ПРИОРИТЕТНЫЙ НАЦИОНАЛЬНЫЙ ПРОЕКТ «ОБРАЗОВАНИЕ»
РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

А.И. ДИВЕЕВ

СОВРЕМЕННЫЕ ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Учебное пособие

Москва
2008

Лекция 1. Общие сведения об интеллектуальных системах. Классификация интеллектуальных систем. Интеллектуальные системы управления.

В 1950 году британский математик Алан Тьюринг опубликовал в журнале «Mind» свою работу «Вычислительная машина и интеллект», в которой описал тест для проверки программы на интеллектуальность. Он предложил поместить исследователя и программу в разные комнаты и до тех пор, пока исследователь не определит, кто за стеной – человек или программа, считать поведение программы разумным. Это было одно из первых определений интеллектуальности, то есть А. Тьюринг предложил называть интеллектуальным такое поведение программы, которое будет моделировать разумное поведение человека.

С тех пор появилось много определений интеллектуальных систем и искусственного интеллекта. Сам термин «искусственный интеллект» (AI – Artificial Intelligence) был предложен в 1956 году на семинаре в Дартсмутском колледже (США). Приведем некоторые из этих определений. Д. Люгер [14] в своей книге определяет «искусственный интеллект как область компьютерных наук, занимающуюся исследованием и автоматизацией разумного поведения».

В другом учебнике по интеллектуальным системам дается такое определение: «искусственный интеллект – это одно из направлений информатики, целью которого является разработка аппаратно-программных средств, позволяющих пользователю-непрограммисту ставить и решать свои, традиционно считающиеся интеллектуальными задачи, общаясь с ЭВМ на ограниченном подмножестве естественного языка».

Введем еще определение. Предметом информатики является обработка информации по известным законам. Предметом искусственного интеллекта [6] является изучение интеллектуальной деятельности человека, подчиняющейся заранее неизвестным законам. Искусственный интеллект – это все то, что не может быть обработано с помощью алгоритмических методов.

Системой будем называть множество элементов, находящихся в отношениях друг с другом и образующих причинно-следственную связь.

Адаптивная система – это система, которая сохраняет работоспособность при непредвиденных изменениях свойств управляемого объекта, целей управления или окружающей среды путем смены алгоритма функционирования, программы поведения или поиска оптимальных, в некоторых случаях просто эффективных, решений и состояний. Традиционно по способу адаптации различают самонастраивающиеся, самообучающиеся и самоорганизующиеся системы.

Под алгоритмом будем понимать последовательность заданных действий, которые однозначно определены и выполнимы на современных ЭВМ за приемлемое время для решаемой задачи.

Под интеллектуальной системой будем понимать адаптивную систему, позволяющую строить программы целесообразной деятельности по решению поставленных перед ними задач на основании конкретной ситуации, складывающейся на данный момент в окружающей их среде.

Сделаем два важных дополнения к данному определению. К сфере решаемых интеллектуальными системами задач относятся задачи, обладающие, как правило, следующими особенностями: в них неизвестен алгоритм решения задач [16]. Такие задачи будем называть интеллектуальными задачами; в них используется помимо традиционных данных в числовом формате информация в виде изображений, рисунков, знаков, букв, слов, звуков; в них предполагается наличие выбора.

Если не существует алгоритма – это значит, что нужно сделать выбор между многими вариантами в условиях неопределенности. Свобода действий является существенной составляющей интеллектуальных задач.

Интеллектуальные робототехнические системы содержат настраиваемую модель внешнего мира и реальной исполнительной системы с объектом управления. Цель и управляющие воздействия формируются в интеллектуальных робототехнических системах на основе знаний о внешней среде, объекте управления и на основе моделирования ситуаций в реальной системе.

О каких признаках интеллекта уместно говорить применительно к интеллектуальным системам? Интеллектуальная система должна уметь в наборе фактов распознать существенные факты.

Интеллектуальные системы способны из имеющихся фактов и знаний сделать выводы не только с использованием дедукции, но и с помощью аналогии, индукции и т.д. Кроме того, интеллектуальные системы должны быть способны к самооценке – обладать рефлексией, то есть средствами для оценки результатов собственной работы. С помощью подсистем объяснения интеллектуальная система может ответить на вопрос, почему получен тот или иной результат. Наконец, интеллектуальная система должна уметь обобщать, улавливая сходство между имеющимися фактами.

Можно ли считать шахматную программу интеллектуальной системой? Если шахматная программа при повторной игре делает одну и ту же ошибку – то нельзя. Обучаемость, адаптивность, накопление опыта и знаний – важнейшие свойства интеллекта. Если шахматная программа реализована на компьютере с бесконечно высоким быстродействием и обыгрывает человека за счет просчета всех возможных вариантов игры по жестким алгоритмам, то такую программу мы также не назовем интеллектуальной. Но если шахматная программа осуществляет выбор и принятие решений в условиях неопределенности на основе эффективных методов принятия решений и эвристик, корректируя свою игру от партии к партии в лучшую сторону, то такую программу можно считать достаточно интеллектуальной.

Всякий раз, как только возникают сомнения в интеллектуальности некоторой системы, договоримся вспоминать тест Алана Тьюринга на интеллектуальность. После этого сомнения и дальнейшие споры, как правило, прекращаются.

Следует определить также понятие знания – центрального понятия в интеллектуальной системе. Рассмотрим несколько определений.

Знания есть результат, полученный познанием окружающего мира и его объектов.

Знания – система суждений с принципиальной и единой организацией, основанная на объективной закономерности.

Знания – это формализованная информация, на которую ссылаются или которую используют в процессе логического вывода.

Под знаниями будем понимать совокупность фактов и правил. Понятие правила, представляющего фрагмент знаний, имеет вид:

если <условие>, то <действие>.

Например, если X истинно и Y истинно, то Z истинно с достоверностью P.

Среди важнейших классов задач, которые ставились перед искусственным интеллектом, следует выделить следующие трудно формализуемые задачи:

- доказательство теорем,

- распознавание изображений,

- экспертные системы логического вывода,

- машинный перевод и понимание текстов на естественном языке,

- игровые программы,

- машинное творчество.

Доказательство теорем.

Изучение приемов доказательства теорем сыграло важную роль в развитии искусственного интеллекта. Формализация дедуктивного процесса с использованием логики предикатов помогает глубже понять некоторые компоненты рассуждений. Многие неформальные задачи, например медицинская диагностика, допускают формализацию как задачу на доказательство теорем. Поиск доказательства математической теоремы требует не только произвести дедукцию, исходя из гипотез, но также создать интуитивные догадки и гипотезы о том, какие промежуточные утверждения следует доказать для вывода доказательства основной теоремы.

В 1954 году А. Ньюэлл задумал создать программу для игры в шахматы. Дж. Шоу и Г. Саймон объединились в работе по проекту Ньюэлла, и в 1956 году они создали язык программирования IPL-I (предшественник LISPa) для работы с символьной информацией. Их первыми программами стала программа LT (Logic Theorist) для доказательства теорем и исчисления высказываний (1956 год), а также программа NSS (Newell, Shaw, Simon) для игры в шахматы (1957 год). LT и NSS привели к созданию А. Ньюэллом, Дж. Шоу и Г. Саймоном программы GPS (General Problem Solver) в 1957 – 1972 годах.

Программа GPS моделировала общие стратегии решения задач и могла применяться для решения шахматных и логических задач, доказательства теорем, грамматического разбора предложений, математического интегрирования, головоломок типа «Ханойская башня» и т. д.

Процесс работы GPS воспроизводит методы решения задач, применяемые человеком: выдвигаются подцели, приближающие к решению, применяется эвристический метод (один, другой и т. д.), пока не будет получено решение. Попытки прекращаются, если получить решение не удается. Программа GPS могла решать только относительно простые задачи. Ее универсальность достигалась за счет эффективности. Специализированные «решатели задач» – STUDENT (Bobrov, 1964) и др. лучше проявляли себя при поиске решения в своих предметных областях. GPS оказалась первой программой (написана на языке IPL-V), в которой предусматривалось планирование стратегии решения задач.

Для решения трудно формализуемых задач и, в частности, для работы со знаниями были созданы языки программирования для задач искусственного интеллекта: LISP (1960 год, J. MacCatthy), Пролог (1975 – 1979, D. Warren, F. Pereira), ИнтерLISP, FRL, KRL, SMALLTALK, OPS5, PLANNER, QA4, MACSYMA, REDUCE, РЕФАЛ, CLIPS. К числу наиболее популярных традиционных языков программирования для создания ИС следует также отнести C++ и Паскаль.

Распознавание изображений.

В традиционном распознавании образов появился хорошо разработанный математический аппарат, и для не очень сложных объектов оказалось возможным строить практически работающие системы классификации по признакам, по аналогии и т.д. В качестве признаков могут рассматриваться любые характеристики распознаваемых объектов. Признаки должны быть инвариантны к ориентации, размеру и вариациям формы объектов. Алфавит признаков придумывается разработчиком системы. Качество распознавания во многом зависит от того, насколько удачно придуман алфавит признаков. Распознавание состоит в априорном получении вектора признаков для выделенного на изображении отдельного распознаваемого объекта и лишь затем – в определении того, какому из эталонов этот вектор соответствует.

П. Уинстон в начале 80-х годов обратил внимание на необходимость реализации целенаправленного процесса машинного восприятия. Цель должна управлять работой всех процедур, в том числе и процедур нижнего уровня, т.е. процедур предварительной обработки и выделения признаков. Должна иметься возможность на любой стадии процесса в зависимости от получаемого результата возвращаться к его началу для уточнения результатов работы процедур предшествующих уровней. У П. Уинстона, так же как и у других исследователей, до решения практических задач дело не дошло, хотя в 80-е годы вычислительные мощности больших машин позволяли начать решение подобных задач.

Экспертные системы.

Методы искусственного интеллекта нашли применение при создании автоматических консультирующих систем. До 1968 года исследователи в области искусственного интеллекта работали на основе общего подхода – упрощения комбинаторики, базирующегося на уменьшении перебора альтернатив исходя из здравого смысла, применения числовых функций оценивания и различных эвристик.

В начале 70-х годов произошел качественный скачок, и пришло понимание, что необходимы глубокие знания в соответствующей области и выделение знаний из данных, получаемых от эксперта. Появляются экспертные системы , или системы, основанные на знаниях.

DENDRAL (середина 60-х годов, Стэнфордский университет) расшифровывала данные масс-спектрографического анализа.

MYCIN (середина 70-х годов, Стэнфордский университет) ставила диагноз при инфекционных заболеваниях крови.

PROSPECTOR (1974 – 1983 годы, Стэнфордский университет) обнаруживала полезные ископаемые.

SOPHIE обучала диагностированию неисправностей в электрических цепях. ЭС XCON помогала конфигурировать оборудование для систем VAX фирмы DEC, ЭС PALLADIO помогала проектировать и тестировать СБИС-схемы.

JUDITH помогает специалистам по гражданским делам и вместе с юристом и с его слов усваивает фактические и юридические предпосылки дела, а затем предлагает рассмотреть различные варианты подходов к его разрешению.

LRS оказывает помощь в подборе и анализе информации о судебных решениях и правовых актах в области кредитно-денежного законодательства, связанного с использованием векселей и чеков.

«Ущерб» на основе российского трудового законодательства обеспечивает юридический анализ ситуации привлечения рабочих и служащих к материальной ответственности при нанесении предприятию материального ущерба действием или бездействием.

Список созданных экспертных систем можно перечислять очень долго. Были разработаны и внедрены тысячи реально работающих экспертных систем.

Разработка инструментальных средств для создания экспертных систем ведется постоянно. Появляются экспертные системы оболочки, совершенствуются технологии создания экспертных систем. Язык Пролог (1975 – 1979) становится одним из основных инструментов создания экспертных систем. Язык CLIPS (C Language Integrated Production System) начал разрабатываться в космическом центре Джонсона NASA в 1984 году. Язык CLIPS свободен от недостатков предыдущих инструментальных средств для создания ЭС, основанных на языке LISP. Появляется инструментарий EXSYS, ставший в начале 90-х годов одним из лидеров по созданию экспертных систем. В начале XXI века появляется теория интеллектуальных агентов и экспертных систем на их основе. Web-ориентированный инструментарий JESS (Java Expert System Shell), использующий язык представления знаний CLIPS, приобрел достаточную известность в настоящее время. Среди отечественных инструментальных средств следует отметить web-ориентированную версию комплекса АТ-ТЕХНОЛОГИЯ, разработанного на кафедре Кибернетики МИФИ. В этом комплексе вся прикладная логика как комплекса в целом, так и разработанных в нем web-интегрированных экспертных систем, сосредоточена на стороне сервера.

Практика внедрения экспертных систем показала, что нет чудодейственных рецептов – нужна кропотливая работа по вводу в ЭВМ опыта и знаний специалистов всех областей науки.

Машинный перевод и понимание текстов на естественном языке.

Началом работ по машинному переводу следует считать 1954 год, когда в США с помощью ЭВМ было переведено шестьдесят фраз. Этот известный «Джорджтаунский эксперимент» произвел неизгладимое впечатление на специалистов. Тогда казалось, что достаточно создать большие хранилища словарей для перевода с одного языка на другой, разработать правила перевода – и проблема будет решена. Когда выяснилось, что проблема не так проста, был создан язык-посредник, облегчающий сопоставление фраз на разных языках. Во второй половине семидесятых годов этот язык-посредник превратился в семантическую модель представления смысла переводимых текстов. Таким образом, достижения в области машинного перевода оказались очень полезными для искусственного интеллекта, так как они показали, что анализ естественно-языковых конструкций невозможен без создания семантической модели, в которой интерпретируется смысл слов. Создание семантической модели привело к пониманию необходимости создания языка для внутреннего представления знаний. Важным достижением явилось понимание того, что анализ текстов на естественном языке состоит из четырех основных этапов: морфологический анализ, синтаксический анализ, семантический анализ, прагматический анализ.

Задача разбора предложения на естественном языке даже с ограниченным словарем в сотню слов является типичной задачей искусственного интеллекта. Для достаточно формальных предметных областей и конкретных наборов предметов, обладающих однозначными, недвусмысленными характеристиками и названиями, такая система была создана Т. Виноградом в 1971 году. Его знаменитая программа SHRDLU работает в абстрактном мире, состоящем из разноцветных кубиков и пирамид. В этом мире функционирует «однорукий» робот Робби, способный общаться с человеком на естественном (английском) языке и совершать манипуляции с объектами этого мира. Этот робот также способен понимать достаточно сложные выражения, самостоятельно планировать свою деятельность и объяснять мотивы тех или иных действий.

Вот небольшой пример – фрагмент реального диалога с этой системой:

– Найди блок, который выше, чем тот, что в твоей руке, и положи его в коробку.

– ОК.

– Поставь, пожалуйста, один из красных блоков на другой, а сверху либо зеленый блок, либо пирамиду.

– ОК.

– Служит ли самая короткая вещь, которая опирается на опору самой высокой пирамиды, опорой для чего-то еще?

– Да, для зеленой пирамиды.

– Когда ты брался за зеленую пирамиду?

– В то время, когда я строил башенку из красного куба, большого красного блока и большого зеленого куба.

– Почему?

– Чтобы от нее избавиться.

– Почему?

– Чтобы освободить красный куб.

Самым интересным здесь является то, что программа не отвечает на заранее заданные типовые фразы. SHRDLU, можно сказать, «понимает», о чем ее спрашивают, и отвечает достаточно разумно. Сам алгоритм Винограда занимает всего несколько сотен строк кода на языке LISP.

Надо отметить, что даже для английского языка, который служит основой для всех современных языков программирования в силу своей лаконичности и достаточно формальной семантики, до сего дня не удалось создать более-менее эффективную программную систему, способную адекватно понимать смысл фраз из достаточно больших областей знаний, например, нашего обыденного мира.

В разборе и понимании естественного русского языка массу проблем создает сложная падежная система, склонения, времена, отсутствие формального порядка следования членов предложения. Тем не менее российскими учеными созданы эффективные системы разбора фраз ограниченного естественного языка.

Игровые программы.

К числу первых игровых программ можно отнести программу Артура Самуэля по игре в чекерс (американские шашки), написанную в 1947 году, причем в ней использовался ряд основополагающих идей искусственного интеллекта, таких, как перебор вариантов и самообучение.

Научить компьютер играть в шахматы – одна из интереснейших задач в сфере игровых программ, использующих методы искусственного интеллекта. Она была поставлена уже на заре вычислительной техники, в конце 50-х годов. В шахматах существуют определенные уровни мастерства, степени качества игры, которые могут дать четкие критерии интеллектуального роста машины. Поэтому компьютерными шахматами активно занимались ученые умы во всем мире. Но шахматы – игра, соревнование, и чтобы продемонстрировать свои логические способности, компьютеру необходим непосредственный противник. В 1974 году впервые прошел чемпионат мира среди шахматных программ в рамках очередного конгресса IFIP (International Federation of Information Processing) в Стокгольме. Победителем этого состязания стала советская шахматная программа «Каисса» (Каисса - богиня, покровительница шахмат). Эта программа была создана в Москве, в Институте проблем управления Академии наук в команде разработчиков программы-чемпиона, лидерами были Владимир Арлазаров, Михаил Донской и Георгий Адельсон-Вельский. «Каисса» показала всему миру способности русских специалистов в области эвристического программирования.

Машинное творчество.

В 1957 году американские исследователи М. Мэтьюз и Н. Гутман посетили концерт одного малоизвестного пианиста. Концерт им обоим не понравился, и, придя домой, М. Мэтьюз тут же стал писать программу, играющую музыку. Идея Мэтьюза, развиваясь, породила целый класс музыкальных языков программирования, которые вначале назывались MUSIC с номером версии. Язык C-Sound произошел как раз из этих программ. А отделение Стэндфордского института исследований, где работал тогда М. Мэтьюз, выросло в музыкальный исследовательский центр под названием CCRMA.

В 1959 году советский математик Рудольф Зарипов начал «сочинять» одноголосные музыкальные пьесы на ЭВМ «Урал» [16]. Они назывались «Уральские напевы» и носили характер эксперимента. При их сочинении использовались случайные процессы для различных элементов музыкальной фактуры (форма, ритм, высота звука и т.д.). С тех пор появилось очень много программ для алгоритмической композиции. Для различных музыкальных задач было создано специальное программное обеспечение: системы многоканального сведения; системы обработки звука; системы синтеза звука; системы интерактивной композиции; программы алгоритмической композиции и др.

В 1975 – 1976 гг. были проведены эксперименты по сравнению машинной и «человеческой» музыки. Для эксперимента были выбраны мелодии песен известных советских композиторов, опубликованные в сборниках избранных песен, и мелодии, сочиненные на вычислительной машине «Урал-2» по программе Р. Зарипова. Результаты экспериментов таковы: машинные сочинения жюри признало в большинстве случаев наиболее интересными и, «без сомнения, написанными человеком». Таким образом, деятельность машины удовлетворяла критерию Тьюринга – слушатели-эксперты не узнали ее.

Д.А. Поспелов в своем интервью «Литературной газете» [№ 1, 1976] слегка иронизирует над методом Р. Зарипова, вспоминая, что примерно такой же способ «творчества» предложил еще Остап Бендер в «Золотом теленке», продав журналисту Ухудшанскому свое «Незаменимое пособие для сочинения юбилейных статей, табельных фельетонов, а также парадных стихотворений, од и тропарей», избавляющее от «необходимости ждать, покуда вас окатит потный вал вдохновенья». Из раздела первого (словарь) берутся нужные существительные, прилагательные, глаголы, смешиваются по образцам раздела второго (творческая часть), и получается «шедевр». Такой метод можно запрограммировать, и можно написать повести, рассказы, стихи. Но вряд ли это можно назвать творчеством. Практически, очевидно, что таким образом не будет создано гениальное в общечеловеческом смысле произведение.

Не будем требовать от интеллектуальных систем гениальности. ИС уже сейчас способны делать много полезной и разумной работы, которая требует какой-то доли интеллекта.

Среди направлений работ в области искусственного интеллекта следует также выделить нейрокибернетику, или, иначе говоря, подход к разработке машин, демонстрирующих «разумное» поведение, на основе архитектур, напоминающих устройство мозга и называемых нейронными сетями. В 1942 году, когда Н. Винер определил концепции кибернетики, В. Мак-Каллок и В. Питс опубликовали первый фундаментальный труд по нейронным сетям, где говорилось о том, что любое хорошо заданное отношение вход – выход может быть представлено в виде формальной НС.

Одна из ключевых особенностей нейронных сетей состоит в том, что они способны обучаться на основе опыта, полученного в обучающей среде. В 1957 году Ф. Розенблат [21] изобрел устройство для распознавания на основе нейронной сети – персептрон, который успешно различал буквы алфавита, хотя и отличался высокой чувствительностью к их написанию.

Читателю, возможно, интересно узнать, что у рядовых муравьев и пчел примерно 80 нейронов на особь (у царицы – 200 – 300 нейронов), у тараканов – 300 нейронов. Эти существа показывают отличные – адаптационные свойства в процессе эволюции. У человека число нейронов более 1010.

Пик интереса к нейронным сетям приходится на 60-е и 70-е годы, но в последние десять лет наблюдается резко возросший объем исследований и разработок. Это стало возможным в связи с появлением нового аппаратного обеспечения, повысившего производительность вычислений в нейронных сетях (нейропроцессоры, транспьютеры и т.п.). Нейронные сети хорошо подходят для распознавания образов и решения задач классификации, оптимизации и прогнозирования. Поэтому основными областями применения нейронных сетей являются:

- промышленное производство и робототехника;

- военная промышленность и аэронавтика;

- банки и страховые компании;

- службы безопасности;

- биомедицинская промышленность;

- телевидение и связь; и другие области.

В 1981 г. японские специалисты, объединившие свои усилия под эгидой научно-исследовательского центра по обработке информации JIPDEC, опубликовали программу НИОКР с целью создания к 1991 году прототипа ЭВМ нового поколения. Эта программа, получившая на Западе название «японский вызов», была представлена как попытка построить интеллектуальный компьютер, к которому можно было бы обращаться на естественном языке и вести беседу.

Серьезность, с которой основные конкуренты Японии откликнулись на брошенный им вызов, объясняется тем, что прежде переход от одного поколения к другому характеризовался изменением элементной базы, ростом производительности и расширением сервисных возможностей для пользователей. Переход к ЭВМ пятого поколения означал резкий рост «интеллектуальных» способностей компьютера и возможность диалога между компьютером и непрофессиональным пользователем на естественном языке, в том числе в речевой форме или путем обмена графической информацией – с помощью чертежей, схем, графиков, рисунков. В

состав ЭВМ пятого поколения также должна войти система решения задач и логического мышления, обеспечивающая способность машины к самообучению, ассоциативной обработке информации и получению логических выводов. Уровень «дружелюбия» ЭВМ по отношению к пользователю повысится настолько, что специалист из любой предметной области, не имеющий навыков работы с компьютером, сможет пользоваться ЭВМ при помощи естественных для человека средств общения – речи, рукописного текста, изображений и образов.

Лекция 2. Основные компоненты нейронных сетей. Правило распространения сигналов в сети. Области применения нейронных сетей. Персептрон. Функции активации. Обучение нейронных сетей. Алгоритм обратного распространения ошибки. Обучение с учителем и без учителя. Градиентные методы обучения. Эвристические алгоритмы для обучения нейронной сети.

Нейронная сеть является совокупностью элементов, соединенных некоторым образом так, чтобы между ними обеспечивалось взаимодействие [12]. Эти элементы, называемые также нейронами или узлами, представляют собой простые процессоры, вычислительные возможности которых обычно ограничиваются некоторым правилом комбинирования входных сигналов и правилом активизации, позволяющим вычислить выходной сигнал по совокупности входных сигналов. Выходной сигнал элемента может посылаться другим элементам по взвешенным связям, с каждой из которых связан весовой коэффициент или вес. В зависимости от значения весового коэффициента передаваемый сигнал или усиливается, или подавляется. Элемент нейронной сети схематически показан на следующем рисунке.

Один из самых привлекательных аспектов использования нейронных сетей заключается в том, что, хотя элементы такой сети имеют очень ограниченные вычислительные возможности, вся сеть в целом, объединяя большое число таких элементов, оказывается способной выполнять довольно сложные задачи [13].

Структура связей отражает детали конструкции сети, а именно то, какие элементы соединены, в каком направлении работают соединения и каков уровень значимости (вес) каждого из соединений. Задача, которую понимает сеть (или ее программа), описывается в терминах весовых значений связей, связывающих элементы. Структура связей обычно определяется в два этапа: сначала разработчик системы указывает, какие элементы должны быть связаны и в каком направлении, а затем в процессе фазы обучения определяются значения соответствующих весовых коэффициентов.

Весовые коэффициенты можно определить и без проведения обучения, но как раз самое большое преимущество нейронных сетей заключается в их способности обучаться выполнению задачи на основе тех данных, которые сеть будет получать в процессе реальной работы. Существует множество различных типов нейронных сетей, но большинство сетей обладают рядом общих характеристик, которые можно представить с помощью следующих абстракций:

- множество простых процессоров,

- структура связей,

- правило распространения сигналов в сети,

- правило комбинирования входящих сигналов,

- правило вычисления сигнала активности,

- правило обучения, корректирующее связи.

Искусственные нейронные сети различаются своей архитектурой: структурой связи между нейронами, числом слоев, функцией активации нейронов, алгоритмом обучения. С этой точки зрения среди известных искусственных нейронных сетей можно выделить статические, динамические сети и нечеткие.

Различия вычислительных процессов в сетях часто обусловлены способом взаимосвязи нейронов, поэтому выделяют следующие виды сетей:

- сети прямого распространения – сигнал проходит по сети от входа к выходу в одном направлении;

- сети с обратными связями;

- сети с боковыми обратными связями;

- гибридные сети.

В целом, по структуре связей нейронные сети могут быть сгруппированы в два класса, сети прямого распространения – без обратных связей в структуре и рекуррентные сети – с обратными связями. В первом классе наиболее известными и чаще используемыми являются многослойные нейронные сети, где искусственные нейроны расположены слоями. Связь между слоями однонаправленная, и в общем случае выход каждого нейрона связан со всеми входами нейронов последующего слоя. Такие сети являются статическими, т.к. не имеют в своей структуре ни обратных связей, ни динамических элементов, а выход зависит от заданного множества на входе и не зависит от предыдущих состояний сети. Сети второго класса являются динамическими, т.к. из-за обратных связей состояние сети в каждый момент времени зависит от предшествующего состояния.

Простейшая сеть (см. рис. 2.1) состоит из группы нейронов, образующих слой.

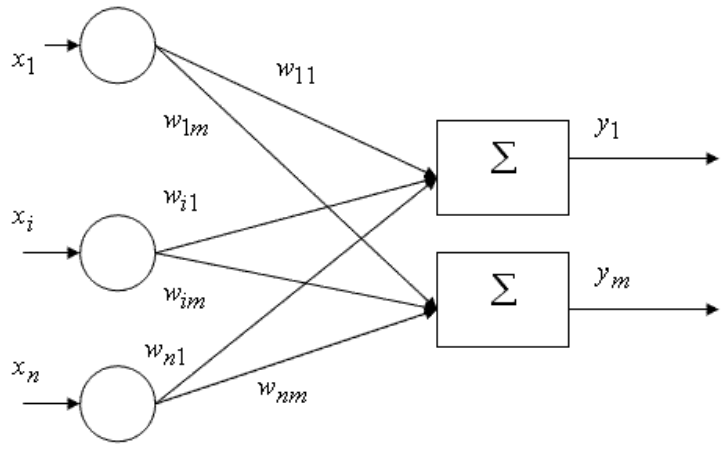


Рис. 2.1. Простейшая нейронная сеть

На рис. 2.1. вершины-круги слева служат для распределения входных сигналов. Они не выполняют каких-либо вычислений, и поэтому не считаются слоем. Они обозначены кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами. Каждый элемент из множества входов отдельным весом соединен с каждым искусственным нейроном. Каждый нейрон выдает взвешенную сумму входов в сеть. Могут иметь место также соединения между выходами и входами элементов в слое.

Удобно считать веса элементами матрицы **W**. Матрица имеет *m* строк и *n* столбцов, где *m* – число входов, а *n* – число нейронов. Например, w_{23} – это вес, связывающий второй вход с третьим нейроном. Таким образом, вычисление выходного вектора **y**, компонентами которого являются выходы нейронов, сводится к матричному умножению

$$\mathbf{y} = \mathbf{W}^T \mathbf{x}.$$

Как правило, веса w_{ij} являются параметрами сети и могут изменяться. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы – как внешние выходы сети.

При подаче любых чисел на входы сети получается какой-то набор чисел на выходах сети. Таким образом, работа нейронной сети состоит в преобразовании входного вектора в выходной вектор, причем это преобразование задается весами сети. Практически любую задачу можно свести к задаче, решаемой нейронной сетью.

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

В многослойной сети (см. рис. 2.2) нейроны расположены в несколько слоев. Нейроны первого слоя получают входные сигналы, преобразуют их и через точки ветвления передают нейронам второго слоя. Далее срабатывает второй слой и т.д. до последнего слоя, который выдает выходные сигналы для интерпретатора и пользователя. Если не оговорено противное, то каждый выходной сигнал *i*-го слоя подается на вход всех нейронов *i+1*-го слоя.

