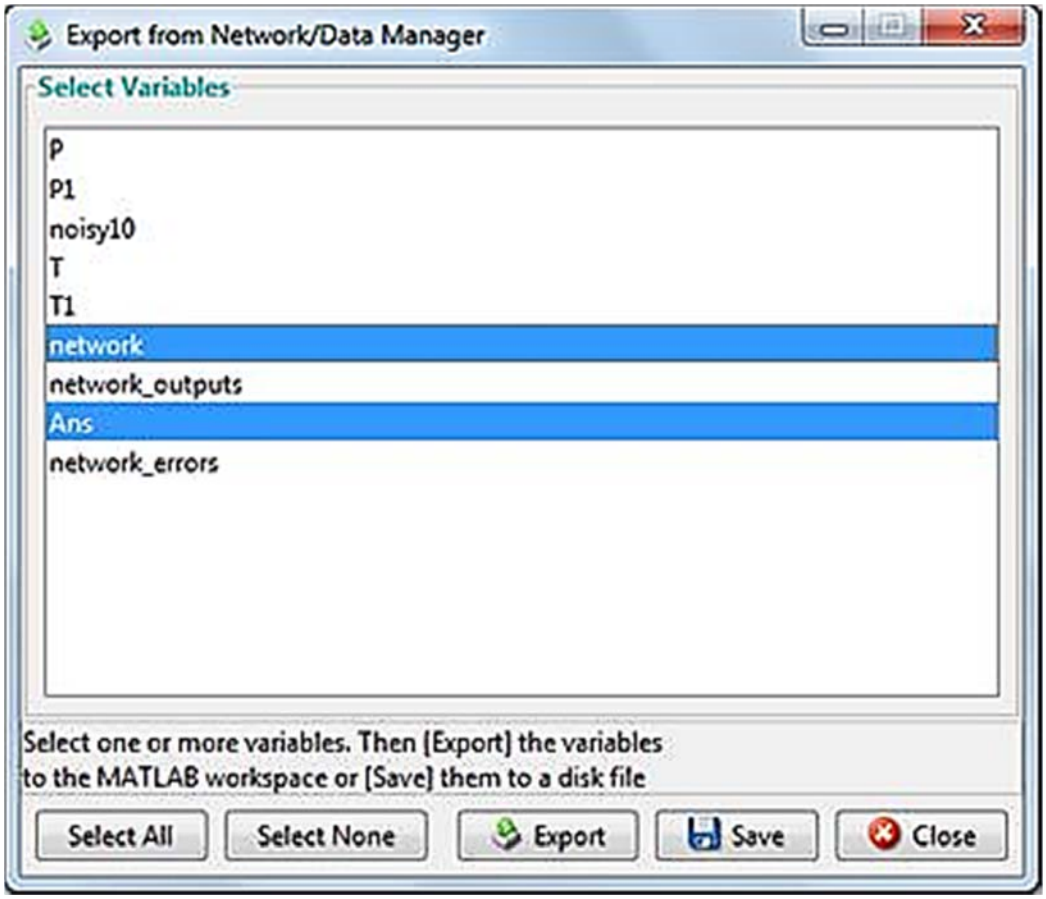

Рис. 2.6.22. Экспорт переменных

После экспорта в окно команд *Matlab* вводим следующий фрагмент кода:

```
%Проверяем результат распознания

Ans = compet(Ans);
answer = find(compet(Ans) = = 1)
plotchar(P(:,answer)); % Распознанный символ И
```

Перед нами появится окно (рис. 2.6.23), отображающее распознанный символ.

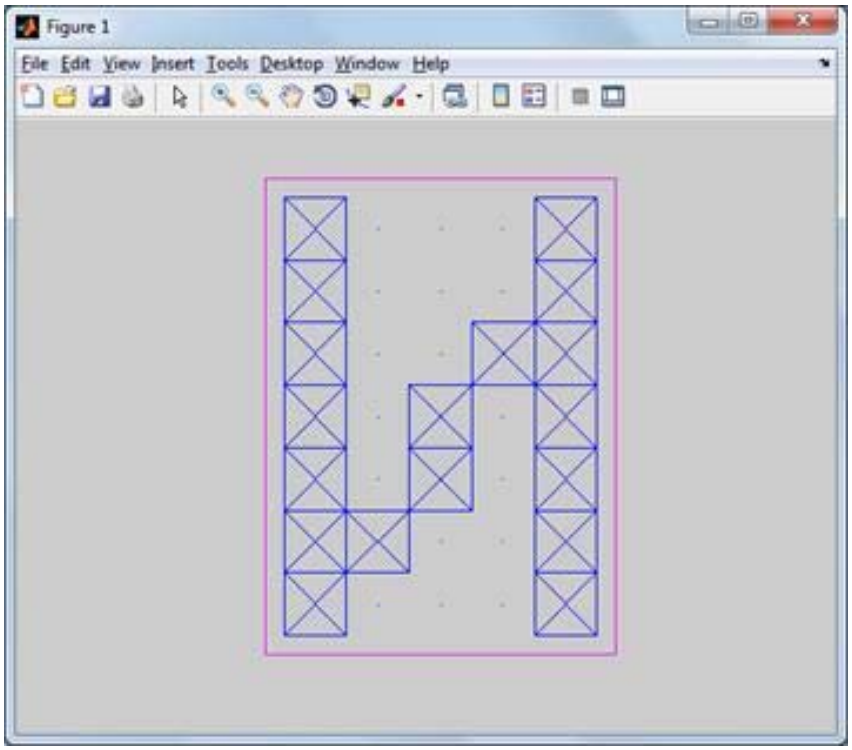

Рис. 2.6.23. Распознанный символ «И»

А в окне команд появится строчка,

```
answer =
10
```

означающая, что поступивший символ – это символ под номером 10 в нашем алфавите.

Таким образом, созданная НС выполняет поставленную задачу.

Пример листинг кода приведен в приложении к лабораторной работе № 6.

Задание

1. Подготовить графические файлы эталонных образов для символов, заданных преподавателем.

2. В среде *MATLAB* создать и обучить НС, предназначенную для распознавания печатных символов.

3. Исследовать зависимость качества работы НС:

– от степени искажения символов (параметр *P*);

– числа нейронов в скрытом слое.

