А. Д. ОБУХОВ, И. Л. КОРОБОВА

СИСТЕМНЫЙ АНАЛИЗ И ОБРАБОТКА ИНФОРМАЦИИ В ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ

Тамбов
♦ Издательский центр ФГБОУ ВО «ТГТУ» ♦
2020

Министерство науки и высшего образования Российской Федерации

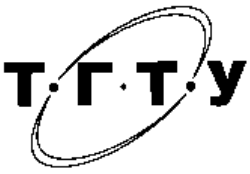
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тамбовский государственный технический университет»

А. Д. ОБУХОВ, И. Л. КОРОБОВА

СИСТЕМНЫЙ АНАЛИЗ И ОБРАБОТКА ИНФОРМАЦИИ В ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ

Утверждено Учёным советом университета
в качестве учебного пособия
для бакалавров направления 09.03.01
«Информатика и вычислительная техника»
и магистрантов направления 09.04.01
«Информатика и вычислительная техника»

Учебное электронное издание



Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2020

УДК 004.89(082)
ББК 32.813.5
О-26

Р е ц е н з е н т ы :

Кандидат педагогических наук, доцент,
доцент кафедры «Математическое моделирование и информационные технологии» ФГБОУ ВО «Тамбовский государственный университет имени Г.Р. Державина»
Е. В. Клыгина

Кандидат технических наук, доцент,
доцент кафедры «Информационные процессы и управление»
ФГБОУ ВО «Тамбовский государственный технический университет»
И. А. Дьяков

Обухов, А. Д.

О-26 Системный анализ и обработка информации в интеллектуальных системах [Электронный ресурс] : учебное пособие / А. Д. Обухов, И. Л. Коробова. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2020. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium II ; CD-ROM-дисковод ; 36,3 Mb ; RAM ; Windows 95/98/XP ; мышь. – Загл. с экрана.

ISBN 978-5-8265-2217-2

Рассмотрены общие сведения об интеллектуальных системах, подходы к представлению знаний, методы машинного обучения. Приводятся примеры реализации методов машинного обучения на языке Python.

Предназначено для бакалавров направления 09.03.01 «Информатика и вычислительная техника» и магистрантов направления 09.04.01 «Информатика и вычислительная техника».

УДК 004.89(082)
ББК 32.813.5

ISBN 978-5-8265-2217-2

ВВЕДЕНИЕ

Современные информационные системы за счет развития передовых технологий искусственного интеллекта позволяют перевести на качественно новый уровень автоматизации деятельность человека. Использование подобных интеллектуальных систем открывает новые возможности по обработке, хранению и передаче информации, организации взаимодействия с системой за счет адаптации системы под индивидуальные особенности каждого пользователя.

Однако разработка интеллектуальных систем отличается более высокой сложностью и требованиями к уровню квалификации разработчиков. Исследование, анализ и конкретизация процесса их проектирования имеет большое значения для современного общества, так как расширение рынка интеллектуальных систем позволит качественно повысить уровень цифровизации.

Реализация и внедрение интеллектуальных систем востребовано как для научно-образовательных организаций, так и промышленных предприятий, поэтому важной задачей является подготовка специалистов и разработчиков к процессу проектирования интеллектуальных систем, осуществлению корректного анализа и обработки информации в подобных системах.

Данное учебное пособие будет использоваться при подготовке студентов и магистров по следующим направлениям: 09.03.01 – Информатика и вычислительная техника (профиль «Модели, методы и программное обеспечение анализа проектных решений»), дисциплины: «Базы знаний», «Офисные технологии», «Системы подготовки документации»; 09.04.01 – Информатика и вычислительная техника (магистерская программа «Модели, методы и программное обеспечение анализа проектных решений»), дисциплины: «Интеллектуальные системы», «Вычислительные системы», «Методы организации информатизации промышленных систем», «Проектирование информационных систем предприятий».

1. ОСНОВНЫЕ СВЕДЕНИЯ О ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ

1.1. ОСНОВНЫЕ ТЕРМИНЫ

Основой эффективного проектирования интеллектуальных систем является применение и адаптация технологий искусственного интеллекта, использование экспертных знаний и обработка больших объёмов данных с целью нахождения в них зависимостей и закономерностей.

Другим аспектом разработки интеллектуальных систем является снижение нагрузки и влияния программиста на процесс реализации интеллектуальных систем. Технологии искусственного интеллекта позволяют освободить человека от выполнения рутинных задач, обработки огромных объёмов данных, поиска скрытых закономерностей в сложно анализируемых массивах информации.

Определим некоторые основные понятия, применяемые при реализации интеллектуальных систем.

Информация – сведения об объектах, явлениях и событиях, процессах окружающего мира, передаваемые различными способами (устно, письменно, графически), что позволяет уменьшить неопределенность знаний о них.

Система – совокупность элементов, объединенная связями между ними и обладающая определенной целостностью.

Информационная система – организационно упорядоченная совокупность информационных объектов и информационных технологий, в том числе и с использованием средств вычислительной техники и связи, реализующих информационные процессы.

Задачи, решаемые информационными системами, определяются предметной областью. Примерами предметных областей являются: промышленность, медицина, экономическая сфера, транспортная отрасль и т.д., т.е. под **предметной областью** понимается фрагмент реального мира.

Экспертные системы – разновидность информационных систем, используемая в процессе принятия решений и основанная на анализе сложно

формализуемых процессов, что приводит к необходимости использования человеческого опыта и знаний. В результате подобного рода системы имитируют работу экспертов в заданной предметной области с некоторой точностью.

Интеллектуальные системы – системы поддержки задач принятия решения, основанные на применении знаний и закономерностей сложно формализуемых процессов. В интеллектуальных системах предпочтительно отказываться от экспертных решений, предпочитая познавательные процедуры, что позволяет повысить их точность.

Искусственный интеллект (ИИ) – это признак или свойство информационных систем, характеризующееся возможностью системы выполнять функции интеллекта человека, выбирать оптимальные решения с учетом предыдущего опыта, анализа информации и окружающей среды.

Для оценки искусственного интеллекта математиком А. Тьюрингом предложен тест, основанный на определении экспертом разумности программы, т.е. соответствия её поведения такому у реального человека.

Интеллектуальная информационная система (ИИС) – это автоматизированная система, основанная на знаниях, включающая совокупность программных, логических, математических и лингвистических средств для осуществления процесса поддержки принятия решений в различных предметных областях.

Исследование интеллектуальных систем и искусственного интеллекта связано с привлечением специалистов различных областей: информатики, программирования, психологии, лингвистики и т.д.

ИИС используют при решении следующих задач:

– с неизвестным алгоритмом действий;

– с анализом не только численной, но и графической, текстовой, звуковой и т.п. информации;

– с необходимостью выбора из нескольких альтернатив на основе имеющихся данных, знаний, правил и т.д.

Тогда ИИС должны обладать следующей функциональностью:

– анализ имеющихся фактов;
– возможность делать из фактов выводы;
– проведение оценки собственной работы;
– возможность определения получен результат или нет;
– обобщение фактов, поиск закономерностей.

Таким образом, основными свойствами ИИС являются обучаемость, адаптивность, накопление опыта и знаний.

1.2. КЛАССИФИКАЦИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

На основе представленных выше свойств ИИС можно осуществить следующую их классификацию **по типу** (рис. 1.1) [1].

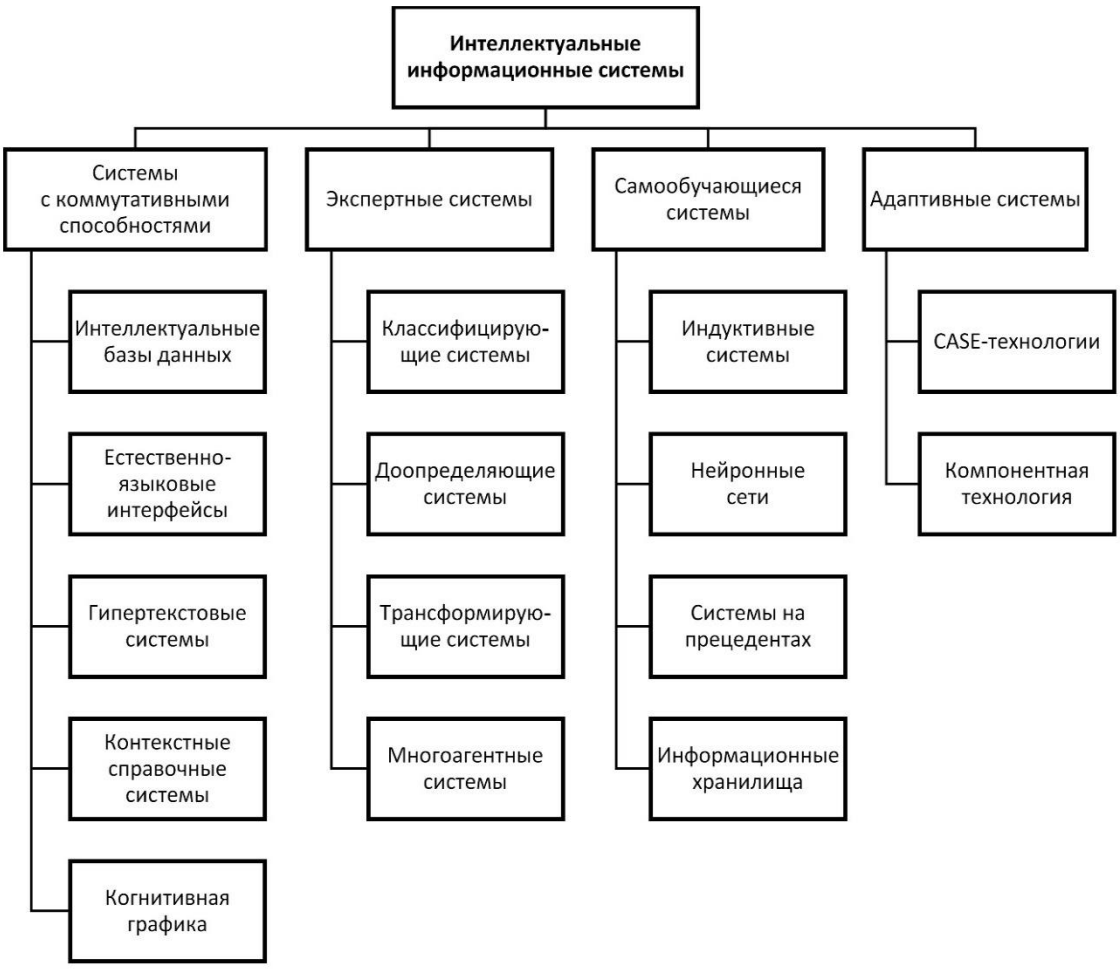


Рис. 1.1. Классификация интеллектуальных информационных систем по типам систем

Интеллектуальные базы данных позволяют не просто осуществить запрос информации из базы данных, но и осуществить её обработку и преобразование для решения конкретных задач.

Естественно-языковой интерфейс осуществляет трансляцию естественно-языковых конструкций на внутримашинный уровень представления знаний путем решения задач морфологического, синтаксического и семантического анализа и синтеза высказываний на естественном языке.

Гипертекстовые системы используются для поиска текстовой информации по ключевым словам.

Системы контекстной помощи являются развитием справочных систем, которые на основе диалогов с пользователем обеспечивают ему доступ к нужной информации.

Системы когнитивной графики формируют графические образы интерфейса в соответствии с происходящими событиями.

Экспертные системы используются для решения задач на основе собранной базы знаний, отражающей опыт работы экспертов в рассматриваемой предметной области. Разделяются на классифицирующие (определение классов объектов), доопределяющие (дополнение недостающих знаний), трансформирующие системы (генерация новых данных и знаний), многоагентные системы (самоорганизующиеся системы для поиска оптимальных решений без внешних вмешательств).

Самообучающиеся системы на основе реальных фактов или примеров формируют новые знания или осуществляют выбор оптимального решения.

Индуктивные системы используют обобщение примеров по принципу от частного к общему.

Нейронные сети являются программным и математическим представлением биологических нейронных сетей. Таким образом, это система соединенных и взаимодействующих между собой простых процессоров (искусственных нейронов). Нейронные сети активно применяются для классификации данных, их распознавания и получения новых знаний и закономерностей.

Системы на прецедентах основаны на хранении данных о конкретных ситуациях, а не об их обобщенном представлении.

Информационные хранилища – хранилище информации для оперативного анализа данных.

Адаптивная информационная система адаптирует свою структуру в соответствии с изменениями в предметной области.

По решаемым задачам ИИС можно распределить следующим образом: системы управления и справочные системы, системы компьютерной лингвистики, системы распознавания, игровые системы и системы создания интеллектуальных информационных систем (рис. 1.2).