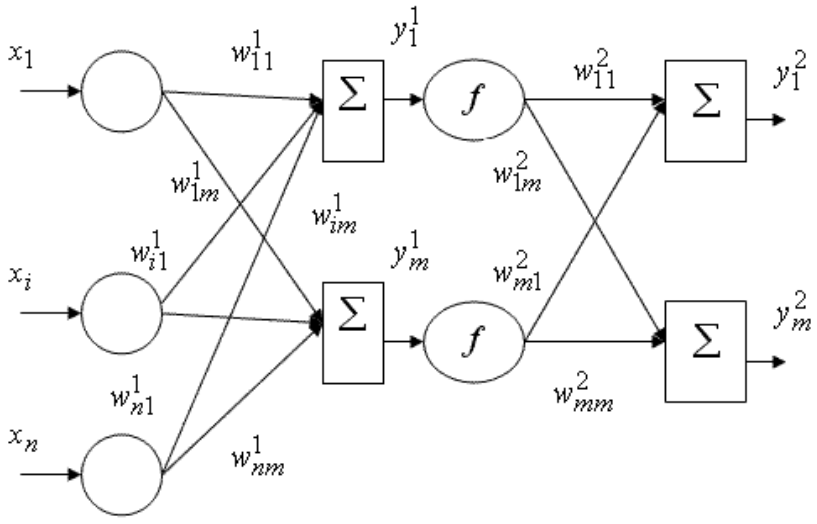


Рис. 2.2. Многослойная нейронная сеть

Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. Теоретически число слоев может быть произвольным, однако фактически оно ограничено ресурсами компьютера, на котором реализуется нейронная сеть.

Стандартный способ подачи входных сигналов, когда все нейроны первого слоя получают каждый входной сигнал.

На рисунке f обозначает нелинейную функцию, которая называется активационной функцией. Преобразования сигналов в многослойной сети имеет следующее описание

$$\mathbf{y}^2 = \mathbf{W}^{2^T}\mathbf{f}\left(\mathbf{y}^1\right),\ \mathbf{y}^1 = \mathbf{W}^{1^T}\mathbf{x},$$

где

$$\mathbf{f}\left(\mathbf{y}^1\right) = \left[f\left(y_1^1\right)\ldots f\left(y_m^1\right)\right]^T.$$

Нелинейная активационная функция $f(y)$ может иметь различный вид, но чаще всего используется либо пороговая функция

$$f(y) = \begin{cases} 1, & \text{если } y > y^+ \\ 0, & \text{иначе} \end{cases},$$

либо логистическая сигмоидная

$$f(y) = \frac{1}{1+e^{-y}}.$$

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

Обучение искусственных нейронных сетей.

Сеть обучается, чтобы для некоторого множества входов давать желаемое (или, по крайней мере, сообразное с ним) множество выходов.

Существуют три парадигмы обучения: "с учителем", "без учителя" (самообучение) и смешанная.

Обучение с учителем (см. рис. 2.3) предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором числе таких обучающих пар. Предъявляется выходной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором, разность (ошибка) с помощью обратной связи подается в сеть, и веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку. Векторы обучающего множества

предъявляются последовательно, вычисляются ошибки, и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

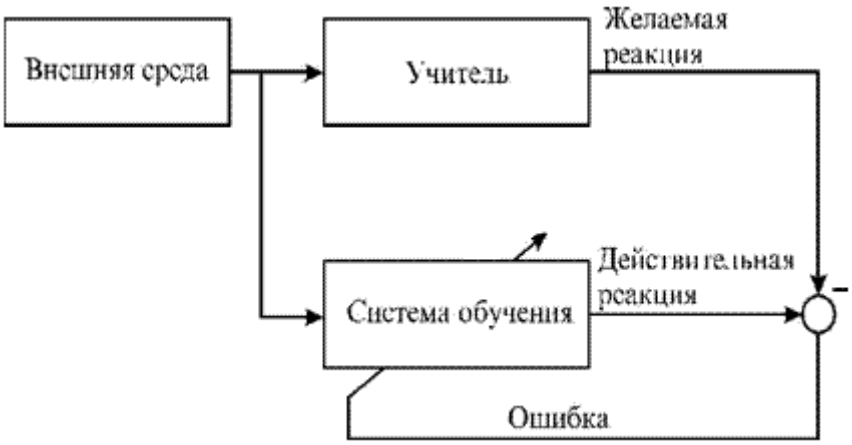


Рис. 2.3. Схема обучения нейронной сети с учителем

Обучение без учителя (см. рис. 2.3) является намного более правдоподобной моделью обучения в биологической системе. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, т.е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения невозможно предсказать, какой выход будет производиться данным классом входных векторов. Следовательно, выходы подобной сети должны трансформироваться в некоторую понятную форму, обусловленную процессом обучения.

Большинство современных алгоритмов строится на концепции Хэбба. Им предложена модель обучения без учителя, в которой синаптическая сила (вес) возрастает, если активированы оба нейрона, источник и приемник. Таким образом, часто используемые пути в сети усиливаются и феномен привычки и обучения через повторение получает объяснение.

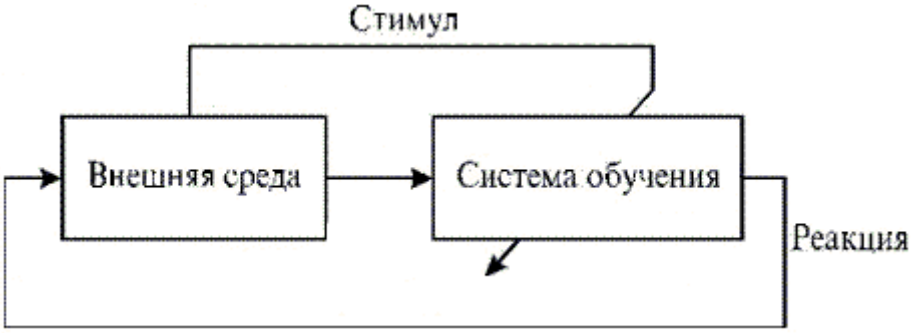


Рис. 2.4. Схема обучения нейронной сети без учителя

Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки опирается на обобщение дельта-правила.

Представим производную ошибки в виде

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

и определим δ_j с помощью формулы

$$\delta_j = \frac{\partial E}{\partial net_j}.$$

Исходное дельта-правило определяет δ_j как $\delta_j = \partial E / \partial o_j$, но новое определение оказывается согласованным с исходным, так как исходное дельта-правило предназначается для линейных элементов, в которых выход оказывается равным входу.

Перепишем последнее правило

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j}.$$

Поскольку

$$E_p = \frac{1}{2} \sum_j (t_j - o_j)^2,$$

то получаем

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j).$$

Для функции активации, которая часто оказывается логистической функцией, имеем

$$o_j = f(net_j).$$

Поэтому для производной получаем

$$\frac{\partial o_j}{\partial net_j} = f'(net_j).$$

Таким образом,

$$\delta_j = (t_j - o_j) f'(net_j).$$

Для нахождения комбинированного входа используется обычное суммирование произведений

$$net_j = \sum_{i=0} x_j w_{ij}.$$

Поэтому

$$\frac{\partial net_j}{\partial w_{ij}} = x_i .$$

Из полученных производных следует

$$\frac{\partial E}{\partial w_{ij}} = -(t_j - o_j) f'(net_j) x_j .$$

С учетом того, что вес будет изменяться в направлении противоположному тому, которое указывает производная поверхности ошибок, и с учетом нормы обучения η изменение веса должно производиться по формуле

$$\Delta w_{ij} = \eta \delta_j x_i .$$

Указанная ошибка соответствует ошибке выходного элемента, но ошибка скрытого элемента не связана с целевым выходным значением непосредственно. Поэтому весовые значения скрытого элемента следует скорректировать пропорционально его «вкладу» в величину ошибки следующего слоя, т.е. выходного слоя в случае сети с одним слоем.

В сети с одним скрытым слоем при распространении сигналов ошибок в обратном направлении ошибка каждого выходного слоя вносит свой вклад в ошибку каждого элемента скрытого слоя. Этот вклад для элементов скрытого слоя зависит от величины ошибки выходного элемента и весового коэффициента связи, соединяющей элементы. Другими словами, выходной элемент с большей ошибкой делает больший вклад в ошибку того элемента скрытого слоя, который связан с данным выходным элементом большим по величине весом. Для скрытого элемента ошибка вычисляется по формуле

$$\delta_j = f'(net_j) \sum_k \delta_k w_{kj} ,$$

где k -номер слоя, посылающего ошибку обратно.

Подходящей функцией активации является логистическая функция

$$f(net_j) = \frac{1}{1 + e^{-net_j}} .$$

Производная этой функции активации равна

$$f'(net_j) = \frac{e^{-net_j}}{\left(1 + e^{-net_j}\right)^2} =$$

$$= \frac{1}{1 + e^{-net_j}} \left(1 - \frac{1}{1 + e^{-net_j}}\right) = f(net_j)(1 - f(net_j)).$$

Алгоритм обратного распространения ошибки

Шаг 1. Прочитать первый входной образец.
Шаг 2. Для входного слоя установить совокупный ввод каждого элемента равным соответствующему элементу входного вектора. Значение вывода каждого элемента установить равным вводу.
Шаг 3. Для первого скрытого слоя вычислить совокупный ввод и вывод

$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij}, \quad o_j = \frac{1}{1 + e^{-net_j}}.$$

Повторить шаг 3 для всех последующих скрытых слоев
Шаг 4. Для элементов выходного слоя вычислить совокупный ввод и вывод

$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij}, \quad o_j = \frac{1}{1 + e^{-net_j}}.$$

Шаг 5. Попадает ли разность между целевым выходным образцом и реальным выводом сети в допустимые рамки.
Шаг 6. Для каждого выходного элемента вычислить его ошибку

$$\delta_j = (t_j - o_j) o_j (1 - o_j).$$

Шаг 7. Для последнего скрытого слоя вычислить ошибку каждого элемента

$$\delta_k = o_j (t_j - o_j) \sum_k \delta_k w_{kj}.$$

Повторить шаг 7 для всех остальных скрытых слоев.
Шаг 8. Для всех слоев обновить значения весов каждого элемента

$$\Delta w_{ij}(n+1) = \eta(\delta_j o_i) + \alpha \Delta w_{ij}(n).$$

Градиентные методы обучения. Эвристические алгоритмы для обучения нейронной сети.

Процесс обучения можно считать процессом решения оптимизационной задачи, в которой ищется вектор весов нейронной сети, обеспечивающий минимум ошибки между желаемыми значениями и реальными значениями на выходах сети.

Такая оптимизационная задача может решаться с использованием обычных методов нелинейного программирования. К методам, успешно используемым для обучения нейронной сети, можно отнести методы нулевого порядка, Хука-Дживса, покоординатного спуска, случайного поиска, и методы первого порядка – метод наискорейшего координатного спуска. Основной проблемой при использовании методов нелинейного программирования для обучения сети является большое количество параметров. Поэтому наиболее предпочтительными являются методы, использующие строго определенное количество итераций. К таким методам следует отнести метод случайного поиска и генетический алгоритм.

Генетический алгоритм наиболее успешно применяется в последнее время для обучения нейронных сетей. Суть генетического алгоритма заключается в том, что на множестве случайно сгенерированных закодированных специальным образом решениях выполняются генетические операции. В результате действия этих операций появляются новые решения, которые

сравниваются по функции пригодности с остальными. Если значения функции пригодности неудовлетворительны, то данное возможное решение с большой вероятностью не будет участвовать в генетических операциях и, скорее всего, будет отброшено, как непригодное.

Возможные решения кодируются чаще всего с помощью кода Грея, поэтому каждое решение представляет собой битовую строку. Основной генетической операцией является операция скрещивания. Данная операция выполняется аналогично скрещиванию хромосом в живых организмах. Выбираются две хромосомы. Затем находят точку скрещивания и обменивают части хромосом после точки скрещивания. Получают две новые хромосомы. В каждой из них производят случайное изменение нескольких бит, чаще всего одного. Это изменение называют мутацией. Новые хромосомы после скрещивания называют потомками. Они содержат признаки обоих родителей. Если скрещивание получилось удачно, то потомки сохраняются в течение долгих поколений. Алгоритм заканчивает свою работу после конечного числа циклов, называемых поколениями.

Лекция 3.Использование нейронной сети для исследования взаимосвязей и прогнозирования. Нейронные системы автоматического регулирования.

На нейронных сетях задача прогнозирования [4] формализуется как задача распознавания образов. Данные о прогнозируемой переменной за некоторый промежуток времени образуют образ, класс которого определяется значением прогнозируемой переменной в некоторый момент времени за пределами данного промежутка, т.е. значением переменной через интервал прогнозирования.

Прогнозирование осуществляется по тому же принципу, что и формирование обучающей выборки. При этом выделяются две возможности: одношаговое и многошаговое прогнозирование.

Многошаговое прогнозирование используется для осуществления долгосрочного прогноза и предназначено для определения основного тренда и главных точек изменения тренда для некоторого промежутка времени в будущем. При этом прогнозирующая система использует полученные (выходные) данные для моментов времени k +1, k +2 и т.д. в качестве входных данных для прогнозирования на моменты времени k +2, k +3 и т.д.

Одношаговое прогнозирование используется для краткосрочных прогнозов, обычно - абсолютных значений последовательности. Осуществляется прогноз только на один шаг вперед, но используется реальное, а не прогнозируемое значение для осуществления прогноза на следующем шаге.

Схема использования нейронной сети для одношагового прогнозирования приведена на рис. 3.1.

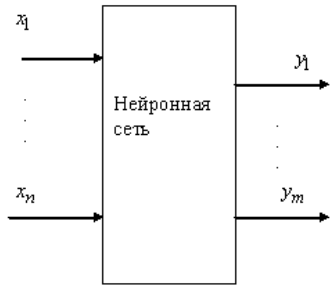


Рис. 3.1. Одношаговое прогнозирование

При одношаговом прогнозировании на вход нейронной сети подаются все параметры, от которых зависит прогнозируемые величины. При этом предыдущие наблюдения входных и выходных переменных используются для обучения нейронной сети.

Многошаговое прогнозирование представлено на рис. 3.2. При многошаговом прогнозировании на вход нейронной сети подаются значения параметров, которые влияют на прогнозируемые величины, для разных моментов времени. Многошаговое прогнозирование используется тогда, когда известно, что прогнозируемая величина зависит не только от значений входных параметров, но и от их изменения.

Как правило, больше, чем изменения за один такт времени, на вход нейронной сети не подается. Многошаговое прогнозирование может выловить медленно меняющийся тренд в прогнозируемых величинах. Когда не ясно, влияет ли динамика входных величин на прогноз, то в этих случаях также лучше использовать многошаговое прогнозирование.

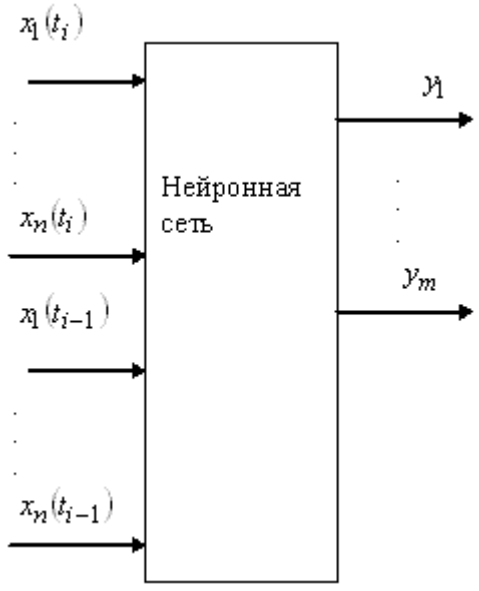


Рис. 3.2. Многошаговое прогнозирование

Управление

Предмет теории управления составляют анализ и синтез динамических систем, в которых изменение одной или нескольких переменных ограничивается в определенных пределах.

Задачи управления можно описать следующим образом. Пусть задан объект управления, описываемый системой обыкновенных дифференциальных уравнений.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u},t),$$

где **x**- вектор состояния объекта, $\mathbf{x} \in \mathbb{R}^n$, **u**- вектор управления, $\mathbf{u} \in \boldsymbol{U} \subseteq \mathbb{R}^m$, $\boldsymbol{U}$ - ограниченное замкнутое множество, $m \le n$.

Задан наблюдатель, обеспечивающий наблюдение выходных сигналов объекта

$$\mathbf{y} = \mathbf{v}(\mathbf{x},\mathbf{u}),$$

где **y** - вектор наблюдения, $\mathbf{y} \in \mathbb{R}^l$.

Необходимо найти управление в виде

$$\mathbf{u} = \mathbf{g}(\mathbf{y},\mathbf{z},t),$$

где **z** - вектор входных воздействий, $\mathbf{z} \in \mathbb{R}^r$.

Управление должно обеспечить минимизацию одних критериев качества

$$J_i(\mathbf{u},\mathbf{x},t) \to \min\ ,\ i = \overline{1,q}$$

и выполнение ограничений по другим критериям качества

$$F_i(\mathbf{u},\mathbf{x},t) \le F_i^+\ ,\ i = \overline{1,k}.$$

Наиболее правильное использование нейронной сети в управлении [18] – это подстройка параметров системы управления в зависимости от изменения внешних условий или объекта управления. Например, пусть объект управления меняет свои свойства от внешних воздействий. Это может быть транспортный робот, который в зависимости от веса груза имеет разные динамические характеристики. При большой нагрузке робот более инертен. Невозможно рассчитать параметры регулятора для всех возможных нагрузок. При этом качество регулирования при каждой нагрузке можно обеспечить только при определенных параметрах регулятора.

Другим примером влияния внешних условий на динамику объекта управления является управление парусным судном или планером. При изменении ветра характеристики управляющего воздействия изменяются. Данные характеристики необходимо учитывать при построении системы управления. Для качественного управления при изменении ветра необходимо строить систему управления с разными оптимальными параметрами.

На рис. 3.3 и 3.4 приведена схема системы управления с нейронной сетью в контуре управления.

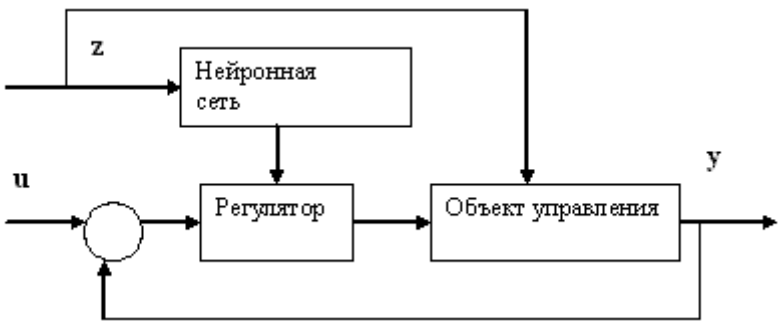


Рис. 3.3. Нейронная сеть для настройки параметров регулятора прямого контура управления

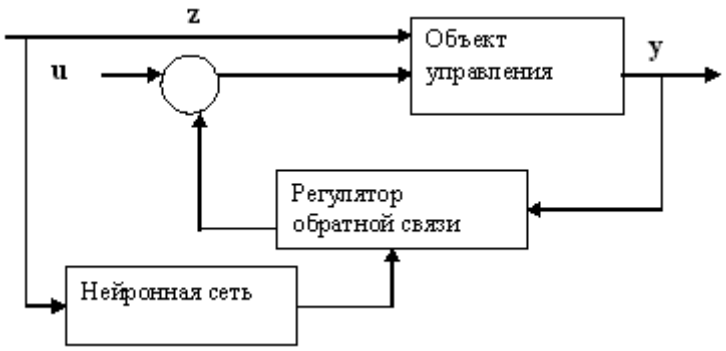


Рис. 3.4. Нейронная сеть для настройки параметров регулятора обратной связи

Если известны оптимальные значения параметров для различных внешних воздействий, точнее, для различных характеристик объекта управления, то нейронная сеть в процессе управления может использоваться для настройки параметров регуляторов. При этом расчеты оптимальных значений параметров для различных внешних воздействий могут использоваться при обучении нейронной сети. В данном случае нейронная сеть настраивает параметры регуляторов для не проанализированных ранее внешних воздействий.

Другим успешным применением нейронной сети - это ее использование для получения внешнего, программного воздействия. Система управления получает внешнее программное воздействие от нейронной сети, обрабатывает его и вырабатывает управление для объекта. Результаты управления обрабатываются и посылаются на нейронную сеть для получения следующего значения программного управления. Схема такой системы управления приведена на рис. 3.5.

Рис. 3.5. Нейронная сеть для выработки программного управления

Анализатор качества управления может подсчитывать значения критериев качества или отклонения траектории объекта управления от оптимальной траектории.

Нейронную сеть можно также использовать для принятия решения по результатам анализа внешних воздействий и поведения объекта управления. В этом случае нейронная сеть используется вместо динамической экспертной системы. По результатам анализа нейронная сеть может выдать сигнал, который должен приниматься как рекомендация по стратегии управления, например выбору контура обратной связи.

Лекция 4. Нейронные сети Хопфилда. Двунаправленная ассоциативная память. Ассоциативное обратное распространение ошибок. Нейронные сети – карта Кохонена. Использование карт Кохонена в задачах классификации.

Сеть Хопфилда (Hopfield) [12] является ассоциативной сетью, ведущей себя подобно памяти, которая может вспомнить сохраненный образец даже по подсказке. Подсказка может представлять собой искаженную помехами версию нужного образца. Ассоциативная память – это память, в которой адрес связан с хранимым значением. Человек использует ассоциативную память, поэтому он не перебирает все хранящиеся в голове слова, чтобы ответить на вопрос, знает он данное слово или нет.

Сеть Хопфилда может использоваться для распознавания букв. Если сети предъявлен искаженная версия сохраненного символа, сеть должна оказаться способной найти истинный экземпляр. Дискретная сеть Хопфилда имеет следующие характеристики:

- один слой элементов, входные элементы, представляющие входной образец, не учитываются;

- каждый элемент связывается с другими элементами, но элемент не связывается с самим собой;

- за один шаг обновляется только один элемент, в отличие от сети с обратным распространением ошибок, где все элементы слоя могут изменяться одновременно, если сеть реализована аппаратно;

- элементы обновляются в случайном порядке, но в среднем каждый элемент должен обновляться в одной и той же мере;

- вывод элемента ограничен значениями 0 и 1.

Сеть Хопфилда является рекуррентной. Для каждого входного образца выход сети повторно используется в качестве ввода до тех пор, пока не будет достигнуто устойчивое состояние. Пример сети Хопфилда показан на рисунке 4.1.

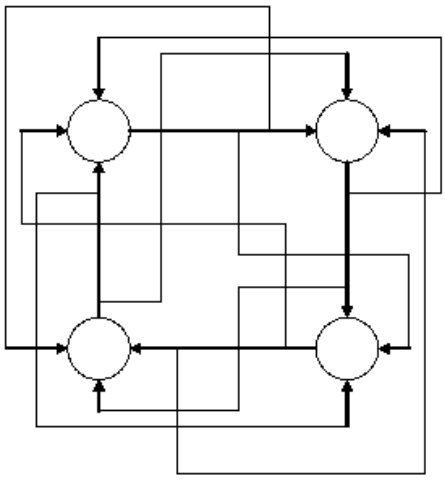


Рис. 4.1. Сеть Хопфилда с четырьмя элементами

На рис. 4.1. каждый из элементов соединяется со входами остальных элементов, но не соединяется со своим входом.

В качестве входных данных можно использовать 1 и 0, а также 1 и –1. Положительная единица обозначает состояние «включено», отрицательная единица – состояние «выключено».

Выход элемента определяется соотношением

$$net_j = \sum_{i=1}^{n} s_i w_{ij}$$

где s_i - обозначает состояние элемента с номером i .

Когда элемент обновляется, его состояние изменяется в соответствии с правилом

$$s_j = \begin{cases} +1, \text{ если } net_j > 0 \\ -1, \text{ если } net_j < 0 \end{cases}$$

Это соотношение можно записать с помощью сигнум-функции или функции знака.

$$s_j = \mathbf{sign}\left(net_j\right)$$

Если комбинированный ввод оказывается равным нулю, то элемент остается в том же состоянии, что и перед обновлением.

Сеть работает следующим образом. Входной сектор задает начальное состояние всех элементов. Элемент для обновления выбирается случайным образом. Выбранный элемент получает взвешенные сигналы от всех остальных элементов и изменяет свое состояние. Выбирается другой элемент и процесс повторяется. Сеть достигает своего предела, когда ни один из его элементов не меняет своего состояния.

Весовые коэффициенты для сети Хопфилда определяются непосредственно из учебных данных без необходимости проведения обучения в более привычном смысле. Сеть Хопфилда ведет себя как память, и процедура сохранения отдельного вектора представляет собой вычисление прямого произведения вектора с ним самим. В результате этой процедуры создается матрица, задающая весовые значения для сети Хопфилда, в которой все диагональные элементы должны быть установлены равными нулю. Таким образом, весовая матрица, соответствующая сохранению вектора, задается формулой

$$\mathbf{W} = \mathbf{x}^T \mathbf{x}$$

Хопфилд доказал, что его сеть должна сходиться к устойчивому состоянию, рассмотрев функцию энергии системы:

$$E = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} s_j s_i w_{ij}$$

Если элемент *j* меняет свое состояние на Δs_j, то изменение энергии будет равно

$$\Delta E = -\Delta s_j \sum_{i=1}^{n} s_i w_{ij}$$

Можно рассматривать изменение энергии как функцию Δs_j и $\sum_{i=1}^{n} s_i w_{ij}$. Эти величины имеют одинаковые знаки, поэтому энергия при переходе от итерации к итерации всегда уменьшается. В итоге уменьшение энергии должно прекратиться, что будет соответствовать устойчивому состоянию.

Стандартная сеть с прямой связью и обратным распространением ошибок может быть обучена автоассоциативным методом выполнению задач типа сжатия изображений. Этот метод дает возможность сделать целевой образец таким же, как и учебный образец. Сеть учится воспроизводить в выходном слое то, что подается ей на рассмотрение во входном слое. Сеть обучается стандартным образом, но целью обучения является ассоциация каждого учебного образца с самим собой. После успешного окончания обучения сеть может работать как два механизма: первый слой весов может восстанавливать полный образец из его сжатого представления. Конечно, в результате реконструкции могут быть и потери информации, поскольку сеть не будет в совершенстве создавать на выходном уровне все учебные экземпляры. Образец сжимается после его подачи на вход сети и распространения сигналов, по которым вычисляются активности скрытых элементов.

Для сети с архитектурой типа 20 - 10 – 20 (20 входных элементов, 10 скрытых элементов, 20 выходных элементов) отношением сжатия будет 2:1, поскольку для данного входного вектора сжатый вектор задается значениями активности скрытых элементов. Чтобы восстановить входной вектор, сжатый вектор предъявляется скрытому слою, и активность распространяется на выходной слой.

Иногда может оказаться выгодным обработать данные с помощью автоассоциативной сети. Известно, например, что автоаасоциативная сеть с прямой связью, имеющая один скрытый слой, вычисляет главные компоненты. Анализ главных компонент является основным методом статистической обработки данных и используется для того, чтобы удалить избыточность данных и провести кластеризацию. В результате анализа главных компонент происходит также декорреляция признаков векторов, что иногда оказывается полезным при обучении другой сети с прямой связью. Идея заключается в преобразовании учебных данных в некоторое сокращенное описание, где компонентами такого сокращения выступают элементы скрытого слоя.

Анализ главных компонент может использоваться при сжатии с потерей информации. Такое сжатие происходит при обработке изображений.

Карта Кохонена.

Основная задача, которую решают сети Кохонена – это кластеризация образцов.

При кластеризации вводятся два допущения:

- образцы внутри одного класса должны быть в некотором смысле подобны;

- кластеры, подобные в некотором смысле, должны размещаться близко друг от друга.

Самоорганизующая карта признаков (SOFM – Self-Organizing Feature Map) имеет набор входных элементов, число которых соответствует размерности учебных векторов, и набор выходных элементов, которые служат в качестве прототипов. Базовая архитектура сети SOFM показана на рис. 4.2.

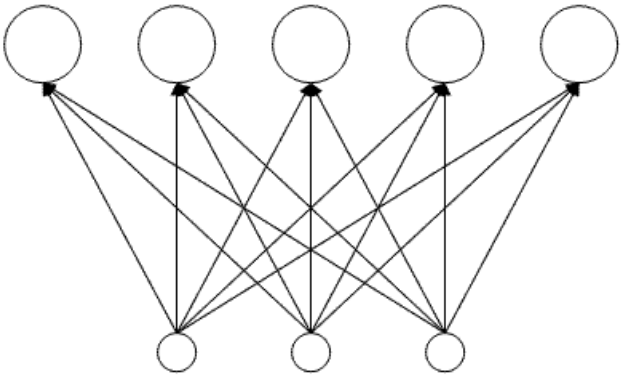


Рис. 4.2. Базовая архитектура сети SOFM

Входные элементы предназначены только для того, чтобы распределять данные входного вектора между выходными элементами сети. Выходные элементы называют кластерными элементами. Так как число входных элементов соответствует размерности вводимых векторов, а каждый входной элемент связан со всеми кластерными элементами, общее число элементов, влияющих на кластерный элемент весовых значений, тоже оказывается равным размерности входных векторов. Кластерные элементы размещаются в виде одно- или двумерного массива. В ходе обучения все элементы могут рассматриваться как претенденты на награды в виде учебных векторов. Когда на конкурс выставляется какой-либо учебный вектор, вычисляются расстояния от него до всех кластерных элементов, и элемент, который находится к данному учебному вектору ближе всех, объявляется элементом-победителем. Для элемента-победителя выполняется корректировка весовых значений так, чтобы кластерный элемент стал к учебному еще ближе. Обычно корректировка весовых значений выполняется и для элементов, близких к элементу-победителю. Весовые значения элемента подлежат обновлению, если элемент лежит внутри круга заданного радиуса с центром в элементе-победителе. В ходе обучения радиус постепенно уменьшается.