Качество работы НС характеризуется вероятностями правильной классификации $P_{пр}(i)$ образа i-го класса, $i = 1 \ldots M$. Оценка вероятностей $P_{пр}(i)$ производится по формуле

$$\hat{P}_{np}(i) = \frac{N_{np}}{N_0},$$

где $N_{пр}$ – число правильных распознаваний образа i-го класса; N_0 – общее число распознаваний образов i-го класса. Число $N_{пр}$ определяется экспериментально при запуске программы *sr_work* при значениях $N_0 = 10 \ldots 100$.

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Что включает подготовка данных, необходимых для создания НС?
2. Чем характеризуется качество работы НС?
3. За что отвечает переменная *Ans*?
4. С помощью какой программы происходит обучение НС?

Лабораторная работа № 7
ФРЕЙМОВАЯ МОДЕЛЬ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Цель работы: изучение фреймовой модели, процесса разработки фреймов в виде взаимосвязанных таблиц и сложной иерархической структуры.

Теория

Начиная с 1960-х годов в области ИИ использовалось понятие фрейма знаний, или просто фрейма. Каждый фрейм имеет свое собственное имя и набор атрибутов, или слотов, которые содержат значения; например, фрейм «дом» мог бы содержать слоты «цвет», «количество этажей» и т. д.

Использование фреймов в экспертных системах – пример объектно-ориентированного программирования с наследованием свойства, которое описывается связью *is-a* («является»). Однако в использовании связи *is-a* существовало немало противоречий: Рональд Брахман написал работу, озаглавленную «Чем является и не является *IS-A*», в которой были найдены 29 различных семантик связи *is-a* в проектах, чьи схемы представления знаний включали связь *is-a*. Другие связи включают, например, *has-part* («имеет своей частью»).

Фреймовые структуры хорошо подходят для представления знаний в виде схем и стереотипных когнитивных паттернов. Элементы подобных паттернов обладают разными весами, причем большие веса назначаются тем элементам, которые соответствуют текущей когнитивной схеме (*schema*). Паттерн активизируется при определенных условиях: если человек видит большую птицу, при условии что сейчас активна его «морская схема», а «земная схема» – нет, то он классифицирует ее скорее как морского орлана, а не сухопутного беркута.

Фреймовые представления объектно-центрированы в том же смысле, что и семантическая сеть: все факты и свойства, связанные с одной концепцией, размещаются в одном месте, поэтому не требуется тратить ресурсы на поиск по базе данных.

Фреймы – это структуры данных, в которых в определенном порядке представлены сведения о свойствах объекта.

Когда человек оказывается в новой ситуации, он извлекает из памяти ранее накопленные блоки знаний, имеющие отношение к текущей ситуации, и пытается применить их. Эти блоки знаний и представляют собой фреймы. Вероятно, знания человека организованы в виде сети фреймов, отражающих его прошлый опыт. Например: типовой номер в гостинице имеет кровать, ванную комнату, шкаф для одежды, телефон и т. д. Детали каждого конкретного номера могут отличаться от приведенного описания, но они легко уточняются, когда человек оказывается в конкретном номере: цвет обоев, положение выключателей.

Таким образом, любое представление о предмете, объекте, стереотипной ситуации у человека всегда обрамлено (отсюда *frame* – рамка) характеристиками и свойствами объекта или ситуации.

Основная *структурная единица* фрейма – **слот;** это вложенная во фрейм структура данных, который представляется в виде

‹имя слота›:$\{(A_i, v^i)\},\{r_i\}$,

где A_i – имя признака, v^i – его значение, r_i – связь с другими слотами.

Слоты – это некоторые незаполненные подструктуры фрейма, после заполнения которых конкретными данными фрейм будет представлять ту или иную ситуацию, явление или объект предметной области. При конкретизации фрейма ему и его слотам присваивают конкретные имена и происходит заполнение слотов.

В качестве значений слотов могут выступать имена других фреймов, что обеспечивает построение сети фреймов.

В общем виде фрейм выглядит следующим образом:

‹Имя фрейма›:
[‹роль 1›] (‹имя слота 1› : ‹значение слота 1›);
[‹роль 2›] (‹имя слота 2› : ‹значение слота 2›);
…….
[‹роль *n*›] (‹имя слота *n*› : ‹значение слота *n*›).

В общем случае структура данных фрейма может содержать более широкий набор информации, в который входят следующие **атрибуты**.

Имя фрейма. Оно служит для идентификации фрейма в системе; имя должно быть уникальным. Фрейм представляет собой совокупность слотов, число которых может быть произвольным. Число слотов в каждом фрейме устанавливается проектировщиком системы, при этом часть слотов определяется самой системой для выполнения специфических функций, примерами которых являются: слот – указатель родителя данного фрейма, слот – указатель дочерних фреймов, слот для ввода имени пользователя, слот для ввода даты определения фрейма, слот для ввода даты изменения фрейма и т. д.

Имя слота. Оно должно быть уникальным в пределах фрейма.

Значение слота. Оно должно соответствовать указанному типу данных и условию наследования. Значением слота могут быть числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов.

Пример

Разработать фреймы в виде взаимосвязанных таблиц и сложной иерархической структуры.

Ход работы

Для построения фреймовой модели представления знаний необходимо выполнить следующие шаги:

1) определить абстрактные объекты и понятия предметной области, необходимые для решения поставленной задачи. Оформить их в виде фреймов-прототипов (фреймов-объектов, фреймов-ролей);

2) задать конкретные объекты предметной области. Оформить их в виде фреймов-экземпляров (фреймов-объектов, фреймов-ролей);

3) определить набор возможных ситуаций. Оформить их в виде фреймов-ситуаций (прототипы). Если существуют прецеденты по ситуациям в предметной области, добавить фреймы-экземпляры (фреймы-ситуации);

4) описать динамику развития ситуаций (переход от одних к другим) через набор сцен. Оформить их в виде фреймов-сценариев;

5) добавить фреймы – объекты сценариев и сцен, которые отражают данные конкретной задачи.

Решение

1. Ключевые понятия данной предметной области – ресторан; тот, кто посещает ресторан (клиент), и те, кто его обслуживает (для простоты ограничимся только официантами). У обслуживающего персонала и клиентов есть общие характеристики, поэтому целесообразно выделить общее абстрактное понятие «человек». Тогда фреймы «ресторан» и «человек» являются прототипами-образцами, а фреймы «официант» и «клиент» – прототипами-ролями. Также нужно определить основные слоты фреймов – характеристики, имеющие значения для решаемой задачи.