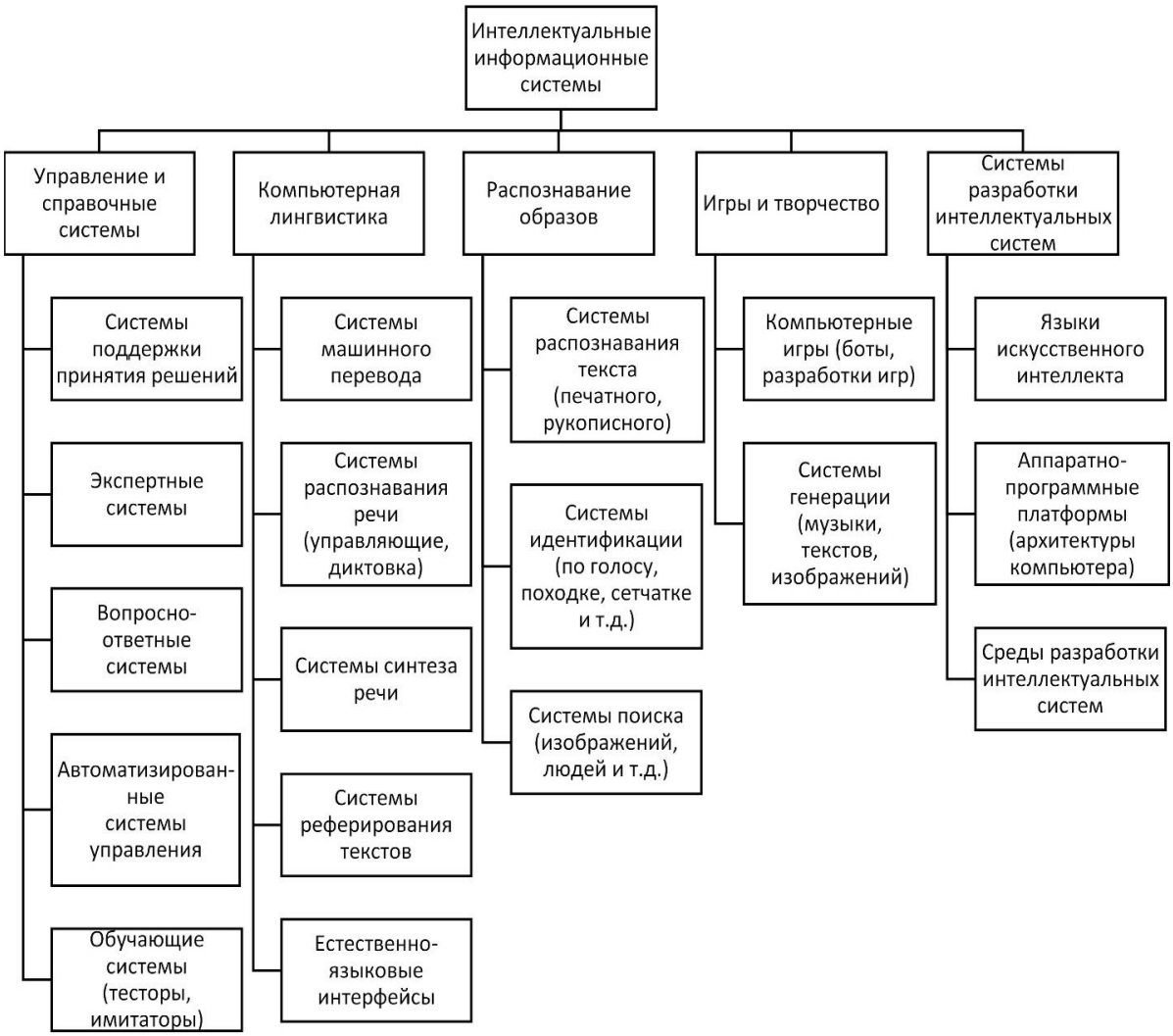


Рис. 1.2. Классификация интеллектуальных информационных систем по решаемым задачам

По используемым методам ИИС можно разделить на мягкие, жесткие и гибридные (рис. 1.3).

Мягкие вычисления – это сложная компьютерная методология, основанная на нечеткой логике, генетических вычислениях, нейровычислениях и вероятностных вычислениях. Жесткие вычисления – традиционные компьютерные вычисления. Гибридные системы – системы, использующие более чем одну компьютерную технологию.

Рис. 1.3. Классификация интеллектуальных информационных систем по методам

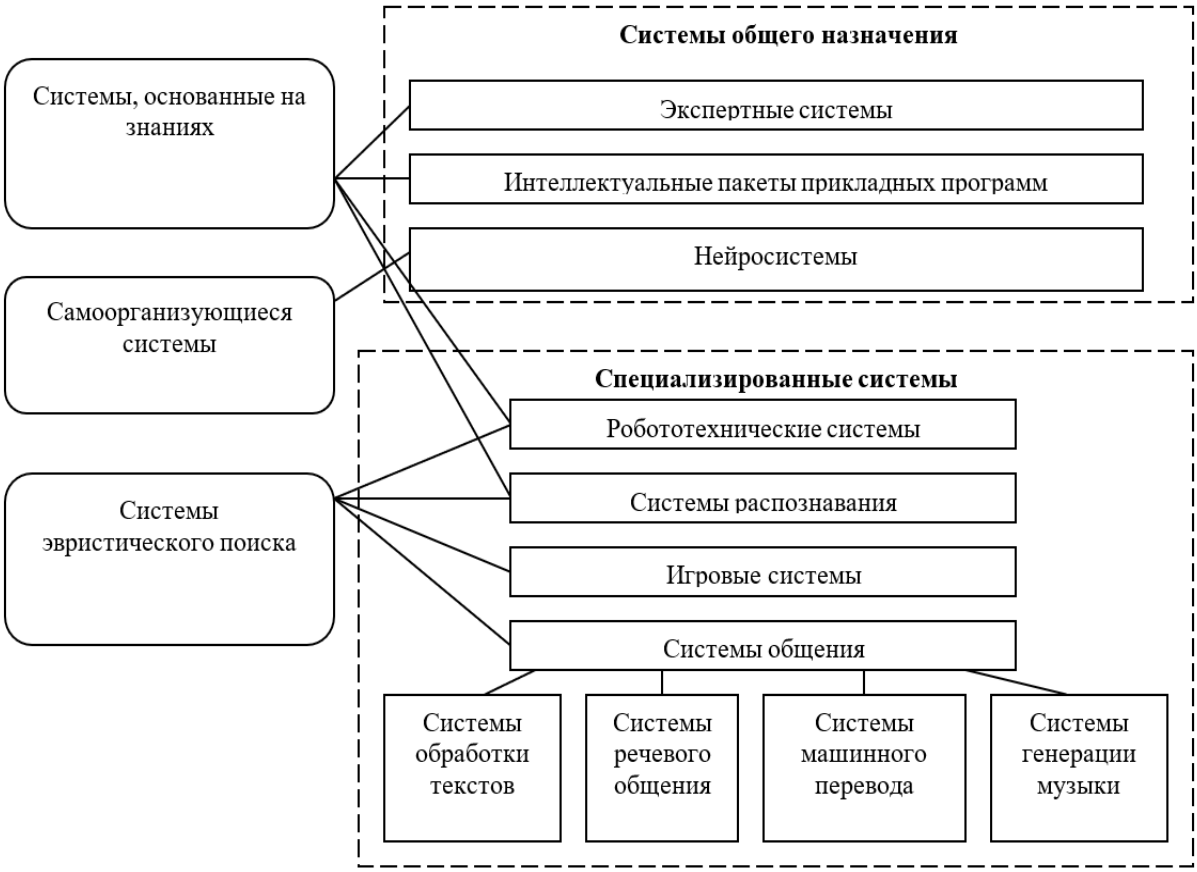


Рис. 1.4. Классификация интеллектуальных систем по назначению

Классификация по назначению ИИС представлена на рис. 1.4.

Таким образом, существует множеству подходов к классификации ИИС. В зависимости от того, какой признак является определяющим, выбирается соответствующая классификация.

1.3. ИСТОРИЯ РАЗВИТИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Искусственный интеллект и интеллектуальные системы зародились ещё во времена Древнего мира, например, в «Илиаде» Гомера упоминается бог Гефест, создающий человекоподобных существ, а в Древнем Египте была создана «оживающая» механическая статуя бога Амона.

В XIII веке средневековый испанский философ, математик и поэт Р. Луллий описывает процесс создания машины для ряда задач на основе

предопределённых правил и понятий, что можно считать предпосылками к реализации искусственного интеллекта.

Теория искусственного интеллекта берет свое начало с работ Декарта и Лейбница, разработавших такие понятия как теория игр, когнитивная психология, теория принятия решений, являющихся фундаментом для будущих систем искусственного интеллекта. Однако, само направление ИИ выделилось только в 1940-х годах и связано с появлением и распространением первых ЭВМ и появлением науки кибернетики.

Вопросами искусственного интеллекта и определением, может ли машина быть разумным, занимался английский ученый А. Тьюринг, разработавший специальный тест для определения, насколько точно и достоверно машина может имитировать поведение человека («тест Тьюринга»).

В 1956 году Д. Маккарти, М. Минский, К. Шеннон и Н. Рочестер организовали конференцию, где впервые прозвучал термин «искусственный интеллект».

Направление искусственного интеллекта интенсивно развивается благодаря развитию как теоретического аппарата, так и мощности ЭВМ. Алгоритмы ИИ применяют для разработки компьютерного интеллекта для игры в шахматы (А. Ньюэлл, Г. Саймон и Дж. Шоу), что привело к появлению нового языка программирования ИПЛ1. Полученные научные результаты применяются далеко за рамками решаемой задачи – в математике, при вычислении интегралов, доказательстве теорем и т.д. Следом появились и другие языки, ориентированные на ИИ, например, Lisp (Д. Маккарти), Пролог (Д. Робинсон).

Первые нейронные сети появились в конце 50-х годов. В 1957 году Ф. Розенблатт попытался создать персептрон – систему, моделирующую человеческий глаз и его взаимодействие с мозгом. В 1969 году в Вашингтоне состоялась первая международная конференция по искусственному интеллекту (IJCAI).

В середине 1970-х годов в США появились первые коммерческие системы, основанные на знаниях, – экспертные системы. С середины 1980-х годов происходит коммерциализация искусственного интеллекта, проявляется интерес к самообучающимся системам и методам представления знаний, растут ежегодные капиталовложения, создаются промышленные экспертные системы.

Первые реализации таких систем показали ошеломительный успех, позволяя экономить миллионы долларов. Однако, имеющаяся в то время вычислительная мощность значительно ограничивала создателей. Ситуация

изменилась с выходом нового поколения ЭВМ в 1990-х годах. Так, в 1997 году компьютер «Deep Blue» победил в игре в шахматы чемпиона мира Г. Каспарова, доказав, что в некоторых интеллектуальных задачах (пусть и в ограниченных условиях) искусственный интеллект может сравняться с человеком или даже превзойти его.

В СССР направление ИИ развивали такие ученые как А. И. Берг и Г. С. Поспелов, М. Л. Цетлин, В. Н. Пушкин, М. А. Гаврилов. К достижениям советских учёных относятся: разработка обратного вывода С. Маслова, реализация алгоритмов, моделирующих деятельность человеческого мозга при распознавании образов, появление направления ситуационного управления, разработка языка символьной обработки данных Refal.

Вопросы для закрепления

1. Дайте определение термину «информация».
2. Дайте определение термину «система».
3. Дайте определение термину «информационная система».
4. Дайте определение термину «экспертная система».
5. Дайте определение термину «искусственный интеллект».
6. Дайте определение термину «интеллектуальная система».
7. Как можно классифицировать по типу интеллектуальные системы?
8. Как классифицируются интеллектуальные системы по решаемым задачам?
9. Как классифицируются интеллектуальные системы по используемым методам?
10. Как классифицируются интеллектуальные системы по используемым методам?
11. Назовите основные этапы развития интеллектуальных систем.
12. Кто стоял у истоков развития современных интеллектуальных систем?

2. ПОДХОДЫ К ПРЕДСТАВЛЕНИЮ ИНФОРМАЦИИ В ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ

2.1. ЭКСПЕРТНЫЕ СИСТЕМЫ

Экспертные системы (ЭС, англ. expert system) – это информационные системы, используемые для замены человека-эксперта в процессе принятия решений. Современные экспертные системы начали разрабатываться исследователями искусственного интеллекта в 1970-х годах, а в 1980-х получили коммерческое подкрепление.

Понятия «экспертные системы» и «базы знаний» в информатике неразделимы. Под первым понятием обычно подразумевают модель поведения человека в выбранной предметной области, в состав которой входят функции логического вывода, поддержки принятия решений. Под вторым – объединение существующих фактов и правил их вывода в рассматриваемой области.

Схожие действия выполняет такой программный инструмент как «Мастер» (англ. Wizard), который, в отличие от экспертных систем, не содержит базы знаний – все действия строго запрограммированы. По сути, это просто набор форм для заполнения пользователем. Мастера применяются как в системных программах, так и в прикладных для упрощения интерактивного общения с пользователем.

Поисковые или справочные (энциклопедические) системы, похожие на Мастер, предоставляют наиболее подходящие (релевантные) разделы базы статей (представления об объектах областей знаний, их виртуальную модель) в соответствии с запросом пользователя.

В 1970 – 1980-е годы прошлого века экспертные системы ориентировались преимущественно на общепринятый в те годы текстовый человеко-машинный интерфейс. В настоящее же время такая «классическая» концепция переживает серьёзный кризис, так как в пользовательских приложениях преобладает графический интерфейс (GUI). Рост мощности вычислительного оборудования, появление сенсорных экранов и виртуальной реальности значительно снизил востребованность классического текстового вывода, несмотря на то что принципы работы экспертных систем одинаковы во всех случаях, меняется только способ отображения результата.

Еще одним минусом классического подхода, разработанного в XX веке, является их плохая интеграция с реляционными СУБД. Основы экспертных систем, заложенные в 1980-х годах, а также примеры их реализации

достаточно сложно интерпретировать в современных условиях ввиду их устаревания. Однако, необходимо понимать, что многие из существующих информационных систем, позиционируемые как экспертные, могут таковыми на самом деле не являться, оставаясь только справочными, но в силу маркетинговых соображений им придается большая значимость или интеллектуальность.

В настоящее время экспертные системы в различных отраслях набирают все большую популярность и могут составить конкуренцию юристам, экономистам, hr-менеджерам, врачам и другим специалистам. Фактически, экспертная система – это симуляция действий эксперта при решении определенной задачи.

Основные характеристики экспертных систем:

– ядро, которое представлено базой знаний;
– накопление и организация знаний;
– формализованный высококачественный опыт;
– возможности к прогнозированию.

Врач, диагностирующий заболевания и назначающий курс лечения, делает это хорошо только при хорошем специализированном образовании и накопленном опыте в медицине. Поэтому качество экспертной системы сводится к качеству формализованных знаний и унификации используемого опыта.

В настоящий момент недостаточно просто принимать эффективные решения, крайне важна скорость их принятия. Экспертная система способна обрабатывать огромный объём знаний за доли секунд, что порой может спасти жизнь человека либо компании.

Однако база знаний экспертной системы довольно ограничена, тогда как может использовать больший спектр органов чувств, символьную, графическую и другие виды информации, т.е. у экспертных систем существуют границы возможностей. Сейчас результаты работы этих системы ведут себя ненадежно на границах применимости или в нестандартных ситуациях. Однако предпринимаются попытки создать экспертные системы со способностью к обучению и к аргументации методов принятия решения.

Этапы разработки экспертных систем:

1. Идентификация области применения и круга решаемых задач.

2. Получение знаний.

3. Содержательный анализ проблемной области, определение методов решения задач.

4. Формализация – перевод в формализованный язык, код;

5. Реализация – создание прототипа системы.