Обычно число кластерных элементов выбирается меньше, чем число учебных образцов. К концу обучения кластерные элементы обеспечивают информационную сводку по пространству входных образцов. Кластерные элементы выступают в роли карты признаков пространства входных данных.

Карта признаков проходит два этапа обучения. На первом этапе элементы упорядочиваются так, чтобы отображать пространство входных элементов, а на втором этапе происходит уточнение их позиций. Как правило, процесс представляется визуально путем использования двумерных данных и построения соответствующей поверхности. Например, входные векторы выбираются случайно в некотором квадрате. В определенные моменты строится изображение путем использования соответствия. Элементы соединяются линиями, чтобы показать их относительное размещение.

Сначала карта выглядит сильно измятой, но затем она разворачивается и расправляется. Конечным результатом обучения является карта, покрывающая все входное пространство и являющаяся достаточно регулярной.

Лекция 5. Генетический алгоритм. Основные определения, генетические операции. Структуры исходных данных при решении задач оптимизации и обучения нейронных сетей. Генетический алгоритм для решения многокритериальных задач оптимизации.

История эволюционных вычислений началась в 60-е годы с разработки различных моделей организации адаптивного случайного поиска. Основополагающим направлением методов эволюционного вычисления является эволюционное программирование, разработанное в 1960 г. Л.Фогелем (L. Vogel). Затем эволюционное вычисление было дополнено в 1964 г. И. Регенбергом (I. Rechenberg) и Г.-П. Швефелем (H.-P. Schwefel) и получило название «эволюционные стратегии». Наиболее популярным направлением в эволюционном программировании является генетический алгоритм, который получил свое развитие после выхода в 1975 г. книги Дж. Голланда (J.H. Holland) «Адаптация в естественных и искусственных системах» ("Adaptation in Natural and Artificial Systems") В начале 90-х годов Дж. Козой (J. Koza) был предложен метод генетического программирования, который представляет собой развитие генетического алгоритма с целью его использования для автоматического написания программ.

По аналогии с естественными процессами в природе в эволюционном программировании используются понятия: наследственность, изменчивость, борьба за существование и отбор. Как и в биологических популяциях, поиск оптимальных решений происходит в течение нескольких поколений, подчиняясь законам естественного отбора и принципу "выживает наиболее приспособленный". В отличие от эволюции, происходящей в природе, методы эволюционного поиска моделируют только те процессы в популяциях, которые являются существенными для развития.

Эволюционное программирование работает с совокупностью особей – популяцией. Каждая особь представляет собой возможное решение задачи и оценивается мерой ее приспособленности согласно тому, насколько хорошо она соответствует решению задачи. Появление новых особей в популяции возможно благодаря сочетанию полезных свойств родителей. Наименее приспособленные особи с меньшей вероятностью смогут воспроизвести потомков, так что те свойства, которыми они обладали, будут постепенно исчезать из популяции в процессе эволюции.

Новая популяция допустимых решений строится на основании предыдущих популяций, их непосредственного скрещивания и мутации. В результате этого хорошие характеристики распространяются по всей популяции, а сам процесс скрещивания исследует наиболее перспективные участки пространства поиска. В конечном итоге в популяции будет увеличиваться число особей, обеспечивающих решения задачи, близкие к оптимальному.

Выделим следующие достоинства и недостатки эволюционных вычислений.

Достоинства:

- возможность проблемно-ориентированного кодирования решений;

- возможность подбора начальной популяции;

- комбинирование эволюционных вычислений с неэволюционными алгоритмами;

- пригодность для поиска в сложном пространстве решений большой размерности;

- отсутствие ограничений на вид целевой функции;

- ясность схемы и базовых принципов эволюционных вычислений;

- интегрируемость эволюционных вычислений с другими неклассическими парадигмами искусственного интеллекта, такими, как искусственные нейронные сети и нечеткая логика.

Недостатки:

- эвристический характер эволюционных вычислений не гарантирует оптимальности полученного решения;

- относительно высокая вычислительная трудоемкость, которая, однако, преодолевается за счет распараллеливания на уровне организации эволюционных вычислений и на уровне их непосредственной реализации в вычислительной системе;

- относительно невысокая эффективность на заключительных фазах моделирования эволюции;

- нерешенность вопросов самоадаптации.

Таким образом, эволюционные методы применяются в тех случаях, когда трудно, а иногда и просто невозможно решить задачу иными способами. Хотя эволюционные методы не гарантируют обнаружения глобального решения за конечное время и того, что глобальное решение будет найдено, но они хороши для поиска удовлетворительного решения задачи за приемлемое время. И даже там, где хорошо работают существующие методики, можно достигнуть улучшения сочетанием их с методами эволюционного программирования.

Методы эволюционного вычисления являются эффективными при решении трудноформализуемых задач искусственного интеллекта (распознавание образов, кластеризация, ассоциативный поиск) и трудоемких задач оптимизации, аппроксимации, интеллектуальной обработки данных.

Генетический алгоритм – эволюционный метод параметрического поиска оптимального решения задачи в кодовом пространстве искомых параметров. Характерные особенности генетического алгоритма:

- варианты решения кодируются чаще всего битовой строкой, которая называется хромосомой;

- поиск осуществляется в кодовом пространстве параметров;

- оценка варианта решения осуществляется по значению функции приспособленности, которая строится на основе целевой функции и ограничений оптимизационной задачи;

- разнообразие множества кодов возможных решений достигается за счет специальных операций селекции, скрещивания и мутации;

- селекция имеет вероятностный характер и осуществляет отбор пар кодов возможных решений для скрещивания по значению функции приспособленности;

- скрещивание обеспечивает построение пары новых кодов решений на основе двух отобранных;

- мутация обеспечивает случайную вариацию кода возможного решения;

- завершение вычислений осуществляется по удовлетворительному значению функции приспособленности или по конечному числу циклов воспроизводства популяций возможных решений, которые называются поколениями;

- решением является хромосома, дающая наилучшее значение функции приспособленности.

Основные этапы генетического алгоритма представлены на рис. 5.1. [8]

Рассмотрим подробно этапы генетического алгоритма на примере решения задачи безусловной оптимизации. В задаче оптимизации требуется определить вектор параметров $\mathbf{q} = \left[q_1 \dots q_p\right]^T \in \boldsymbol{R}^p$, который обеспечивает минимум целевой функции:

$$f_0(\mathbf{q}) \to \min_{\mathbf{q} \in R^p}.$$

В генетическом алгоритме чаще всего применяется двоичное кодирование хромосом кодом Грея. Он характеризуется тем, что двоичные последовательности, соответствующие двум идущим по порядку целым числам, отличаются только одним битом.

Рис. 5.1. Основные этапы генетического алгоритма

Чтобы представить десятичное число в коде Грея, необходимо сначала его представить в двоичной системе счисления, а затем с помощью логической операции исключающего ИЛИ перевести в код Грея.

В качестве примера рассмотрим перевод в код Грея десятичного числа 29,35.

Переводим целую часть числа в двоичный код. Для этого делим число 29 на основание двоичной системы счисления (2) до тех пор, пока оно делится, фиксируя остатки 1 или 0. Затем остатки записываются в обратном порядке в виде двоичного числа.

Получаем $29_{10} = 11101_2$.

Теперь переведем в двоичный код дробную часть числа. Для этого ее многократно умножаем на 2, запоминаем и отбрасываем целую часть результатов умножений. Количество умножений определяется желаемой точностью вычислений.

В данном случае получаем: $0{,}35 \times 2 = \underline{0}{,}7\ ;\ 0{,}7 \times 2 = \underline{1}{,}4\ ;\ 0{,}4 \times 2 = \underline{0}{,}8\ ;\ 0{,}8 \times 2 = \underline{1}{,}6$ и т.д.

В результате, собирая целые части сверху вниз, получаем двоичный код дробной части $0{,}35_{10} = 0101_2$:

$$29{,}35_{10} = 11101{,}0101_2.$$

Для перевода в код Грея отдельно для дробной и целой части сохраняем первый бит, а остальные биты последовательно складываем с предыдущим с помощью операции исключающего ИЛИ. В общем виде перевод из двоичного кода в код Грея осуществляется с помощью следующих соотношений:

$$y_0 = x_0,$$

$$y_i = (x_i + x_{i-1}) \bmod 2,\ i = \overline{1,n},$$

где y_i бит числа в коде Грея в позиции i слева, x_i код двоичного кода числа в позиции i слева, $i = \overline{1,n}$, n - число бит в двоичном коде, $a \bmod b$ - остаток от деления a на b.

Для рассматриваемого примера из соотношений (2), (3) получаем:

$$11101{,}0101_2 = 10011{,}0111_G,$$

где индекс G обозначает, что число представлено в коде Грея.

Обратный перевод из кода Грея в двоичный код осуществляется с помощью следующих соотношений:

$$x_0 = y_0,$$

$$x_i = (y_i + x_{i-1}) \bmod 2,\ i = \overline{1,n}.$$

В некоторых случаях кодирование отсутствует, и хромосома состоит из действительных чисел.

Для решения задачи (1) выполняем кодировку всех компонент вектора параметров $\mathbf{q} = \left[q_1 \ldots q_p\right]^T \in \boldsymbol{R}^p$.

Для каждой компоненты определяем одинаковое число позиций под целую и дробную части. После этого соединяем все коды в одну единую битовую строку, хромосому длинной N символов.

На первом этапе генерируем случайным образом H хромосом или H векторов параметров с их последующей кодировкой. Получаем первую популяцию хромосом.

$$\mathbf{s}^h = \left(s_1^h, \ldots, s_N^h\right),\ s_i^h \in \{0,1\},\ i = \overline{1,N},\ h = \overline{1,H},$$

где N - количество бит в хромосоме.

В качестве функции приспособленности для задачи используем значение целевой функции, так как рассматриваем задачу безусловной оптимизации.

$$F(\mathbf{s}) = f_0(\varphi(\mathbf{s})) + \Omega,$$

где $\varphi(\mathbf{s})$ - взаимно однозначное отображение из множества $\boldsymbol{S}^N$ битовых строк длинной N символов в пространство $\boldsymbol{R}^p$ значений искомых параметров $\varphi(\mathbf{s}): \boldsymbol{S}^N \to \boldsymbol{R}^p$, Ω - постоянная величина для обеспечения положительности функции приспособленности, $\Omega > 0$.

Для каждой хромосомы вычисляем значение функции приспособленности.

$$F^h = F\left(\mathbf{s}^h\right),\ h = \overline{\mathbf{1, H}}.$$

В популяции находим хромосому, дающую наилучшее значение функции приспособленности.

$$F\left(\mathbf{s}^{h\_}\right) = \min\left\{F^h, h = \overline{\mathbf{1, H}}\right\},$$

где $h\_$- индекс наилучшей хромосомы.

Отбираем случайным образом пару хромосом для скрещивания. Для отобранной пары определяем вероятность скрещивания из следующего соотношения:

$$P_i = \max\left\{\frac{F^{h\_}}{F^{h_1}}, \frac{F^{h\_}}{F^{h_2}}\right\},$$

где P_i- вероятность скрещивания *i*-й пары хромосом, h_1, h_2- индексы отобранных хромосом.

В случае поиска максимума в задаче (1) соотношение (10) должно определять хромосому, дающую максимальное значение функции приспособленности, а в соотношении (11) дроби обращаются. Описанная селекция относится к разновидности метода рулетки (roulette-wheel selection), в котором вероятность скрещивания пропорциональна значению функции приспособленности хромосомы.

Существует много других видов селекции. При турнирной селекции (tournament selection) все особи популяции разбиваются на подгруппы с последующим выбором в каждой из них особи с наилучшей приспособленностью. Различаются два способа такого выбора: детерминированный выбор (deterministic tournament selection) и случайный выбор (stochastic tournament selection). Детерминированный выбор осуществляется с вероятностью, равной 1, а случайный выбор - с вероятностью, меньшей 1. Подгруппы могут иметь произвольный размер, но чаще всего популяция разделяется на подгруппы по 2 - 3 особи в каждой. В турнирном методе допускается изменение размера подгрупп, на которые подразделяется популяция.

При ранговой селекции (ranking selection) особи популяции ранжируются по значениям их функции приспособленности. Это можно представить себе как отсортированный список особей, упорядоченных по направлению от наиболее приспособленных к наименее приспособленным, в котором каждой особи приписывается число, определяющее ее место в списке, называемое рангом. Количество копий каждой особи, введенных в родительскую популяцию, рассчитывается по априорно заданной функции в зависимости от ранга особи. Ранговая селекция уменьшает потенциально возможный эффект доминирования в популяции индивидов со сравнительно высокой степенью пригодности путем установления ожидаемого, ограниченного выбора в пользу таких индивидов. В то же время эта селекция преувеличивает разницу между близко рассоложенными значениями пригодности так, что лучшие будут выбраны с большей вероятностью.

Элитный отбор заключается в том, что элитные хромосомы с наилучшим значением функции приспособленности отбираются и обязательно скрещиваются между собой и другими хромосомами.

После операции селекции выполняется операция скрещивания. Для выполнения операции одноточечного (простого) скрещивания случайным образом выбираем номер позиции в

хромосоме, которая называется точкой скрещивания. Затем обмениваем части хромосом друг с другом до точки скрещивания. Получаем две новые хромосомы.

$$\mathbf{s}^{H+1} = \left(s_1^{h_1}, \ldots, s_{k-1}^{h_1}, s_k^{h_2}, \ldots, s_N^{h_2} \right),$$

$$\mathbf{s}^{H+2} = \left(s_1^{h_2}, \ldots, s_{k-1}^{h_2}, s_k^{h_1}, \ldots, s_N^{h_1} \right),$$

где k - точка скрещивания.

Скрещивание также бывает двухточечным, многоточечным, равномерным. Двухточечное скрещивание, как следует из его названия, отличается от одноточечного скрещивания тем, что потомки наследуют фрагменты родительских хромосом, определяемые двумя случайно выбранными точками скрещивания.

После скрещивания для каждой новой хромосомы потомка выполняем операцию мутации. Обычно при выполнении данной операции изменяется случайно выбранный бит в хромосоме.

Мутация обеспечивает вариацию возможного решения с целью выхода из локального экстремума. Ее можно производить для нескольких бит хромосом, но следует помнить, что частая мутация ухудшает скорость поиска оптимального решения.

Мутация может быть точечной и абсолютной. Точечная мутация изменяет с заданной вероятностью в допустимых пределах один случайно выбранный бит в хромосоме, а при абсолютной мутации меняются все биты хромосомы.

Изменение значения бита при двоичной кодировке происходит на противоположное (с 1 на 0 или наоборот). При представлении варианта решения вектором вещественных значений изменение параметра производится случайно. Чаще всего используется нормальное или равномерное распределения.

Операции селекции, скрещивания и мутации производим R раз, где R - заданный параметр, определяющий количество отобранных пар.

Из всей популяции либо оставляем H хромосом, дающих наилучшие значения функции приспособленности, либо отбираем $H_1 < H$ хромосом, а остальные $H - H_1$ хромосом генерируем случайным образом. Полный цикл операций генерации, селекции, скрещивания, мутации, включения и исключения хромосом называется поколением. Повторяем циклы операций заданное число W раз.

Решение находим по хромосоме из последнего поколения, у которой наилучшая величина функции приспособленности.

$$\tilde{\mathbf{q}} = \varphi\left(\mathbf{s}^{h_-}\right),$$

где $\mathbf{s}^{h_-}$ - хромосома, имеющая наименьшее значение функции приспособленности.

Основными параметрами, влияющими на скорость сходимости генетического алгоритма, являются: количество поколений (W), количество хромосом в популяции (H), количество

родительских пар в поколении (R).На работу алгоритма влияет также выбор видов селекции, скрещивания и мутации.

Общее количество K проверяемых решений можно определить из соотношения

$$K = H + 2RW.$$

Оптимальность полученного решения в результате работы генетического алгоритма является предметом дополнительных исследований. Можно утверждать, что найденное решение точно является наилучшим среди K просмотренных решений. Однако такая оценка оптимальности является неудовлетворительной. Многочисленные эксперименты на разных задачах показали, что генетический алгоритм работает на порядок эффективнее алгоритма случайного поиска.

При решении задач многокритериальной оптимизации необходимо минимизировать вектор критериев $\mathbf{J}^h = \left[J_1^h \ldots J_M^h\right]^T$. Решением задачи является построение множества Парето. Отношение Парето $\mathbf{J}^{h_1} \geq \mathbf{J}^{h_2}$справедливо, если выполняются условия

$$J_i^{h_1} \geq J_i^{h_2},\ i = \overline{1,M},\ \exists k \in k\{1,\ldots,M\},\ J_i^{h_1} > J_i^{h_2}.$$

С помощью генетического алгоритма строится условное множество Парето $\boldsymbol{P}_C$, которое представляет собой совокупность возможных хромосом в популяции, для которых не существует хромосом, более лучших в смысле отношения Парето.

Чтобы построить условное множество Парето на популяции хромосом, введем характеристику, «расстояние от текущей хромосомы до условного множества Парето»

$$\Lambda^k = \sum_{h=1}^{H} \lambda_{hk},$$

где $\lambda_{hk} = \begin{cases} 1, \text{если } \mathbf{J}^h \leq \mathbf{J}^k \\ 0, \text{иначе} \end{cases}$.

В качестве функции приспособленности для оценки решения используем расстояние до условного множества Парето. Хромосомы, имеющие нулевое расстояние, принадлежат условному множеству Парето

$$\boldsymbol{P}_C = \left\{\left(\boldsymbol{W}^k, \mathbf{s}^k\right), 1 \leq k \leq H : \Lambda^k = 0\right\}.$$

Для репродукции хромосом в популяции случайно отбираем пару хромосом и определяем вероятность скрещивания по условию

$$p_S = \max\left\{\frac{1+\gamma\Lambda^{h_1}}{1+\Lambda^{h_1}}, \frac{1+\gamma\Lambda^{h_2}}{1+\Lambda^{h_2}}\right\},$$

где $\gamma \in [0,1)$ выбирается произвольно.

Лекция 6. Генетическое программирование. Структуры данных. Польская запись. Символьные генетические операции. Сетевой оператор. Матрица сетевого оператора. Вариации сетевого оператора. Принцип базисной структуры. Сетевой оператор как нейронная сеть с нелинейной структурой.

Генетическое программирование [8] является развитием метода генетического алгоритма [3]. Оно появилось при использовании генетического алгоритма для автоматического написания программ. Автоматическое написание программ подразумевает создание такой программы, которая могла бы создавать другие программы без детального описания алгоритма, используя только набор требований и условий.

В генетическом программировании возможным решением является код программы, которая удовлетворяет заданным требованиям или решает поставленную задачу. Для универсального описания программы и ее кодировки в виде записи используется польская запись программного кода.

Польская запись является промежуточным кодом, к которому преобразуют трансляторы исходные тексты программ при их переводе в машинные команды. Польская запись является удобным и универсальным видом записи программы.

Рассмотрим пример польской записи для арифметического выражения

$$A + B - \frac{C^2}{D \cdot E}$$

С учетом скобок получаем следующую запись:

$$\left((A + B) - \left((C)^2 / (D \cdot E)\right)\right).$$

Скобки определяют порядок выполнения вычислений. В обычной польской записи скобки не используются. Вычисления выполняются слева направо. Первоначально определяется тип операции, например, унарная, бинарная и т.п. В приведенном примере четыре бинарные операции сложения, вычитания, умножения и деления и одна унарная ? возведение в квадрат. Затем из записи берется необходимое число операндов для вычисления.

Для получения польской записи сначала рассматриваем самые внутренние скобки, имеющие наивысший приоритет порядка вычислений. В нашем случае это выражения $(A + B),\ (D \cdot E)$ и унарная операция возведения в квадрат C^2 Обозначим операцию возведения в квадрат значком

«^». Тогда в польской записи они имеют вид соответственно $+AB$, $*DE$ и $\wedge C$. В результате получаем промежуточную запись $(+AB-(\wedge C/*DE))$.

Затем вновь раскрываем внутренние скобки $(+AB-/\wedge C*DE)$, а после раскрытия последних скобок и получаем польскую запись рассмотренного выражения

$$+AB/\wedge C*DE.$$

Данный пример польской записи называется префиксной, т.к. в записи символ операции указывается перед символами операндов. Наряду с префиксной существует и постфиксная запись, в которой символ операции стоит после символов операндов.

Постфиксная запись данного выражения описывается следующей строкой:

$$((AB)+((C)\wedge(DE)*)/)-.$$

При чтении польской записи операции выполняются в обратном порядке. Для реализации алгоритма можно использовать стек для хранения промежуточных символов, регистр операции и регистры операндов для хранения символов операций и операндов соответственно.

При увеличении арности операций в форме польской записи можно представить все основные конструкции программирования, операторы условия, циклы, массивы, подпрограммы, арифметические операции, математические функции, булевы операции и обратные связи.

В генетическом программировании чаще используется префиксная запись.

Математические выражения удобно изображать в виде деревьев. Пусть внутренние узлы дерева представляют собой операции, а внешние узлы (листья) – операнды. Для бинарных операций левая ветвь соответствует первому операнду, а правая ветвь – второму, причем в каждой ветви дерева узлы, которые лежат ниже, соответствуют операциям, которые выполняются раньше.

Например, последнему выражению соответствует дерево, изображенное на рис. 6.1.

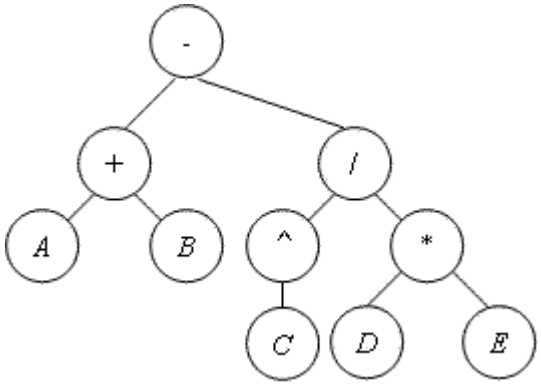


Рис. 6.1. Дерево выражения $-(+(AB)/(\wedge(C)*(DE)))$

Рассмотрим подробнее операции генетического программирования при представлении возможных решений в виде строк польской записи.

Начальная популяция записей представляет собой множество случайных, правильно сгенерированных строк в польской записи со скобками.

Для вычисления значения функции приспособленности необходимо дать количественную оценку разницы между решением, полученным с помощью польской записи, и требуемым результатом. При написании сложных логических программ формирование такой оценки не тривиально. В большинстве случаев проблема оценки пригодности хромосомы должна решаться непосредственно для конкретной задачи.

При скрещивании в генетическом программировании из двух родительских записей формируем две новые записи. Родители выбираются по правилам селекции в генетическом алгоритме. Для осуществления операции скрещивания случайным образом выбираем точку скрещивания для каждого родителя. Для этой цели генерируем случайные целые числа не больше чем длины строк родительских записей. Если оба символа в позициях, соответствующих случайным числам, в родительских записях оба являются операндами или операциями, то данные символы могут являться точками скрещивания. Если в одном родителе символ является операндом, а в другом – операцией, то скрещивание не выполняется. В результате точками скрещивания являются либо два операнда в обеих родительских записях, либо две операции.

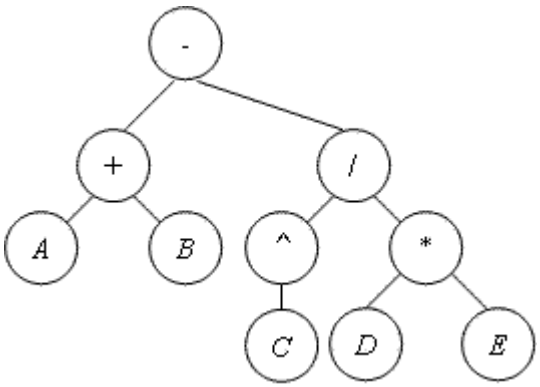
Первая запись-потомок получается путем удаления фрагмента записи из первого родителя и вставки на это место фрагмента записи из второго родителя. Второй потомок получается симметрично. Фрагментом записи является либо символ-операнд, если точки скрещивания указывают на операнды, либо подстрока, начинающаяся с символа операции и заканчивающаяся закрывающей скобкой для данной операции, если точки скрещивания указывают на операции.

В качестве примера рассмотрим скрещивание записей выражений $A+B-C^2/(D\cdot E)$ и $\sqrt{B^2-4\cdot A\cdot C}$. Префиксные записи данных выражений имеют вид соответственно $-(+(AB)/(\wedge(C)*(DE)))$ и $!(-(\wedge(B)*(4*(AC))))$, где «!» - символ операции извлечения корня.

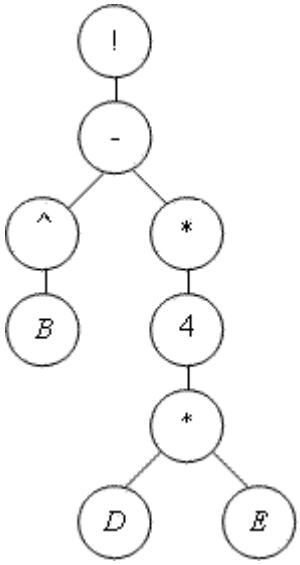
Данные выражения изображены в виде деревьев на рис. 6.2.

Пусть точками скрещивания в первой записи является восьмой символ «/», а во второй записи пятый символ – «^». Переставляя соответствующие фрагменты записей, получаем следующие новые записи: $-(+(AB)\wedge(B))$, $!(-(/(\wedge(C)*(DE))*(4*(AC))))$.

Полученные польские записи являются правильными и соответствуют следующим выражениям: $A+B-B^2$ и $\sqrt{C^2/(D\cdot E)-4\cdot A\cdot C}$.



a)



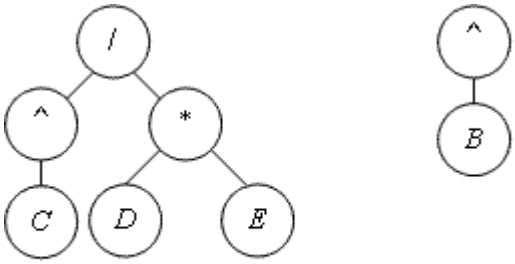
b)

Рис. 6.2. Деревья родительских записей

a) $-(+(AB)/(\hat{}(C)*(DE)))$

b) $!(-(\hat{}(B)*(4*(AC))))$

Графически весь процесс скрещивания представлен на рис. 6.2, 6.3 и 6.4.



а) б)

Рис. 6.3. Два фрагмента записей:

а) – фрагмент первой родительской записи
б) – фрагмент второй родительской записи

Из-за того, что поддерево одного родителя полностью меняется на поддерево другого родителя и свойства используемых операций, выбранных в качестве точек скрещивания, схожи, то в результате скрещивания получились правильные записи-потомки.

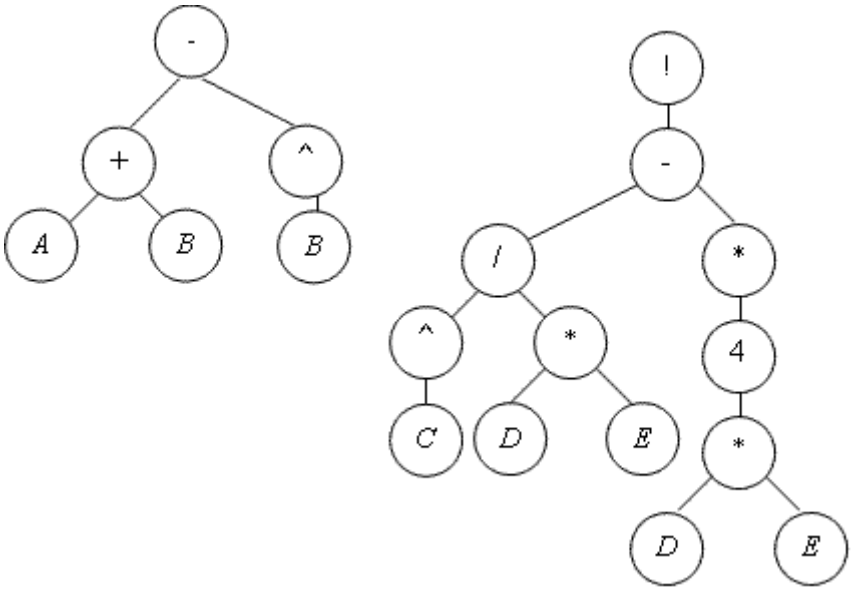


Рис. 6.4. Результат операции скрещивания - две записи-потомка

В случае если корень первой родительской записи выбирается в качестве точки скрещивания, вся эта запись полностью перемещается в точку скрещивания второй записи и становится его поддеревом. Запись-потомок, полученная таким образом, часто имеет значительную глубину. Другой же потомок будет состоять только из фрагмента записи второго родителя.

Мутация представляет собой случайные изменения в особях популяции. В обычных генетических алгоритмах мутация привносит разнообразие в популяцию, стремящуюся к преждевременной сходимости.

Мутация начинается с выбора случайной точки в записи. Точкой мутации может быть символ операции, внутренний узел, или операнд, внешний узел дерева. То, что находится в этой точке мутации и ниже, извлекается, а на это место помещается случайно сгенерированная правильная польская запись, поддерево.

Другим альтернативным способом описания математических выражений является сетевой оператор.

Сетевой оператор – это граф, который является аналогом сетевого графика.