ЧЕЛОВЕК			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
пол	Мужской или	из внешних источников	
возраст	От 0 до 120 лет	из внешних источников	

РЕСТОРАН			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Название		из внешних источников	
Адрес		из внешних источников	
Часы работы		из внешних источников	
Специализация		из внешних источников	
Класс	Средний или высший	из внешних источников	

Фреймы-наследники содержат все слоты своих родителей, они явно прописываются только в случае изменения какого-либо параметра.

ОФИЦИАНТ (АКО ЧЕЛОВЕК)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
возраст	От 18 до 55 лет	из внешних источников	
стаж работы		из внешних источников	
зарплата		из внешних источников	
график работы		из внешних источников	
место работы	Фрейм-объект	из внешних источников	

КЛИЕНТ (АКО ЧЕЛОВЕК)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Вид оплаты	Наличные или карточка	По умолчанию (наличные)	
Статус	Обычный или Vip	По умолчанию (обычный)	
Форма заказа	Заказ есть или нет	По умолчанию (заказа нет)	
Чаевые		Из внешних источников	

2. Фреймы-образцы описывают конкретную ситуацию: какие рестораны имеются в городе, как именно организовывается посещение, кто является посетителем, кто работает в выбранном ресторане и т. д. Поэтому определим следующие фреймы-образцы, являющиеся наследниками фреймов-прототипов.

КАФЕ-РЕСТОРАН "ВКУСНЯТИНА" (АКО РЕСТОРАН)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Название	Вкуснятина	из внешних источников	
Адрес	г. Ульяновск, улица Минаева, 15	из внешних источников	
Часы работы	9:00-00:00	из внешних источников	
Специализация	Пиццерия	из внешних источников	
Класс	Средний или высший	из внешних источников	

КАФЕ "ВКУСНАЯ ЕДА" (АКО РЕСТОРАН)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Название	Вкусная еда	из внешних источников	
Адрес	г. Ульяновск, улица Карла Маркса, 5	из внешних источников	
Часы работы	9:00-00:00	из внешних источников	
Специализация	Паб	из внешних источников	
Класс	Средний	из внешних источников	

СЕРГЕЙ (АКО ОФИЦИАНТ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
возраст	27	из внешних источников	
пол	мужской	из внешних источников	
стаж работы	5	из внешних источников	
зарплата	7 000	из внешних источников	
график работы	Через день с 18:00 до 00:00	из внешних источников	
место работы	КАФЕ "ВКУСНАЯ ЕДА"	из внешних источников	

МАРИНА (АКО ОФИЦИАНТ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
возраст	24	из внешних источников	
Пол	женский	из внешних источников	
стаж работы	2	из внешних источников	
зарплата	8 200	из внешних источников	
график работы	Каждый день с 9:00 до 14:00	из внешних источников	
место работы	**КАФЕ-РЕСТОРАН "ВКУСНЯТИНА"**	из внешних источников	

ПЁТР (АКО КЛИЕНТ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
пол	мужской	из внешних источников	
возраст	19	из внешних источников	
Вид оплаты	Наличные	По умолчанию (наличные)	
Статус	Обычный	По умолчанию (обычный)	
Форма заказа	Заказа нет	По умолчанию (заказа нет)	
Чаевые	7 % от суммы заказа	Из внешних источников	

3. Фреймы-ситуации описывают возможные ситуации. В ресторане клиент попадает в несколько типичных: заказ и оплата. Возможны и другие, нетипичные: клиент подавился, у клиента нет наличности для оплаты счета и т. д. Рассмотрим типичные ситуации (их может быть больше).

ЗАКАЗ			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Перечень блюд		из внешних источников	IF-ADDED (изменяет слот «Перечень цен»)
Перечень цен		Присоединенная процедура	IF-ADDED (изменяет слот «Сумма заказчик»)
Сумма заказа		Присоединенная	
Принял заказ	Фрейм-образец	из внешнего источника	
Сделал заказ	Фрейм-образец	из внешнего источника	

ОПЛАТА			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Вид платежа		из внешних источников	IF-ADDED (изменяет слот «Чаевые»)
Чаевые		Присоединенная	
Оплатил	Фрейм-образец	Присоединенная процедура	
Заказ	Фрейм-образец	из внешних источников	IF-ADDED (изменяет слот «Оплатил»)

4. Ситуации возникают после наступления каких-то событий, выполнения условий и могут следовать одна за другой. Динамику предметной области можно отобразить в фреймах-сценариях. Их может быть множество, опишем наиболее общий и типичный сценарий посещения ресторана.

ПОСЕЩЕНИЕ РЕСТОРАНА			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Посетитель	Фрейм-объект	из внешних источников	
Ресторан	Фрейм-объект	из внешних источников	IF-ADDED, IF-REMOVED (изменяют слот «Официант»)
Официант	Фрейм-объект	присоединенная процедура (определяет по выбранному ресторану)	
Сцена 1	Вход, выбор	из внешних источников	
Сцена 2	Заказ	из внешних источников	
Сцена 3	Еда	из внешних источников	
Сцена 4	Оплата	из внешних источников	
Сцена 5	Выход	из внешних источников	

5. Пусть в рамках нашей задачи Петр посетил ресторан «Вкусная еда». Тогда фреймы будут заполнены следующим образом.

ПОСЕЩЕНИЕ «Вкусной еды» (АКО ПОСЕЩЕНИЕ РЕСТОРАНА)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Посетитель	**ПЁТР**	из внешних источников	
Ресторан	**КАФЕ "ВКУСНАЯ ЕДА"**	из внешних источников	IF-ADDED, IF-REMOVED (изменяют слот «Официант»)
Официант	**СЕРГЕЙ**	присоединенная процедура (определяет по выбранному ресторану)	
Сцена 1	Вход, выбор	из внешних источников	
Сцена 2	**ЗАКАЗ ПЕТРА**	из внешних источников	
Сцена 3	Еда	из внешних источников	
Сцена 4	**ОПЛАТА ПЕТРА**	из внешних источников	
Сцена 5	Выход	из внешних источников	

ЗАКАЗ ПЕТРА (АКО ЗАКАЗ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Перечень блюд	Отбивная, темное пиво	из внешних источников	IF-ADDED (изменяет слот «Перечень цен»)
Перечень цен	250, 75	Присоединенная процедура	IF-ADDED (изменяет слот «Сумма заказа»)
Сумма заказа	325	Присоединенная	
Принял заказ	**СЕРГЕЙ**	из внешнего источника	
Сделал заказ	**ПЕТР**	из внешнего источника	

ОПЛАТА ПЕТРА (АКО ОПЛАТА)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Вид платежа	Наличные	из внешних источников	IF-ADDED (изменяет слот «Чаевые»)
Чаевые	30	Присоединенная процедура	
Оплатил	**ПЕТР**	из внешних источников	
Заказ	**ЗАКАЗ ПЕТРА**	из внешних источников	IF-ADDED (изменяет слот «Оплатил»)

Взаимосвязь различных видов фреймов отображается графически в виде графа (рис. 2.7).