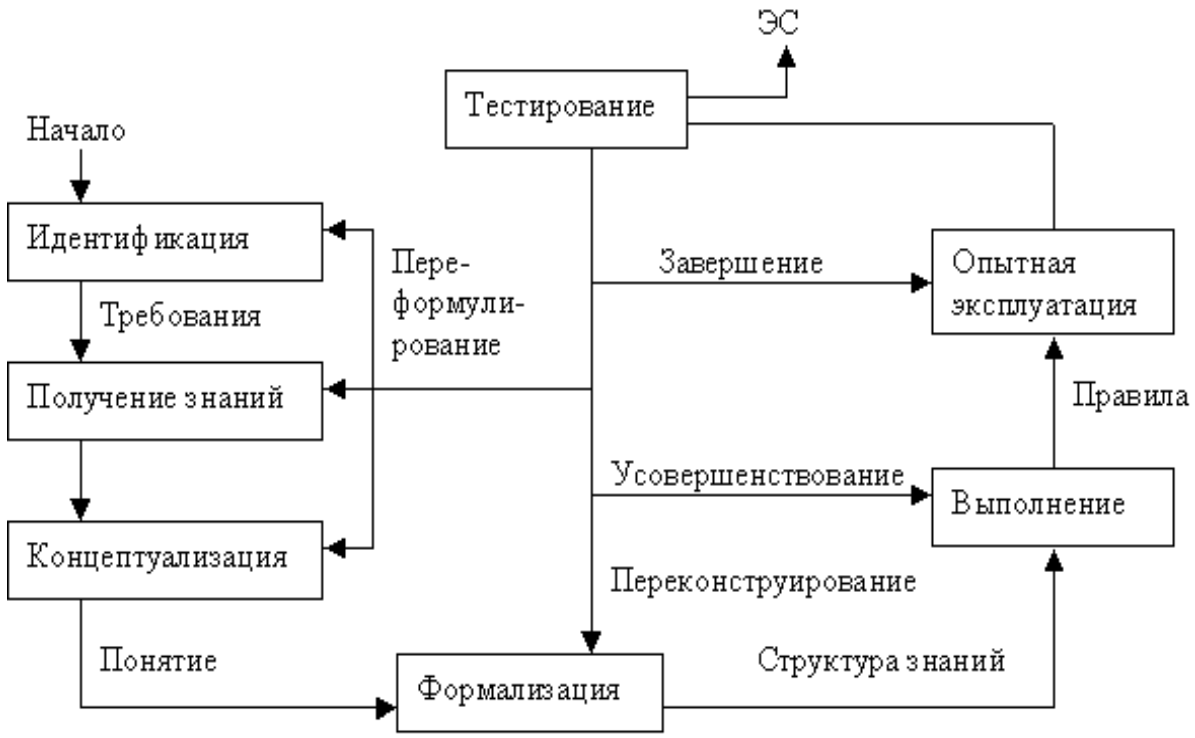


Рис. 2.1. Схема процесса реализации экспертных систем

В графическом виде это можно отобразить следующей схемой (рис 2.1).

В заключение стоит заметить, что экспертные системы уже эффективно применяются во многих отраслях, в настоящий момент многие организации в мире занимаются разработкой, тестированием и внедрением аналогичных систем в более сложные области нашей профессиональной жизни.

2.2. БАЗЫ ЗНАНИЙ

Ранее упоминалось, что базы знаний тесто связаны с экспертными системами. Можно дать следующее определение: базы знаний – это некоторая форма базы данных, используемая для работы со знаниями и фактами. Обязательными требованиями к базам знаний являются строгая структура хранения информации, ограничение определенной предметной областью, возможность логического вывода фактов по системе правил, корректность и полнота хранимых данных, их качество и актуальность.

Можно выделить следующие виды знаний – факторы, закономерности и алгоритмы.

Базы знаний можно разделять по масштабу и сложности систем, в которых их применяют, начиная от простых (для узких специалистов и небольших экспертных систем) и заканчивая глобальными, функционирующими в рамках всего Интернета.

2.3. СЕМАНТИЧЕСКИЕ СЕТИ

Семантическая сеть – это информационная модель предметной области, представленная в виде ориентированного графа. Вершинам графа соответствуют объекты предметной области, а дуги (рёбра) задают отношения между ними. Можно выделить следующие отношения:

- класс – подкласс;
- свойство – значение;
- пример элемента класса.

По количеству типов отношений выделяют однородные и неоднородные семантические сети. Однородные имеют один тип отношения между всеми понятиями, следовательно, тогда как неоднородные содержат множество типов отношений.

Все типы отношений:

- часть – целое;
- класс – подкласс;
- элемент – количество;
- атрибутивный;
- логический;
- лингвистический.

Пример семантической сети представлен на рис. 2.2. Можно отметить, что извлекать данные из семантической сети сложнее, чем из базы данных, так как этот процесс связан с анализом графовых структур.

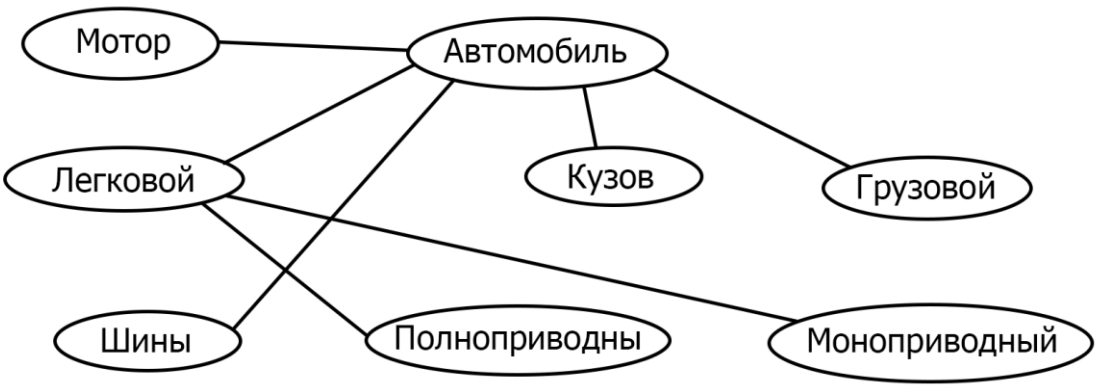


Рис. 2.2. Пример семантической сети

2.4. ФРЕЙМЫ

Фрейм – это некоторый образ, представляющий объект выбранной области с помощью слотов (атрибутов). Слот имеет имя, значение, тип хранимых данных, демон. Демон – процедура, выполняющаяся при определенных условиях. Имя фрейма должно быть уникальным в пределах одной фреймовой модели. Имя слота должно быть уникальным в пределах одного фрейма.

Отметим, что фреймы могут быть вложенными, то есть фрейм может содержаться в слоте. Пример фрейма представлен на рис. 2.3.

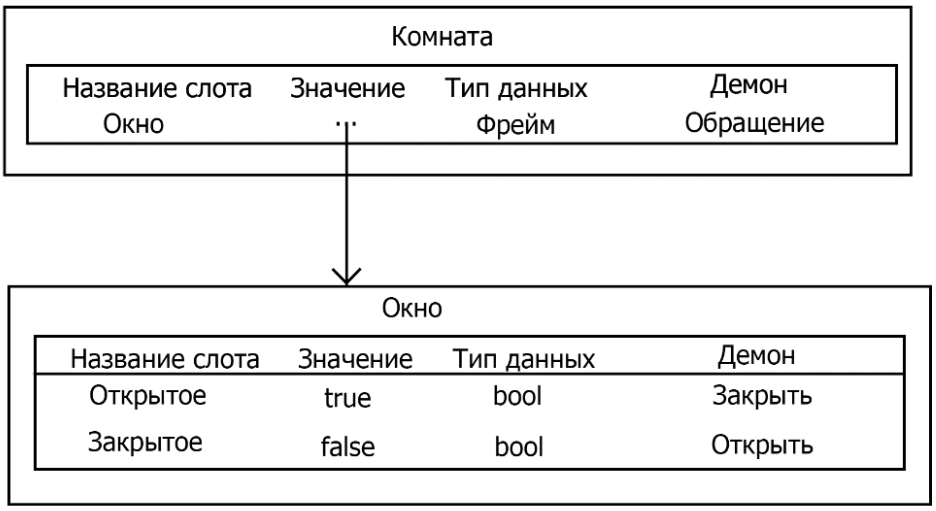


Рис. 2.3. Пример фрейма

2.5. ПРОДУКЦИОННЫЕ И ЛОГИЧЕСКИЕ МОДЕЛИ

В основе продукционной модели представления знаний находится конструктивная часть, продукция (правило):

IF <условие>, THEN <действие>.

Продукция состоит из двух частей: условие – антецедент, действие – консеквент. Условия можно сочетать с помощью логических функций AND, OR. Антецеденты и консеквенты составленных правил формируются из атрибутов и значений. *Пример*: IF температура реактора поднимается THEN добавить стержни в реактор.

В базе данных продукционной системы хранятся правила, истинность которых установлена заранее при решении определенной задачи. Правило срабатывает, если при сопоставлении фактов, содержащихся в базе данных с антецедентом правила, которое подвергается проверке, имеет место совпадение. Результат работы правила заносится в базу данных.

Пример продукции: IF Температура = 39 AND Кашель = Есть AND Давление = 110-130 THEN Бронхит

2.6. СИСТЕМНЫЙ АНАЛИЗ ДАННЫХ ДЛЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Таким образом, из обзора рассмотренных способов представления данных можно сделать вывод, что именно анализ и обработка данных являются ключевым моментом при реализации интеллектуальных систем. Без тщательной проработки данного этапа получить систему, адекватно осуществляющую поддержку принятия решений невозможно.

Исходные данные могут быть получены из разных мест, иметь различные форматы и структуру, более того, часть из них может иметь ошибки и неточности, быть искажена. Наиболее распространенными проблемами для данных является их неполнота (часть значений или атрибутов отсутствует), зашумленность (ошибочность некоторых значений), несогласованность (конфликты и противоречия между данными).

Поэтому качество данных является обязательным условием для реализации интеллектуальных систем. Для исключения искажения данных, их зашумленности и некорректности необходимо осуществлять процедуру системного анализа.

На первом этапе необходимо комплексно оценить данные. Для этого нужно понимать, что именно будет анализироваться. Интерес представляют объём записей, количество атрибутов, типы данных каждого атрибута, количество пустых значений, правильность формата данных.

Необходимо обращать внимание на специфику хранения данных: если они хранятся в формате CSV/TSV, то требуется проверить везде ли правильно расставлены разделители строк и столбцов, для XML убедиться в правильности синтаксиса и т.д.

Необходимо осуществить проверку соответствия значений атрибутов заданному диапазону, является положительным или целым числом для тех атрибутов, где это требуется.

При обнаружении проблем необходимо осуществить предварительную обработку данных, которая включает следующие операции:

Очистка данных путем исключения неправильных, некорректных данных.

Преобразование данных, например, нормализация, округление до целого.

Уплотнение данных: сокращение информации за счет выбора определённого подмножества.

Дискретизация данных: переход от непрерывных значений к категориальным, преобразование значений к разреженным матрицам.

Очистка текста: удаление лишних символов, пробелов, знаков, либо добавление необходимых разделителей, упрощающих анализ текста.

Более подробно остановимся на проблеме пропущенных значений атрибутов. Во-первых, такие записи можно исключить из области анализа целиком. Во-вторых, можно подставить вместо них нулевые или заданные значения, если это возможно в условиях задачи. Следующий способ – использовать средние значения соседних записей, либо наиболее часто встречающее значение. Наконец, можно предсказывать значения на основе регрессии по значениям остальных атрибутов.

Процесс анализа и обработки данных зачастую включает этап нормализации данных, т.е. их масштабирования в заданном диапазоне. Для нормализации могут использоваться различные подходы, например, минимакс, Z-показатель, десятичное масштабирование и другие.

Если объём анализируемых данных слишком велик, алгоритмы машинного обучения могут работать медленно или некорректно. В таком случае требуется уменьшение размерности либо путем сокращения количества данных, либо с помощью отбрасывания менее полезных и важных атрибутов.

Подводя итог, можно сформулировать процесс системного анализа данных в интеллектуальных системах следующим образом.

На этапе подготовки данных необходимо определить структуру извлекаемых данных или фактов, привести их к виду, пригодному для последующего использования в алгоритмах машинного обучения. В подготовку данных входят следующие процессы:

1) очистка данных от аномальных, неправильных значений, устранение несогласованности и зашумленности в информации;

2) выбор данных, используемых в процессе анализа, что приводит к разделению информации на актуальную/полезную и устаревшую/ненужную;

3) предварительная обработка данных является продолжением процесса очистки и используется для дополнения отсутствующих данных, их округления, уточнения;

4) преобразование данных используется для подготовки собранных данных, изменения их формата, типа, структуры, осуществления операций группировки, нормализации и т.д.

Обработанные данные анализируются. В ходе данного этапа определяется размер данных, подаваемых на вход алгоритмов машинного обучения, их типы, размерность. Если это требуется условием задачи, то на данном этапе осуществляется переход от численных значений к разреженным матрицам, где 1 соответствует наличие некоторого признака, а 0 – его отсутствие. Если работа осуществляется с символьными данными, то зачастую используют различные подходы к преобразованию данных к численному типу с помощью словарей, хэш-таблиц, таблиц соответствия и других способов.

Обработанные данные, как правило, разделяют на два подмножества (выборки). Основное, составляющее 75 – 90% от общего объёма, называется тренировочным. Именно на этих данных обучаются алгоритмы машинного обучения, т.е. осуществляют настройку своих внутренних параметров, чтобы обеспечить соответствия заданного входа и выхода. Вторая часть (10 – 25%) называется тестовой и используется для проверки работоспособности и точности работы алгоритма машинного обучения. Объективность этой оценки обеспечивается тем, что элементы тестовой выборки не входят в тренировочную. Иногда добавляется ещё одно подмножество данных – контрольное, на котором дополнительно проверяется работоспособность программы.

Вопросы для закрепления

1. Назовите основные этапы развития экспертных систем.
2. Назовите основные характеристики экспертных систем.
3. Перечислите этапы разработки экспертных систем.
4. Дайте определению понятию «базы знаний».
5. Дайте определению понятию «семантические сети».
6. Дайте определению понятию «фрейм».
7. Дайте определению понятию «продукционная модель».
8. Опишите процедуру анализа и обработки данных в интеллектуальных системах.

3. МЕТОДЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Все задачи, решаемые с помощью машинного обучения, относятся к одной из следующих категорий.

1) Регрессия – прогноз на основе выборки объектов с различными признаками. На основе входных данных строится предположение о выходном значении, выраженном вещественным числом. Примером может быть предсказание цены предмета по его характеристикам.

2) Классификация – определение класса (категории) объекта по значениям его признаков. Обычно выражается в вещественном значении от 0 (не является представителем класса) до 1 (совершенно точно является). Используется для распознавания цифр, объектов, определения наличия или отсутствия объектов на изображениях, определения тональности или тематики текстов и т.д.