Для построения сетевого оператора введем в рассмотрение несколько конечных упорядоченных множеств, из элементов которых состоит формула.

Множество переменных – это упорядоченное множество символов, вместо которых в процессе вычисления могут подставляться числа из множества вещественных чисел $\mathbb{R}^1$:

$$\boldsymbol{V} = (v_1, \ldots, v_P),\ v_i \in \mathbb{R}^1,\ i = \overline{\mathbf{1}, P}.$$

Множество параметров – это упорядоченное множество чисел, не меняющихся в процессе вычислений:

$$\boldsymbol{C} = (c_1, \ldots, c_R),\ c_i = \mathbf{const},\ i = \overline{\mathbf{1}, R}.$$

Множество унарных операций - это упорядоченное множество функций или однозначных отображений, заданных на числовом множестве:

$$\boldsymbol{O}_1 = (\rho_1(z), \rho_2(z), \ldots, \rho_W(z)),$$

где $\rho_i(z)$: $\mathbb{R}^1 \to \mathbb{R}^1,\ \forall z \in \mathbb{R}^1, \exists y \in \mathbb{R}^1 \Rightarrow y = \rho_i(z),\ i = \mathbf{1}, W$.

Множество бинарных операций - это упорядоченное множество функций двух аргументов или однозначных отображений декартового произведения пары одинаковых числовых множеств в одно такое же числовое множество:

$$\boldsymbol{O}_2 = (\chi_1(z', z''), \chi_2(z', z''), \ldots, \chi_V(z', z'')),$$

где $\chi_i(z', z'')$: $\mathbb{R}^1 \times \mathbb{R}^1 = \mathbb{R}^2 \to \mathbb{R}^1,\ \forall z', z'' \in \mathbb{R}^1,\ \exists y \in \mathbb{R}^1 \Rightarrow y = \chi_i(z', z''),\ i = \overline{\mathbf{1}, V}$.

Сетевой оператор - это ориентированный граф, обладающий следующими свойствами:

а) в графе отсутствуют циклы;

б) к любому узлу, который не является источником, имеется хотя бы один путь от узла-источника;

в) от любого узла, который не является стоком, имеется хотя бы один путь до узла-стока;

г) каждому узлу-источнику соответствует элемент из множества переменных $\boldsymbol{V}$ или из множества параметров $\boldsymbol{C}$;

д) каждому узлу соответствует бинарная операция из множества бинарных операций;

е) каждой дуге графа соответствует унарная операция из множества унарных операций.

Вычисления по сетевому оператору включают три этапа. На первом этапе находим узел, который имеет выходящие дуги, но не имеет ни одной входящей дуги. На втором этапе выполняем унарную и бинарную операции. Унарная операция соответствует дуге, выходящей из найденного узла. В качестве аргумента унарной операции используем значение, которое хранится в найденном узле. Бинарная операция соответствует узлу, в который дуга входит. В качестве первого аргумента бинарной операции используем либо единицу, либо результат последнего вычисления, который хранится в данном узле. В качестве второго аргумента используем результат вычисления унарной операции. На третьем этапе исключаем узел и дугу из графа. Найденный узел исключаем в том случае, если этот узел не является узлом-стоком и не имеет ни одной выходящей дуги. Дугу исключаем во всех случаях, если операция, которая соответствует данной дуге, выполнена.

Этапы вычислений выполняем до тех пор, пока в сетевом операторе не останутся только узлы-стоки, в которых хранятся результаты вычислений. Выполнение унарной операции согласно выходящей из узла дуге осуществляем только в том случае, если в узел, из которого данная дуга выходит, не входит ни одна дуга.

Для формального описания сетевого оператора в матрице смежности сетевого оператора вместо единиц, которые соответствуют дугам графа, записываем соответствующие номера унарных операций из множества $\boldsymbol{O}_1$, а на диагонали – соответствующие номера бинарных операций из множества $\boldsymbol{O}_2$.

Матрица сетевого оператора – это целочисленная матрица, на диагонали которой расположены номера бинарных операций, а остальные элементы либо нули, либо номера унарных операций, причем при замене диагональных элементов на нули, а ненулевых недиагональных элементов на единицы получаем матрицу смежности графа сети, удовлетворяющей свойствам а – в сетевого оператора.

Верхний треугольный вид матрицы сетевого оператора $\boldsymbol{\Psi} = \left[\psi_{ij}\right], i, j = \overline{1, L}$ позволяет легко осуществлять вычисления согласно сетевому оператору. Для этой цели необходимо последовательно проходить все строки матрицы и совершать вычисления с помощью унарных операций, определяемых ненулевыми недиагональными элементами в строке, и бинарные операции, определяемые номерами диагональных элементов, которые расположены в столбцах с ненулевыми недиагональными элементами.

Для удобства вычислений и хранения промежуточных результатов введем вектор узлов $\mathbf{z} = \left[z_1 \ldots z_L\right]^T$, размерность L которого равна количеству узлов сетевого оператора. Зададим начальные значения вектору узлов. Если узел является источником, то в качестве начального значения возьмем соответствующий элемент из множеств переменных или параметров $\mathbf{z}^{(0)} = \left[z_1^{(0)} \ldots z_L^{(0)}\right]^T$, $z_{s_i}^{(0)} = c_i \in \boldsymbol{C}, i = \overline{1, P}$, $z_{b_i}^{(0)} = a_i \in \boldsymbol{V}, i = \overline{1, R}$.

Определим номера узлов-источников сетевого оператора. Остальные компоненты вектора узлов представляют собой единичные элементы для бинарных операций, соответствующих узлов. Если бинарной операцией является сложение, то единицей для данной операции является ноль, а если умножение, то единица.

После задания начального значения вектора узлов $z_i^{(0)}, \quad i = \overline{1, L}$ последовательно проходим в матрице сетевого оператора $\boldsymbol{\Psi}$ все строки по столбцам. Если встречаем в строке i в столбце j не нулевой элемент, то выполняем сначала унарную операцию, номер которой определяем по элементу ψ_{ij} матрицы сетевого оператора, над значением элемента z_i вектора узлов. Затем над результатом вычислений $\boldsymbol{\rho}_{\psi_{ij}}\left(z_i\right)$ выполняем бинарную операцию, номер которой определяем по элементу $\boldsymbol{\Psi}_{jj}$ матрицы сетевого оператора, причем первым аргументом бинарной операции является последнее значение элемента z_j вектора узлов. Результат вычислений присваиваем элементу z_j вектора узлов $z_j^{(i)} = \boldsymbol{\chi}_{\Psi_{jj}}\left(z_j^{(i-1)}, \boldsymbol{\rho}_{ij}\left(z_i^{(i-1)}\right)\right)$, где $z_j^{(i)}$ - значение j -й компоненты вектора узлов после прохождения i -й строки, $i = \overline{1, L}$.

После прохождения всех строк матрицы сетевого оператора результат вычислений будут содержать компоненты вектора узлов, которые определяются по номерам выходных переменных $z_{d_i}^{(L)}$, $i = \overline{1,M}$.

При построении генетического алгоритма, работающего с матрицами сетевых операторов, используем универсальный принцип базисного решения. Принцип используется для того, чтобы не нарушать структуру решения. Генетические операции выполняются не над самой структурой, а над множеством допустимых вариаций структуры.

Для использования принципа базисного решения сначала необходимо определить малые вариации графа сетевого оператора. Каждая вариация должна изменять граф сетевого оператора, сохраняя его свойства. Малые вариации графа приведены в таблице 6.1.

Таблица 6.1.

Малые вариации графа

Название малой вариации	Номер вариации
Замена унарной операции на дуге графа	0
Замена бинарной операции в узле графа	1
Добавление дуги	2
Удаление дуги	3
Удаление узла	4
Добавление узла	5

Все вариации формально могут быть описаны целочисленным вектором вариаций из четырех компонент $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}^T$, где w_1- номер вариации, w_2- номер строки матрицы сетевого оператора, w_3- номер столбца матрицы сетевого оператора, w_4- номер бинарной или унарной операции, в зависимости от номера вариации.

При реализации генетического алгоритма на основе принципа базисного решения в качестве хромосомы используем упорядоченный набор векторов вариаций.

Помимо основных параметров генетического алгоритма, количества хромосом, поколений и скрещиваемых пар количество векторов вариаций в хромосоме является еще одним настраиваемым параметром. Этот параметр называем длиной хромосомы. Большая длина хромосомы увеличивает вариабельность решений, но усложняет вычисления.

Чтобы обеспечить достаточную вариабельность решений при малой длине хромосомы, необходимо периодически изменять базисное решение. Базисное решение целесообразно менять после нескольких поколений, которые являются основными циклами генетического алгоритма.

Для смены базисного решения введем еще один настраиваемый параметр генетического алгоритма – эпоха.

Эпоха – это количество поколений в генетическом алгоритме, при котором базисное решение гарантированно остается неизменным.

При смене эпохи базисное решение следует заменять наилучшим найденным решением, при этом в популяции необходимо создать тождественную хромосому, состоящую из векторов вариаций, не меняющих решение.

Сетевой оператор может рассматриваться как нейронная сеть с нелинейной переменной структурой. Если рассматривать множество обучающих пар, в которых учитываются входные и выходные переменные, как некий образец, а отклонение от образца считать критерием оптимизации, то генетический алгоритм найдет такую структуру сети, которая обеспечит наилучшее совпадение с образцами.

В сетевом операторе основная адаптация производится не за счет изменения значений параметров, а за счет нелинейного преобразования входных сигналов в выходные. Можно утверждать, что сетевой оператор всегда даст решение не хуже любой нейронной сети, так как любая нейронная сеть может быть описана с помощью сетевого оператора.

Преимуществом сетевого оператора перед нейронной сетью является то, что он может обеспечить настройку меньшего числа весовых коэффициентов, которые в сетевом операторе соответствуют параметрам. Сложные нелинейные преобразования дают описания более сложных процессов, к которым должна приспособиться нейронная сеть.

Лекция 7. Реализация нейронных сетей в среде MatLab. Пакет Neural Network ToolBox. Использование функций среды MatLab для построения и обучения нейронных сетей. Функции инициализации слоя. Функции для анализа. Функции отклонения. Обучающие функции смещения и градиентного спуска. Функция Хэбба. Функции вычисления производных.

Пакет фирмы “The MathWorks” MATLAB [10, 11] предоставляет пользователям возможность работы с нейронными сетями. Входящий в стандартную поставку MATLAB toolbox предоставляет широкие возможности для работы с нейронными сетями всех типов. Использование “Neural Network Toolbox” совместно с другими средствами MATLAB открывает широкий простор для эффективного комплексного использования современных математических методов при решении самых разных задач прикладного и научного характера.

На русском языке практически отсутствует методическая литература по применению “Neural Network Toolbox”. Это создает определенные сложности, и пользователи предпочитают

использовать специализированные нейронно-сетевые программные средства, такие, как BrainMaker, NeuroOffice, NeuroPro, и др.

В качестве примера рассмотрим следующую задачу:

Задан массив, состоящий из нескольких значений функции $y = Ce^{-\frac{(x-A)^2}{S}}$ ($S>0$) на интервале (0,1). Необходимо создать нейронную сеть - такую, что при вводе этих значений на входы сети на выходах получались бы значения параметров C, A и S.

В первую очередь, необходимо определиться с размерностью входного массива.

Выберем количество значений функции, равным N=21, т.е. в качестве входных векторов массива используем значения функции y в точках x=0.05; …1.0. Для обучения сети необходимо сформировать массив входных векторов для различных наборов параметров C, A и S. Каждый набор этих параметров является вектором-эталоном для соответствующего входного вектора.

Для подготовки входного и эталонного массивов воспользуемся следующим алгоритмом. Выбираем случайным образом значения компонент вектора-эталона C, A, S и вычисляем компоненты соответствующего входного вектора. Повторяем эту процедуру M раз и получаем массив входных векторов в виде матрицы размерностью $N x M$ и массив векторов-эталонов в виде матрицы размерностью в нашем случае $3x M$. Полученные массивы мы можем использовать для обучения сети.

Прежде чем приступать к формированию обучающих массивов, необходимо определиться с их некоторыми свойствами.

Выберем диапазоны изменения параметров C, A, S равными (0.1, 1). Значения, близкие к 0, и сам 0 исключим в связи с тем, что функция не определена при S = 0. Второе ограничение связано с тем, что при использовании типичных передаточных функций желательно, чтобы компоненты входных и выходных векторов не выходили за пределы диапазона (-1,1). В дальнейшем мы познакомимся с методами нормировки, которые позволяют обойти это ограничение.

Количество входных и эталонных векторов выберем равным M =100. Этого достаточно для обучения, а процесс обучения не займет много времени.

Тестовые массивы и эталоны подготовим с помощью программы mas1:

```
% формирование входных массивов (входной массив P) и (эталоны T)
P=zeros(100,21);

T=zeros(3,100);

x=0:5.e-2:1;

for i=1:100

c=0.9*rand+0.1;

a=0.9*rand+0.1;
```

```
s=0.9*rand+0.1;

T(1,i)=c;

T(2,i)=a;

T(3,i)=s;

P(i,:)=c*exp(-((x-a).^2/s));

end;

P=P';
```

С помощью этой программы формируется матрица P из $M = 100$ входных векторов-столбцов, каждый из которых сформирован из 21 точки исходной функции со случайно выбранными значениями параметров C, A, S и матрица T эталонов из 100 эталонных векторов-столбцов, каждый из которых сформирован из 3 соответствующих эталонных значений. Матрицы P и T будут использованы при обучении сети. Следует отметить, что при каждом новом запуске этой программы будут формироваться массивы с новыми значениями компонентов векторов, как входных, так и эталонных.

Создание сети

Вообще, выбор архитектуры сети для решения конкретной задачи основывается на опыте разработчика. Поэтому предложенная ниже архитектура сети является одним вариантом из множества возможных конфигураций.

Для решения поставленной задачи сформируем трехслойную сеть обратного распространения, включающую 21 нейрон во входном слое (по числу компонент входного вектора) с передаточной функцией `logsig`, 15 нейронов во втором слое с передаточной функцией `logsig` и 3 нейрона в выходном слое (по числу компонентов выходного вектора) с передаточной функцией `purelin`. При этом в качестве обучающего алгоритма выбран алгоритм Levenberg-Marquardt (`trainlm`). Этот алгоритм обеспечивает быстрое обучение, но требует много ресурсов. В случае, если для реализации этого алгоритма не хватит оперативной памяти, можно использовать другие алгоритмы (`trainbfg, trainrp, trainscg, traincgb, traincgf, traincgp, trainoss, traingdx`). По умолчанию используется `trainlm`. Указанная сеть формируется с помощью процедуры:

```
net=newff(minmax(P),[21,15,3],{'logsig' 'logsig'
'purelin'},'trainlm');
```

Первый аргумент - матрица * M x*2 минимальных и максимальных значений компонент входных векторов вычисляется с помощью процедуры `minmax`.

Результатом выполнения процедуры `newff` является объект – нейронная сеть `net` заданной конфигурации. Сеть можно сохранить на диске в виде `mat.` файла с помощью команды `save` и загрузить снова с помощью команды `load`. Более подробную информацию о процедуре можно получить, воспользовавшись командой `help`.

Обучение сети

Следующий шаг – обучение созданной сети. Перед обучением необходимо задать параметры обучения. Задаем функцию оценки функционирования `sse`.

```
net.performFcn='sse';
```

В этом случае в качестве оценки вычисляется сумма квадратичных отклонений выходов сети от эталонов. Задаем критерий окончания обучения – значение отклонения, при котором обучение будет считаться законченным:

```
net.trainParam.goal=0.01;
```

Задаем максимальное количество циклов обучения. После того, как будет выполнено это количество циклов, обучение будет завершено:

```
net.trainParam.epochs=1000;
```

Теперь можно начинать обучение:

```
[net,tr]=train(net,P,T);
```

Процесс обучения иллюстрируется графиком зависимости оценки функционирования от номера цикла обучения (см. рис. 7.1).

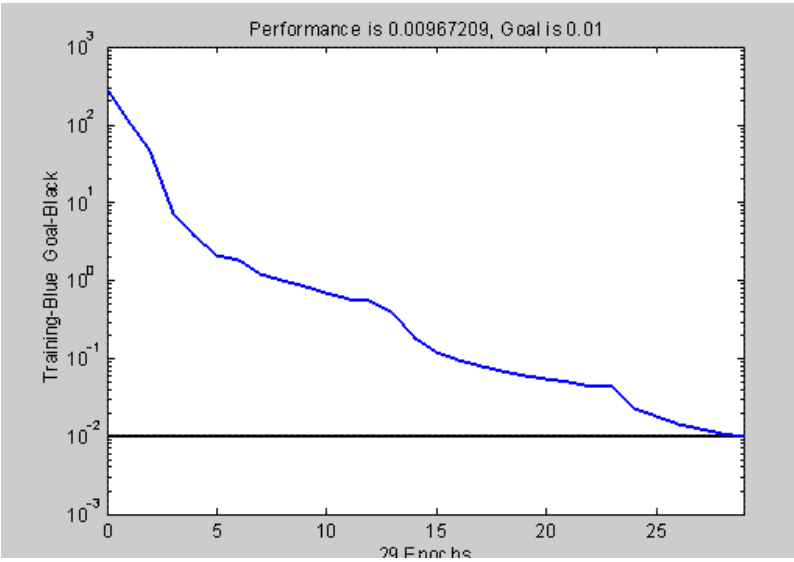


Рис. 7.1

Таким образом, обучение сети окончено. Теперь эту сеть можно сохранить в файле nn1.mat:

```
save nn1 net;
```

Тестирование сети

Перед тем, как воспользоваться нейронной сетью, необходимо исследовать степень достоверности результатов вычислений сети на тестовом массиве входных векторов. В качестве тестового массива необходимо использовать массив, компоненты которого отличаются от компонентов массива, использованного для обучения. В нашем случае для получения тестового массива достаточно воспользоваться еще раз программой mas1.

Для оценки достоверности результатов работы сети можно воспользоваться результатами регрессионного анализа, полученными при сравнении эталонных значений со значениями, полученными на выходе сети когда на вход поданы входные векторы тестового массива. В среде MATLAB для этого можно воспользоваться функцией postreg. Следующий набор команд иллюстрирует описанную процедуру:

```
>> mas1
```
-создание тестового массива P;

```
>> y=sim(net,P);
```
-обработка тестового массива;

```
>> [m,b,r]=postreg(y(1,:),T(1,:));
```
- регрессионный анализ результатов обработки.

Сравнение компонентов C эталонных векторов с соответствующими компонентами выходных векторов (рис. 7.2-7.4) показывает, что все точки легли на прямую, поэтому сеть на тестовом массиве работает правильно.

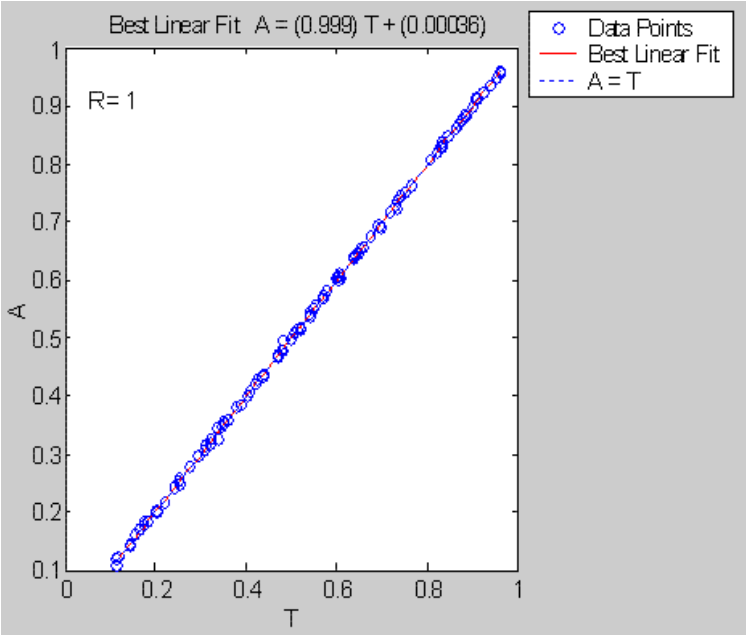


Рис. 7.2

```
>> [m,b,r]=postreg(y(2,:),T(2,:));
```

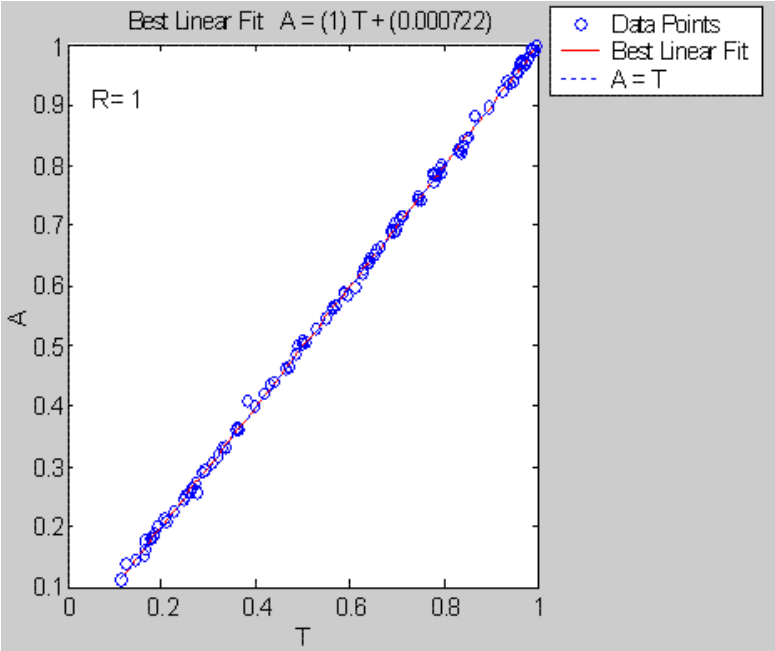


Рис. 7.3

```
>> [m,b,r]=postreg(y(3,:),T(3,:));
```

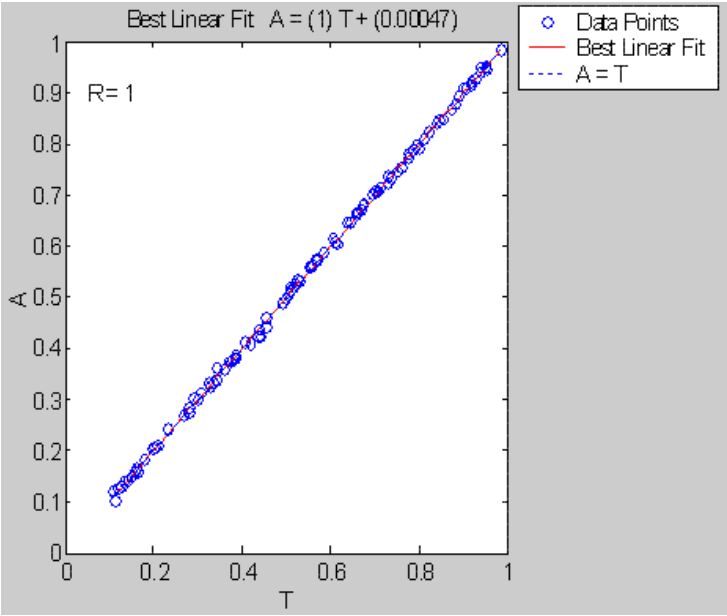


Рис. 7.4

Как видно из рисунков, наша сеть отлично решает поставленную задачу для всех трех выходных параметров. Сохраним обученную сеть `net` на диске в файл `nn1.mat`

```
save nn1 net
```

Моделирование сети. Использование сети для решения поставленной задачи.

Для того, чтобы применить обученную сеть для обработки данных, необходимо воспользоваться функцией `sim`:

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

где p – набор входных векторов, Y – результат анализа в виде набора выходных векторов. Например, пусть C =0.2, A =0.8, S =0.7, тогда

```
p=0.2*exp(-((x-0.8).^2/0.7));
```

Подставив этот входной вектор в качестве аргумента функции `sim`:

```
Y=sim(net,p)
```

получим: `Y =0.2048 (C) 0.8150 (A) 0.7048 (S)`.

Лекция 8. Использование функций среды MatLab при построении и обучении нейронных сетей.

Перечислим основные функции пакета Neural Net toolbox MatLab [15, 23]

Эти свойства определяют количество подобъектов сети (таких, как слои, выходы, эталоны, смещения и веса), и каким образом они объединяются в сеть:

- `numInputs` - определяет количество входов сети,

- `net.numInputs` - может принимать значения 0 или положительное число.

Количества входов сети и размерность входа разные вещи. количество входов определяет, сколько групп векторов принимает сеть в качестве входов. Размерность каждого входа (т.е. количество элементов в каждом векторе) определяется размерностью входа (`net.inputs{i}.size`).

Большинство сетей имеет один вход, размерность которого определяется поставленной задачей.

Сопутствующие эффекты

Любое изменение этого свойства приводит к изменению размера матрицы, определяющей связи входов со слоями, (`net.inputConnect`) и размер массива ячеек входных подобъектов (`net.inputs`):

- `numLayers` - определяет количество слоев сети;

- `net.numLayers` - может принимать значения от 0 до любого положительного целого.

Сопутствующие эффекты

Любое изменение этого свойства приводит к изменению размера каждой из булевых матриц, которые определяют связи между слоями:

- `net.biasConnect`,

- `net.inputConnect`,

- `net.layerConnect`,

- `net.outputConnect`,

- `net.targetConnect`

и изменяют размерности каждого из массивов ячеек структур подобъектов, чьи размеры зависят от количества слоев:

- `net.biases`,

- `net.inputWeights`,

- `net.layerWeights`,

- `net.outputs`,

- `net.targets`.

Также изменяет размеры каждого из настраиваемых параметров:

- `net.IW`,

- `net.LW`,

- `net.b`,

- `biasConnect` - определяет, какие слои имеют смещения.

- `net.biasConnect` - может принимать значения Булевой матрицы N_1x 1, где N_1– количество слоев (`net.numLayers`). Наличие (или отсутствие) смещения у *i* -го слоя определяется 1 (или 0) в: `net.biasConnect(i)`.

Сопутствующие эффекты

Любое изменение этого свойства влияет на наличие или отсутствие структур в массиве ячеек смещений (`net.biases`) и, в наличии или отсутствии в массиве элементов векторов смещений (`net.b`):

- `inputConnect` - определяет, какие слои обладают связями со входами,

- `net.inputConnect` – определяет Булеву матрицу $N_1 \times N_i$,

где N_1– количество слоев сети (`net.numLayers`), а N_i– количество входов сети (`net.numInputs`). Наличие (или отсутствие) связи *i* -го слоя с *j* -м слоем определяется 1 (или 0) в `net.inputConnect(i,j)`.

Сопутствующие эффекты

Любое изменение этого свойства влияет на наличие или отсутствие структур в массивах ячеек подобъектов входных весов (`net.inputWeights`) и на наличие или отсутствие матриц в массиве ячеек входных весов (`net.IW`):

- `layerConnect` - определяет какие слои связаны с другими слоями,

- `net.layerConnect` - определяет Булеву матрицу $N_1 \times N_1$,

где N_1– количество слоев в сети (`net.numLayers`). Наличие (или отсутствие связи *i* -го слоя с *j* -м слоем определяется 1 (или 0) в `net.layerConnect(i,j)`.

Сопутствующие эффекты

Любое изменение этого свойства влияет на наличие или отсутствие структур в массивах ячеек подобъектов весов слоев (`net.layerWeights`) и в наличии или отсутствии матриц в массиве ячеек весов слоев (`net.LW`):

- `outputConnect` - определяет какие слои генерируют выходы сети,

- `net.outputConnect` – задается в виде матрицы Булевых значений $\mathbf{1} \times N$,

где N – количество слоев в сети (`net.numLayers`). Наличие (или отсутствие) выхода от *i* -го слоя определяется 1 (или 0) в net.outputConnect(i).

Сопутствующие эффекты

Любые изменения этого свойства будут влиять на количество выходов сети (`net.numOutputs`) и наличие или отсутствие структур в массиве ячеек выходных подобъектов (`net.outputs`).

- `targetConnect` - определяет какие слои связаны с эталонами,

- `net.targetConnect` - задается в виде матрицы Булевых значений $\mathbf{1} \times N$,

где N – количество слоев в сети (`net.numLayers`). Наличие (или отсутствие) эталонного значения, связанного с *i* -м слоем определяется 1 (или 0) в `net.targetConnect(i)`.

Сопутствующие эффекты

Любые изменения этого свойства будут изменять количество эталонных значений сети (`net.numTargets`) и наличие или отсутствие структур в массиве ячеек эталонных подобъектов (`net.targets`):

- `numOutputs` (только для чтения) - указывает сколько выходов имеет сеть,

- `net.numOutputs` - возвращает значение, соответствующее количеству единиц в матрице выходных соединений:

```
numOutputs=sum(net.outputConnect);
```

- `numTargets` (только для чтения) - указывает сколько эталонных значений имеет сеть,

- `net.numTargets` - возвращает значение, соответствующее количеству единиц в матрице соединений с эталонами:

```
numTargets=sum(net.targetConnect);
```

- `numInputDelays` (только для чтения) - указывает какое количество временных шагов последних входов должно быть реализовано для того, чтобы смоделировать сеть,

- `net.numInputDelays` - возвращает значение величины максимальной задержки, связанной с входными весами сети:

```
numInputDelays = 0;

for i=1:net.numLayers

for j=1:net.numInputs

if net.inputConnect(i,j)

numInputDelays = max( ...

[numInputDelays net.inputWeights{i,j}.delays]);

end

end

end
```

- `numLayerDelays` (только для чтения) - указывает какое количество временных шагов последних выходов должно быть реализовано для того, чтобы смоделировать сеть,

- `net.numLayerDelays` - возвращает значение величины максимальной задержки, связанной с весами слоев сети:

```
numLayerDelays = 0;

for i=1:net.numLayers

for j=1:net.numLayers

if net.layerConnect(i,j)

numLayerDelays = max( ...
```

```
[numLayerDelays net.layerWeights{i,j}.delays]);

end

end

end
```

Эти свойства определяют массивы ячеек структур, которые определяют каждый из входов сети, слои, выходы, эталоны, смещения и веса:

- `inputs` - содержит структуры свойств для каждого из входов сети.