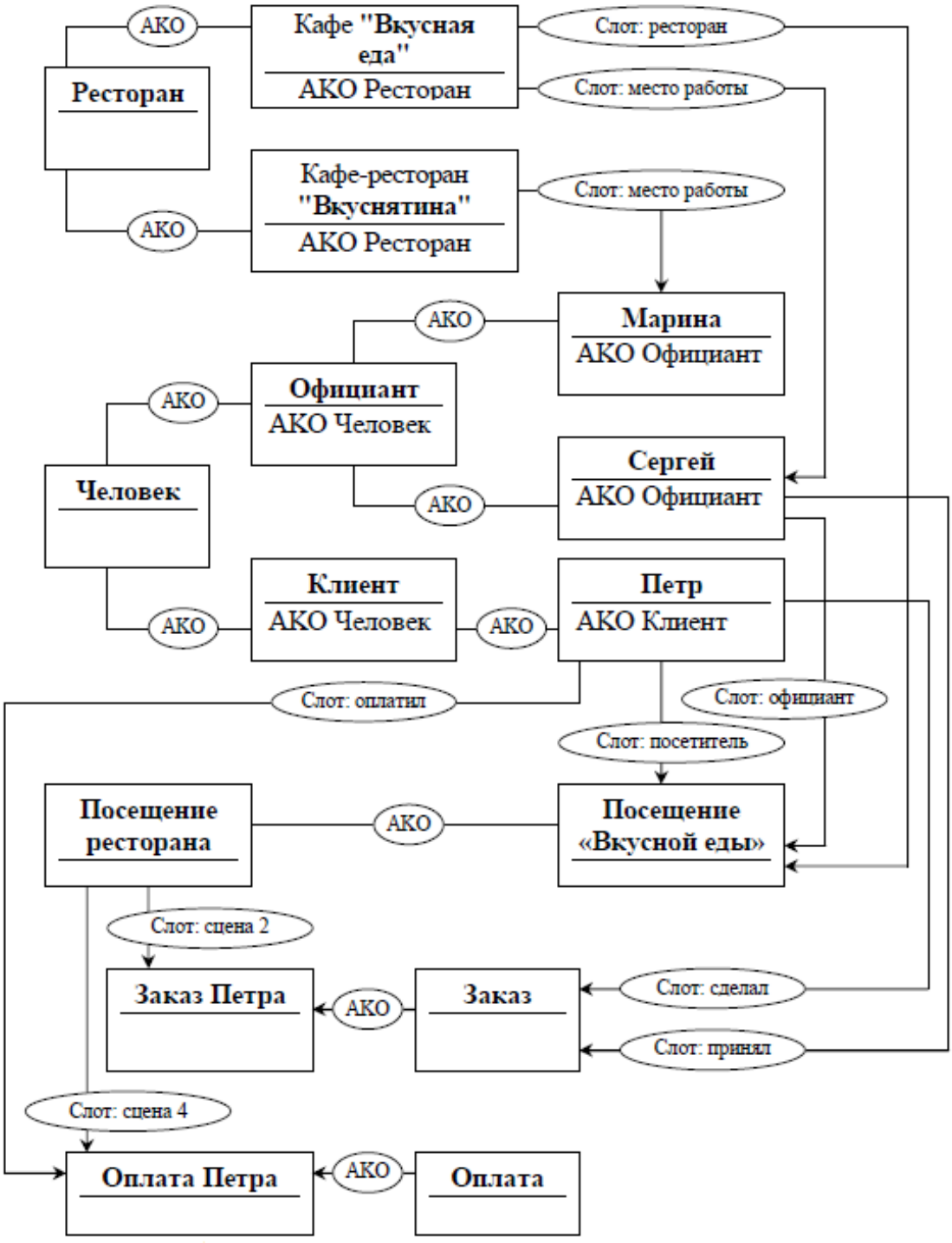

Рис. 2.7. Схема фреймов для предметной области «Ресторан»

Задание

Для самостоятельно выбранной предметной области разработать фреймы в виде взаимосвязанных таблиц и сложной иерархической структуры.

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Что такое фрейм? Опишите его структуру и приведите классификацию фреймов.

2. Какие бывают виды присоединенных процедур и каковы принципы их функционирования?

3. Каковы принципы организации фреймовых систем?

4. Что такое семантическая структура? Приведите классификацию семантических структур и опишите принципы их построения.

Лабораторная работа № 8
ПРОДУКЦИОННАЯ МОДЕЛЬ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Цель работы: приобретение студентами умений и навыков реализации пополняемой динамической базы знаний, не включаемой непосредственно в текст программы.

Необходимые для достижения поставленной цели **задачи** состоят в следующем:

- научиться оценивать возможности применения современных языков высокого уровня для реализации баз знаний экспертных систем (ЭС);
- изучить технические аспекты реализации продукционной модели представления знаний.

Теория

Системы продукций – это набор правил, используемый как база знаний, поэтому его еще называют базой правил. В стэнфордской теории фактор уверенности *CF* (*certainty factor*) принимает значения от +1 (максимум доверия к гипотезе) до –1 (минимум доверия).

А. Ньюэлл и Г. Саймон отмечали в *GPS*, что продукции соответствуют навыкам решения задач человеком в долгосрочной памяти человека. Подобно навыкам в долгосрочной памяти эти продукции не изменяются при работе системы. Они вызываются «по образцу» для решения данной специфической проблемы. Рабочая память продукционной системы соответствует краткосрочной памяти, или текущей области внимания человека. Содержание рабочей области памяти после решения задачи не сохраняется.