3) Кластеризация – разделение данных на группы по их признакам, без заранее определённого перечня этих групп. При таком подходе искусственный интеллект может выявить закономерности группировки объектов по ранее незамеченным признакам.

4) Уменьшение размерности – переход от большого числа признаков к меньшему для удобства их последующей визуализации и упрощения анализа данных.

5) Выявление аномалий – отделение аномалий от стандартных случаев. В отличии от задачи классификации направлено на выявление редких, аномальных случаев в анализируемых данных, количество которых в обучающей выборке может быть мало либо просто отсутствовать.

Далее рассмотрим основные виды машинного обучения и используемые алгоритмы для их решения [6].

3.1. АЛГОРИТМЫ МАШИННОГО ОБУЧЕНИЯ С УЧИТЕЛЕМ

Метод Байеса (Naive Bayes, NB) относится к вероятностным методам классификации. Преимущества метода состоят в следующем: высокая скорость работы, поддержка инкрементного обучения, простая реализация алгоритма в виде программы, легкая интерпретируемость результатов работы алгоритма. Несмотря на приведенные достоинства, метод Байеса имеет так же и минусы в своей реализации. Относительно низкое качество классификации и неспособность учитывать зависимость результата классификации от сочетания признаков являются главными недостатками этого метода.

Метод k ближайших соседей (k Nearest Neighbors, KNN) относится к метрическим методам и считается простейшим классификатором. Объект присваивается тому классу, который является наиболее распространенным среди соседей данного элемента. Достоинства данного метода: простая реализация, проработанная теоретическая база, адаптация под нужную задачу выбором метрики или ядра, интерпретируемость. К недостаткам относятся: недостаточная производительность в реальных задачах, так как число соседей, используемых для классификации, будет достаточно большим; трудность в наборе подходящих весов и определением, какие признаки необходимы для классификации; зависимость от выбранной метрики расстояния между примерами.

Метод ближайшего соседа использует следующий подход к классификации: если сходный по признакам ближайший объект выборки (сосед) относится к определенному классу, то и исходный объект скорее всего относится к тому же классу.

Метод к-ближайших соседей основан на предположении, что если анализировать не одного соседа, а некоторое их количество (к), то можно повысить точность классификации. Если у каждого соседа будет коэффициент (вес), определяющий его вклад (ценность), то метод называется к-взвешенных ближайших соседей.

Метод опорных векторов (Support Vector Machine, SVM) является линейным методом классификации. Потенциальные недостатки метода опорных векторов заключаются в следующем: невозможность калибровки вероятности попадания в определенный класс, подходит только для решения задач с 2 классами, параметры модели сложно интерпретировать.

Метод деревьев решений (Decision Trees, DT) относится к логическим методам классификации. Деревом решений называют граф, по которому производится классификация объектов, описанных набором признаков. Каждый узел дерева содержит условие ветвления по одному из признаков. У каждого узла столько ветвлений, сколько значений имеет выбранный признак. Главным преимуществом метода является высокая производительность обучения и прогнозирования, такие деревья решений можно легко визуализировать и интерпретировать.

3.2. АЛГОРИТМЫ МАШИННОГО ОБУЧЕНИЯ БЕЗ УЧИТЕЛЯ

В обучении без учителя (unsupervised learning) у модели есть набор данных, и нет явных указаний, что с ним делать. ИИ извлекает полезные признаки и анализирует их, не учитывая остальные признаки. К данному

направлению относятся задачи кластеризации и обнаружения аномалий, рассмотренные ранее, а также следующие:

- **Ассоциации.** На основе нескольких признаков объекта выбираются другие объекты, с которыми у него может быть связь. Так работают рекомендательные алгоритмы интернет-магазинов.
- **Автоэнкодеры.** На основе входных данных формируется некоторое кодированное представление, из которого затем воссоздаются исходные данные. Таким образом можно генерировать картины, убирать шум с видео и изображений.

Большой проблемой данного направления является сложность определения точности полученной модели, правильности результата, так как нет набора размеченных данных с известными выходными значениями. С другой стороны, данные алгоритмы позволяют получать интересные, полезные результаты в тех областях, где алгоритмы обучения с учителем бессильны.

3.3. НЕЙРОННЫЕ СЕТИ

В последние годы направление машинного обучения – нейронные сети – активно развивается. В основе данного направления лежит предположение, что для решения нетривиальных и творческих задач можно сымитировать структуру человеческого мозга и механизм нейронных связей, что позволит машине решать задачу также эффективно, как это мог бы сделать человек. Нейронные сети успешно применяются для решения следующих задач: классификация, предсказание, распознавание, генерация новых данных.

Рассмотрим основные составляющие нейронных сетей [7].

Нейрон – это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает её дальше. Они делятся на три основных типа: входной (синий), скрытый (красный) и выходной (зеленый) (рис. 3.1).

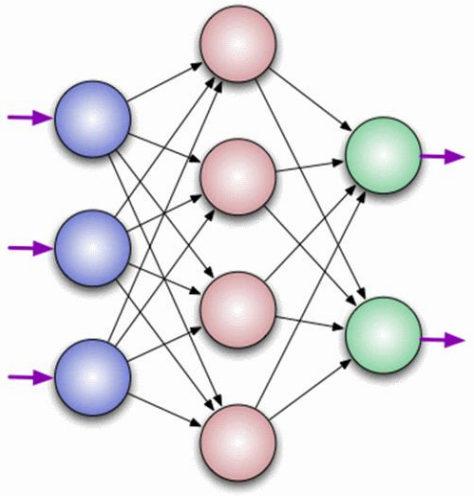


Рис. 3.1. Виды нейронов

Однако, помимо трех основных типов, существует множество других, используемых для решения специфических задач. Множество нейронов объединяются в понятие «слой».

На вход каждого нейронного приходят входные данные, на выходе получаем

выходные. У входных нейронов выход равен входу. В остальных нейронов на вход попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации и передается на выход нейрона.

Нейроны работают с числами в диапазоне от 0 до 1 или от –1 до 1. Однако, при решении задач мы обработаем широкие диапазоны чисел. Для этого используется понятие нормализация – 1 делится на некоторое число, позволяя получить любые значения.

Общая структура нейрона представлена на рис. 3.2.

Синапс – это связь между двумя нейронами. У синапсов есть 1 параметр – вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне. Совокупность весов нейронной сети или матрица весов – это своеобразный мозг всей системы. В начале работы нейронной сети веса распределяются случайным образом.

Нейронная сеть работает следующим образом. Пусть есть значения I входных нейронов и каждому из них соответствует вес w_i. Тогда на следующий слой на каждый нейрон будет поступать следующая информация $w_1*I_1+w_2*I_2$. Рассчитав это выражение, мы передам его в функцию активации. Она преобразует данные и передаст их на выход нейрона. Далее ситуация повторится на следующем слое. Изначально веса заданы неправильно, но при тренировке сети они будут изменятся в сторону правильных и выходной результат будет стремится к нужному.

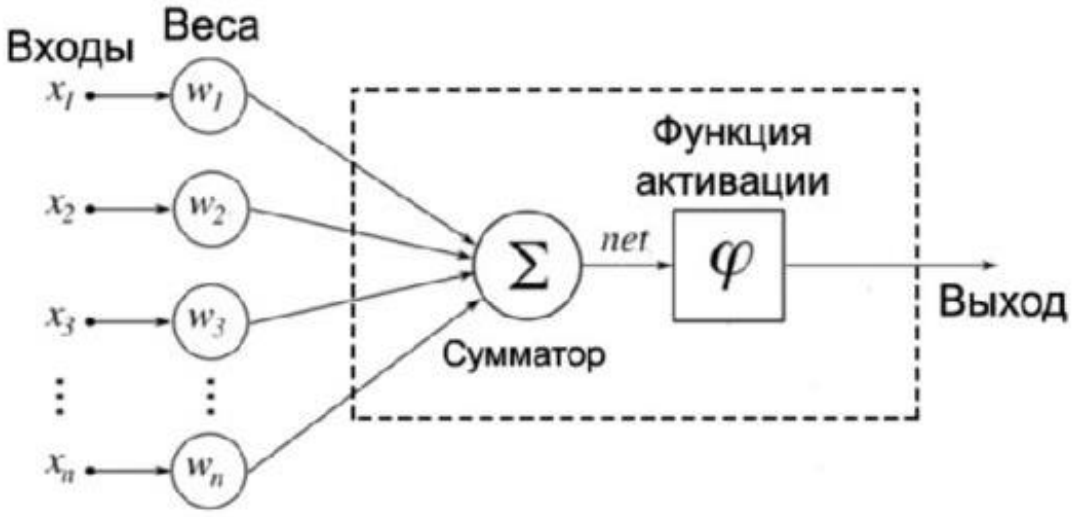


Рис. 3.2. Структура нейрона

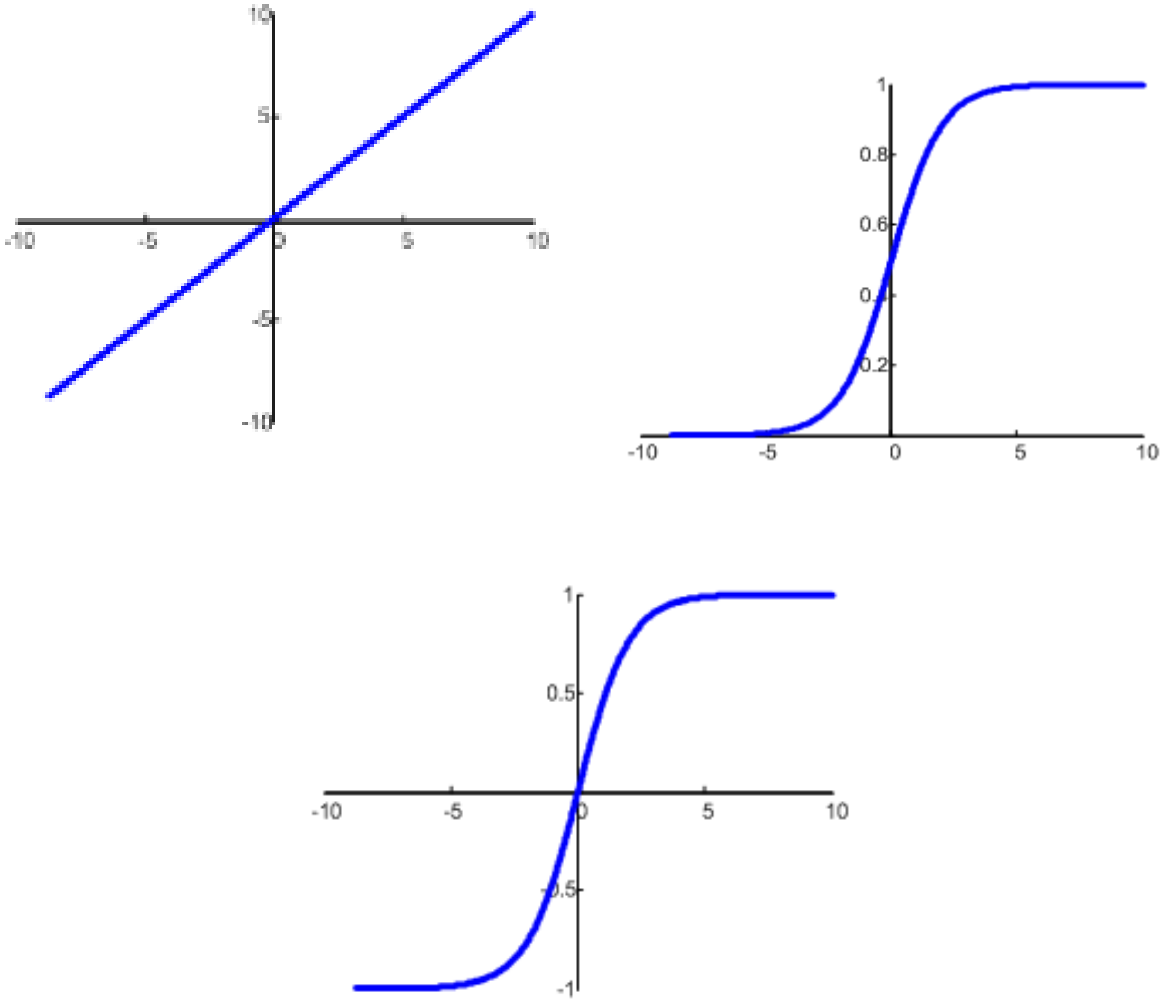


Рис. 3.3. Виды функций активации
(линейная, сигмоид, гиперболический тангенс)

Функция активации – это способ обработки данных, их нормализация. Проходя через функцию активации, числа приводятся к некоторому заданному диапазону, что облегчает их обработку. Функций активации достаточно много, рассмотрим основные: Линейная, Сигмоид (Логистическая) и Гиперболический тангенс (рис. 3.3). Главные их отличия – это диапазон возможных значений.

Линейная функция передает данные без преобразования.

Сигмоид преобразует значения в диапазон от 0 до 1, таким образом отбрасывая отрицательные значения.

Гиперболический тангенс может принимать и отрицательные, и положительные значения (от –1 до 1).

Далее перейдём к понятиям, относящимся к процессу обучения нейронных сетей.

Тренировочная выборка (сет) – это набор данных, которыми оперирует нейронная сеть.

Итерация – счетчик, который увеличивается каждый раз, когда нейронная сеть обучается на тренировочной выборке. Отражает общее количество тренировочных сетов.

Эпоха – количество проходов по всем тренировочным сетам, т.е. сначала мы последовательно проходим по всем данным, увеличивая количество итераций, когда данные закончились, мы начинаем заново, увеличивая количество эпох на 1.

Ошибка – это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка с каждой эпохой должна уменьшаться, иначе достигнуть приемлемого результата не получится. Ошибку можно вычислить разными путями, например, Mean Squared Error (MSE), Root MSE и Arctan. Каждый метод считает ошибку по-разному. Например, MSE считает среднее арифметическое суммы квадратов отклонений полученных значений от правильных.

Когда ошибка стала минимальной и удовлетворяющей поставленным требованиям, задача решена и обученную сеть можно использовать для определения нужных данных.