- `net.inputs` - матрица $N_i \times 1$, ячеек входных структур, где N_i- число входов сети (`net.numInputs`).

Структура для свойства *i* -го входа, определена в: `net.inputs{i}`:

- `layers` - содержит структуры свойств для каждого из слоев сети,

- `net.layers` - массив $N_i \times 1$ячеек входных структур, где N_i- число слоев сети (`net.numLayers`).

Структура, определяющая свойства *i* -го слоя определена в: `net.layers{i}`:

- `outputs` - содержит структуры свойств для каждого из выходов сети,

- `net.outputs` - массив $1 \times N_i$ячеек входных структур, где N_i- число слоев сети (`net.numLayers`).

Структура, определяющая свойства *i* -го выхода, определена в: `net.outputs{i}` если соответствующее выходное соединение `net.outputConnect(i)` – 1 (или 0):

- `targets` – содержит структуры свойств для каждого из эталонов сети,

- `net.targets` – массив $1 \times N_i$ячеек входных структур, где N_i- число слоев сети (`net.numLayers`).

Структура, определяющая свойства эталона связанного с *i* -ым слоем (или нулевая матрица) определена в `net.targets{i}` если соответствующее выходное соединение `net.targetConnect(i)` – 1 (или 0):

- `biases` - содержит структуры свойств для каждого из смещений сети,

- `net.biases` – массив $N_i \times 1$ячеек, где N_i– число слоев сети (`net.numLayers`).

Структура, определяющая свойства смещений *i* -го слоя (или нулевая матрица) расположена в `net.biases{i}` если соответствующее соединение для смещения `net.biasConnect(i)` – 1 (или 0):

- `inputWeights` – содержит структуры свойств для каждого из входных весов,

- `net.inputWeights` – массив $N_1 \times N_i$ ячеек, где N_1 – число слоев сети (`net.numLayers`), а N_i – число входов сети (`net.numInputs`).

Структура, определяющая свойства весов к *i* -му слоя от *j* -го слоя (или нулевая матрица) определена в `net.inputWeights{i,j}`, если соответствующее соединение входа `net.inputConnect(i,j)` – 1 (или 0):

- `layerWeights` – содержит свойства для каждого из весов слоя сети,

  – `net.layerWeights` – массив $N_1 \times N_1$ ячеек, где x*N1* – число слоев сети (`net.numLayers`).

Лекция 9. Функции математического программирования, используемые для обучения нейронной сети. Методы связанных градиентов Пауэлла-Била (Powell-Beale), Флетчера-Пауэлла (Fletcher-Powell), Полака-Рибира (Polak-Ribiere). Методы градиентного спуска. Метод Левенберга-Маркара (Levenberg-Marquardt). Одноступенчатый метод секущих. Метод случайных приращений. Алгоритм упругого обратного распространения. Метод последовательных приращений.

Эти свойства определяют алгоритмы, которые используются при адаптации, инициализации, тренировке и оценке ее функционирования:

- `adaptFcn` – определяет функцию, которая будет использована для адаптации сети,

- `net.adaptFcn` – задается в виде названия функции адаптации сети из настоящего пакета,

- `trains` – функция адаптации весов и смещений.

Функция адаптации сети используется, чтобы выполнить адаптацию всякий раз, когда вызывается `adapt`:

```
[net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)
```

Сопутствующие эффекты

Всякий раз, когда это свойство изменено, параметры адаптации сети (`net.adaptParam`) устанавливаются таким образом, чтобы содержать параметры и значения по умолчанию новой функции:

- `initFcn` – определяет функцию, используемую для инициализации матриц весов и векторов смещений сети;

- `net.initFcn` – задается в виде названия функции инициализации сети из настоящего пакета;

- `initlay` – функция инициализации весов и смещений.

Функция инициализации сети используется, чтобы выполнить инициализацию всякий раз, когда вызывается `init`:

```
net = init(net)
```

Сопутствующие эффекты

Всякий раз, когда это свойство изменено, параметры сети (`net.initParam`) устанавливаются таким образом, чтобы содержать параметры и значения по умолчанию новой функции:

- `performFcn` – определяет функцию, используемую для оценки функционирования сети,

- `net.performFcn` – задается в виде названия функции функционирования сети из настоящего пакета: Функции функционирования,

- `mae` – средняя абсолютная ошибка,

- `mse` – средняя квадратичная ошибка,

- `msereg` – средняя квадратичная ошибка `w/reg`,

- `sse` – суммарная квадратичная ошибка.

Функция функционирования сети используется, чтобы вычислить оценку функционирования сети всякий раз, когда вызывается `train`:

```
[net,tr] = train(NET,P,T,Pi,Ai)
```

Сопутствующие эффекты

Всякий раз, когда это свойство изменено, параметры сети (`net.performParam`) устанавливаются таким образом, чтобы содержать параметры и значения по умолчанию новой функции:

- `trainFcn` – определяет функцию, используемую для тренировки сети,

- `net.trainFcn` – задается в виде названия функции тренировки сети из настоящего пакета.

Функции тренировки:

- `trainb`– пакетная тренировка с использованием правил обучения для весов и смещений,

- `trainbfg` – тренировка сети с использованием квази –Ньютоновского метода BFGS,

- `trainbr` – регуляризация `Bayesian`,

- `trainc` – использование приращений циклического порядка,

- `traincgb` – метод связанных градиентов Пауэлла-Била (Powell-Beale),

- `traincgf` – метод связанных градиентов Флетчера-Пауэлла (Fletcher-Powell),

- `traincgp` – метод связанных градиентов Полака-Рибира (Polak-Ribiere),

- `traingd` – метод градиентного спуска,

- `traingda` – метод градиентного спуска с адаптивным обучением,

- `traingdm` – метод градиентного спуска с учетом моментов,

- `traingdx` – метод градиентного спуска с учетом моментов и с адаптивным обучением,

- `trainlm` – метод Левенберга-Маркара (Levenberg-Marquardt),

- `trainoss` – одноступенчатый метод секущих,

- `trainr` – метод случайных приращений,

- `trainrp` – алгоритм упругого обратного распространения,

- `trains` – метод последовательных приращений.

Функция используется, чтобы тренировать сеть всякий раз, когда вызывается `train`:

```
[net,tr] = train(NET,P,T,Pi,Ai)
```

Сопутствующие эффекты

Всякий раз, когда это свойство изменено, параметры сети (`net.trainParam`) устанавливаются таким образом, чтобы содержать параметры и значения по умолчанию новой функции:

- `adaptParam` – определяет параметры и значения текущей функции адаптации,

- `net.adaptParam` – поля этого свойства зависят от текущей функции адаптации (`net.adaptFcn`).

Вызовите вышеупомянутую ссылку, чтобы увидеть поля текущей функции адаптации.

Вызовите help на текущую функцию адаптации, чтобы получить описание каждого из полей: `help(net.adaptFcn)`:

- `initParam` – определяет параметры и значения текущей функции инициализации,

- `net.initParam` – поля этого свойства зависят от текущей функции инициализации (`net.initFcn`).

Вызовите вышеупомянутую ссылку, чтобы увидеть поля текущей функции адаптации.

Вызовите help на текущую функцию инициализации, чтобы получить описание каждого из полей: `help(net.initFcn)`:

- `performParam` – определяет параметры и значения текущей функции функционирования,

- `net.performParam` – поля этого свойства зависят от текущей функции функционирования (`net.performFcn`).

Вызовите вышеупомянутую ссылку, чтобы увидеть поля текущей функции функционирования.

Вызовите help на текущую функцию функционирования, чтобы получить описание каждого из полей: `help(net.performFcn)`:

- `trainParam` – определяет параметры и значения текущей функции тренировки,

- `net.trainParam` – поля этого свойства зависят от текущей функции тренировки (`net.trainFcn`).

Вызовите вышеупомянутую ссылку, чтобы увидеть поля текущей функции функционирования.

Вызовите help на текущую функцию тренировки, чтобы получить описание каждого из полей: `help(net.trainFcn)`.

Эти свойства определяют модифицируемые параметры сети: матрицы весов и векторы смещений:

- `IW` – определяет матрицы входных весов,

- `net.IW` – массив $N_1 \times N_i$ ячеек, где N_i – число слоев в сети (`net.numLayers`), N_i – число входов сети (`net.numInputs`).

Матрица весов для связи *i* -го слоя с *j* -м входом (или нулевая матрица) описывается в: `net.IW{i,j}`, если соответствующее входное соединение: `net.inputConnect(i,j)` – 1 (или 0).

Матрица весов имеет количество строк, равное размеру слоя, к которому она относится (`net.layers{i}.size`). Содержит количество столбцов равное произведению размерности входа на количество задержек, ассоциированных с весом:

`net.inputs{j}.size * length(net.inputWeights{i,j}.delays)`

Эти размеры могут также быть получены из свойств входных весов:

`net.inputWeights{i,j}.size`:

- `LW` – определяет матрицы весов связей мехду слоями,

- `net.LW` – массив $N_1 \times N_1$ячеек, где N_1- число слоев в сети (`net.numLayers`).

Матрица весов для связи *i* -го слоя. с *j* -м слоем (или нулевая матрица) представлена в: `net.LW{i,j}`, если соответствующее соединение слоя `net.layerConnect(i,j)` – 1 (или 0).

Матрица весов имеет количество строк, равное размеру слоя, к которому она относится (`net.layers{i}.size`). Содержит количество столбцов равное произведению размерности слоя на количество задержек, ассоциированных с весом:

```
net.layers{j}.size * length(net.layerWeights{i,j}.delays)
```

Эти размеры могут также быть получены из свойств веса слоя:

```
net.layerWeights{i,j}.size
```

Параметр `b` - определяет векторы смещений для каждого слоя со смещением:

`net.b` - массив $N_1 \times 1$ячеек, где N_1- число слоев в сети (`net.numLayers`).

Вектор смещения для *i* -го слоя (или нулевая матрица) определен в `net.b{i}` если соответствующее соединение смещения `net.biasConnect(i)` - 1 (или 0).

Число элементов в векторе смещения всегда равно размеру слоя, с которым он связан (`net.layers{i}.size`). Этот размер может также быть получен из свойств смещения: `net.biases{i}.size`.

Единственное другое свойство - свойство для данных пользователя.

- `userdata` - это свойство обеспечивает место для пользователей.

Чтобы добавить пользовательскую информацию о сети, необходимо ввести

```
net.userdata
```

Только одно поле предопределено. Оно содержит секретное сообщение всем пользователям Neural Network Toolbox:

```
net.userdata.note
```

Лекция 10. Построение нейронной сети пользователя в среде MatLab. Создание конкурентного слоя. Создание каскадной направленной сети. Создание сети обратного распространения Элмана (Elman). Создание рекуррентной сети Хопфилда. Создание самоорганизующейся карты. Конструирование вероятностной нейронной сети.

Рассмотрим построение LVQ сети и обучение для классификации признаков согласно заданному правилу.

Пусть **P** будет содержать 10x2 элементов. Вектор **C** будет классифицировать эти вектора внутри **P**:

```
P = [-3 -2 -2 0 0 0 0 +2 +2 +3;

       0 +1 -1 +2 +1 -1 -2 +1 -1 0];

C = [1 1 1 2 2 2 2 1 1 1];

T = ind2vec(C);
```

Изобразим данные точки на графике. Красный цвет (сплошной) соответствует классу 1, бирюзовый (пунктир) – классу 2. LVQ-сеть представляет собой кластеризацию вектора с помощью скрытых нейронов и группирует точки согласно кластерам:

```
colormap(hsv);

plotvec(P,C)

title('Input Vectors');

xlabel('P(1)');

ylabel('P(2)');
```

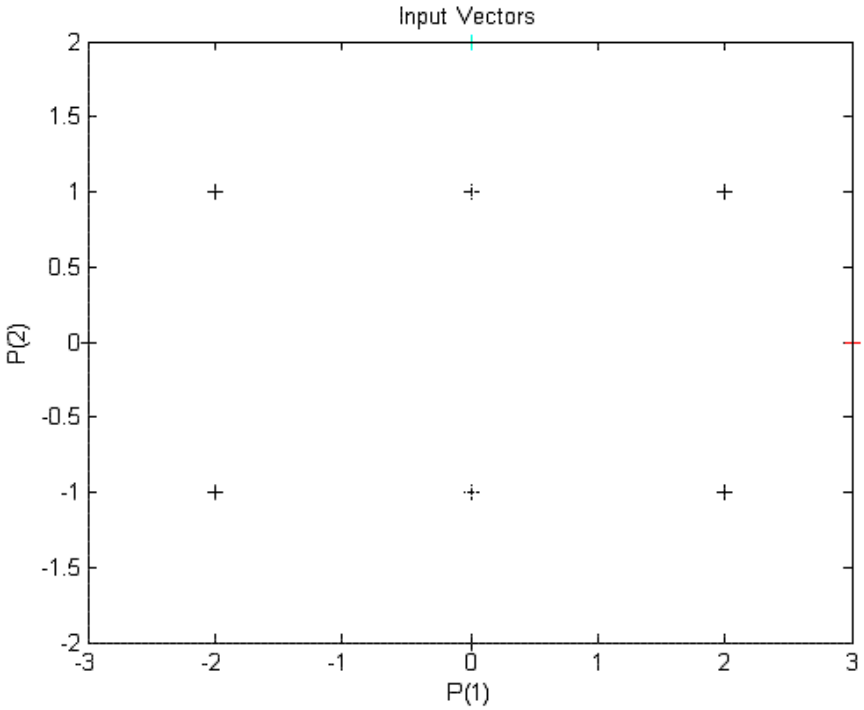


Рис. 10.1. Кластеризация вектора LVQ сетью

NEWLVQ создает в LVQ слой. Возьмем четыре аргумента: *R* x2 матрица минимальных и максимальных значений для R входных элементов, число скрытых нейронов, процент элементов векторов типового класса и количество обучающих циклов

```
net = newlvq(minmax(P),4,[.6 .4],0.1);
```

Сравним нейронные веса на графике следующим образом

```
hold on

W1 = net.IW{1};

plot(W1(1,1),W1(1,2),'ow')

title('Input/Weight Vectors');

xlabel('P(1), W(1)');

ylabel('P(2), W(3)');
```

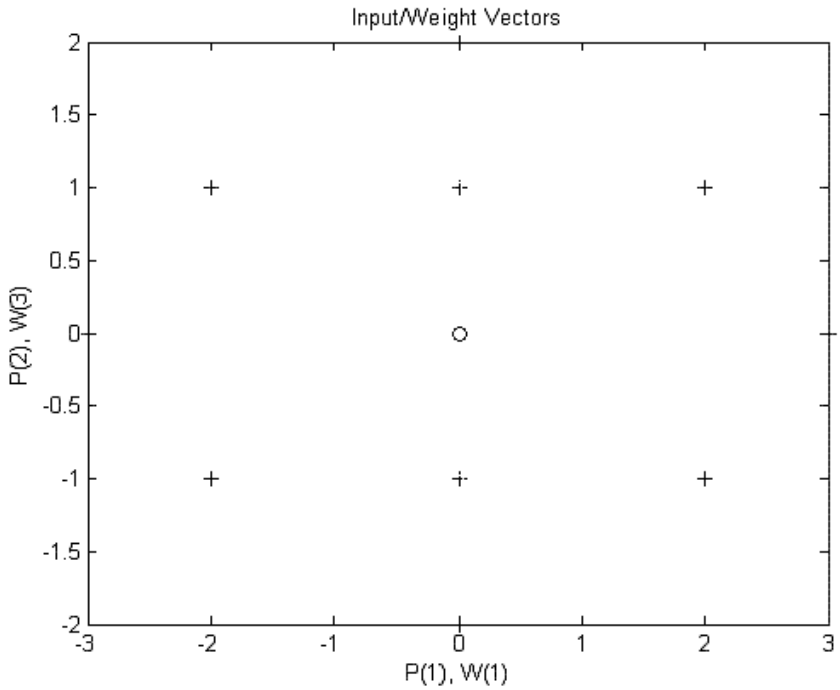


Рис. 10.2. Сравнение нейронных весов

Для того, чтобы обучить нейронную сеть, во-первых, перепишем число заданных по умолчанию эпох и затем запустим обучение. Когда оно закончится, перерисуем график, выведем график с входными векторами, отмеченными знаком «+», а конкурирующие нейроны – «0». Красным цветом (сплошным)– класс 1, бирюзовым (пунктиром)– класс 2

```
net.trainParam.epochs=150;

net.trainParam.show=Inf;

net=train(net,P,T);

cla;

plotvec(P,C);

hold on;

plotvec(net.IW{1}',vec2ind(net.LW{2}),'o');
```

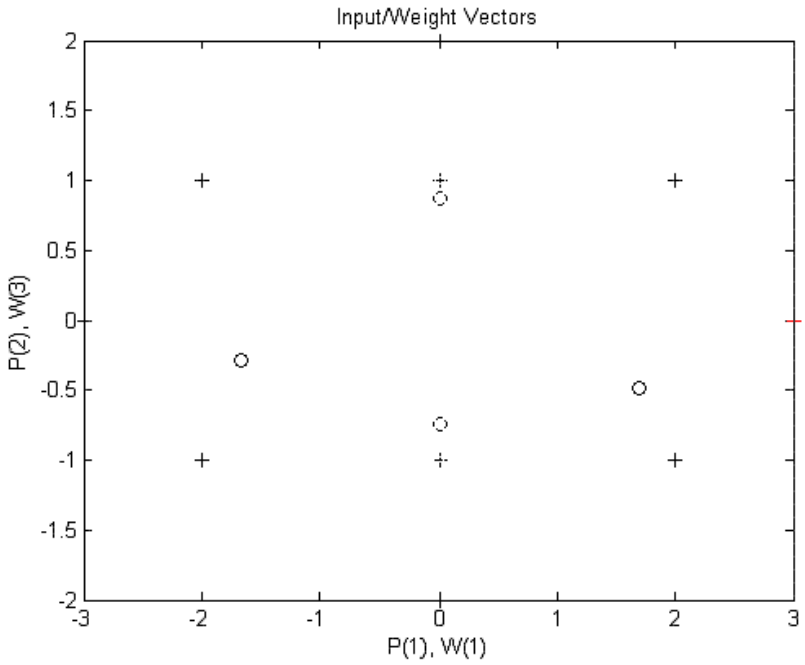


Рис. 10.3. Сеть после обучения

LVQ-сеть используется как классификатор, и каждый нейрон согласован со своей категорией. Представим вектор [0,2; 1]. Красный – класс 1, Бирюзовый – класс 2.

```
p = [0.2; 1];

a = vec2ind(sim(net,p))

a =2
```

Сети Хопфилда

Рассмотрим сеть Хопфилда, состоящую из двух нейронов и имеющую два устойчивых положения равновесия.

Для того, чтобы получить сеть Хопфилда с двумя точками равновесия используем два целевых вектора

```
T = [+1 -1; ...

     -1 +1];
```

На графике равновесные точки показаны в углах. Все возможные состояния двух нейронов сети Хопфилда расположены внутри границ графика

```
plot(T(1,:),T(2,:),'r*')
```

```
axis([-1.1 1.1 -1.1 1.1])

title('Hopfield Network State Space')

xlabel('a(1)');

ylabel('a(2)');
```

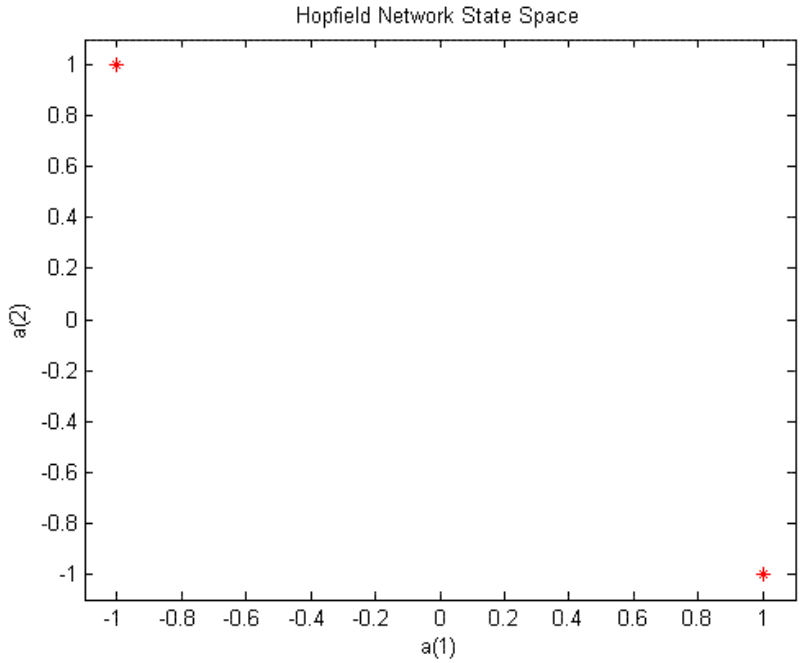


Рис. 10.4. Возможные состояния сети Хопфилда

Функция NEWHOP создает сеть Хопфилда заданную устойчивыми точками в векторе T

```
net = newhop(T);
```

Во-первых, мы проконтролируем целевой вектор, действительно ли он устойчив. Проверим целевой вектор для сети Хопфилда

```
[Y,Pf,Af] = sim(net,2,[ ],T);

Y

Y =

1 -1

-1 1
```

Тепрь мы определим случайно начальные точки и промоделируем сеть Хопфилда для 20 шагов. Видим, что достигаем одну устойчивую точку

```
a = {rands(2,1)};
```

```
[y,Pf,Af] = sim(net,{1 20},{},a);
```

Мы можем активизировать рисование графика:

```
record = [cell2mat(a) cell2mat(y)];

start = cell2mat(a);

hold on

plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
```

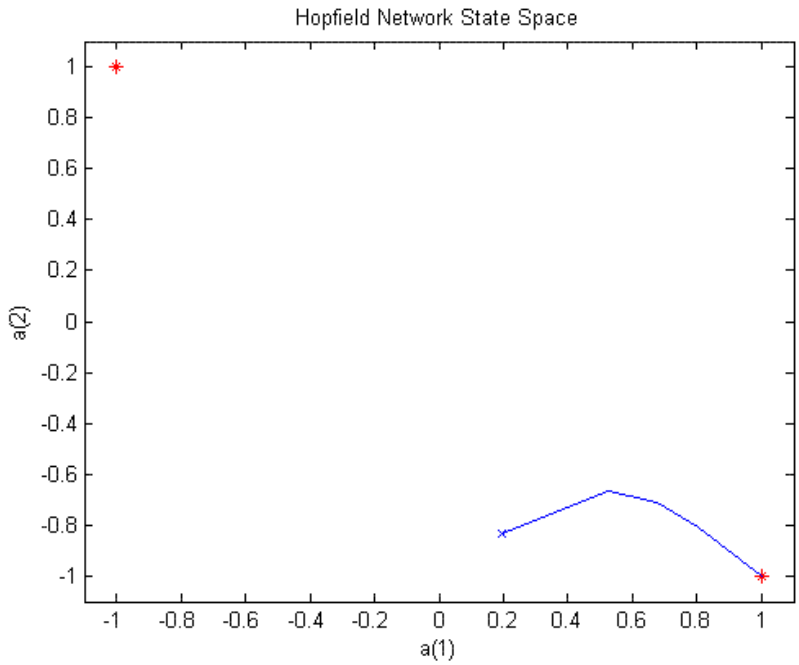


Рис. 10.5. Моделирование сети Хопфилда для 20 шагов

Повторим моделирования для 25 начальных условий:

```
color = 'rgbmy';

for i=1:25

a = {rands(2,1)};

[y,Pf,Af] = sim(net,{1 20},{},a);

record=[cell2mat(a) cell2mat(y)];

start=cell2mat(a);

plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:),color(rem(i,5)
+1))

end
```

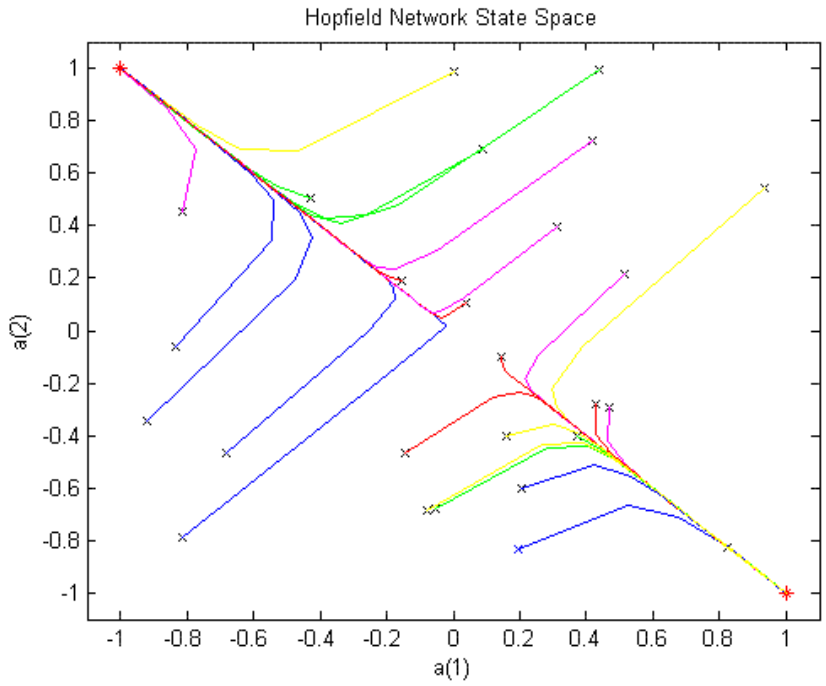


Рис. 10.6. Моделирование сети Хопфилда для 25 шагов

Лекция 11. Функции моделирования нейронной сети с помощью блока Simulink в MatLab. Генерация блока Simulink для моделирования нейронной сети. Топологические функции в виде сеточного, гексагонального и случайного слоев.

Пакет фирмы “The MathWorks” MATLAB [10, 11] предоставляет пользователям возможность работы с нейронными сетями. Входящий в стандартную поставку MATLAB toolbox предоставляет широкие возможности для работы с нейронными сетями всех типов. Использование “Neural Network Toolbox” совместно с другими средствами MATLAB открывает широкий простор для эффективного комплексного использования современных математических методов при решении самых разных задач прикладного и научного характера.

Рассмотрим построение нейронной сети с помощью пакета MatLab Simulink [9]. Пакет Simulink содержит все необходимые блоки для создания и моделирования нейронной сети.

Вызов пакета Simulink осуществляется командой `neural`. В результате получим окно (рис. 11.1), описывающее используемые блоки.

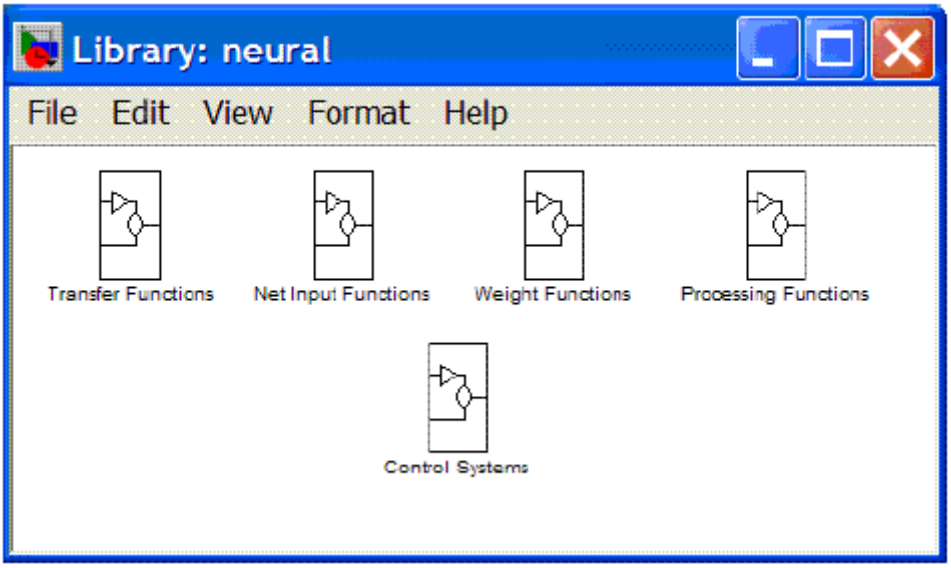


Рис. 11.1. Окно пакета Simulink

Двойное нажатие на клавишу «мыши» приведет к получению блока передаточных функций (рис. 11.2)

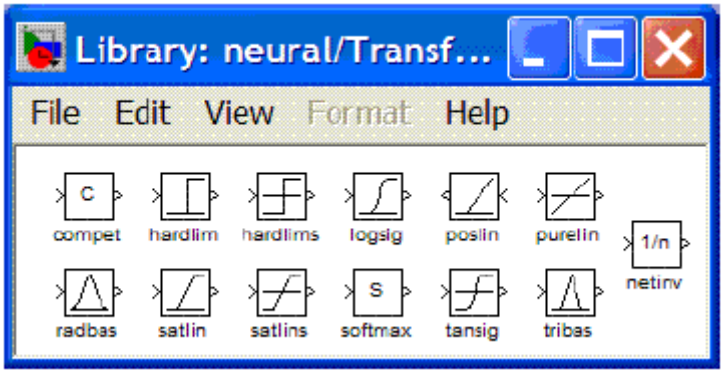


Рис. 11.2. Блок передаточных функций

Каждый из этих блоков получает входной вектор и преобразует его в выходной. Размерности векторов совпадают.