Работа продукционной системы инициируется начальным описанием (состоянием) задачи. Из продукционного множества правил выбирают правила, пригодные для применения на очередном шаге. Эти правила создают так называемое конфликтное множество. Для выбора правил из конфликтного множества существуют стратегии разрешения конфликтов, которые могут быть и достаточно простыми, например выбор первого правила, а могут быть и сложными эвристическими правилами. Продукционная модель в чистом виде не имеет механизма выхода из тупиковых состояний в процессе поиска. Она продолжает работать, пока не будут исчерпаны все допустимые продукции. Практические реализации продукционных систем содержат механизмы возврата в предыдущее состояние для управления алгоритмом поиска.

Задание

Реализовать продукционную систему со следующей структурой:

- *база правил* – область памяти, которая содержит базу знаний (совокупность знаний, представленных в форме правил вида «ЕСЛИ… ТО»);
- *глобальная база данных* – область памяти, содержащая факты, которые описывают вводимые данные и состояния системы;
- *интерпретатор правил* (механизм логического вывода) – компонент системы, который формирует заключения, используя базу правил и базу данных.

Для синтаксического представления продукций в модели следует использовать язык исчисления предикатов первого порядка, т. е. основными формализмами представления продукций должны быть:

а) терм, устанавливающий соответствие знаковых символов описываемому объекту;

б) предикат для описания отношения сущностей в виде реляционной формулы, содержащей в себе термы.

Термы должны быть двух видов: терм-переменная и терм-константа.

В записи условной части должна быть предусмотрена возможность наличия логических связок «И» и «ИЛИ».

Требования к системе, реализующей продукционную модель представления знаний:

а) наличие механизма заполнения базы правил и глобальной базы данных, а также отображение результата логического вывода (интерфейс с пользователем);

б) механизм логического вывода

- нечетные номера – прямой вывод;
- четные номера – обратный вывод;

в) представление системы продукций графом «И/ИЛИ»;

г) предвидение механизма разрешения конфликта на этапе вывода;

д) обнаружение ошибочных правил в случае, когда либо доказательство заключения закончилось неуспехом, либо получено неверное заключение;

е) протоколирование поиска на графе: описать процесс вывода в системе, используя частичный граф; в случае неуспешного вывода указать невыполнимое условие.

Ход работы

Для построения продукционной модели представления знаний необходимо выполнить следующие шаги:

1) определить целевые действия задачи (являющиеся решениями);

2) определить промежуточные действия или цепочку действий между начальным состояниями и конечным (между тем, что имеется, и целевым действием);

3) опередить условия для каждого действия, при котором его целесообразно и возможно выполнить. Определить порядок выполнения действий;

4) добавить конкретики при необходимости исходя из поставленной задачи;

5) преобразовать полученный порядок действий и соответствующие им условия в продукции;

6) для проверки правильности построения продукций следует записать цепочки продукций, отобразив логические связи между ними.

Этот набор шагов предполагает движение при построении продукционной модели от результата к начальному состоянию, но возможно и движение от начального состояния к результату (шаги 1 и 2).

Решение

1. Обязательное действие, выполняемое в ресторанах, – прием пищи и оплата заказа. Значит, есть уже два целевых действия: «съесть пищу» и «оплатить», которые взаимосвязаны и следуют друг за другом.

2. Прежде чем что-либо съесть в ресторане, туда нужно прийти, дождаться официанта и сделать заказ. Кроме того, нужно выбрать, в какой именно ресторан пойти. Значит, цепочка промежуточных действий: «выбрать ресторан и добраться до него», «сделать заказ официанту».

3. Прежде чем идти в ресторан, необходимо убедиться, что у вас есть необходимая сумма денег. Выбор ресторана может обусловливаться многими причинами, выберем территориальный признак: к какому ближе, в тот и идем. В разных ресторанах работают разные люди, поэтому официанты будут разные в каждом ресторане. Кроме того, разные рестораны специализируются на разных кухнях, поэтому заказанные блюда будут в разных ресторанах отличаться. Значит, вначале

идут действия, позволяющие выбрать ресторан, а уже после – соответственно заказ, прием пищи и оплата заказа.

4. Пусть в задаче будут рассматриваться два ресторана: «Вкусная еда» и «Вкуснятина». Первый – паб, и заказы приносят быстрее, чем во втором, второй – пиццерия. В первом работает официант Сергей, а во втором – официантка Марина. Петр – это клиент.

5. Вышеописанное можно преобразовать в следующие предложения типа «Если… то».

- Если субъект хочет есть и у субъекта есть достаточная сумма денег, то субъект может пойти в ресторан.
- Если субъект ближе к ресторану «Вкусная еда», чем к ресторану «Вкуснятина», и субъект может пойти в ресторан, то субъект идет в ресторан «Вкусная еда».
- Если субъект ближе к ресторану «Вкуснятина», чем к ресторану «Вкусная еда», и субъект может пойти в ресторан, то субъект идет в ресторан «Вкуснятина».
- Если субъект идет в ресторан «Вкуснятина» и в ресторане «Вкуснятина» работает официант Марина, то у субъекта принимает заказ Марина.
- Если субъект идет в ресторан «Вкусная еда» и в ресторане «Вкусная еда» работает официант Сергей, то у субъекта принимает заказ Сергей.
- Если субъект выбрал блюда и у субъекта принимает заказ Марина, то заказ принесут через 20 мин.
- Если субъект выбрал блюда и у субъекта принимает заказ Сергей, то заказ принесут через 10 мин.
- Если заказ принесут через 20 мин или через 10 мин, то субъект может есть.
- Если субъект может есть, то после еды субъект должен оплатить заказ.