Нейронные сети в настоящее время активно развиваются. Появляются новые архитектуры и подходы к построению нейронных сетей. Примеры таких структур представлены на рисунке 3.4 ниже, каждый тип сети предназначен для решения определенного класса задач [8]:

- **Нейронные сети прямого распространения** (feed forward neural networks, **FF** или **FFNN**) и **перцептроны** (perceptrons, **P**) передают информацию от входа к выходу. FFNN обычно обучается по методу обратного распространения ошибки, в котором сеть получает множества входных и выходных данных. Этот процесс называется обучением с учителем.
- **Сети радиально-базисных функций** (radial basis function, **RBF**) – это FFNN, которая использует радиальные базисные функции как функции активации.
- **Нейронная сеть Хопфилда** (Hopfield network, **HN**) – это полносвязная нейронная сеть с симметричной матрицей связей. Во время получения входных данных каждый узел является входом, в процессе обучения он становится скрытым, а затем становится выходом. Сеть обучается так: значения нейронов устанавливаются в соответствии с желаемым шаблоном, после чего вычисляются веса, которые в дальнейшем не меняются. После того, как сеть обучилась на одном или нескольких шаблонах, она всегда будет сводиться к одному из них.

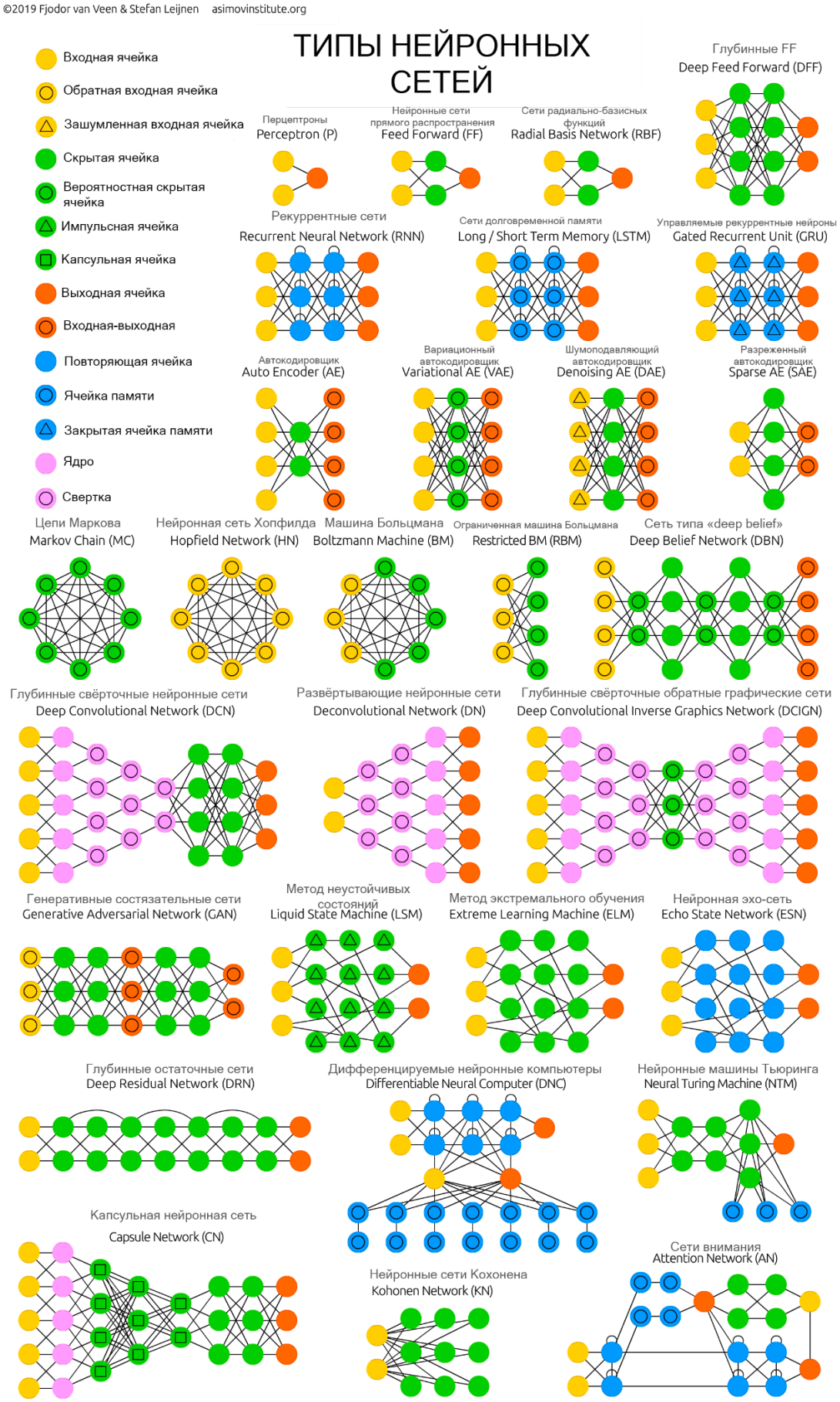


Рис. 3.4. Архитектуры нейронных сетей

- **Цепи Маркова** (Markov chains, **MC** или discrete time Markov Chains, **DTMC**) – оценивают шансы попасть в один из следующих узлов из текущей точки, таким образом, каждое следующее состояние зависит только от предыдущего.
- **Машина Больцмана** (Boltzmann machine, **BM**) основана на сети Хопфилда, но в ней некоторые нейроны помечены как входные, а некоторые – как скрытые. Входные нейроны в дальнейшем становятся выходными.
- **Ограниченная машина Больцмана** (restricted Boltzmann machine, **RBM**) отличаются от предыдущей тем, что нейроны одного типа не связаны между собой. Ограниченную машину Больцмана можно обучать как FFNN, но с одним нюансом: вместо прямой передачи данных и обратного распространения ошибки нужно передавать данные сперва в прямом направлении, затем в обратном. После этого проходит обучение по методу прямого и обратного распространения ошибки.
- **Автокодировщик** (autoencoder, **AE**) осуществляет автоматическое кодирование информации. Полученные на входе данные кодируются в более краткое представление в среднем слое и «восстанавливаются» в выходном слое.
- **Разреженный автокодировщик** (sparse autoencoder, **SAE**) – отличается тем, что вместо обучения сети отображать информацию в меньшем «объёме» узлов, мы увеличиваем их количество. В центральных слоях такая сеть расширяется. Сети такого типа полезны для работы с большим количеством мелких свойств набора данных.
- **Вариационные автокодировщики** (variational autoencoder, **VAE**) являются комбинацией машины Больцмана и автокодировщиков, функционируя на основе вероятностных выводов и расчёта влияния нейронов.
- **Шумоподавляющие автокодировщики** (denoising autoencoder, **DAE**) – это AE, в которые входные данные подаются в зашумленном состоянии. Благодаря этому сеть учится обращать внимание на более широкие свойства, поскольку маленькие могут изменяться вместе с шумом.
- **Сеть типа «deep belief»** (deep belief networks, **DBN**) – соединение нескольких RBM или VAE. Такие сети обучаются поблочно, причём каждому блоку требуется лишь уметь закодировать предыдущий. Такая техника называется “жадным обучением”, которая заключается в выборе локальных оптимальных решений, не гарантирующих оптимальный конечный результат.
- **Свёрточные нейронные сети** (convolutional neural networks, **CNN**) и **глубинные свёрточные нейронные сети** (deep convolutional neural

networks, **DCNN**) используются для обработки изображений, аудио, видео. Типичным способом применения CNN является классификация изображений.

- **Развёртывающие нейронные сети** (deconvolutional networks, **DN**), являются обратным к свёрточным нейронным сетям, на основе слов или последовательностей они генерируют изображения.
- **Капсульная нейронная сеть** (capsule neural network, **CN**) – архитектура искусственных нейронных сетей, которая предназначена для распознавания изображений.
- **Глубинные свёрточные обратные графические сети** (deep convolutional inverse graphics networks, **DCIGN**) моделируют свойства в виде вероятностей, поэтому их можно научить создавать картинку с собакой и кошкой, даже если сеть видела только картинки, на которых было только одно из животных. Возможно и удаление одного из двух объектов. Также были созданы сети, которые могли менять источник освещения и вращать объект. Сети такого типа обычно обучают методом обратного распространения ошибки.
- **Генеративные состязательные сети** (generative adversarial networks, **GAN**) состоят из любых двух сетей (обычно из FF и CNN), одна из которых контент генерирует, а другая – оценивает. Сеть-дискриминатор получает обучающие или созданные генератором данные. Степень угадывания дискриминатором источника данных в дальнейшем участвует в формировании ошибки. Таким образом, возникает состязание между генератором и дискриминатором, где первый учится обманывать первого, а второй – раскрывать обман. Обучать такие сети весьма тяжело, поскольку нужно не только обучить каждую из них, но и настроить баланс.
- **Рекуррентные нейронные сети** (recurrent neural networks, **RNN**) – это сети типа FFNN, но нейроны получают информацию не только от предыдущего слоя, но и от самих себя с предыдущего прохода. Обычно сети такого типа используются для автоматического дополнения информации.
- **Сети с долгой краткосрочной памятью** (long short term memory, **LSTM**) направлены на решение проблемы потери информации, используя фильтры и явно заданную клетку памяти. У каждого нейрона есть клетка памяти и три фильтра: входной, выходной и забывающий. Целью этих фильтров является защита информации. Забывающий фильтр определяет, какая информация с предыдущей итерации стоит «забыть».
- **Управляемые рекуррентные нейроны** (gated recurrent units, **GRU**) – небольшая вариация предыдущей сети, с двумя фильтрами. Фильтр

обновления определяет, сколько информации останется от прошлого состояния и сколько будет взято из предыдущего слоя. Фильтр сброса работает как забывающий фильтр.

- **Нейронные машины Тьюринга** (neural Turing machines, **NTM**) можно рассматривать как абстрактную модель LSTM и попытку показать, что на самом деле происходит внутри нейронной сети. Ячейка памяти не помещена в нейрон, а размещена отдельно с целью объединить эффективность обычного хранилища данных и мощь нейронной сети.
- **Дифференцируемые нейронные компьютеры** (Differentiable Neural Computers, **DNC**) – это усовершенствованные нейронные машины Тьюринга с масштабируемой памятью.
- **Двунаправленные RNN, LSTM и GRU** (bidirectional recurrent neural networks, bidirectional long / short term memory networks и bidirectional gated recurrent units, **BiRNN**, **BiLSTM** и **BiGRU**) вариации предыдущих сетей (не представлены на схеме). Данные сети используют не только данные из «прошлого», но и из «будущего».
- **Глубинные остаточные сети** (deep residual networks, **DRN**) – это очень глубокие сети типа FFNN с дополнительными связями между отделёнными друг от друга слоями. Такие сети можно обучать на шаблонах глубиной до сотен слоёв.
- **Нейронная эхо-сеть** (echo state networks, **ESN**) – разновидность рекуррентных сетей, отличающаяся отсутствием сформированных слоёв, т.е. связи между нейронами случайны.
- **Сети внимания** (attention networks, **AN**) можно рассматривать как класс сетей, основанный на механизме «внимания» для борьбы с потерей информации путем раздельного хранения предыдущих состояний сети и переключения внимания между состояниями. Скрытые состояния каждой итерации в слоях кодирования хранятся в ячейках памяти.
- **Метод экстремального обучения** (extreme learning machines, **ELM**) – сеть типа FFNN со случайными связями.
- **Метод неустойчивых состояний** (liquid state machines, **LSM**) основана на эхо-сети, у которой сигмоидная активация заменена пороговой функцией, а каждый нейрон является накопительной ячейкой памяти.
- **Метод опорных векторов** (support vector machines, **SVM**) находит оптимальные решения задачи оптимизации. Классическая версия способна категоризировать линейно разделяемые данные. В процессе обучения сеть как бы размещает все данные на 2D-графике и пытается разделить данные

прямой линией так, чтобы с каждой стороны были данные только одного класса.

- **Нейронные сети Кохонена** (Kohonen networks, KN) также известные как **самоорганизующиеся карты** (self organising (feature) maps, **SOM**, **SOFM**). Эти сети используют соревновательное обучение для классификации данных без учителя. Сети подаются входные данные, после чего сеть определяет, какие из нейронов максимально совпадают с ними. После этого эти нейроны изменяются для ещё большей точности совпадения, в процессе двигая за собой соседей.

Вопросы для закрепления

1. Перечислите основные задачи, решаемые с помощью машинного обучения.
2. Перечислите алгоритмы машинного обучения с учителем.
3. Перечислите алгоритмы машинного обучения без учителя.
4. Что такое искусственный нейрон?
5. Опишите принцип работы нейронной сети.
6. Что такое функция активации?
7. Что такое итерация и эпоха?
8. Что такое ошибка при обучении?
9. Какие типы нейронных сетей вы знаете? Для решения каких задач их используют?

4. МАШИННОЕ ОБУЧЕНИЕ НА PYTHON

4.1. ЗНАКОМСТВО С БИБЛИОТЕКОЙ SCIKIT-LEARN

Далее рассмотрим практические подходы к машинному обучению. В качестве языка программирования для решения задач машинного обучения будет использовать Python, включающий все необходимые библиотеки для сбора, анализа и обработки данных.

Одним из популярных инструментов машинного обучения является библиотека scikit-learn. Для своей работы она требует наличия ещё двух пакетов NumPy и SciPy. Полезными также могут оказаться пакеты matplotlib, IPython и Jupyter Notebook.

Кратко рассмотрим данные библиотеки.

NumPy – библиотека для научных и математических вычислений, позволяющая работать с многомерными массивами, высокоуровневыми математическими функциями. Массивы NumPy являются основой для хранения данных в scikit-learn (и не только в ней). Таким образом, данные хранятся в классе ndarray, многомерном массиве.

SciPy – это библиотека для научных вычислений в Python. В работе мы будем использовать такие возможности пакета, как создание разреженных матриц (sparse matrices), что позволяет снизить расход памяти и улучшить результат машинного машинного обучения.

matplotlib – это библиотека для построения графиков, включающая функции для создания визуализаций диаграмм, гистограмм, диаграмм разброса и т.д.

pandas – библиотека Python для обработки и анализа данных. В её основе применение структур данных DataFrame, похожие на таблицы Excel. Библиотека позволяет работать с многими форматами файлом, быстро импортировать данные из них, хранить в каждом столбце данные разных типов и т.д.

В качестве примера мы будем обучать модель классификатора на базе дерева решений [2].

Деревья решений для классификации и регрессии очень просты в использовании в Scikit-Learn. Сначала мы загрузим наш датасет (набор данных), который фактически встроен в библиотеку. Затем мы инициализируем наше дерево решений для классификации. Обучение модели выполняется одной строкой: ***fit(X, Y),*** где ***X*** – обучающая выборка в формате массива

NumPy, а ***Y*** – массив целевых значений, также в формате массива NumPy. Для загрузки данных используется следующий код:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
iris = load_iris()
```

Scikit-Learn также позволяет нам визуализировать наше дерево. Для этого есть несколько настраиваемых опций, которые помогут визуализировать узлы принятия решений и разбить изученную модель, что очень полезно для понимания того, как она работает. Ниже мы раскрасим узлы на основе имен признаков и отобразим информацию о классе и объектах каждого узла.