Нажатие Double-click на блок входных функций покажет окно (рис. 11.3) для выбора блока-смесителя двух сигналов, блока суммирования или блока перемножения.

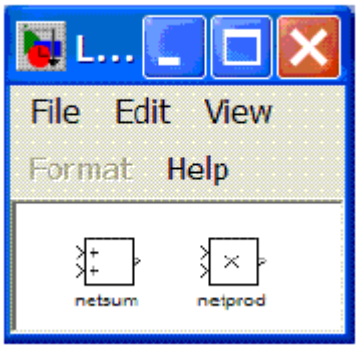


Рис. 11.3. Окно для выбора блока-смесителя

Нажатие Double-click на блок весовых функций в окне Neural приведет к окну выбора блока весовых функций (рис. 11.4).

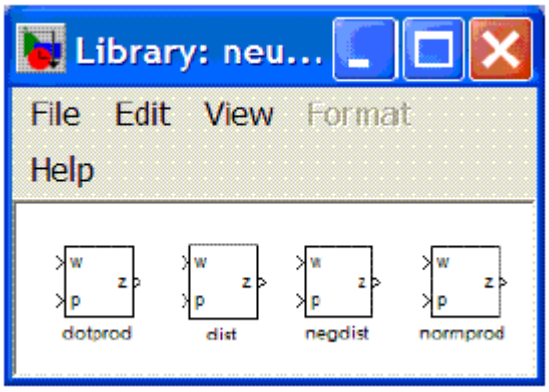


Рис. 11.4. Выбор блока весовых функций

Каждый из этих блоков задает веса вектора входного или выходного слоев. Важно отметить, что в пакете Simulink весовые векторы являются векторами столбцами, они не могут быть матрицами или векторами строками.

Вы должны создать S блоков весовых функций, для того, чтобы работать с весами слоя с S нейронами.

Нажатие Double-click на блок выполняемых процессов приведет к появлению окна с различными процессами (рис. 11.5).

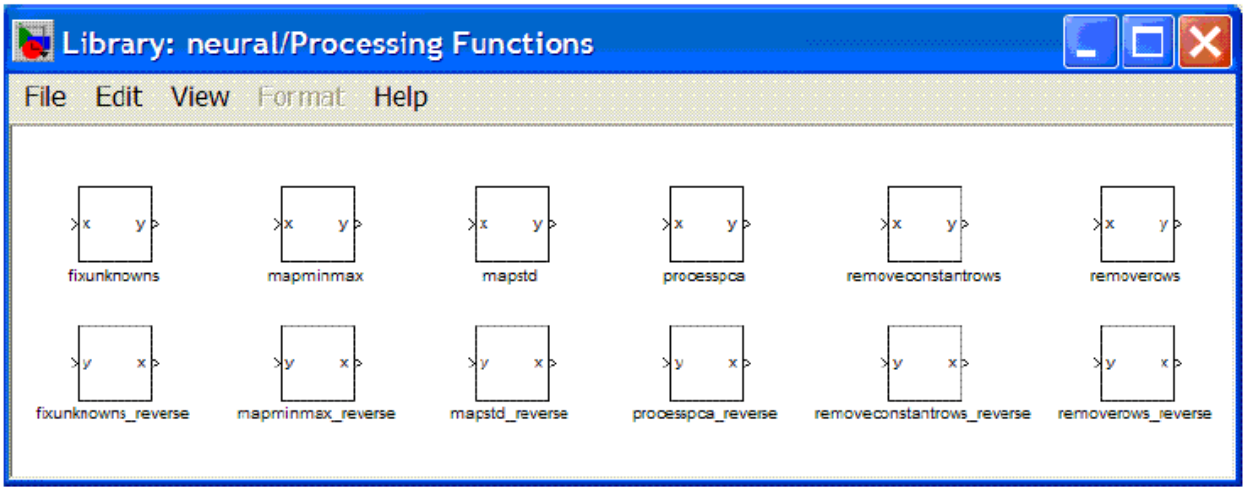


Рис. 11.5. Блок выбора процессов

Каждый из этих блоков используется для описания предпроцесса входа и постпроцесса выхода.

Для описания создаваемой сети используется функция `Gensim(net,st)`.

Рассмотрим пример

Пусть, например, задано множество входов **P** и множество целей **T**.

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

```
T=[1 3 5 7 9];
```

Зададим один линейный слой для данной задачи

```
net=newlind(p,t)
```

Мы можем протестировать сеть для наших первоначальных входов

```
y=sim(net,p)
```

В результате сеть выдаст решение задачи.

```
y= 1.000 3.000 5.000 7.000 9.000
```

Для моделирования сети (рис. 11.6) надо вызвать simulink следующим образом:

```
gensim(net,-1)
```

Второй аргумент задает таким образом продолжение работы сети.

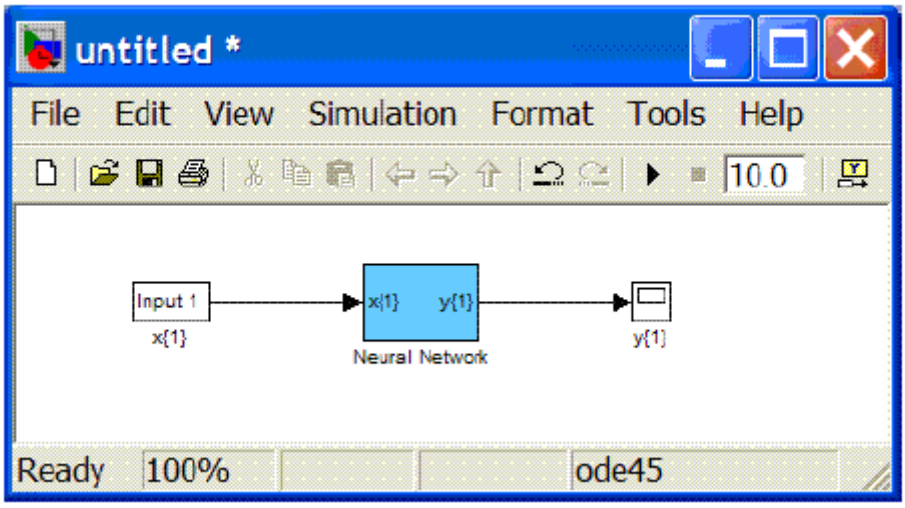


Рис. 11.6. Пример нейронной сети на пакете Simulink

Для того, чтобы протестировать работу сети дважды следует нажать на блок Input1, получим окно, представленное на рис. 11.7.

Входной блок в действительности является блоком констант. Для изменения значения констант, необходимо нажать клавишу OK

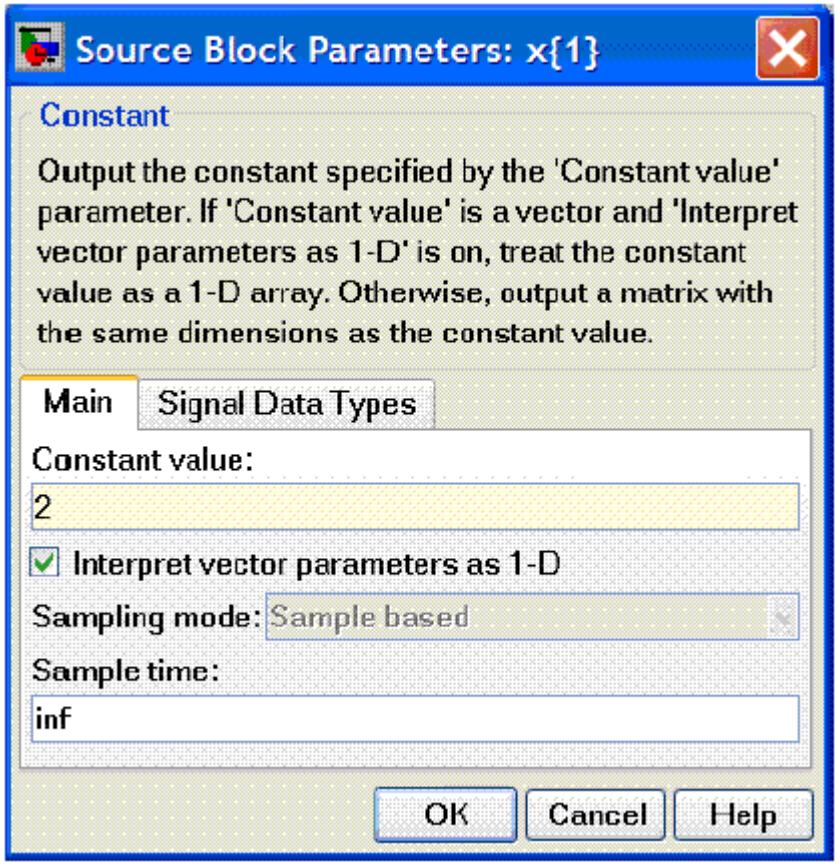


Рис. 11.7. Блок констант

После запуска моделирования из меню Simulation получим результат моделирования созданной нейронной сети (рис. 11.8)

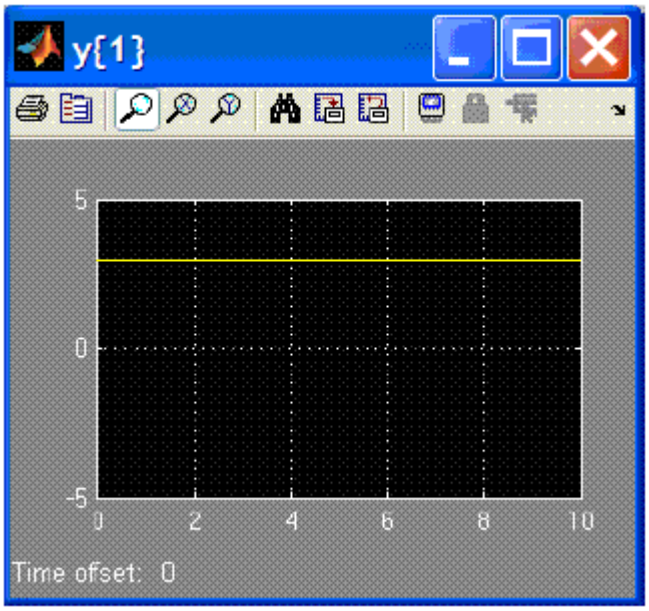


Рис. 11.8. Результат моделирования нейронной сети

Лекция 12. Экспертные системы. Назначение экспертных систем. Структуры экспертных систем. Компоненты экспертных систем. Применение экспертных систем в задачах управления. Интеллектуальные системы управления. Динамические экспертные системы реального времени.

Экспертные системы [7] – это прикладные системы искусственного интеллекта, в которых база знаний представляет собой формализованные эмпирические знания высококвалифицированных специалистов (экспертов) в какой-либо узкой предметной области. Экспертные системы предназначены для замены при решении задач экспертов в силу их недостаточного количества, недостаточной оперативности в решении задачи или в опасных (вредных) для них условиях.

Обычно экспертные системы рассматриваются с точки зрения их применения в двух аспектах: для решения каких задач они могут быть использованы и, в какой области деятельности. Эти два аспекта накладывают свой отпечаток на архитектуру разрабатываемой экспертной системы.

Можно выделить следующие основные классы задач, решаемых экспертными системами:

- диагностика,

- прогнозирование,

- идентификация,

- управление,

- проектирование,

- мониторинг.

Наиболее широко встречающиеся области деятельности, где используются экспертные системы:

- медицина,

- вычислительная техника,

- военное дело,

- микроэлектроника,

- радиоэлектроника,

- юриспруденция,

- экономика,

- экология,

- геология (поиск полезных ископаемых),

- математика.

Примеры широко известных и эффективно используемых (или использованных в свое время) экспертных систем:

– DENDRAL – ЭС для распознавания структуры сложных органических молекул по результатам их спектрального анализа (считается первой в мире экспертной системой);

– MOLGEN – ЭС для выработке гипотез о структуре ДНК на основе экспериментов с ферментами;

– XCON – ЭС для конфигурирования (проектирования) вычислительных комплексов VAX-11 в корпорации DEC в соответствии с заказом покупателя;

– MYCIN – ЭС диагностики кишечных заболеваний;

– PUFF – ЭС диагностики легочных заболеваний;

– MACSYMA – ЭС для символьных преобразований алгебраических выражений;

– YES/MVS – ЭС для управления многозадачной операционной системой MVS больших ЭВМ корпорации IBM;

– DART – ЭС для диагностики больших НМД корпорации IBM;

– PROSPECTOR – ЭС для консультаций при поиске залежей полезных ископаемых;

– POMME – ЭС для выдачи рекомендаций по уходу за яблоневым садом;

– набор экспертных систем для управления планированием, запуском и полетом космических аппаратов типа "челнок";

– ЭСПЛАН- ЭС для планирования производства на Бакинском нефтеперерабатывающем заводе;

– МОДИС- ЭС диагностики различных форм гипертонии.

Структура экспертных систем

На рис.12.1. изображена обобщенная структура экспертной системы.

База знаний [22] предназначена для хранения экспертных знаний о предметной области, используемых при решении задач экспертной системой.

Рис. 12.1. Структура экспертной системы

База данных предназначена для временного хранения фактов или гипотез, являющихся промежуточными решениями или результатом общения системы с внешней средой, в качестве которой обычно выступает человек, ведущий диалог с экспертной системой.

Машина логического вывода – механизм рассуждений, оперирующий знаниями и данными с целью получения новых данных из знаний и других данных, имеющихся в рабочей памяти. Для этого обычно используется программно реализованный механизм дедуктивного логического вывода (какая-либо его разновидность) или механизм поиска решения в сети фреймов или семантической сети.

Машина логического вывода может реализовывать рассуждения в виде:

- дедуктивного вывода (прямого, обратного, смешанного);

- нечеткого вывода;

- вероятностного вывода;

- унификации (подобно тому, как это реализовано в Прологе);

- поиска решения с разбиением на последовательность подзадач;

- поиска решения с использованием стратегии разбиения пространства поиска с учетом уровней абстрагирования решения или понятий, с ними связанных;

- монотонного или немонотонного рассуждения,

- рассуждений с использованием механизма аргументации;

- ассоциативного поиска с использованием нейронных сетей;

- вывода с использованием механизма лингвистической переменной.

Подсистема общения служит для ведения диалога с пользователем, в ходе которого ЭС запрашивает у пользователя необходимые факты для процесса рассуждения, а также, дает возможность пользователю в какой-то степени контролировать и корректировать ход рассуждений экспертной системы.

Подсистема объяснений необходима для того, чтобы дать возможность пользователю контролировать ход рассуждений и, может быть, учиться у экспертной системы. Если нет этой подсистемы, экспертная система выглядит для пользователя как "вещь в себе", решениям которой можно либо верить, либо нет. Нормальный пользователь выбирает последнее, и такая ЭС не имеет перспектив для использования.

Подсистема приобретения знаний служит для корректировки и пополнения базы знаний. В простейшем случае это – интеллектуальный редактор базы знаний, в более сложных экспертных системах – средства для извлечения знаний из баз данных, неструктурированного текста, графической информации и т.д.

Все экспертные системы можно разделить на два больших класса [17]:

1) статические экспертные системы;

2) динамические экспертные системы.

Статические экспертные системы имеют неизменную во времени базу знаний и делают выводы на ее основе. Например, экспертная система по решению задач школьного курса математики, так как этот курс хорошо проработан, то знания экспертной системы не изменяются, а многие ученики из года в год ее используют для решения задач.

Динамические экспертные системы имеют изменяемую во времени базу знаний и делают выводы на ее основе. Поэтому в таких динамических экспертных системах одна и та же задача может быть решена по-разному, если вы обратитесь к экспертной системе в разное время. Например, экспертная система, прогнозирующая поведение финансовых рынков. Знания в экспертной системе пополняются и изменяются ежедневно и даже ежечасно. Ясно, что прогноз на первую неделю апреля о котировке доллара на рынке Форекс будет отличаться от прогноза на

первую неделю марта того же года. Хотя алгоритмы прогнозирования останутся те же, но знания экспертной системы о рынках за этот месяц изменятся существенным образом.

Лекция 13. Структура базы знаний в экспертной системе. Семантические сети. Фреймы. Уровни представления знаний и уровни детализации. Организация знаний в базе данных. Логические и ассоциативные связи.

Можно провести параллель между базами данных и базами знаний [2]. На рис. 13.1 приведена схема управления базой знаний

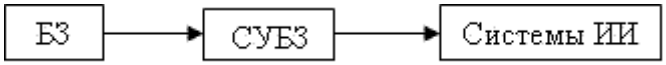


Рис. 13.1. Связь базы знаний с системами управления базой знаний и системами искусственного интеллекта

Концептуальная (интенсиональная) часть БЗ – обобщенное описание Предметной области и метазнания. Цель – представить структуру, свойства и поведение классов объектов предметной области.

Например, Любой сотрудник имеет уникальный табельный номер, занимает должность и имеет некоторый номер телефона.

Фактуальная (экстенсиональная) часть БЗ – свойства конкретных объектов.

Иван Иванович Иванов работает инженером и имеет служебный телефон 45-30.

Для постоения БЗ необходимо ответить на три вопроса:

- какие знания включать в БЗ,

- как их представлять,

- как использовать.

Модели представления знаний

1. Логическая – представление знаний в системе логики предикатов 1-го порядка и выведение заключения с помощью силлогизмов.

Факты: СТОЛИЦА (Москва): Москва – столица

ДРУГ (Петя, Вася): Петя дружит с Васей

Формулы: $(\forall x)[СЛОН(x) \rightarrow ЦВЕТ(x,Серый)]$: все слоны серые.

Логические модели. В основе моделей такого типа лежит формальная система, задаваемая четверкой вида: M = <T, P, A, B>. Множество T есть множество базовых элементов различной природы, например слов из некоторого ограниченного словаря, деталей детского конструктора, входящих в состав некоторого набора, и т.п. Важно, что для множества T существует некоторый способ определения принадлежности или непринадлежности произвольного элемента к этому множеству. Процедура такой проверки может быть любой, но за конечное число шагов она должна давать положительный или отрицательный ответ на вопрос, является ли X элементом множества T. Обозначим эту процедуру П(T).

Множество P есть множество синтаксических правил. С их помощью из элементов T образуют синтаксически правильные совокупности. Например, из слов ограниченного словаря строятся синтаксически правильные фразы, из деталей детского конструктора с помощью гаек и болтов собираются новые конструкции. Декларируется существование процедуры П(P), с помощью которой за конечное число шагов можно получить ответ на вопрос, является ли совокупность X синтаксически правильной.

В множестве синтаксически правильных совокупностей выделяется некоторое подмножество A. Элементы A называются аксиомами. Как и для других составляющих формальной системы, должна существовать процедура П(A), с помощью которой для любой синтаксически правильной совокупности можно получить ответ на вопрос о принадлежности ее к множеству A.

Множество B есть множество правил вывода. Применяя их к элементам A, можно получать новые синтаксически правильные совокупности, к которым снова можно применять правила из B. Так формируется множество выводимых в данной формальной системе совокупностей. Если имеется процедура П(B), с помощью которой можно определить для любой синтаксически правильной совокупности, является ли она выводимой, то соответствующая формальная система называется разрешимой. Это показывает, что именно правило вывода является наиболее сложной составляющей формальной системы.

Для знаний, входящих в базу знаний, можно считать, что множество A образуют все информационные единицы, которые введены в базу знаний извне, а с помощью правил вывода из них выводятся новые производные знания. Другими словами, формальная система представляет собой генератор порождения новых знаний, образующих множество выводимых в данной системе знаний. Это свойство логических моделей делает их притягательными для использования в базах знаний. Оно позволяет хранить в базе лишь те знания, которые образуют множество A, а все остальные знания получать из них по правилам вывода.

Метод резолюций – способ вывода

Метод имеет следующие особенности:

- точность и строгость определений и выводов, ясная запись фактов;

- большие модели плохо поддаются обработке, т. к. нет четких принципов организации фактов в БЗ.

- человек мыслит не формальной логикой.

Продукционная модель на основе правил

Правило 1: ЕСЛИ “намерение - отдых” и “дорога ухабистая”

ТО “использовать Ниву”.

Правило 2: ЕСЛИ “место отдыха - горы”

ТО “дорога ухабистая”.

В продукционной модели используются прямой вывод, от условий (ЭС для медицинской диагностики MYCIN) и обратный вывод, от цели (ЭС OPS5 для определения конфигурации ЭВМ VAX-II).

Продукционные модели. В моделях этого типа используются некоторые элементы логических и сетевых моделей. Из логических моделей заимствована идея правил вывода, которые здесь называются продукциями, а из сетевых моделей – описание знаний в виде семантической сети. В результате применения правил вывода к фрагментам сетевого описания происходит трансформация семантической сети за счет смены ее фрагментов, наращивания сети и исключения из нее ненужных фрагментов. Таким образом, в продукционных моделях процедурная информация явно выделена и описывается иными средствами, чем декларативная информация. Вместо логического вывода, характерного для логических моделей, в продукционных моделях появляется вывод на знаниях.

Обычно из правил строят дерево и/или. Оценка этого дерева на основе фактов в БЗ и есть логический вывод (рис. 13.2).

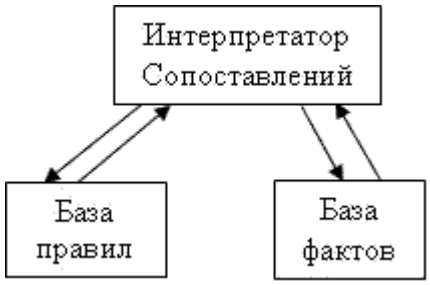


Рис. 13.2. Дерево правил

Модель имеет следующие преимущества:

- простота создания и понимания правил,

- простота пополнения и модификации,

- простота логического вывода;

и недостатки:

- сложность целостной оценки

- неясность взаимозависимости правил,

- низкая эффективность обработки,

- отличие от человеческой структуры знаний,

- отсутствие гибкости в логическом выводе.

Фреймовая (М. Минский) модель – систематизированная психологическая модель памяти и сознания человека.

Frame ~ рамка – структура данных для представления некоторого концептуального объекта.

Фреймовые модели. В отличие от моделей других типов во фреймовых моделях фиксируется жесткая структура информационных единиц, которая называется протофреймом. В общем виде она выглядит следующим образом:

(Имя фрейма:

Имя слота 1(значение слота 1);

Имя слота 2(значение слота 2);

. . . . . .

Слот – составляющая фрейма, содержащая информацию.

Фреймы связаны в систему, включая декларативные и процедурные знания.

Имя слота К (значение слота К)).

Значением слота может быть практически что угодно (числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов). В качестве значения слота может выступать набор слотов более низкого уровня, что позволяет во фреймовых представлениях реализовать "принцип матрешки".

При конкретизации фрейма ему и слотам присваиваются конкретные имена и происходит заполнение слотов. Таким образом, из протофреймов получаются фреймы-экземпляры. Переход от исходного протофрейма к фрейму-экземпляру может быть многошаговым, за счет постепенного уточнения значений слотов.

Например, структура, записанная в виде протофрейма, имеет вид:

(Список работников:

Фамилия (значение слота 1);

Год рождения (значение слота 2);

Специальность (значение слота 3);

Стаж (значение слота 4)).

Если в качестве значений слотов использовать конкретные данные, то получится фрейм - экземпляр

(Список работников:

Фамилия (Попов – Сидоров – Иванов – Петров);

Год рождения (1965 – 1946 – 1925 – 1937);

Специальность (слесарь – токарь – токарь – сантехник);

Стаж (5 – 20 – 30 – 25)).

Связи между фреймами задаются значениями специального слота с именем "Связь". Часть специалистов по интеллектуальным системам считает, что нет необходимости специально выделять фреймовые модели в представлении знаний, т.к. в них объединены все основные особенности моделей остальных типов.

Фрейма может быть записан в виде схемы. Пример такого фрейма приведен на рис. 13.3.

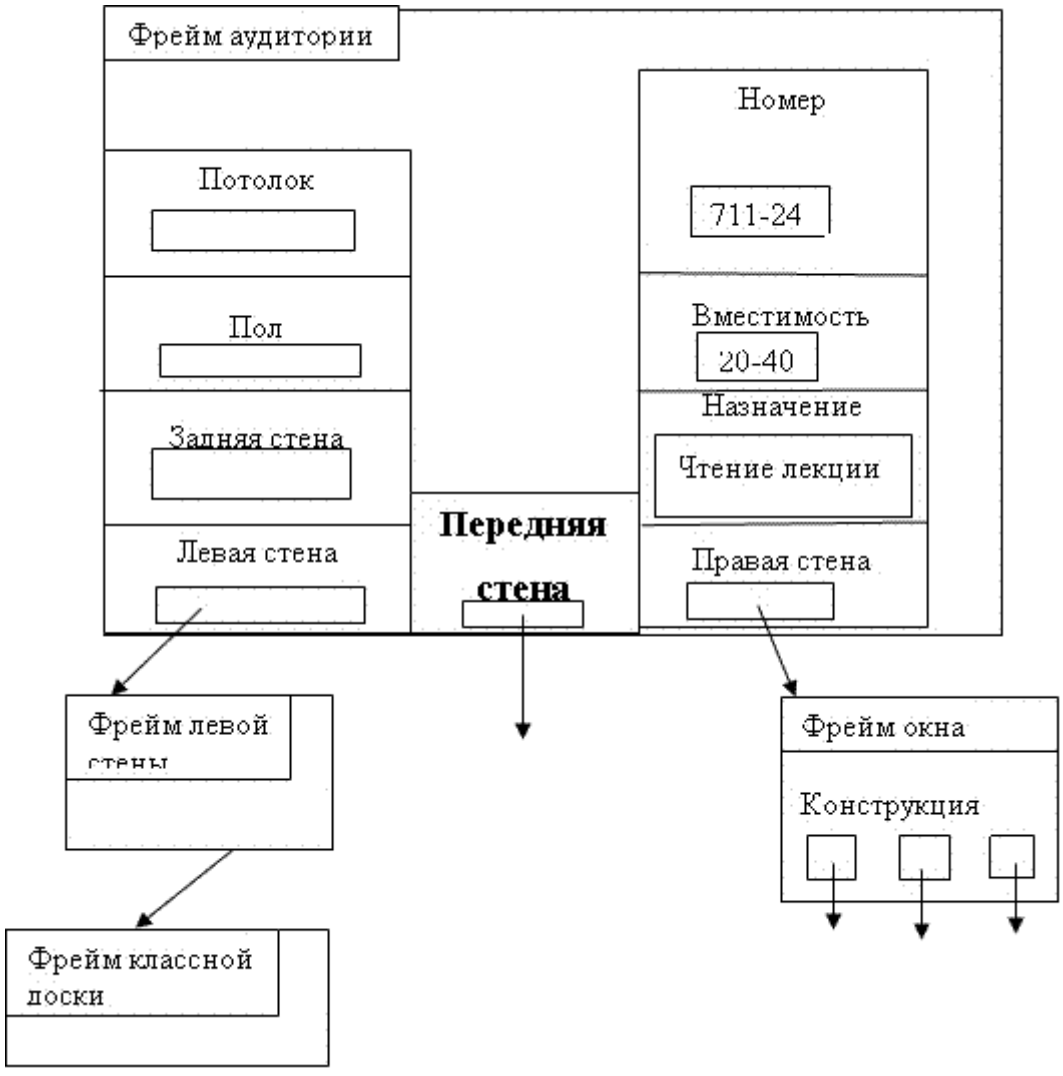


Рис. 13.3. Схема описания фрейма

Фреймы привели к появлению многих языков программирования заданий: FMS, FRL, KRL, LOOPS, COMEX и др.

Преимуществом языка программирования заданий на основе фреймов является то, что он эффективен для описания сложных объектов и понятий.

Недостатком языка программирования заданий на основе фреймов является трудность контролирования завершенности и постоянства целостного образа.

Сетевые модели. В основе моделей этого типа лежит конструкция, названная ранее семантической сетью.

Семантическая сеть – определение отношений между символами и объектами, представленными этими символами.

Рассмотрим пример, приведенный на рис. 13.4.

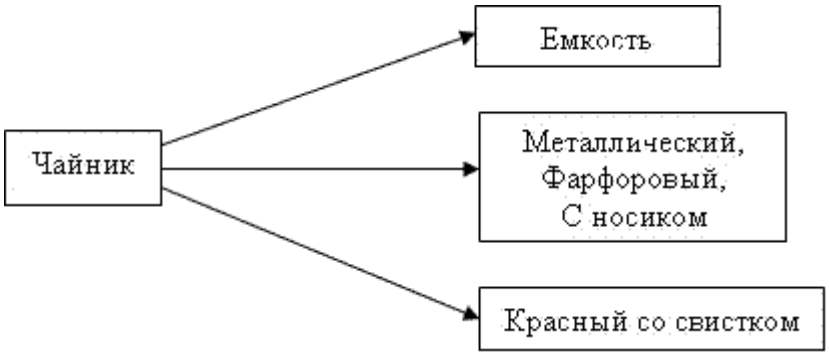


Рис. 13.4. Семантическая сеть

Сетевые модели формально можно задать в виде $H = <I, C_1, C_2, ..., C_n, \Gamma>$. Здесь *I* есть множество информационных единиц: $C_1, C_2, ..., C_n$–множество типов связей между информационными единицами. Отображение **Γ** задает между информационными единицами, входящими в *I*, связи из заданного набора типов связей.