Введем обозначения для фактов (Ф), действий (Д) и продукций (П), тогда:

субъект = *Петр*;

Ф1 = *субъект хочет есть*;

Ф2 = *у субъекта есть достаточная сумма денег*;

Ф3 = *субъект ближе к ресторану «Вкусная еда», чем к «Вкуснятине»*;

Ф4 = *в ресторане «Вкуснятина» работает официант Марина*;

Ф5 = *в ресторане «Вкусная еда» работает официант Сергей*;

Ф6 = *субъект выбрал блюда*;

Д1 = *субъект может пойти в ресторан*;

Д2 = *субъект идет в ресторан «Вкусная еда»;*

Д3 = *субъект идет в ресторан «Вкуснятина»;*

Д4 = *у субъекта принимает заказ Марина;*

Д5 = *у субъекта принимает заказ Сергей;*

Д6 = *заказ принесут через 20 мин;*

Д7 = *заказ принесут через 10 мин;*

Д8 = *после еды субъект должен оплатить заказ.*

Для продукций установим приоритет (в скобках перед запятой, чем выше приоритет, чем раньше проверяется правило):

П1(4, Ф1 и Ф2) = Д1;

П2(5, Ф3 и Д1) = Д2;

П3(4, не Ф3 и Д1) = Д3;

П4(3, Д3 и Ф4) = Д4;

П5(3, Д2 и Ф5) = Д5;

П6(2, Д4) = Д6;

П7(2, Д5) = Д7;

П8(1, Д6 или Д7) = Д8.

6. Для отображения взаимосвязи продукций построим граф (рис. 2.8).

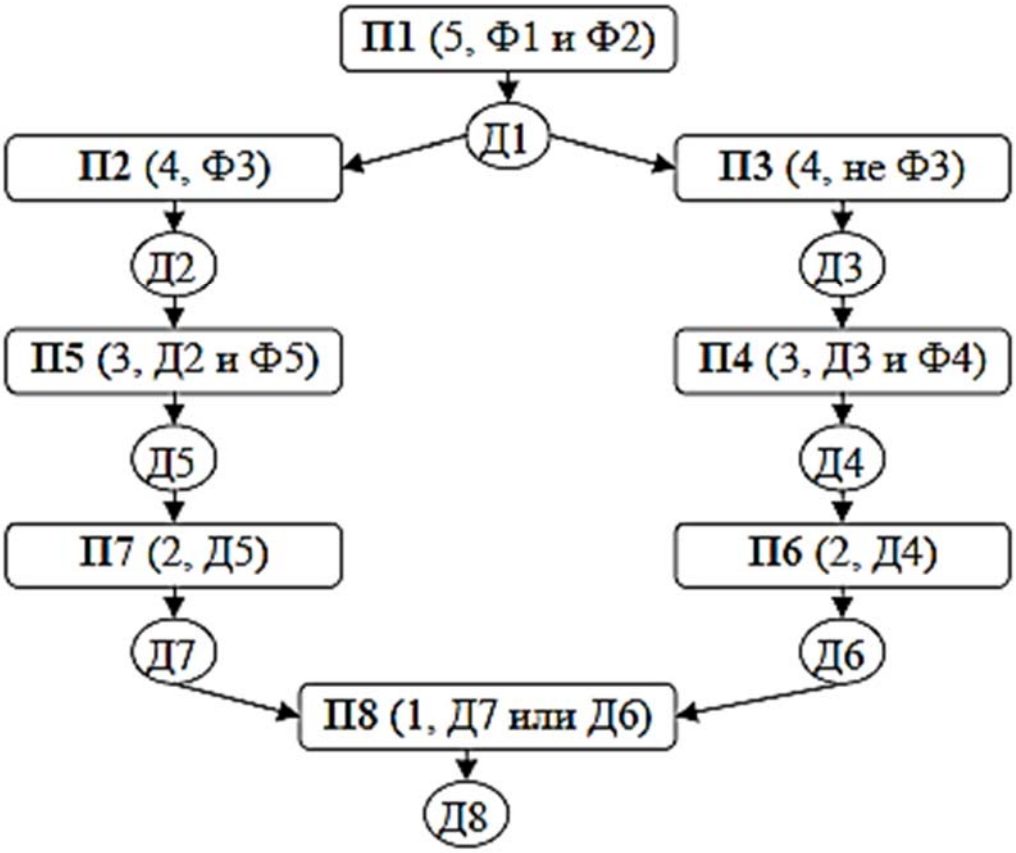

Рис. 2.8. Схема продукций предметной области «Ресторан»

Задание

Реализовать продукционную систему со следующей структурой:

- *база правил* – область памяти, которая содержит базу знаний (совокупность знаний, представленных в форме правил вида «ЕСЛИ… ТО»);
- *глобальная база данных* – область памяти, содержащая факты, которые описывают вводимые данные и состояния системы;
- *интерпретатор правил* (механизм логического вывода) – компонент системы, который формирует заключения, используя базу правил и базу данных.

Варианты предметных областей для разработки экспертной системы представлены в табл. 2.8.

Таблица 2.8

Варианты предметных областей для разработки экспертной системы

№ варианта	Предметная область
1	Профориентация школьника
2	Выбор оптимального языка программирования в зависимости от поставленной задачи
3	Подбор учебных курсов, которые необходимо изучить для получения определенных навыков (веб-программист, *windows*-программист, *unix*-программист, тестировщик, архитектор ПО и т. д.)
4	Выбор оптимального способа вложения денег в соответствии с потребностями инвестора
5	Выбор вуза для обучения по направлению «Информационные системы и технологии»
6	Ошибки системы *Windows*

Содержание работы

1. Цель работы.
2. Вариант задания.
3. Исходный текст программы.
4. Результаты выполнения.
5. Выводы.

Контрольные вопросы

1. Что называют продукционной моделью построения знаний?
2. Что такое продукционные правила? Какие бывают типы и основные структуры продукционных правил?
3. Что такое продукционные системы? Какова их структура?
4. Каковы основные принципы организации и функционирования продукционной системы?

ЗАКЛЮЧЕНИЕ

Развитие искусственного интеллекта как науки и технологии создания машин началось с 1956 года, и успехов на данный момент удалось достичь ошеломительных. Искусственный интеллект окружает человека практически везде. У этих технологий есть особенность: человек считает их интеллектуальными только первое время, затем он привыкает и они кажутся ему естественными.

Важно помнить, что наука об искусственном интеллекте находится в тесной связи с математикой, комбинаторикой, статистикой и другими науками; но они оказывают влияние не только на развитие искусственного интеллекта, но и позволяют человеку переосмыслить то, что уже было создано, как это было с программой *Logic Theorist*.

Важную роль в развитии технологий искусственного интеллекта играет появление новых технологий в смежных областях: нейропсихологии, квантовой физике. Едва ли можно представить серьезную программу интеллектуального анализа данных, которой бы хватило 100 Кбайт оперативной памяти. Компьютеры позволили технологиям развиваться экстенсивно, в то время как теоретические исследования послужили предпосылками для их интенсивного развития. Можно сказать, что развитие науки об искусственном интеллекте явилось следствием развития компьютеров.