В зависимости от типов связей, используемых в модели, различают классифицирующие сети, функциональные сети и сценарии. В классифицирующих сетях используются отношения структуризации. Такие сети позволяют в базах знаний вводить разные иерархические отношения между информационными единицами. Функциональные сети характеризуются наличием функциональных отношений. Их часто называют вычислительными моделями, т.к. они позволяют описывать процедуры "вычислений" одних информационных единиц через другие. В сценариях используются каузальные отношения, а также отношения типов "средство – результат", "орудие – действие" и т.п. Если в сетевой модели допускаются связи различного типа, то ее обычно называют семантической сетью.

Лекция 14.Методы поиска решений в экспертной системе. Методы поиска решений в иерархических пространствах. Методы поиска решений при неточных и неполных данных. Методы поиска решений, использующие несколько моделей, предназначенные для работы с областями, для адекватного описания которых одной модели недостаточно.

Методы решения задач, основанные на сведении их к поиску, зависят от особенностей предметной области, в которой решается задача, и от требований, предъявляемых пользователем к решению. Особенности предметной области [1] с точки зрения методов решения можно характеризовать следующими параметрами:

- размер, определяющий объем пространства, в котором предстоит искать решение;

- изменяемость области, характеризует степень изменяемости области во времени и пространстве (здесь будем выделять статические и динамические области);

- полнота модели, описывающей область, характеризует адекватность модели, используемой для описания данной области. Обычно если модель не полна, то для описания области используют несколько моделей, дополняющих друг друга за счет отражения различных свойств предметной области;

- определенность данных о решаемой задаче, характеризует степень точности (ошибочности) и полноты (неполноты) данных. Точность (ошибочность) является показателем того, что предметная область с точки зрения решаемых задач описана точными или неточными данными; под полнотой (неполнотой) данных понимается достаточность (недостаточность) входных данных для однозначного решения задачи.

Требования пользователя к результату задачи, решаемой с помощью поиска, можно характеризовать количеством решений и свойствами результата и (или) способом его получения. Параметр "количество решений" может принимать следующие основные значения: одно решение, несколько решений, все решения. Параметр "свойства" задает ограничения, которым должен удовлетворять полученный результат или способ его получения. Так, например, для системы, выдающей рекомендации по лечению больных, пользователь может указать требование не использовать некоторое лекарство (в связи с его отсутствием или в связи с тем, что оно противопоказано данному пациенту). Параметр "свойства" может определять и такие особенности, как время решения ("не более чем", "диапазон времени" и т.п.), объем памяти, используемой для получения результата, указание об обязательности (невозможности) использования каких-либо знаний (данных) и т.п.

Итак, сложность задачи, определяемая вышеприведенным набором параметров, варьируется от простых задач малой размерности с неизменяемыми определенными данными и отсутствием ограничений на результат и способ его получения до сложных задач большой размерности с

изменяемыми, ошибочными и неполными данными и произвольными ограничениями на результат и способ его получения. Из общих соображений ясно, что каким-либо одним методом нельзя решить все задачи. Обычно одни методы превосходят другие только по некоторым из перечисленных параметров.

Рассмотренные ниже методы могут работать в статических и динамических проблемных средах. Для того чтобы они работали в условиях динамики, необходимо учитывать время жизни значений переменных, источник данных для переменных, а также обеспечивать возможность хранения истории значений переменных, моделирования внешнего окружения и оперирования временными категориями в правилах.

Существующие методы решения задач, используемые в экспертных системах, можно классифицировать следующим образом:

- методы поиска в одном пространстве – методы, предназначенные для использования в следующих условиях: области небольшой размерности, полнота модели, точные и полные данные;

- методы поиска в иерархических пространствах – методы, предназначенные для работы в областях большой размерности;

- методы поиска при неточных и неполных данных ;

- методы поиска, использующие несколько моделей, предназначенные для работы с областями, для адекватного описания которых одной модели недостаточно.

Предполагается, что перечисленные методы при необходимости должны объединяться для того, чтобы позволить решать задачи, сложность которых возрастает одновременно по нескольким параметрам.

Теория фреймов – это парадигма для представления знаний с целью использования этих знаний компьютером. Впервые теория была представлена в 70-е годы Марвиным Минским, одним из пионеров искусственного интеллекта. для обозначения структуры знаний для восприятия пространственных сцен, как попытка построить фреймовую сеть, или парадигму с целью достижения большего эффекта понимания. С одной стороны М. Минский пытался сконструировать базу данных, содержащую энциклопедические знания, но, с другой стороны, он хотел создать наиболее описывающую базу, содержащую информацию в структурированной и упорядоченной форме. Эта структура позволила бы компьютеру вводить информацию в более гибкой форме, имея доступ к тому разделу, который требуется в данный момент. Минский разработал такую схему, в которой информация содержится в специальных ячейках, называемых фреймами, объединенными в сеть, называемую системой фреймов. Новый фрейм активизируется с наступлением новой ситуации. Отличительной его чертой является то, что он одновременно содержит большой объем знаний и в то же время является достаточно гибким для того, чтобы быть использованным как отдельный элемент БД. Термин «фрейм» был наиболее популярен в середине семидесятых годов, когда существовало много его толкований, отличных от интерпретации Минского.

Как было сказано выше фреймы – это фрагменты знания, предназначенные для представления стандартных ситуаций. Термин «фрейм» (Frame – рамка) был предложен Минским. Фреймы имеют вид структурированных компонентов ситуаций, называемых слотами. Слот может указывать на другой фрейм, устанавливая, таким образом, связь между двумя фреймами. Могут устанавливаться общие связи типа связи по общению. С каждым фреймом ассоциируется

разнообразная информация (в том числе и процедуры), например ожидаемые процедуры ситуации, способы получения информации о слотах, значения принимаемые по умолчанию, правила вывода.

Формальная структура фрейма имеет вид:

$$f\ [<N1, V1>, <N2, V2>, \ldots, <Nk,Vk>],$$

где f – имя фрейма; пара – i-й слот, Ni – имя слота и Vi – его значение.

Значение слота может быть представлено последовательностью

$$<K1><Li>;\ldots; <Kn><Ln>; <R1>; \ldots; <Rm>,$$

где Ki – имена атрибутов, характерных для данного слота; Li – значение этих атрибутов, характерных для данного слота; Rj – различные ссылки на другие слоты.

Каждый фрейм, как структура, хранит знания о предметной области (фрейм–прототип), а при заполнении слотов знаниями превращается в конкретный фрейм события или явления. Фреймы можно разделить на две группы: фреймы-описания; ролевые фреймы.

Рассмотрим пример.

Фрейм-описание: [<программное обеспечение>, <программа 1С бухгалтерия, версия 7.5>, <программа 1С торговля, версия 7.5>, <правовая программа «Консультант +» проф.>].

Ролевой фрейм: [<заявка на продажу>, <что, установка и покупка программы 1С торговля, версия 7.5>, <откуда, фирма ВМИ>, <куда, фирма «Лукойл»>, <кто, курьер Иванова>, <когда, 27 октября 1998г.>].

Во фрейме-описании в качестве имен слотов задан вид программного обеспечения, а значение слота характеризует массу и производителя конкретного вида продукции. В ролевом фрейме в качестве имен слотов выступают вопросительные слова, ответы на которые являются значениями слотов. Для данного примера представлены уже описания конкретных фреймов, которые могут называться либо фреймами-примерами, либо фреймами-экземплярами. Если в приведенном примере убрать значения слотов, оставив только имена, то получим так называемый фрейм-прототип.

Достоинство фрейма-представления во многом основываются на включении в него предположений и ожиданий. Это достигается за счет присвоения по умолчанию слотам фрейма стандартных ситуаций. В процессе поиска решений эти значения могут быть заменены более достоверными.

Лекция 15. Инструментальный программный комплекс G2 для создания экспертных систем. Структура комплекса G2. Задачи, успешно решаемые с помощью экспертных систем, изготовленных на основе комплекса G2.

G2 – это объектно-ориентированная [5, 19, 20], графическая среда от компании Gensym для разработки и сопровождения динамических экспертных систем реального времени. Эта инструментальная среда применяется в широком диапазоне предметных областей, где необходимо принимать решения, включающих в себя:

- мониторинг, системную диагностику и предупреждения о внештатных ситуациях (анализ и верификация сенсоров, фильтрация предупреждающих сигналов);

- планирование и логистику;

- усовершенствованное управление процессами;

- проектирование процессов, имитационное моделирование, реинжиринг;

- управление интеллектуальными сетями, поддержку принятия решений для систем масштаба предприятия.

G2 объединяет в себе как универсальные технологии построения современных информационных систем (стандарты открытых систем, архитектура клиент/сервер, объектно-ориентированное программирование, использование ОС, обеспечивающих параллельное выполнение в реальном времени многих независимых процессов), так и специализированные методы (рассуждения, основанные на правилах, рассуждения, основанные на динамических моделях, или имитационное моделирование, процедурные рассуждения, активная объектная графика, структурированный естественный язык для представления базы знаний), а также интегрирует технологии систем, основанных на знаниях с технологией традиционного программирования (с пакетами программ, с СУБД, с контроллерами и концентраторами данных и т.д.).

Функциональные возможности G2 включают в себя:

- возможности G2-приложения (работа в реальном времени, функции экспертной системы, поддержка процедурного языка программирования, интерфейс пользователя, переносимость баз знаний, распределенная обработка);

- средства разработки G2-приложений (естественный язык, использование объектов с атрибутами и иерархии классов, текстовый редактор с грамматическим разбором, редактор пиктограмм, средство инспекции, средство отладки).

Прикладные программы, написанные в среде G2, могут существенно повысить эффективность выполнения операций благодаря следующим факторам:

- непрерывному контролю над потенциальными проблемами прежде, чем они проявят неблагоприятное воздействие;

- принятие комплексных оперативных решений на основе информации, полученной посредством рассуждений и анализа данных, содержащихся в интеллектуальной модели процесса;

- диагностирование основных случаев возникновения проблем, критичных ко времени выполнения и выработки последовательности правильных действий;

- поддержание оптимальных рабочих условий;

- координирование действий и информации в выполнении сложных оперативных процессах.

Платформа G2 сочетает технологии принятия решений в реальном времени, включая правила, последовательности выполняемых действий, процедуры, объектное моделирование, имитационное моделирование и графику в одной среде разработки. G2 преобразует данные в реальном времени в автоматизированное принятие решений и выполнение действий. Приложения G2 работают совместно с операционными системами, включая системы управления предприятием, базами данных, автоматизированными системами, системами управления сетями, системами дистанционного измерения и др. В G2 разработчики приложений могут быстро проектировать с помощью прототипов и моделирования, разрабатывать и изменять правила принятия решений, основанных на рассуждениях.

G2 эмулирует в реальном времени рассуждения, как если бы это делали эксперты, которые оценивают, диагностируют и реагируют на необычные ситуации или ищут оптимальное решение.

Основными особенностями оболочки G2 является:

- работа в реальном времени с распараллеливанием процессов рассуждений;

- структурированный естественно-языковый интерфейс с управлением по меню и автоматической проверкой синтаксиса;

- обратный и прямой вывод, использование метазнаний, сканирование и фокусирование;

- интеграция подсистемы моделирования с динамическими моделями для различных классов объектов;

- структурирование БЗ, наследование свойств, понимание связей между объектами;

- библиотеки знаний являются ASCII-файлами и легко переносятся на любые платформы и типы ЭВМ;

- развитый редактор для сопровождения БЗ без программирования, средства трассировки и отладки БЗ;

- управление доступом с помощью механизмов авторизации пользователя и обеспечения желаемого взгляда на приложение;

- гибкий интерфейс оператора, включающий графики, диаграммы, кнопки, пиктограммы и т.п.;

- интеграция с другими приложениями (по TCP/IP) и базами данных, возможность удаленной и многопользовательской работы.

В отличие от систем, ориентированных на какую-то одну методологию или на конкретную предметную область, G2 интегрирует в себе множество взаимодополняющих методов искусственного интеллекта, что упрощает и ускоряет процесс разработки приложений и позволяет делать их универсальными.

Основное предназначение программных продуктов семейства G2 – помочь предприятиям сохранять и использовать знания и опыт их наиболее талантливых и квалифицированных сотрудников в интеллектуальных системах реального времени, повышающих качество продукции, надежность и безопасность производства и снижающих производственные издержки. Инструментальные средства G2 являются эволюционным шагом в развитии традиционных экспертных систем от статических предметных областей к динамическим.

Большим достоинством оболочки экспертных систем G2 является возможность применять ее как интегрирующий компонент, позволяющий за счет открытости интерфейсов и поддержки широкого спектра вычислительных платформ легко объединить уже существующие, разрозненные средства автоматизации в единую комплексную систему управления, охватывающую все аспекты производственной деятельности – от формирования портфеля заказов до управления технологическим процессом и отгрузки готовой продукции. Это особенно важно для отечественных предприятий, парк технических и программных средств которых формировался по большей части бессистемно, под влиянием резких колебаний в экономике. G2 – кроссплатформенное приложение, ее базы знаний сохраняются в обычном ASCII-файле, что позволяет безболезненно переносить ее из одной операционной системы в другую.

Кроме, системы G2, как базового средства разработки, у Gensym есть комплекс проблемно/предметно-ориентированных расширений для быстрой реализации сложных динамических систем на основе специализированных графических языков, включающих параметризуемые операторные блоки для представления элементов технологического процесса и типовых задач обработки информации. Набор инструментальных сред фирмы Gensym, сгруппированный по проблемной ориентации, охватывает все стадии производственного процесса и выглядит следующим образом:

- интеллектуальное управление производством – G2, G2 Diagnostic Assistant (GDA), NeurOn-Line (NOL), Statistical Process Control (SPC), BatchDesign_Kit;

- оперативное планирование – G2, G2 Scheduling Toolkit (GST), Dynamic Scheduling Packadge (DSP);

- разработка и моделирование производственных процессов – G2, ReThink, BatchDesign_Kit;

- управление операциями и корпоративными сетями – G2, Fault Expert.

Лекция 16. Объектно-ориентированная технология проектирования экспертных систем. Иерархия классов в программном комплексе G2. Типовые правила и процедуры. Рабочие области организации данных.

Объекты в G2 являются средством представления материальных и абстрактных сущностей в прикладных программах. Объектно-ориентированный подход к разработке программ позволяет быстро и легко:

- встроить модули и объекты из других прикладных программ;

- графически определить объекты, их свойства и действия;

- создать новые образцы объектов, имитируя существующие объекты.

Объекты или класс объектов, определенные один раз, могут использоваться многократно. Любой объект или группа объектов могут иметь несколько экземпляров. Каждый экземпляр наследует все свойства и поведение первоначального объекта(ов). Объекты (а также правила и процедуры) можно группировать в библиотеки, которые будут общими для всех прикладных программ.

Для моделирования широкого разнообразия действий типа производственных процессов, сетевых топологий, информационных маршрутизаций, или логических потоков объекты можно объединять графически на экране дисплея.

Представление знаний

Существует два типа файлов, в которых могут храниться знания в G2: базы знаний и библиотеки знаний. В файлах баз знаний хранятся знания о приложениях: определения всех объектов, объекты, правила, процедуры и т.п. В файлах библиотек знаний хранятся общие знания, которые могут быть использованы более, чем в одном приложении, например определение стандартных объектов.

В G2 для представления знаний используется структурный естественный язык, что позволяет облегчить чтение, редактирование и поддержку баз знаний. Это облегчает использование и редактирование приложений пользователем-непрограммистом. Для создания и редактирования баз знаний используется Редактор баз знаний (KB Editor).

Классы и иерархия классов

Представление знаний в G2 осуществляется с помощью классов. Структуры данных представляются в виде классов объектов (или определений объектов), имеющих определенные атрибуты. Классы наследуют атрибуты от суперклассов и передают свои атрибуты подклассам.

Все, что хранится в базе знаний и с чем оперирует система, является экземпляром того или иного класса. Все синтаксические конструкции G2 также являются классами. Для сохранения общности даже базовые типы данных – символьные, числовые, булевы и истинностные значения нечеткой логики – представлены соответствующими классами. Описание класса включает ссылку на суперкласс и содержит перечень атрибутов, специфичных для класса.

В G2 входит большой набор системных классов, многие из которых можно использовать для создания пользовательских классов. Каждый класс в иерархии классов G2 является либо системным, либо пользовательским.

Классы имеют атрибуты, которые определяют унаследованные или заданные пользователем свойства класса. Атрибуты задаются в таблице атрибутов.

В классовой иерархии G2 возможно множественное наследование: любой заданный пользователем класс может наследовать атрибуты и методы любого суперкласса.

В G2 существует специальная функция Configurations (Конфигурации) для создания пользовательских режимов баз знаний и управления свойствами элементов баз знаний. Можно изменять свойства отдельных элементов или иерархически задавать свойства группам и классам элементов, обозначенных разными способами.

С помощью G2 новые классы могут создаваться не только в процессе разработки, но и динамически, во время работы приложения. Во время исполнения приложения может быть создан, модифицирован или уничтожен экземпляр любого класса или целый класс. Это касается как объектов, так и правил и процедур.

Правила и процедуры

В G2 можно эффективно создавать и применять общие знания, создавая универсальные правила, процедуры, формулы и зависимости, которые являются применимыми для полных классов объектов. В результате сокращается время на разработку и увеличивается эффективность приложений.

Для представления знаний эксперта о проблемной области используются правила. Правила могут быть как общими, то есть относящимися ко всему классу, так и специфическими, относящимися к конкретным экземплярам класса.

Обычной целью правила является проверка, имеет ли переменная или параметр конкретное значение, и если да, то следует предпринять определенные действия, возможно, использующие другие переменные или параметры.

Правила определяют, как база знаний будет реагировать на различные условия. Правила описывают знания таким образом, что позволяют базе знаний вывести заключение, основанное на существующих знаниях, и реагировать на конкретные события.

Правила в G2 имеют традиционный вид: условие (антецедент) и заключение (консеквент). Пример:

```
if the level of any tank = 0

then conclude that the status of the tank is empty
```

В G2 реализованы 5 типов правил:

- `if` (если): применяется в большинстве случаев;

- `initially` (в начале): реагирует на активацию рабочего пространства;

- `unconditionally` (безусловно): выполняет действия при запуске;

- `when` (когда): также как и правило `if`, но не участвует в выводе;

- `whenever` (всякий раз когда): реагирует на события.

Можно создать:

- специальные правила, которые буду применяться к особым объектам и значениям;

- общие правила, которые будут применяться к набору объектов и значений. в таких правилах в тексте в качестве префикса используется `for`.

Процедура – это заданная последовательность операций, которые выполняются последовательно или параллельно каждый раз, когда запускается процедура. Процедуры используются для выполнения операций, повторяющихся при различных условиях и/или с различными значениями данных.

G2 запускает процедуру, когда имя процедуры и ее аргументы (если они есть) появляются в операторе `call` или действии `start`. Процедуры выполняются синхронно, когда они вызываются (`called`), и асинхронно, если запускаются (`started`). Вызванная процедура возвращает одно или несколько значений.

Процедуры и правила во многом похожи, но их предназначение различается, также немного отличается и их синтаксис.

Процедуры используют для выполнения конкретного (явно заданного) набора операций, а также для контроля потоками событий. Например: to perform the steps involved in starting up or shutting down a plant (выполнять определенные действия при включении или выключении устройства).

Правила используют для слежения за асинхронными событиями, а также для обнаружения и предупреждения проблем, которые могут произойти. Пример: to watch for conditions that exceed specified limits (следить за параметрами, превышающими заданные ограничения).

Для слежения за событиями правила лучше процедур, так как они не потребляют ресурсы активным ожиданием, как процедуры.

В процедуре вы точно контролируете ожидание G2-приложения и разрешаете запуск других процессов. Это отличается от того, как работают правила, потому что вы не можете контролировать, когда правило получит то или иное значение. В результате G2 гарантированно выполняет задачи процедуры без прерываний, за исключением случаев, когда вы указываете в процедуре выполнение других процессов.

Подобно правилам, процедуры в G2 выполняются в реальном времени. Процедуры, правила и модели выполняются одновременно согласно их приоритетам. Процедуры могут использоваться для эффективного представления поведения объектов. В результате это позволяет формировать мощные прикладные системы реального времени проще и быстрее, чем с помощью традиционных инструментальных средств программирования.

Язык программирования, используемый в G2 для представления процедурных знаний, является достаточно близким к Паскалю. Кроме стандартных управляющих конструкций язык расширен элементами, учитывающими работу процедуры в реальном времени: ожидание наступления

событий, разрешение другим задачам прерывать выполнение данной процедуры, директивы, задающие последовательное или параллельное выполнение операторов. Еще одна интересная особенность языка – итераторы, позволяющие организовать цикл над множеством экземпляров класса. Перечисленные свойства языка позволяют системе одновременно выполнять множество различных процедур или множество копий одной и той же процедуры для множества различных объектов.

Модули и рабочие пространства

Знания в G2 структурируются через иерархию классов, иерархию модулей и иерархию рабочих пространств. Несмотря на то, что функции модулей и рабочих пространств похожи, между ними есть существенные различия.

G2-приложение может быть организовано в виде одной базы знаний или в виде нескольких баз знаний, называемых модулями. Модули приложения организованы в древовидную иерархию с одним модулем верхнего уровня. Модули следующего уровня состоят из тех модулей, без которых не может работать модуль предыдущего уровня (рис. 16.1.)

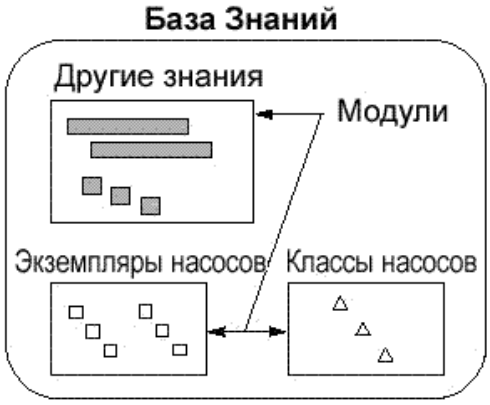


Рис. 16.1. Модуль G2

Существует два способа создать G2-приложения:

- разрабатывается одномодульное приложение, которое затем при необходимости разделяется на отдельные модули;

- приложение изначально создается, как состоящее из нескольких модулей.

Некоторые из этих модулей разрабатываются впервые, а другие могут выбираться из библиотеки знаний.

Каждый созданный модуль или загруженный модуль имеет свой собственный набор системных таблиц. Разработчик определяет каждый модуль в системной таблице Module Information (Информация о модулях).

Системные таблицы определяют значения по умолчанию глобальных параметров, относящихся ко всей базе знаний. Атрибуты системной таблицы влияют на значения соответствующих системных параметров. При создании нового модуля G2 автоматически создает новый набор системных таблиц и привязывает их к новому модулю.

Сущности базы знаний привязываются к модулю присваиванием этого модуля к одному или нескольким рабочим пространствам верхнего уровня в базе знаний. Таким образом, рабочее пространство, все элементы рабочего пространства и все элементы ниже по иерархии будут связаны с этим модулем.

Рабочие пространства являются контейнерным классом, в котором размещаются другие классы и их экземпляры, например объекты, связи, правила, процедуры и т.д. Каждый модуль (база знаний) может содержать любое количество рабочих пространств. Объекты рабочего пространства могут иметь свои дополнительные рабочие пространства.

Рабочие пространства образуют одну или несколько древовидных иерархий с отношением "is-a-part-of" ("является частью"). С каждым модулем (базой знаний) ассоциируется одно или несколько рабочих пространств верхнего (нулевого) уровня, каждое из этих рабочих пространств является корнем соответствующей древовидной иерархии. В свою очередь, с каждым объектом (определением объекта или связи), расположенным в нулевом уровне, может быть ассоциировано рабочее пространство первого уровня, связанное с ним отношением "является частью", и т.д.

Таким образом, можно создавать логическую иерархию объектов и рабочих пространств, чтобы группировать и организовывать данные вашей базы знаний.

Рабочие пространства могут содержать любые объекты, от текстовых сообщений до целых схем, моделирующих процессы в реальном времени.

Различие между "модулями" и "рабочими пространствами" состоит в следующем. Модули разделяют приложение на отдельные базы знаний, совместно используемые в различных приложениях. Динамические модули (аналог библиотек динамического связывания) могут подгружаться и вытесняться из оперативной памяти во время исполнения программно и одновременно использоваться несколькими приложениями. Рабочие пространства выполняют свою роль при исполнении приложения. Они содержат в себе (и в своих подпространствах) различные сущности и обеспечивают деление приложения на небольшие части, которые легче понять и обрабатывать. Например, весь процесс разбивается на подпроцессы и с каждым подпроцессом ассоциируется свое подпространство.

Рабочие пространства могут устанавливаться (вручную или действием в правиле/процедуре) в активное или неактивное состояние (т.е. объекты, находящиеся в этом пространстве и в его подпространствах становятся невидимыми для механизма вывода). Механизм активации/деактивации рабочих пространств используется, например, при наличии альтернативных групп правил, когда активной должна быть только одна из альтернативных групп.

Кроме того, рабочие пространства используются для определения пользовательских ограничений, определяющих различное поведение приложения для различных категорий пользователей.

На рис. 16.2 представлен пример иерархии классов G2 с помощью функции `Inspect` (средства для доступа к базе знаний и ее просмотра). Иерархия начинается с класса ITEM слева и расширяется направо. Все классы, представленные на рисунке, являются системными.

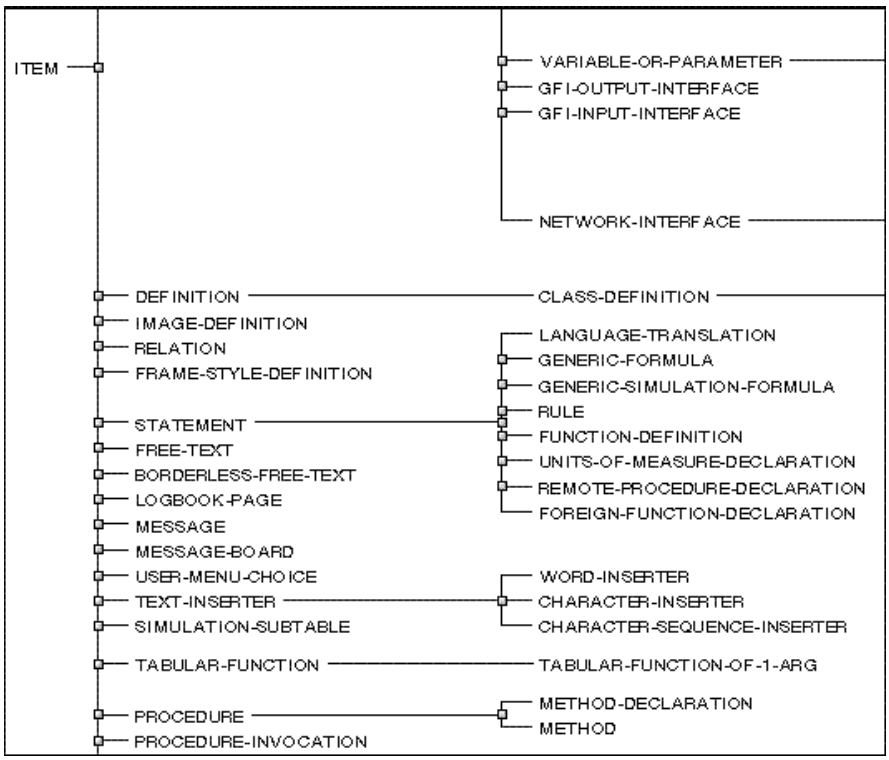


Рис. 16.2. Иерархия классов G2

Лекция 17. Параллельная обработка в реальном времени. Машина логического вывода реального времени. Сканирование. Фокусирование и вызов знаний. Динамическое моделирование и имитация.

Объектно-ориентированное программирование обеспечивает параллельное выполнение в реальном времени многих независимых процессов в G2 [20, 24]. В первую очередь это достигается за счет параллельной обработки данных в программных модулях и процедурах, а также за счет параллельного выполнения правил.

Одной из основных компонент G2 является машина вывода, выполняющая рассуждения на основании:

- знаний, содержащихся в базе знаний;

- данных, поступающих от подсистемы имитационного моделирования;

- данных, поступающих от внешних источников (контрольно-измерительной аппаратуры, СУБД и т.п.).

Правила вызываются машиной вывода. При этом проверяется истинность условия, находящегося в антецеденте правила. Если оно истинно, то машина вывода выполняет действия, находящиеся в консеквенте. При проверке условия правила машине вывода необходимо найти значения всех переменных и параметров, содержащихся в них. Параметры в любой момент времени имеют определенное значение, в то время как значение переменных может отсутствовать, поскольку для них определено время жизни.

В связи с тем что G2 ориентирована на приложения, работающие в реальном времени, в машине вывода должны быть средства для сокращения перебора, для реакции на непредвиденные события и т.п. Для машины вывода G2 характерен богатый набор способов вызова правил. Предусмотрено девять случаев:

1. Данные, входящие в антецедент правила изменились (прямой вывод).