История развития искусственного интеллекта не закончена, она пишется прямо сейчас. Постоянно совершенствуются технологии, создаются новые алгоритмы, открываются новые области их применения. Время постоянно открывает перед исследователями новые возможности и ставит новые вопросы и задачи.

Приложение к лабораторной работе № 6
НЕЙРОСЕТЕВОЕ РАСПОЗНАВАНИЕ ПЕЧАТНЫХ СИМВОЛОВ В СРЕДЕ *MATLAB*

Листинг код

```
clear, [alphabet, targets] = prprob;
% Определим шаблон для символа А, который является первым элементом алфавита:
i = 6; ti = alphabet(:, i); % и – порядковый номер символа в алфавите
letter{i} = reshape(ti, 5, 7)'; letter{i} % число столбцов, число строк, не меняется
% Вызовем М-файл prprob, который формирует массив векторов входа alphabet размера 35,26 с шаблонами символов алфавита и массив целевых векторов targets:

[alphabet,targets] = prprob;
[R,Q] = size(alphabet); [S2,Q] = size(targets);
% Двухслойная нейронная сеть создается с помощью команды newff:
% S1 – число нейронов  в скрытом слое

S1 = 10;
net=newff(minmax(alphabet),[S1 S2],...
{'logsig' 'logsig'}, 'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
gensim(net)
% конец создания нейросети
% Обучение
% Обучение в отсутствие шума
% Сеть первоначально обучается в отсутствие шума с максимальным числом циклов обучения 5000 либо до достижения допустимой средней квадратичной погрешности, равной 0.1:

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

```
net.trainParam.show = 20; net.trainParam.epochs = 500;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);

% Обучение в присутствии шума
% При обучении с шумом максимальное число циклов обучения сократим до 300, а допустимую погрешность увеличим до 0.6:

netn = net; netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];
for pass = 1:10
P = [alphabet, alphabet, ...
(alphabet + randn(R,Q)*0.1), ...
(alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
% конец обучения с шумом
% Повторное обучение в отсутствие шума

netn.trainParam.goal = 0.1; % Предельная среднеквадратичная погрешность
netn.trainParam.epochs = 500; % Максимальное количество циклов обучения
net.trainParam.show = 5; % Частота вывода результатов на экран
[netn, tr] = train(netn,P,T);
% конец обучения без шума
% Эффективность функционирования системы

tic, noise_range = 0:.05:.5; max_test = 20;
network1 = []; network2 = []; T = targets;
% Выполнить тест

for noiselevel = noise_range
errors1=0; errors2=0;
for i=1:max_test
```

```
P = alphabet + randn(35,26)*noiselevel;
A = sim(net,P);   AA = compet(A);
errors1 = errors1 + sum(sum(abs(AA - T)))/2;
An = sim(netn,P);  AAn = compet(An);
errors2 = errors2 + sum(sum(abs(AAn - T)))/2; echo off
end
% Средние значения ошибок (100 последовательностей из 26 векторов)

network1 = [network1 errors1/26/100];
network2 = [network2 errors2/26/100];
end
toc
figure(1),clf
% Отображаем файл
% t = Tiff('lab1.tif', 'r');
imageData = imread('lab1.jpg','jpg');
% figure
image(imageData);
title('Образец')
grid on

      figure(2),clf
% Соответствующий график погрешности сети от уровня входного шума показан на
      plot(noise_range,network1*100,'--',noise_range,network2*100);
      grid on
% Проверим работу нейронной сети для распознавания символов. Сформируем зашумленный вектор входа для символа

% noisy = alphabet(:,6) + randn(35,1)*0.2;  распознаем из массива
noisy=imageData; % распознаем из изображения
plotchar(noisy); % Зашумленный символ
A2 = sim(net,noisy); A2 = compet(A2);
answer = find(compet(A2) == 1)
plotchar(alphabet(:,answer)); % Распознанный символ
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Чабан, Л. Н. Теория и алгоритмы распознавания образов : учеб. пособие / Л. Н. Чабан. – М. : МИИГАиК, 2004. – 70 с.

2. Лепский, А. Е. Математические методы распознавания образов : курс лекций / А. Е. Лепский, А. Г. Броневич. – Таганрог : Изд-во ТТИ ЮФУ, 2009. – 155 с. – ISBN 978-5-8327-0306-0.

3. Shapiro L. G. Computer Vision / L. G. Shapiro, G. C. Stockman. – New Jersey : Prentice-Hall, 2001. – P. 279 – 325.

4. Canny, J. F. Finding edges and lines in images / J. F. Canny ; MIT. – USA, 1983. – P. 50 – 67.

5. Детектор границ Канни [Электронный ресурс]. – URL: https://habr.com/ru/post/114589/ (дата обращения: 01.10.2021).

6. Engel, K. Real-time volume graphics / K. Engel // Eurographics. – 2006. – P. 112 – 114.

7. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М. : Техносфера, 2012. – 1103 с. – ISBN 978-5-94836-331-8.

8. Методы и алгоритмы обработки цифровых изображений / Е. Петров [и др.] // Инфокоммуникационные технологии. – 2011. – С. 94 – 99.

9. Анисимов, Б. В. Распознавание и цифровая обработка изображений / Б. В. Анисимов. – М. : Высш. шк., 1983. – 295 с.

10. Марчук, Ю. Н. Модели перевода : учеб. пособие для студентов учреждений высш. проф. образования / Ю. Н. Марчук. – М. : Академия, 2010. – 176 с. – ISBN 978-5-7695-6991-3.

11. Карасев, И. В. Системы машинного перевода / И. В. Карасев, Е. А. Артюшина // Успехи современного естествознания. – 2011. – № 7. – С. 117 – 118.

12. Морозкина, Е. А. Использование информационных технологий для оптимизации процесса перевода / Е. А. Морозкина, Н. Р. Шакирова // Вестник Башкирского университета. – 2012. – № 1 (1). – С. 544 – 546.

13. Дроздова, К. А. Машинный перевод: история, классификация, методы / К. А. Дроздова // Вестник Омского государственного педагогического университета. Гуманитарные исследования. – 2015. – № 3 (7). – С. 156 – 158.