Прямой вывод – это процесс выполнения правила, если хотя бы одно из условий в антецеденте выполняется. Прямой вывод является формой дедуктивного рассуждения. При прямом выводе G2 идентифицирует все правила, в чьи антецеденты входят изменившееся значение. Если в заключении правила выводится новое значение, это правило должно вызывать прямой вывод. Новое значение переменной может вызвать прямой вывод антецедента одного или нескольких правил. G2-приложение вызывает такие правила параллельно и назначает выполнение каждого правила в соответствии с заявленным приоритетом.

G2 возбуждает правила прямым выводом, когда значения в антецеденте меняются по любым причинам, а не только потому, что они являются результатом вывода другого правила. Значения в антецеденте могут меняться у атрибутов, переменных или параметров. G2-приложение вызывает прямой вывод правил, когда переменная или параметр в правиле получает новое значение, отличающееся от предыдущего; в то же время G2-приложение вызывает прямой вывод всякий раз, когда атрибут в правиле получает новое значение, независимо от того, изменилось оно или нет.

В основном прямой вывод работает только с правилами `if`. И хотя в G2 нельзя сделать прямой вывод для правила `whenever`, его можно сделать от правила `whenever`. В этом случае прямой вывод происходит, когда правило whenever выводит значение переменной, заявленной в прямом выводе.

2. Правило определяет значение переменной, которое требуется другому правилу или процедуре (обратный вывод).

Обратный вывод выполняется, если у переменной сущности БЗ нет текущего значения. G2-приложение может вызывать через обратный вывод одно или несколько правил, чей консеквент выводит значение переменной.

3. Каждые n секунд, где n – число, определенное для данного правила (сканирование).

G2 вызывает правила в течение указанного интервала времени, если для них указан атрибут Scan-interval. Этот способ называется сканированием, потому что G2 вызывает правила с течением времени, вне зависимости от состояния знаний в БЗ.

Для вызова правила сканированием надо задать интервал времени в атрибуте правила Scan-interval. Например, если требуется, чтобы G2-приложение проверяло температуру в резервуаре (temperature of tank-4) каждые 5 минут, для следующего правила в атрибуте Scan-interval следует указать 5:

```
if the temperature of tank-4 > 40 F

then inform the operator that "Tank-4 is overheating."
```

G2 будет вызывать это правило каждые пять минут; таким образом, каждые пять минут G2-приложение будет измерять текущую температуру резервуара, сравнивать ее с `40 F`, и сообщит оператору о перегреве резервуара.

Сканирование является менее эффективным способом для вызова правил по сравнению с обнаружением событий. Вызов правила сканированием может побудить G2-приложение выполнить действия, которые не будут иметь отношения к условиям, описанным в каждом просканированном правиле.

Если G2-приложение сканирует много правил, при некоторых условиях представление БЗ может быть ограничено. Важной задачей при анализе приложения является идентификация объектов, чьи знания напрямую зависят от течения времени в указанном интервале. Управлять с помощью вызова сканированных правил следует только такими объектами.

4. Явный или неявный вызов другим правилом – путем применения действий фокусирования и вызова правил.

Если большинство правил являются общими, можно вызывать эти правила таким образом, чтобы они применялись только к определенной сущности или набору сущностей. Это называется фокусировкой. В G2 поддерживается 2 способа вызова правил фокусировкой:

- фокусировка на правилах, связанных с определенными объектами или классами объектов;

- фокусировка на правилах, заданных для определенной категории.

Фокусный объект или фокусный класс объектов определяется с помощью оператора focus (фокусировка). В результате выполнения действия focus G2-приложение вызывает все правила, связанные с фокусным объектом или фокусным классом.

Определение фокусного объекта подходит для специальных правил, которые относятся только к конкретной сущности. Однако фокусный объект также можно связать и с общими правилами.

Определение фокусного класса подходит для общих правил, связанных с набором сущностей или значений.

Когда G2-приложение вызывает правило с помощью фокусировки, правило применяется к каждому объекту, являющемуся аргументом оператора focus.

В операторе `invoke` (вызов) в качестве аргумента выступает одна или несколько категорий правил. В результате этого оператора G2-приложение вызывает все правила из указанной категории. Для обозначения категории правила надо определить атрибут правила Categories (категории).

Если G2-приложение выполняет действие `invoke`, чьим аргументом является набор объектов, то вызывается копия правила для каждой категории правила и для каждого объекта.

5. Каждый раз при запуске приложения.

6. Входящей в антецедент переменной присвоено значение, независимо от того, изменилось оно или нет.

7. Определенный объект на экране перемещен пользователем или другим правилом.

8. Определенное отношение между объектами установлено или уничтожено.

9. Переменная не получила значения в результате обращения к своему источнику данных.

В G2 существуют функции отладки и слежения за выполнением правил. Эти функции уведомляют пользователя, когда запускается, приостанавливается и выполняется конкретное правило.

Динамическое моделирование и планировщики

В связи с тем что G2-приложение управляет множеством одновременно возникающих задач, необходим планировщик. Планировщик управляет всеми процессами в G2. Хотя пользователь никогда не взаимодействует с ним, планировщик контролирует как всю активность, видимую пользователем, так и активность фоновых задач. Планировщик определяет порядок обработки задач, взаимодействует с источниками данных и пользователями, запускает процессы и осуществляет коммуникацию с другими процессами в сети.

Подсистема моделирования G2 является автономной, но важной частью системы. Во время разработки подсистема моделирования используется вместо объектов реального мира для имитации показаний датчиков.

G2 позволяют динамически моделировать системы и процессы, используя объекты, правила, процедуры и формулы. Во время разработки модели используются вместо объектов реального мира, что позволяет непрерывно проверять прикладные программы. Модели могут использоваться на этапе эксплуатации как часть прикладной программы G2 для сравнения фактических и модельных значений.

Модели можно также использовать для проведения анализа и ответа на вопросы "что, если", определения, например, лучших рабочих условий или лучших проектов. Модели можно также использовать для предсказания важных параметров действий в реальном времени, например, качество изделия или затрат.

На этапе эксплуатации прикладной системы процедуры моделирования выполняются параллельно функциям мониторинга и управления процессом, что обеспечивает следующие возможности:

- верификация показаний датчиков во время исполнения приложения;

- подстановка модельных значений переменных при невозможности получения реальных (выход из строя датчика или длительное время получения ответа на запрос).

Играя роль самостоятельного агента знаний, подсистема моделирования повышает надежность приложений на базе G2.

Интервалы цикла работы планировщика системы позволяют контролировать процессы со скоростью протекания на уровне миллисекунд. Подсистема сбора профиля работы приложения дает возможность на этапе отладки и во время эксплуатации системы целенаправленно проводить модификации для достижения требуемой производительности. Использование принципов статической и условной компиляции повышает производительность критических участков приложения при сохранении гибкости системы в целом.

Лекция 18. Создание экспертной системы на основе комплекса G2. Среда разработки. Структурированный естественный язык. Создание экспертной системы для сети клиент/сервер. Инструментарий G2 для связи с основными стандартами. G2 JavaLink, G2 webLink. Интерфейсы G2 реального времени. Этапы разработки приложений G2.

В настоящее время оболочка G2 поддерживает различные платформы, обладает всеми возможностями графического интерфейса и визуального проектирования и не имеет конкурентов среди инструментальных средств построения сложных интеллектуальных систем анализа и управления в динамически изменяющемся окружении. В G2 поддержаны все основные принципы объектно-ориентированного подхода.

Среда разработки программного обеспечения G2 во много раз увеличивает производительность разработчика по сравнению с традиционными методами программирования. Изменение описаний классов и отношений во время исполнения позволяет не только экспериментировать со структурами данных непосредственно в процессе отладки, но и открывает возможности использования генетических алгоритмов, самодифицирующихся и обучающихся систем.

Возможность простого управления графическим представлением объектов в G2 и составления схем, являющихся отображением технологических цепочек или абстрактных алгоритмов обработки данных, обеспечивает базовые средства для построения проблемно-ориентированных языков визуального программирования. В данном случае объекты приобретают свойства операторов и в совокупности с различными классами связей формируют грамматику нового языка. Основным преимуществом такого подхода является то, что сформированная диаграмма потоков информации, по сути, и есть исполняемая программа, промежуточные фазы генерации кода и компиляции для ее использования не требуются.

G2 поддерживает естественные языки (английский, японский, корейский, русский). Библиотека знаний language.kl обеспечивает локализацию G2 на нескольких европейских языках.

Таким образом, достоинствами данной среды разработки является интуитивно понятное графическое объектно-ориентированное моделирование; естественные языки на описания правил; логика, основанная на правилах и процедурах; встроенная функция «понимания» времени; возможность незамедлительного изменения программы без прерываний ее выполнения или перекомпоновки кода; многочисленные средства для разработки и утилиты.

В G2 реализована клиент-серверная технология, а легкая интеграция с разнородными источниками информации позволяет использовать ее в качестве связующего звена в гетерогенных распределенных вычислительных средах, объединяющих как технические средства (контроллеры и каналы связи), так и развитые СУБД (ORACLE, INFORMIX, все ODBC совместимые СУБД). Передача объектов и массивов в качестве аргументов упрощает совместное использование данных как независимыми приложениями на базе G2, так и внешними по отношению к G2 программными системами. Конфиденциальность и безопасность распределенной обработки достигаются за счет

системы уровней автоматической проверки прав доступа при установлении сетевого взаимодействия процессов через независимый монитор транзакций – G2 Standard Interface (GSI).

Интерфейс G2 Gateway GSI является ориентированным на сеть инструментальным средством для разработки интерфейсов программного обеспечения или мостов между G2 и другими внешними системами. G2 Gateway позволяет базам знаний осуществлять обмен различными типами данных между процессами G2 и мостами.

Мост G2 Gateway также является процессом, обменивающимся данными с G2 посредством протокола DECnet или TCP/IP.

Интерфейс «G2 to G2» позволяет двум и более G2-процессам обмениваться данными между собой. G2 работает либо с протоколом DECnet, либо TCP/IP. Как только две системы подключаются друг к другу, можно обмениваться различными типами данных. Каждая из систем G2 может посылать и получать данные. Это позволяет создавать приложения, использующие распределенную обработку информации в отдельных системах G2.

Каждый разработчик имеет доступ к базе знаний, находящейся на сервере, при помощи средства, называемого Telewindows, обычно расположенного на компьютере-клиенте.

В этом случае разработчики могут иметь различные авторизованные уровни доступа к приложениям. Приложение может быть реализовано не только на различных ЭВМ, но и с использованием нескольких взаимодействующих оболочек G2.

Telewindows предоставляет параллельный расшаренный (shared) доступ к G2 из любого места. Telewindows обеспечивает удаленное управление системой с помощью полного доступа разработчиков к исполняемым удаленно приложениям. Как только приложение запущено Telewindows позволяет многим пользователям совместно работать с этим приложением.

С помощью Telewindows2 Toolkit можно разрабатывать и выполнять G2-приложение в стандартной программной среде, такой, как Microsoft Visual Basic, Microsoft Explorer и Netscape Navigator. Программные компоненты G2, рабочие пространства и редакторы можно представить соответствующим образом. Telewindows2 Toolkit поддерживает интеграцию JavaBean и ActiveX компонентов в приложение G2 на стороне клиента.

В дополнение к Telewindows G2 WebLink позволяет пользователям взаимодействовать с приложениями G2 удаленно посредством браузеров типа Netscape и Microsoft Explorer. Благодаря поддержке гипертекстового протокола HTTP эта функция обеспечивает обычным пользователям доступ через интернет или интранет.

Продукт G2 JavaLink позволяет приложениям G2 взаимодействовать с другими программными компонентами, разработанными на языке Java и оперативными средствами управления работой программы. С помощью этого продукта G2-приложения могут использовать постоянно растущее число доступных библиотек программного обеспечения Java-компонентов.

Продукт G2 ActiveXLink позволяет G2-приложениям взаимодействовать с компонентами программного обеспечения Microsoft DCOM/ActiveX. С его помощью G2-приложения могут работать с такими программами, как EXCEL, Visual Basic и PowerPoint.

Таким образом, основанная на правилах логика, объекты, структуры данных и пользовательский интерфейс легко интегрируется в операционные системы и внедряется в приложения других

компаний через поддержку стандартов ActiveX/COM, .NET, Java RMI, SQL, ODBC, JMS, OPC, XML, HTTP и др.

Этапы разработки приложений G2

Разработка прототипа приложения. Разработчиком обычно является специалист в конкретной области знаний. Он в ходе обсуждений с конечным пользователем определяет функции, выполняемые прототипом. При разработке прототипа не используется традиционное программирование. Создание прототипа обычно занимает от одной до двух недель (при наличии у разработчика опыта по созданию приложений в данной среде). Прототип, как и приложение, создается на структурированном естественном языке, с использованием объектной графики, иерархии классов объектов, правил, динамических моделей внешнего мира. Многословность языка сведена к минимуму путем введения операции клонирования, позволяющей размножить любую сущность базы знаний.

Расширение прототипа до приложения. Конечный пользователь предлагает этапность проведения работ, направления развития базы знаний, указывает пропуски в ней. Разработчик может расширять и модифицировать базу знаний в присутствии пользователя даже в тот момент, когда приложение исполняется. В ходе этой работы прототип развивается до такого состояния, что начинает удовлетворять представлениям конечного пользователя. В крупных приложениях команда разработчиков может разбить приложение на отдельные модули, которые интегрируются в единую базу знаний.

Возможен и альтернативный подход к созданию приложения. При этом подходе каждый разработчик имеет доступ к базе знаний, находящейся на сервере, при помощи Telewindows.

Тестирование приложения на наличие ошибок. В G2 ошибки в синтаксисе показываются непосредственно при вводе конструкций (структур данных и исполняемых утверждений) в базу данных; эти конструкции анализируются инкрементно. Могут быть введены только конструкции, не содержащие синтаксических ошибок. Таким образом, отпадает целая фаза отладки приложения (свойственная традиционному программированию), что ускоряет разработку приложений.

Разработчик освобожден и от необходимости знать детальный синтаксис языка G2, так как при вводе в базу знаний некоторой конструкции ему в виде подсказки сообщается перечень всех возможных синтаксически правильных продолжений.

Для выявления ошибок и неопределенностей реализована возможность "Inspect", позволяющая просматривать различные аспекты базы знаний, например "показать все утверждения со ссылками на неопределенные сущности (объекты, связи, атрибуты)", "показать графически иерархию заданного класса объектов".

Тестирование логики приложения и ограничений (по времени и памяти). Блок динамического моделирования позволяет при тестировании воссоздать различные ситуации, адекватные внешнему миру. Таким образом, логика приложения будет проверяться в тех условиях, для которых она создавалась. Конечный пользователь может принять непосредственное участие в тестировании благодаря управлению цветом (т.е. изменение цвета при наступлении заданного состояния или выполнения условия) и анимации (т.е. перемещение/вращение сущности при наступлении состояния/условия). Благодаря этому он сможет понять и оценить логику работы приложения, не анализируя правила и процедуры, а рассматривая графическое изображение управляемого процесса, технического сооружения и т.п.

Для проверки выполнения ограничений используется возможность "Meters", вычисляющая статистику по производительности и используемой памяти.

Полученное приложение полностью переносимо на различные платформы в среду UNIX (SUN, DEC, HP, IBM и т.д.), VMS (DEC VAX) и Windows NT (Intel, DEC Alpha). База знаний сохраняется в обычном ASCII-файле, который однозначно интерпретируется на любой из поддерживаемых платформ. Функциональные возможности и внешний вид приложения не претерпевают при этом никаких изменений. Приложение может работать как в "полной" (т.е. предназначенной для разработки) среде, так и под runtime, которая не позволяет модифицировать базу знаний.

Вопросы для самостоятельной проверки по темам

1. Разработка линейной нейронной сети в среде MatLab с помощью Neural Network ToolBox с заданным количеством слоев, нейронов.

2. Разработка нейронной сети Хопфилда в среде MatLab с помощью Neural Network ToolBox с заданным количеством, нейронов.

3. Разработка нейронной сети Кохонена в среде MatLab с помощью Neural Network ToolBox с заданным количеством, нейронов во входном и выходном слоях.

4. Выполнить, вручную, преобразования двух вещественных чисел в двоичный код, в код Грея.

5. Выполнить вручную, обратные преобразования из кода Грея в двоичный код, а затем в вещественные числа.

6. Построить, вручную, сетевой оператор для заданного математического выражения.

7. Построить, вручную, матрицу сетевого оператора для заданного математического выражения.

8. Выполнить несколько вариаций матрицы сетевого оператора и построить соответствующие векторы вариаций.

9. Построить математическое выражение по матрице сетевого оператора.

10. Создать экспертную систему в программной среде G2 и заполнить ее базу знаний по известной студенту конкретной теме, например по работе в пакете Word.

Описание системы контроля знаний

В курсе «Современные инструментальные средства интеллектуальных систем» предусматриваются цикл лекций, практические занятия (лабораторные работы) и курсовая работа.

В систему контроля знаний входит: контроль посещения лекций, контроль выполнения лабораторных работ, контроль поэтапного выполнения курсовой работы. Особо ценится своевременное выполнение лабораторных работ, качество выполнения курсовой работы, и итоговое испытание.

Промежуточная аттестация студентов проводится в конце каждого месяца, и результаты размещаются на учебном портале.

Правила выполнения письменных работ (курсовых, лабораторных):

Список тем курсовых работ предлагается студентам в начале учебного семестра. Студент вправе выбрать тему из данного списка или предложить свою (согласовав с преподавателем). Курсовая работа выполняется и сдается в срок, указанный в календарном плане.

Требования к оформлению работ: полуторный интервал, кегль — 13, цитирование и сноски в соответствии с принятыми стандартами, выверенность грамматики, орфографии, синтаксиса.

Курсовая работа должна содержать обзорную часть проблемы, иметь четкую постановку задачи, содержать теоретические исследования проблемы и материалы математического моделирования.

Текст отчета о лабораторной работе должен содержать краткую теоретическую и развернутую практическую части, с подробными комментариями ко всем этапам моделирования, объем не менее 4—6 страниц.

Балльная структура оценки:

Посещение лекций: 0- 20 баллов
Практические занятия (лабораторные работы): 0-55 баллов
Итоговое испытание: 0 - 25 баллов
Всего - 100 баллов

Курсовая работа: 0 - 100 баллов

Шкала оценок:

Баллы за семестр	Автоматическая оценка
91-100	5
76-90	4
56-75	3
35-55	-
<35	-

Студенты, получившие положительные оценки по результатам работы в семестре, но претендующие на получение более высокой оценки, могут участвовать в сдаче экзаменов в период сессии. Количество баллов за экзамен от 0 до 25 баллов.
Студенты, набравшие в течение семестра 35-55 баллов, обязаны пройти итоговую семестровую аттестацию в установленном порядке.
Студенты, не выполнившие программу изучаемой дисциплины и не набравшие 35 баллов, не допускаются до прохождения итоговой семестровой аттестации.

Темы курсовых работ

1. Разработка и исследование нейронной сети в среде MatLab для динамического выбора параметров регулятора в системе автоматического управления.

2. Разработка и исследование нейронной сети в среде MatLab для динамического выбора параметров корректирующего устройства в системе автоматического управления.

3. Разработка и исследование нейронной сети в среде MatLab для динамического выбора параметров фильтра система автоматического управления в условиях случайных воздействий.

4. Разработка и исследование нейронной сети в среде MatLab для динамического выбора эффективного контура управления в системе автоматического управления.

5. Разработка и исследование нейронной сети в среде MatLab для прогноза параметров возмущающего воздействия.

6. Разработка и исследование нейронной сети в среде MatLab для прогноза параметров функции распределения случайного сигнала.

7. Разработка и исследование нейронной сети в среде MatLab для восстановления ненаблюдаемых координат пространства состояний объекта.

8. Разработка и исследование нейронной сети в среде MatLab для выбора программы оптимального управления

9. Разработка и исследование нейронной сети в среде MatLab для выбора критерия качества в задаче оптимального управления динамическим объектом.

10. Разработка и исследование нейронной сети в среде MatLab для распознавания источника и сигнала внешнего воздействия в системе автоматического управления.

11. Разработка и исследование нейронной сети в среде MatLab для распознавания и прогноза траектории движения объекта управления.

12. Разработка и исследование нейронной сети в среде MatLab для распознавания изображения по визуальному сигналу.

13. Разработка и исследование динамической экспертной системы в среде G2 для принятия решения о выборе регулятора для системы управления.

14. Разработка и исследование динамической экспертной системы в среде G2 для принятия решения о выборе программы управления в системе автоматического управления.

15. Разработка и исследование динамической экспертной системы в среде G2 для выбора фильтра в системе автоматического управления.

16. Разработка и исследование динамической экспертной системы в среде G2 для принятия решения об изменении критерия качества системы управления из-за потери свойств управляемости.

17. Разработка и исследование динамической экспертной системы в среде G2 для мониторинга состояния функционирования объекта.

18. Разработка и исследование динамической экспертной системы в среде G2 для управления удаленным объектом.

19. Разработка и исследование динамической экспертной системы в среде G2 для определения стратегии противодействия.

20. Разработка и исследование динамической экспертной системы в среде G2 для контроля достижения цели в системе автоматического управления.

21. Разработка и исследование динамической экспертной системы в среде G2 для определения состояния объекта после частичной потери свойств из-за аварийной ситуации.

22. Разработка и исследование динамической экспертной системы в среде G2 для выбора стратегии управления в случае изменения модели объекта управления.

23. Разработка и исследование динамической экспертной системы в среде G2 для определения степени соответствия системы при управлении объектом в условиях неопределенности.

24. Разработка и исследование динамической экспертной системы в среде G2 для определения и предсказания чрезвычайных ситуаций при управлении динамическим объектом.

Темы дополнительных домашних заданий

1. Разработка нейронной сети для системы автоматического управления на языке программирования высокого уровня Delphi, C++, VBA, Java, Fortran и др.

2. Разработка динамической экспертной системы для работы в интеллектуальной системе автоматического управления на языке программирования высокого уровня Delphi, C++, VBA, Java, Fortran и др.

3. Разработка динамической экспертной системы на языке программирования Prolog.

4. Разработка динамической экспертной системы в среде CLIPS.

5. Разработка динамической экспертной системы в среде REFAL.

6. Разработка динамической экспертной системы с помощью реляционной базы данных.

7. Разработка сетевой динамической экспертной системы на основе технологии клиент/сервер с помощью распределенной базы данных.

8. Разработка практической экспертной системы для контроля знаний

Литература

1. *Балтрашевич В.Э.* Реализация инструментальной экспертной системы. – СПб.: Политехника, 1993.

2. *Гаврилова Т.А., Хорошевский В.Ф.* Базы знаний интеллектуальных систем. – СПб: Питер, 2000.

3. *Гладков Л.А., Курейчик В.В., Курейчик В.М.* Генетические алгоритмы. – М.: Физматлит, 2006.

4. *Головко В.А.* Нейронные сети: обучение организация и применение. – М.: ИПРЖР, 2001.

5. *Григорьев Л.И., Наталюк Ю.А.* Инструментальные средства G2 для построения экспертных систем. Компьютерный практикум. – М.: РГУ нефти и газа им. И.М. Губкина. – 1998. – 37с.

6. *Девятков В.В.* Системы искусственного интеллекта – М.: Изд-во МГТУ им. Н.Э.Баумана, 2001.

7. *Джонсон П.* Введение в экспертные системы. – М.: «Вильямс», 2001.

8. *Дивеев А.И., Софронова Е.А.* Основы генетического программирования: Учебно-методическое пособие. – М.: Изд-во РУДН, 2006.

9. *Дьяконов В.П.* Matlab 6.5 SP1/7 + Simulink 5/6. Основы применения. М.: СОЛОН-Пресс, 2005. – 800с

10. *Дьяконов В.П.* MATLAB 6: Учебный курс. СПб.: Питер. 2001.

11. *Потемкин В.Г.* Вычисления в среде MATLAB. М.: Диалог-МИФИ. 2004.

12. *Каллан Р.* Основные концепции нейронных сетей. – М.: «Вильямс», 2003.

13. *Комарцова Л.Г., Максимов А.В.* Нейрокомпьютеры. – М.: МГТУ им. Н.Э. Баумана, 2002.

14. *Люгер Дж. Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание: пер. с англ. М.: "Вильямс", 2003.

15. *Медведев В.В., Потемкин В.Г.* Нейронные сети. MATLAB 6. М.: Диалог-МИФИ, 2002.

16. *Нильсон Н.* Принципы искусственного интеллекта. – М.: Радио и связь, 1985.

17. *Попов Э.В. и др.* Статические и динамические экспертные системы. – М.: Финансы и статистика, 1996.

18. *Пупков К.А., Коньков В.Г.* Интеллектуальные системы. – М.: МГТУ им. Н.Э. Баумана, 2003.

19. *Рыбина Г.В.* Инструментальные средства нового поколения для построения прикладных интеллектуальных систем// Авиакосмическое приборостроение. – 2004. – № 10. – С. 14-23.

20. *Рыбина Г.В., Смирнов В.В., Шубинцева М.В.* Введение в инструментальную систему G2. – М.: МГУ, 1997.

21. *Уоссермен Ф.* Нейрокомпьютерная техника. – М.:Мир, 1992.

22. *Уотерман Д.* Руководство по экспертным системам. – М.:Мир, 1989.

23. *Demuth H., Beale M., Hagan M.* Neyral Network ToolBoxTM 6. User's Guid. 2008. MathWork, Inc.

24. Gensym Corp., G2 Reference Manual, Version 4.0. Cambridge (Mass. USA). – 1992.

Темы практических занятий

1. Построение в среде MatLab с помощью Neural Network ToolBox однослойной нейронной сети для прогнозирования статической функции.

2. Построение в среде MatLab многослойной нейронной сети для прогнозирования тренда случайной функции. Обучение сети с учителем. Изучение алгоритма обратного распространения ошибки. Изучение свойств функций активации.

3. Построение в среде MatLab системы автоматического регулирования с нейронной сетью. Анализ эффективности алгоритмов обучения.

4. Построение в среде MatLab ассоциативной нейронной сети Хопфилда для распознавания символьных и цифровых образцов.

5. Построение в среде MatLab нейронной сети Кохонена и ее применение в задачах классификации входных воздействий.

6. Моделирование в среде MatLab с помощью пакета Simulink нейронной сети в системе автоматического управления, работающей в условиях неопределенности.

7. Разработка экспертной системы в среде G2. Формирование базы знаний. Исследование алгоритмов логического вывода.

8. Разработка сетевой экспертной системы с помощью технологии клиент/сервер.

9. Разработка динамической экспертной системы для работы в реальном времени в системе автоматического управления динамическим объектом.

Перечень воросов итоговой аттестации

1. Назначение нейронных сетей. Общий алгоритм работы. Состав однослойной нейронной сети.

2. Многослойные нейронные. Функции активации. Перцептрон.

3. Обучение нейронной сети. Алгоритм Уидроу – Хоффа обучения однослойной нейронной сети.

4. Алгоритм обратного распространения ошибки.

5. Обучение нейронной сети с помощью методов математического программирования.

6. Алгоритм Хебба обучения нейронной сети.

7. Нейронные сети Хопфилда. Алгоритм работы. Область применения.

8. Нейронная сеть Кохонена. Алгоритм работы. Область применения.

9. Генетический алгоритм. Генетические операции. Кодирование решение в генетическом алгоритма. Настраиваемые параметры генетического алгоритма.

10. Генетическое программирование. Постфиксная, префиксная и инфиксная польские записи. Генетические операции с польскими записями. Алгоритм анализа польских записей.

11. Сетевой оператор. Графическое представление математических выражений с помощью сетевого оператора. Матрица сетевого оператора. Алгоритм вычислений значений выражения по матрице сетевого оператора.

12. Вариации сетевого оператора. Принцип базисной структуры в генетическом программировании.

13. Экспертные системы. Назначение. Составные элементы экспертной системы. Режимы работы экспертной системы.

14. Структуры данных для хранения знаний.

15. Алгоритм работы машины вывода в экспертной системе.

16. Семантические сети в экспертных системах.

17. Фреймовая модель знаний.

18. Продукционные экспертные системы. Системы с выводимой структурой.

19. Инструментальные средства для создания экспертных систем. Среда G2. Объектно-ориентированное проектирование экспертных систем

20. Динамические экспертные системы. Особенности проектирования и применения динамических экспертных систем.

21. Разработка и этапы проектирования баз знаний, представление знаний в реляционных базах данных.

22. Соотношение методов представления знаний в базах данных и интеллектуальных информационных системах. Системы управления базами данных и системы управления базами знаний.

23. Байесовский метод принятия решений в экспертных системах. Вероятностная логика.

24. Поиск решений в экспертной системе на основе имитационного моделирования процессов управления в интеллектуальных системах.

Перечень тем рефератов

1. Вероятностные нейронные сети и их реализация в пакете Matlab

2. Использование нейронных сетей для решения задач оптимизации

3. Обобщенные регрессионные нейронные сети и их реализация в пакете Matlab.

4. Точные радиально-базисные нейронные сети, их применение и реализация в пакете Matlab

5. Каскадные прямо-обратные нейронные сети, их применение и реализация в пакете Matlab

6. Решение задачи многомерной интерполяции с помощью метода генетического программирования.

7. Решение задачи идентификации динамической модели с помощью метода генетического программирования.

8. Анализ математических результатов, полученных в исследовании генетического алгоритма и методов адаптивного случайного поиска.

9. Исследование методов оценки эффективности генетических алгоритмов.

10. Исследование методов реализации логических выводов для базы знаний экспертных систем.

Дивеев Асхат Ибрагимович - доктор технических наук, профессор кафедры Кибернетики и мехатроники
Российского университета дружбы народов,
заведующий сектором проблем кибернетики вычислительного центра им. А.А. Дородницына Российской академии наук