14. Koehn, P. Statistical Machine Translation / P. Koehn. – Cambridge : Cambridge University Press, 2010. – P. 15.

15. Колганов, Д. С. Обзор аналитической, статистической и нейронной технологий машинного перевода / Д. С. Колганов, Е. А. Данилов // Международный студенческий научный вестник. – 2018. – № 3 – 2. – С. 301 – 303.

16. Statistical Post-Editing of a Rule-Based Machine Translation System / A.-L. Lagarda [et al] // Proceedings of NAACL HLT 2009: Short Papers. Boulder, Colorado. – Association for Computational Linguistics. – 2009. – P. 217 – 220.

17. Wołk, K. Polish-English Speech Statistical Machine Translation Systems for the IWSLT 2013 / K. Wolk, K. Marasek // Proceedings of the 10th International Workshop on Spoken Language Translation. – Heidelberg, Germany. – 2013. – P. 113 – 119.

18. Хорошилов, А. А. Фразеологический машинный перевод текстов: теоретические основы и технологические решения / А. А. Хорошилов, А. В. Кан, А. А. Хорошилов. – М. ; Берлин : Директ-Медиа, 2019. – 467 с. – ISBN 978-5-4499-0089-0.

19. Cho, K. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches [Электронный ресурс] / K. Cho [et al]. – URL: http://arxiv.org/absl/1409.1259 (дата обращения: 01.10.2021).

20. Баженов, Р. И. Интеллектуальные информационные технологии / Р. И. Баженов. – Биробиджан : ПГУ им. Шолом-Алейхема, 2011. – 176 с.

21. Баженов, Р. И. Информационная безопасность и защита информации : практикум / Р. И. Баженов. – Биробиджан : Изд-во ГОУВПО «ДВГСГА», 2011. – 140 с.

22. Баженов, Р. И. Проектирование методики обучения дисциплины «Информационные технологии в менеджменте» / Р. И. Баженов // Современная педагогика. – 2014. – № 8 (21). – С. 24 – 31.

23. Баженов, Р. И. Об организации научно-исследовательской практики магистрантов направления «Информационные системы и технологии» / Р. И. Баженов // Современные научные исследования и инновации. – 2014. – № 9 – 2 (41). – С. 62 – 69.

24. Иванников, В. Matlab для DSP. Нейронные сети: графический интерфейс пользователя [Электронный ресурс] / В. Иванников, А. Ланнэ. – URL: http://www.chipinfo.ru/literature/chipnews/200108/1.html#lanne8 (дата обращения: 19.09.2021).

25. Медведев, В. С. Нейронные сети. MATLAB 6 / В. С. Медведев, В. Г. Потемкин ; под общ. ред. канд. техн. наук. В. Г. Потемкина. – М. : ДИАЛОГ-МИФИ, 2002. – 496 с. – ISBN 5-86404-163-7.

26. Миронов, И. С. Распознавание образов при помощи нейронной сети [Электронный ресурс] / И. С. Миронов, С. В. Скурлаев. – URL: http://confonline.susu.ru/index.php?option=com_content&view=article&id=57:2011-05-06-04-36-21&catid=16:-2----&Itemid=18 (дата обращения: 01.10.2021).

27. Сахнюк, П. А. Интеллектуальные информационные системы : лаб. работы [Электронный ресурс] / П. А. Сахнюк. – URL: http://www.stgau.ru/company/personal/user/7684/files/lib/202.pdf (дата обращения: 13.09.2021).

28. Шеремет, А. И. Проектирование нейронной сети для распознавания символов в программной среде MATLAB [Электронный ресурс] / А. И. Шеремент, В. В. Перепелица, А. М. Денисова. – URL: http://nauka.zinet.info/13/sheremet.php (дата обращения: 19.09.2021).

29. Principe, J. C. Neural and Adaptive Systems: Fundamentals Through Simulations / J. C. Principe, N. R. Euliano, W. C. Lefebvre. – New York : John Wiley & Sons Inc., 2000. – 656 p.

30. Luo, F-L. Applied Neural Networks for Signal Processing / F.-L. Luo, R. Unbehauln. – Cambridge University Press, 1998. – 367 p.

31. Demuth, H. Neural Network Toolbox. For Use with MATLAB / H. Demuth, V. Beale. – The MathWorks Inc, 1992 – 2000.

32. Awadalla, M. H. A. Spiking neural network-based control chart pattern recognition / M. H. A. Awadula, I. I. Ismaeil, M. A. Sadek // Journal

of Engineering and Technology Research. – 2011. – V. 3. – № 1. – P. 5 – 15.

33. Dede, G. Speech recognition with artificial neural networks / G. Dede, M. H. Sazli // Digital Signal Processing. – 2010. – V. 20. – № 3. – P. 763 – 768.

34. Прикладная статистика. Классификация и снижение размерности / С. А. Айвазян [и др.]. – М. : Финансы и статистика, 1989. – 607 с.

35. Principal Manifolds for Data Visualisation and Dimension Reduction / A. N. Gorban [et al] // Lecture Notes in Computational Science and Engineering 58. – Springer, Berlin – Heidelberg – New York. – 2007. – V. XXIV. – 340 p.

36. Pearson, K. On lines and planes of closest fit to systems of points in space / K. Pearson // Philosophical Magazine. – 1901. – № 2. – P. 559 – 572.

37. Sylvester, J. J. On the reduction of a bilinear quantic of the nth order to the form of a sum of n products by a double orthogonal substitution / J. J. Sylvester // Messenger of Mathematics. – 1889. – № 19. – P. 42 – 46.

38. Онлайн-руководство «Метод Главных Компонент (PCA)» [Электронный ресурс]. – URL: http://www.chemometrics.ru/materials/textbooks/pca.htm (дата обращения: 01.10.2021).

ОГЛАВЛЕНИЕ

Учебное издание

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ

Практикум

Авторы-составители:
ОЗЕРОВА Марина Игоревна
ЖИГАЛОВ Илья Евгеньевич

Редактор Е. А. Платонова
Технические редакторы Ш. В. Абдуллаев, О. В. Балашова
Компьютерная верстка Е. А. Кузьминой
Выпускающий редактор А. А. Амирсейидова

Подписано в печать 28.09.22.
Формат 60×84/16. Усл. печ. л. 7,44. Тираж 57 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.