```
plt.figure(figsize=((20,13)))
clf = DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)
plot_tree(clf,
        filled=True,
        feature_names=iris.feature_names,
        class_names=iris.target_names,
        rounded=True)
plt.show()
```

Классификаторы библиотеки scikit-learn рассматриваются на задаче классификации ирисов Фишера-Андерсона, описание которых дано в виде набора из данных о 150 экземплярах ириса, по 50 экземпляров из трех следующих видов (рис. 4.1):

- ирис щетинистый (iris setosa);
- ирис версиколор (iris versicolor);
- ирис виргинский (iris virginica).

Рис. 4.1. Виды ирисов

Длина чашелистика	Ширина чашелистика	Длина лепестка	Ширина лепестка	Вид ириса
5.1	3.5	1.4	0.2	*setosa*
4.9	3.0	1.4	0.2	*setosa*
4.7	3.2	1.3	0.2	*setosa*
…	…	…	…	…
7.0	3.2	4.7	1.4	*versicolor*
6.4	3.2	4.5	1.5	*versicolor*
6.9	3.1	4.9	1.5	*versicolor*
…	…	…	…	…
6.3	3.3	6.0	2.5	*virginica*
5.8	2.7	5.1	1.9	*virginica*
7.1	3.0	5.9	2.1	*virginica*
…	…	…	…	…

Рис. 4.2. Примеры параметров из файла iris.csv

Для каждого экземпляра приведены 4 следующие характеристики:

- длина чашелистика – отдельной части чашечки цветка (sepal length);
- ширина чашелистика (sepal width);
- длина лепестка (petal length).
- ширина лепестка (petal width).

Значения параметров расположены в файле iris.csv (рис. 4.2).

На основании этого набора данных требуется построить правило классификации, определяющее класс растения. Один из классов (Iris setosa) линейно-разделим от двух остальных (рис. 4.3).

Рис. 4.3. Представление классов ирисов по трем характеристикам

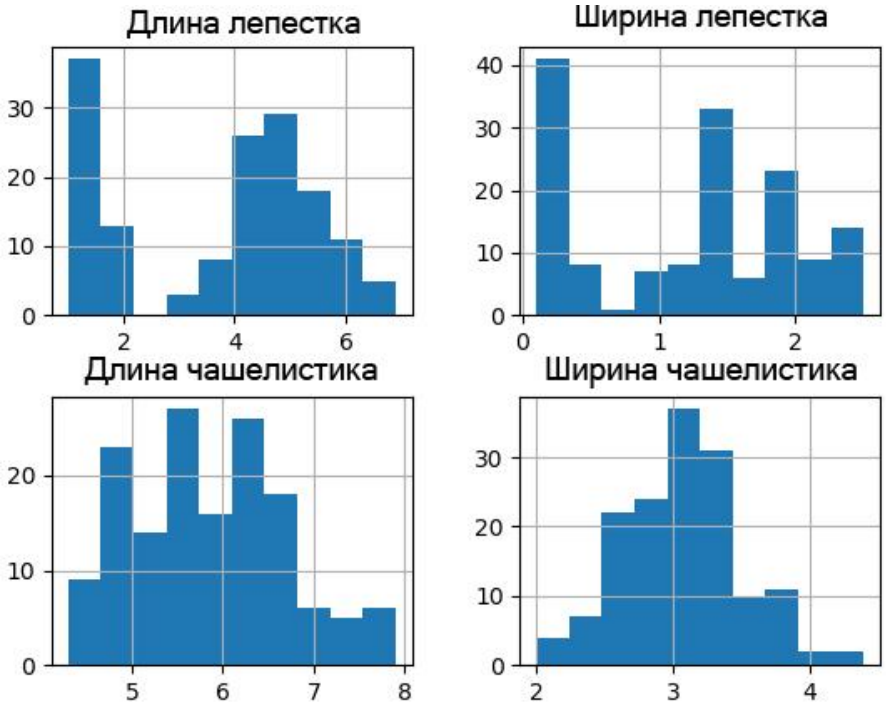


Рис. 4.4. Гистограммы характеристик ирисов (получены по файлу iris.csv)

Рисунок 4.3 будет выведен, если в приводимой далее программе задать groundTruth = True.

Первая строка файла iris.csv содержит заголовки столбцов, далее следуют строки данных.

Гистограммы характеристик ирисов, описанных в вышерассмотренном наборе, показаны на рис. 4.4.

Для построения гистограмм можно применить следующий код:

```
import pandas
import matplotlib.pyplot as plt
iris = pandas.read_csv('iris.csv')
iris.hist()
plt.show()
```

Рассмотрим различные классификаторы, которые мы можем применять.

1. SGD Classifier (SGD) – Линейный классификатор с SGD-обучением (stochastic gradient descent – стохастический градиентный спуск):

Основная идея классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на две полуплоскости, в каждой из которых прогнозируется одно из двух значений целевого класса. Если это можно сделать без ошибок, то обучающая выборка называется линейно разделимой (рис. 4.5)

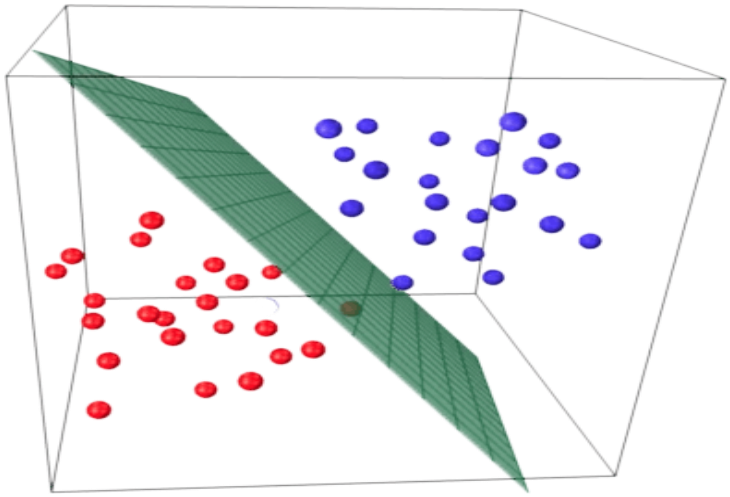


Рис. 4.5. Пример классификации методом SGD

2. Support Vector Machines (SVM) – Метод опорных векторов (kernel = 'linear'):

Основная идея метода – перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

На практике случаи, когда данные можно разделить гиперплоскостью, довольно редки. В этом случае поступают так: все элементы обучающей выборки вкладываются в пространство X более высокой размерности, так, чтобы выборка была линейно разделима (рис.4.6).

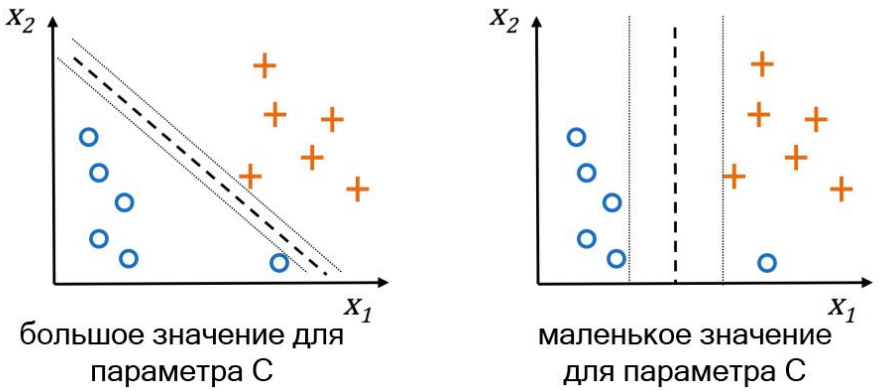


Рис. 4.6. Пример классификации методом SVM

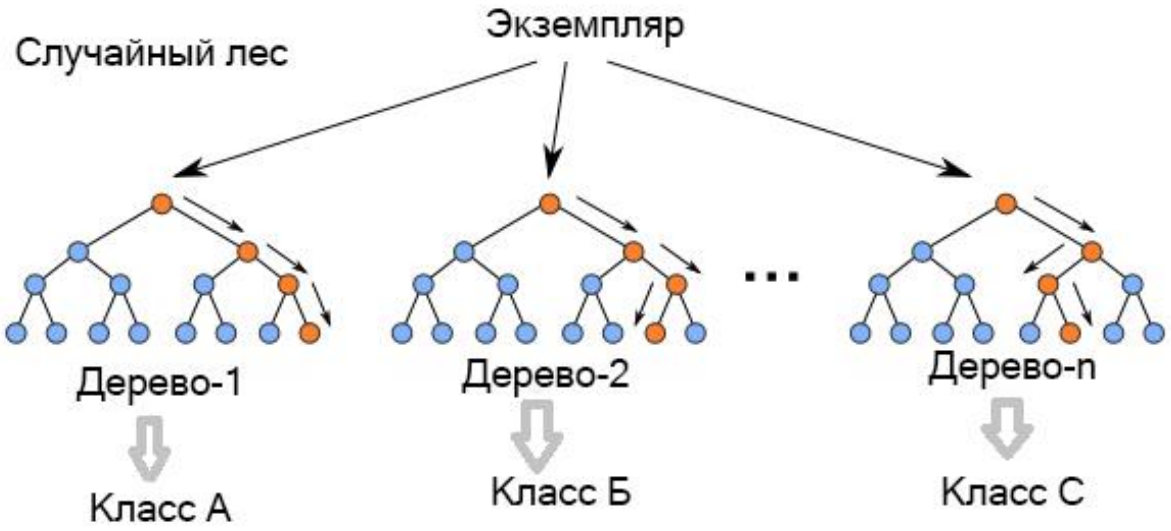


Рис. 4.7. Алгоритм Случайный лес

Kernel (ядро) отвечает за гиперплоскость и может принимать значения linear (для линейной), rbf (для нелинейной) и другие.

3. Decision tree classifier (DT) – Дерево решений:

Этот классификатор разбивает данные на всё меньшие и меньшие подмножества на основе разных критериев, т.е. у каждого подмножества своя сортирующая категория. С каждым разделением количество объектов определённого критерия уменьшается.

Классификация подойдёт к концу, когда сеть дойдёт до подмножества только с одним объектом.

4. Random Forest Classifier (RF) – Случайный лес (объединение нескольких деревьев решений):

Принцип его работы представлен на рис. 4.7.

5. Gaussian process classification (GP) – Гауссовская классификация:

Гауссовские процессы задают априорное распределение на множестве функций, позволяют находить сложные закономерности в данных, автоматически настраивать сложность модели, а также оценивать неопределенность в прогнозе.

6. AdaBoost (Adaptive Boosting) Classifier (AB) – Адаптивное усиление (комбинация нескольких неэффективных классификаторов, что дает высокоточный сильный классификатор).

7. Logistic Regression (LR) – Логистическая регрессия:

Логистическая регрессия является частным случаем линейного классификатора, но она обладает хорошим «умением» – прогнозировать вероятность отнесения наблюдения к классу в бинарном масштабе – нулевом или

единичном. Если значение чего-либо равно или больше 0,5, то объект классифицируется в большую сторону (к единице), если значение меньше 0,5 – в меньшую (к нулю).

8. Gaussian Naive Bayes (NB) – Гауссовский наивный байесовский классификатор:

Такой классификатор вычисляет вероятность принадлежности объекта к какому-то классу. Эта вероятность вычисляется из шанса, что какое-то событие произойдёт, с опорой на уже на произошедшие события. Каждый параметр классифицируемого объекта считается независимым от других параметров.

9. Support Vector (SV) Classification – Метод опорных векторов.

10. MLP (Multi-layer Perceptron) Classifier (MLP) – Многослойный перцептрон:

Многослойными персептронами называют нейронные сети прямого распространения. Входной сигнал в таких сетях распространяется в прямом направлении, от слоя к слою. Многослойный персептрон в общем представлении состоит из множества входных узлов, образующих входной слой, одного или нескольких скрытых слоев вычислительных нейронов, одного выходного слоя нейронов.

11. K-Nearest Neighbors (KNN) – Метод K-ближайших соседей:

Формально основой метода является гипотеза компактности: если метрика расстояния между примерами введена достаточно удачно, то схожие примеры гораздо чаще лежат в одном классе, чем в разных. При такой классификации необходимо вычислить расстояние до каждого из объектов обучающей выборки и отобрать k объектов обучающей выборки, расстояние до которых минимально. Класс классифицируемого объекта – это класс, наиболее часто встречающийся среди k ближайших соседей.

12. Linear Discriminant Analysis (LDA) – Линейный дискриминантный анализ:

Этот метод относится к линейным алгоритмам классификации, т.е. он хорошо подходит для данных с линейной зависимостью. Данный метод работает путём уменьшения размерности набора данных, проецируя все точки данных на линию. Потом он комбинирует эти точки в классы, базируясь на их расстоянии от центральной точки. Метод предназначен для разделения на два класса (рис. 4.8).

13. Quadratic Discriminant Analysis (QDA) – Квадратичный дискриминантный анализ:

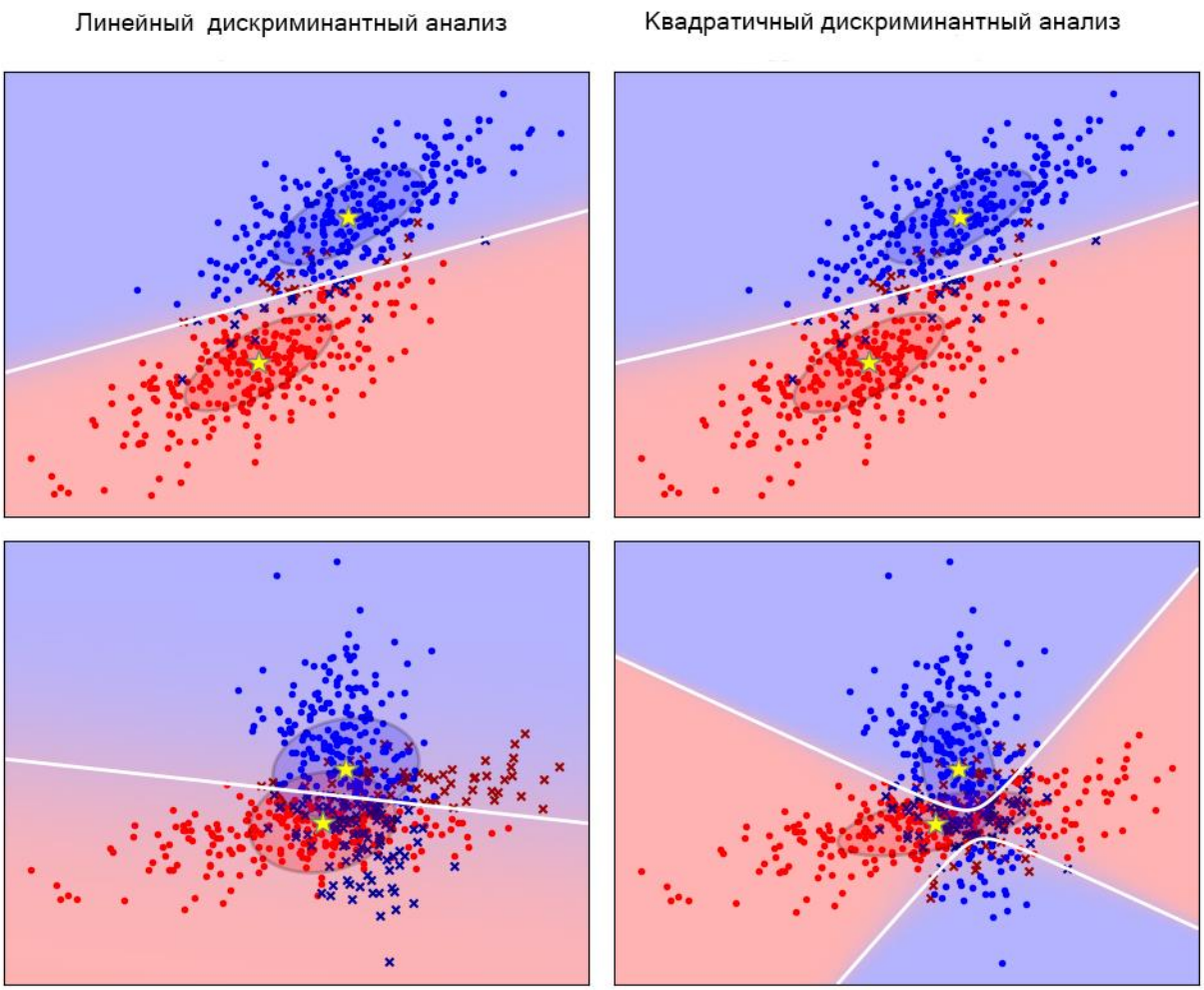


Рис. 4.8. Линейный и квадратичный дискриминантный анализ

Данный метод является естественным обобщением метода LDA. QDA – многоклассный метод, который может использоваться для одновременной классификации нескольких классов (рис. 4.8).

График показывает границы решений для линейного дискриминантного анализа и квадратичного дискриминантного анализа. Нижний ряд демонстрирует, что Линейный Дискриминантный Анализ может выучить только линейные границы, тогда как Квадратичный Дискриминантный Анализ может выучить квадратичные границы и поэтому является более гибким.

Рассмотрим классификацию средствами библиотеки scikit-learn. Исходный код и комментарии к нему представлены ниже:

```
    import numpy as np
import pandas
import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D # Для управления 3D-изображением (посредством мышки)
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
import sys # Для sys.exit()
np.random.seed(348)
# Загрузка данных
iris = pandas.read_csv('iris.csv')
print(iris.shape) # (150, 5)
print(iris.head(10)) ## Печать первых 10 строк данных, загруженных из файла iris.csv
print(iris.describe()) # count, mean, std и др.
print(iris.groupby('variety').size()) # Setosa 50, Versicolor 50, Virginica 50
#
hist = False # True – вывод гистограмм
groundTruth = False # True – вывод классов по трем характеристикам: ширина лепестка, длина чашелистика, длина лепестка
cross_val = True # True, False – вывод карт вероятности по двум характеристикам: длина и ширина чашелистика
#
if hist:
    ## Гистограммы характеристик ирисов
    iris.hist()
    plt.show()
    sys.exit()
if groundTruth:
    # Заменяем имена классов на номера
    iris["variety"] = iris["variety"].map({"Setosa":0, "Versicolor":1,
```

```
"Virginica":2})
    # Выводим результаты наблюдений (по 3-м характеристикам)
    X = iris.values[:, 0:4]
    y = iris.values[:, 4]
    fig = plt.figure(1, figsize = (5, 4))
    ax = Axes3D(fig, rect = [0, 0, .95, 1], elev = 48, azim = 134)
    for name, label in [('Setosa', 0), ('Versicolor', 1), ('Virginica', 2)]:
        ax.text3D(X[y == label, 3].mean(),
                X[y == label, 0].mean(),
                X[y == label, 2].mean() + 2, name,
                horizontalalignment = 'center',
                bbox = dict(alpha = 0.2, edgecolor = 'w', facecolor = 'w'))
    ax.scatter(X[:, 3], X[:, 0], X[:, 2], c = y, edgecolor = 'k')
    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel('Ширина лепестка')
    ax.set_ylabel('Длина чашелистика')
    ax.set_zlabel('Длина лепестка')
    ax.set_title('Ирисы Фишера')
    ax.dist = 12
    fig.show()
    sys.exit()
iris_array = iris.values
x = iris_array[:, 0:4]
y = iris_array[:, 4]
# Выделяем обучающую и тестовую выборки
# x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size
= 0.20, random_state = 348)
classifiers = []
        classifiers.append(('SGD', SGDClassifier(max_iter = 1500, tol = 1e-4)))
classifiers.append(('SVL', SVC(kernel = 'linear', C = 0.025, probability =
True))) # C – штраф в случае ошибки
classifiers.append(('RF', RandomForestClassifier(max_depth = 5, n_estimators
= 10, max_features = 1)))
classifiers.append(('GP', GaussianProcessClassifier()))
classifiers.append(('AB', AdaBoostClassifier()))
```

```
classifiers.append(('DT', DecisionTreeClassifier()))
classifiers.append(('LR', LogisticRegression(solver = 'lbfgs', max_iter = 500,
multi_class = 'auto')))
classifiers.append(('NB', GaussianNB()))
classifiers.append(('SVR', SVC(gamma = 2, C = 1.0))) # gamma – коэффициент ядра для 'rbf' – radial basis function, 'poly' and 'sigmoid'
classifiers.append(('MLP', MLPClassifier(alpha = 0.01, max_iter = 200, solver
= 'lbfgs', tol = 0.001)))
classifiers.append(('KNN', KNeighborsClassifier(3)))
classifiers.append(('QDA', QuadraticDiscriminantAnalysis()))
classifiers.append(('LDA', LinearDiscriminantAnalysis()))
        results = []
        names = []
        k = 10
        for name, classifier in classifiers:
# Генерируем индексы для выделения обучающих и тестовых данных
# k – 1 подвыборок будут использованы для обучения, а одна – для проверки (валидации)
kfold = model_selection.KFold(n_splits = k, random_state = 348)
## cv – стратегия кросс-валидации
cv_results = model_selection.cross_val_score(classifier, x, y, cv = kfold, scoring = 'accuracy')
results.append(cv_results)
names.append(name)
print(name, np.round(cv_results, 3), 'Средняя точность: ', round(cv_results.mean(), 3))
        # Графическое представление результатов
fig = plt.figure()
        fig.suptitle('Точность, показанная классификаторами')
ax = fig.add_subplot(1, 1, 1)
plt.boxplot(results)
ax.set_xticklabels(names)
        plt.show()
```

Тогда получим следующие результаты.

SGD [1. 0.933 1. 0.8 0.267 0. 1. 0.933 1. 1.] Средняя точность: 0.793

SVL [1. 1. 1. 0.867 0.867 0.733 1. 0.8 0.667 0.733] Средняя точность: 0.867

RF [1. 1. 1. 1. 0.867 0.867 1. 0.8 0.6 0.933] Средняя точность: 0.907

GP [1. 1. 1. 1. 0.867 0.867 1. 0.867 0.733 0.933] Средняя точность: 0.927

AB [1. 1. 1. 1. 0.933 0.867 1. 0.867 0.8 0.933] Средняя точность: 0.94

DT [1. 1. 1. 1. 0.933 0.867 1. 0.867 0.8 1.] Средняя точность: 0.947

LR [1. 1. 1. 1. 0.933 0.867 1. 0.867 0.867 0.933] Средняя точность: 0.947

NB [1. 1. 1. 0.933 0.933 0.867 1. 0.867 0.867 1.] Средняя точность: 0.947

SVR [1. 1. 1. 1. 0.867 0.867 1. 0.867 0.867 1.] Средняя точность: 0.947

MLP [1. 1. 1. 1. 0.867 0.933 1. 0.867 0.667 1.] Средняя точность: 0.93

KNN [1. 1. 1. 1. 0.867 0.867 1. 0.867 0.867 1.] Средняя точность: 0.947

QDA [1. 1. 1. 1. 0.8 0.933 1. 1. 0.933 1.] Средняя точность: 0.967

LDA [1. 1. 1. 1. 0.933 0.933 1. 1. 0.8 1.] Средняя точность: 0.967

Сравним эффективность классификаторов (рис 4.9).

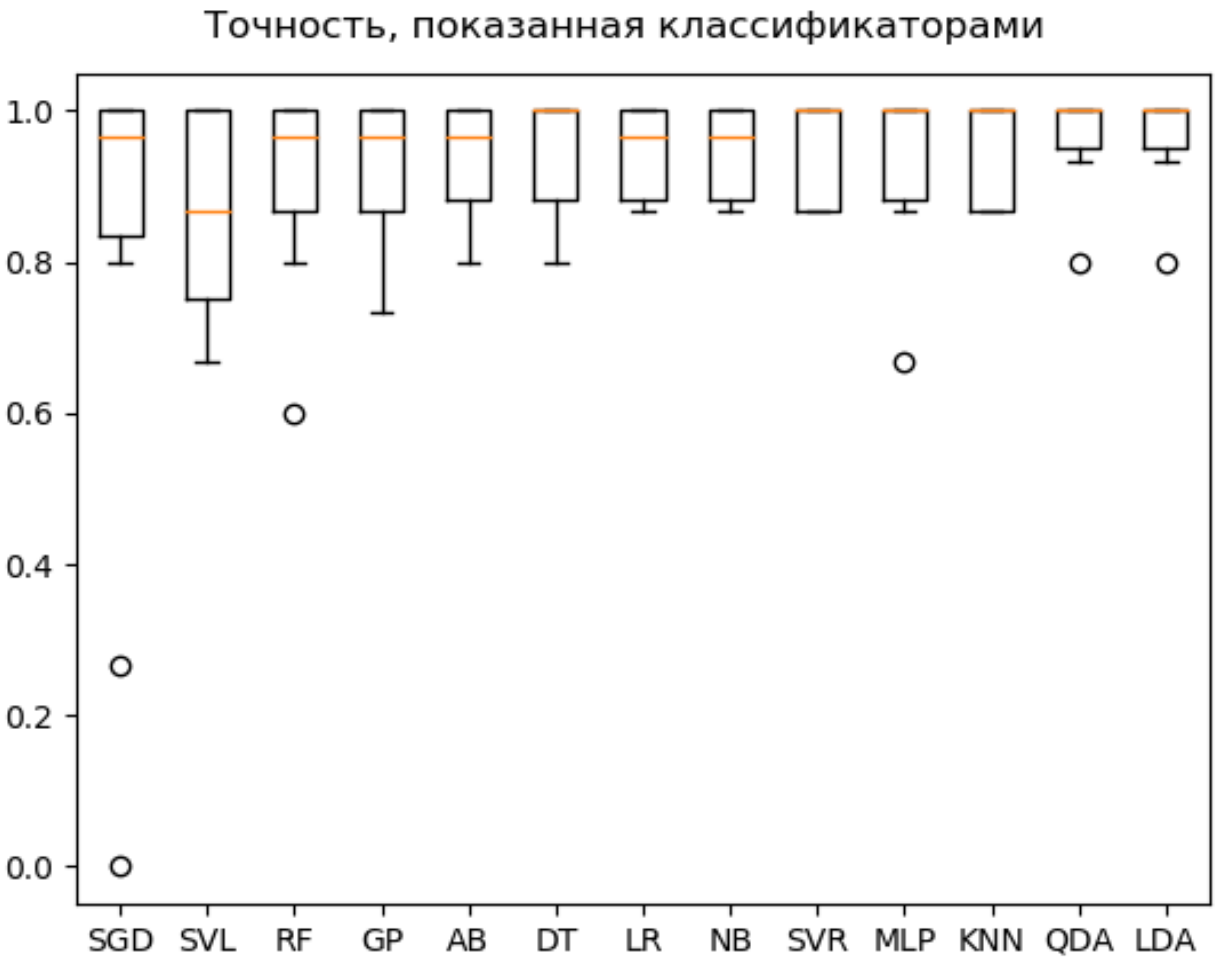


Рис. 4.9. Точность классификации

4.2. ЗНАКОМСТВО С БИБЛИОТЕКОЙ KERAS

Одним из мощных и распространённых инструментов работы с нейронными сетями является библиотека Keras.

В качестве первого примера рассмотрим задачу распознавания цифр. Для этого будем использовать набор данных MNIST, который содержит различные рукописные версии цифр и включающий 60 000 обучающих и 10 000 тестовых примеров. Правильные метки для каждого изображения уже расставлены. Цифре в наборе имеют вид, представленный на рис. 4.10.

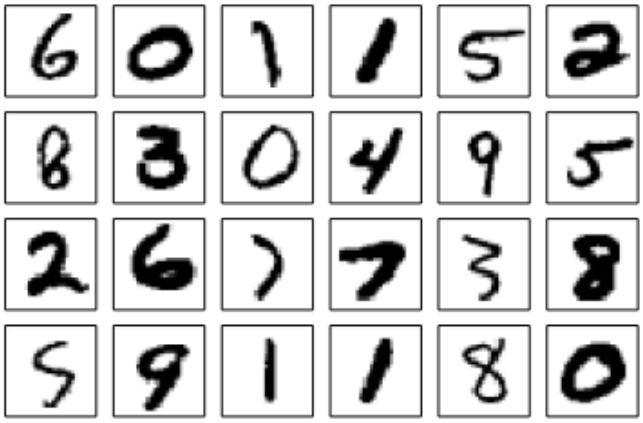


Рис. 4.10. Фрагмент набора данных MNIST

Так как имеется набор размеченных данных (т.е. выходные значения для каждого изображения известны), то будет использоваться алгоритм обучения с учителем.

Воспользуемся библиотекой Keras, чтобы определить сеть, распознающую рукописные цифры из набора MNIST. Начнем с очень простой нейросети и постепенно будем её улучшать.

Keras предоставляет средства для загрузки набора данных и разбиения его на обучающий – x_train, и тестовый – x_test. Для поддержки вычислений на GPU данные преобразуются к типу float32 и нормируются на интервал от 0 до 1. Суть нормирования состоит в переходе к определенному масштабу – стандартизированным единицам измерения.

Подготовка данных осуществляется следующим программным кодом [3]:

```
# подключение необходимых модулей
from _future_ import print_function
import numpy as np
from keras.datasets import mnist
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
np.random.seed(1671) # для воспроизводимости результатов
# сеть и ее обучение
# задаем основные параметры сети, создавая необходимые константы, которые будут регулярно использоваться в нашей сети
NB_EPOCH = 200 # количество эпох
BATCH_SIZE = 128 # количество обучающих примеров за одну итерацию, которые увидит оптимизатор, прежде чем обновит веса.
VERBOSE = 1 #режим вывода процесса обучения (1 – подробный, 0 – фоновый)
NB_CLASSES = 10 # количество результатов (классов объектов) = числу цифр
OPTIMIZER = SGD() # оптимизатор – определенный алгоритм, который обновляет веса в процессе обучения нейросети.
N_HIDDEN = 128 #количество нейронов скрытого слоя
VALIDATION _ SPLIT=0.2  # определяет какая часть обучающего набора зарезервирована для контроля
# далее данные случайно перетасованы и разбиты на обучающий и тестовый набор:
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# X_train содержит 60000 изображений размера 28x28
# преобразуем в массив (матрицу) размером 60000 на 784
RESHAPED = 784
X_train = X_train.reshape(60000, RESHAPED) #обучающий набор
X_test = X_test.reshape(l0000, RESHAPED) #тестовый набор
X_train = X_train.astype('float32') # данные преобразуются к типу float32
X_test = X_test.astype ('float32') # данные преобразуются к типу float32
# Нормируем. Тем самым широкий интервал хаотичных данных приводится к удобному интервалу [0,1]
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
# преобразовать векторы классов в бинарные матрицы классов
```

Y_train = np_utils.to_categorical(y_train, NB_CLASSES)

Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

Во входном слое с каждым пикселем изображения ассоциирован один нейрон, т. е. всего получается 28 x 28 = 784 нейрона.

Обычно значения, ассоциированные с пикселями, нормируются с целью привести их к диапазону [0, 1] (это значит, что яркость каждого пикселя делится на максимально возможную яркость 255). На выходе получается 10 классов, по одному для каждой цифры.

Последний слой состоит из 10 нейронов с функцией активации softmax, являющейся обобщением сигмоиды.

Рассмотрим код для создания модели нейронной сети:

 # Sequential() означает, что используется последовательная модель нейронной сети, в которой слои идут друг за другом

model = Sequential()

 # Добавление слоя из 10 нейронов (т.к. у нас 10 цифр)

model.add(Dense(NB_CLASSES, input shape=(RESHAPED,)))

 # последний добавленный слой состоит из единственного нейрона с функцией активации softmax

model.add(Activation('softmax'))

model.summary()

Определенную таким образом модель необходимо откомпилировать, т.е. привести к виду, допускающему исполнение базовой библиотекой (Theano или TensorFlow). Перед компиляцией необходимо принять несколько решений:

- выбрать оптимизатор, т.е. конкретный алгоритм, который будет обновлять веса в процессе обучения модели;

- выбрать целевую функцию, которую оптимизатор использует для навигации по пространству весов (часто целевая функция называется также функцией потерь, а процесс оптимизации – минимизацией потери);

- оценить качество обученной модели.

Показатели качества похожи на целевые функции, различаются они только тем, что показатели используются не для обучения модели, а для оценки её качества. Компиляция модели в Keras производится следующим образом:

model.compile(loss='categorical crossentropy', optimizer=OPTIMIZER, metrics=('accuracy'])

Для обучения откомпилированной модели служит функция *fit()*, принимающая в частности следующие параметры:

– *epochs*: число эпох (периодов). На каждой эпохе оптимизатор пытается подкорректировать веса, стремясь минимизировать целевую функцию;

– *batch_size*: сколько обучающих примеров обрабатывает оптимизатор, прежде чем обновит веса.

Обучить модель в Keras очень просто. Допустим, что выполняется NB_EPOCH итераций:

history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split=VALIDATION_SPLIT)

Для получения новых данных от обученной нейронной сети используется метод *predict(X)*, где X массив новых входных данных:

predictions = model.predict(X)

Для заданного входного вектора можно вычислить несколько значений:

– *model.evaluate()*: вычисляет потерю;

– *model.predict_classes()*: вычисляет категориальные выходы;

– *model.predict_proba()*: вычисляет вероятности классов.

Необходимо отметить, что представленная модель является наипростейшей. Для улучшения точности нейронной сети в нее добавляют новые слои, варьируют параметры, тем самым увеличивая её сложность и гибкость. Далее рассмотрим несколько подходов к совершенствованию нашей нейронной сети.

Первое улучшение – включить в сеть дополнительные слои. После входного слоя поместим первый плотный слой с *N_HIDDEN* нейронами и функцией активации *relu*. Этот слой называется скрытым, потому что он напрямую не соединен ни с входом, ни с выходом. После первого скрытого слоя добавим ещё один, также содержащий *N_HIDDEN* нейронов, а уже за ним будет расположен выходной слой с 10 нейронами, которые активируются, если распознана соответствующая цифра

В итоге обновленный фрагмент кода для слоев нашей сети будет выглядеть следующим образом:

```
model = Sequential()
model.add(Dense(N_HIDDEN, input shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
```

model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()

В результате нейронная сеть показывает следующие показатели – верность 94,50% на обучающем наборе, 94,63% – на контрольном и 94,41% – на тестовом.

Продолжим работу, следующее улучшение совсем простое. Мы применим прореживание – с вероятностью DR будем случайным образом отбрасывать некоторые значения, распространяющиеся внутри сети, состоящей из плотных скрытых слоев. Это хорошо известная форма регуляризации в машинном обучении. DR может принимать значения от 0 до 1, где 1 даст 100% прореживание предыдущего слоя нейронов. Слой прореживания выглядит следующим образом:

model.add(Dropout(DR))

Отметим, что нежелательно помещать его в конце модели, это может негативно повлиять на результат.

В Keras есть два способа соединения моделей нейронных сетей:

– последовательная композиция (рассмотренная выше);

– функциональная композиция.

При последовательной композиции готовые модели соединяются в линейный конвейер слоев, напоминающий стек или очередь.

Функциональный API позволяет определять более сложные модели, например, ациклические графы, модели с разделяемыми слоями или с несколькими выходами. Примеры будут приведены далее.

Большой проблемой при использовании нейронных сетей является переобучение, т.е. ситуация, когда нейронная сеть адекватно работает на обучающих данных, но совершенно неспособна обрабатывать неизвестные данные.

Цель регуляризации – предотвратить переобучение. В слоях различных типов имеются параметры регуляризации. Ниже приведен список параметров регуляризации, часто используемых в плотных и сверточных модулях.

– kernel_regularizer: функция регуляризации, применяемая к матрице весов;

– bias_regularizer: функция регуляризации, применяемая к вектору смещений;

– activity_regularizer: функция регуляризации, применяемая к выходу слоя (его функции активации).

Кроме того, для регуляризации можно использовать прореживание *Dropout* и зачастую это дает весомый эффект.

4.3. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Сверточная нейронная сеть получила свое название благодаря математической операции **свертки**, которая лежит в её основе её архитектуры. Если говорить математическим языком, то свертка – это математическая операция, применённая к двум функциям f и g, порождающая третью функцию, которая иногда может рассматриваться как модифицированная версия одной из первоначальных.

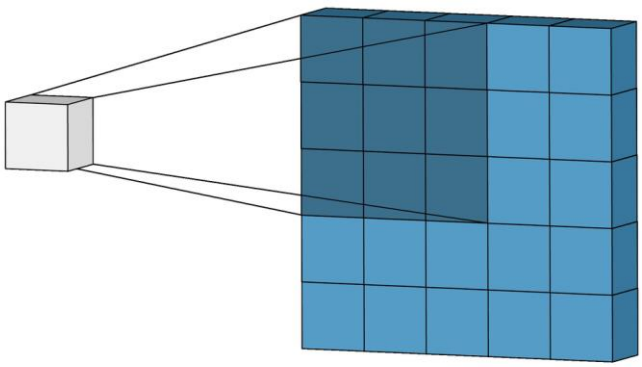


Рис. 4.11. Двумерная свертка

Двумерная свертка (2D convolution) – это довольно простая операция. Её суть состоит в том, что изображения, которые поступают как входные данные, разбиваются на отдельные фрагменты (тем самым образуя матрицу) и умножаются на **ядро** (матрицу весовых коэффициентов). Ядро поэтапно «скользит» по входному изображению. Затем все полученные значения суммируются в один выходной пиксель в ядре на данном этапе.

Процедура повторяется пока ядро не пройдёт по каждой локацией входной матрицы. Когда этот процесс будет выполнен, входная двумерная матрица преобразуется в иную двумерную матрицу – **матрицу признаков**, её размер будет равен размерам ядра, «скользившему» по ней. Признаки на выходе являются взвешенными суммами (где веса являются значениями самого ядра) признаков на входе, расположенных примерно в том же месте, что и выходной пиксель на входном слое.

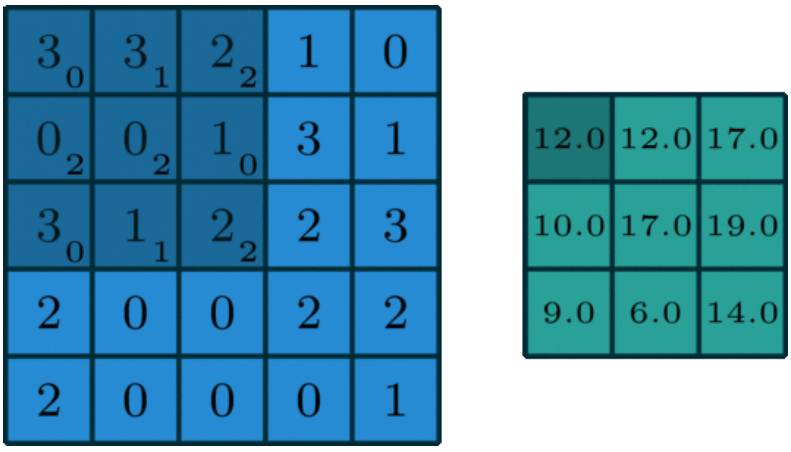


Рис. 4.12. Операция свертки

В примере, приведённом выше, имеется матрица размерностью 5 на 5 и содержащая 25 признаков на входе и матрица размерностью 3 на 3 с 9 признаками на выходе. Если бы мы использовали нейросеть на основе обычных плотных слоев мы бы имели весовую матрицу размерностью 25 на 9, тем самым включающую в себя 225 параметров, а каждый выходной признак являлся бы взвешенной суммой всех признаков на входе. Свертка позволяет произвести такую операцию с всего 9-ю параметрами, ведь каждый признак на выходе получается анализом не каждого признака на входе, а только одного входного, находящегося в «примерно том же месте». Обратите на это внимание, так как это будет иметь важное значение для дальнейшего обсуждения.

В сверточных нейронных сетях в основном применяются две технологии: Padding и Striding.

Padding. Обратите внимание на то, что в процессе скольжения края изображения обрезаются, преобразуя матрицу признаков размером 5 на 5 в матрицу 3 на 3 (рис. 4.12). Это является проблемой, потому что крайние пиксели входной матрицы никогда не оказываются в центре ядра, потому что тогда ядру не над чем будет скользить за краем. Но нам требуется сохранить размеры матрицы признаков. Для этого используется технология padding. Она добавляет к краям поддельные (fake) пиксели (обычно нулевого значения, вследствие этого к ним применяется термин «нулевое дополнение» – “zero padding”). Таким образом, поддельные пиксели позволяют настоящим оказаться в центре ядра, и, в следствии этого, размер матрицы признаков остается таким же, как и размерность входной матрицы.

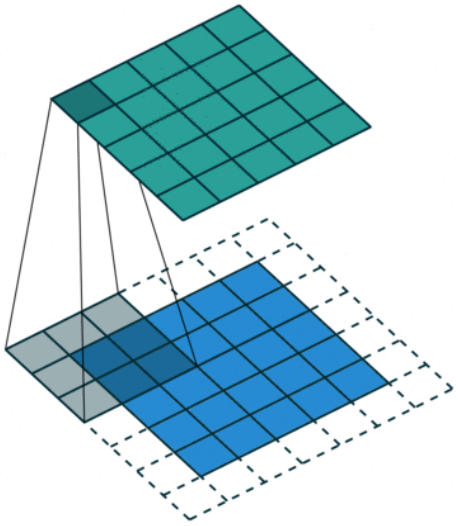


Рис. 4.13. Padding

Striding. Иной случай, когда при работе со сверточным слоем, нужно получить матрицу признаков меньшего размера, чем входная матрица. Это обычно необходимо в сверточных нейронных сетях, где размер пространственных размеров уменьшается при увеличении количества каналов. Чтобы достичь этого требуется использование субдискритизирующих слоев (pooling layer), например, принимать среднее/максимальное в каждой локации размером 2 на 2, при этом избегая их пересечения. Данная методика дает возможность уменьшить все пространственные размеры в два раза.

Другим способом достижения этой цели является stride. Основная её идея в том, чтобы специально пропустить некоторые области на входной матрице. Основным параметром этого метода является шаг. Например, шаг равный 1, означает, что берутся пролеты через пиксель, так же, как и обычной свертке. Шаг равный 2 реализует пролет в 2 пикселя, тем самым уменьшает матрицу признаков в 2 раза. Аналогично будет и с другими значениями шага.

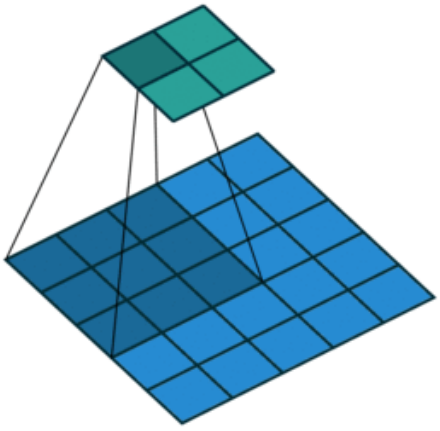


Рис. 4.14. Свертка с шагом

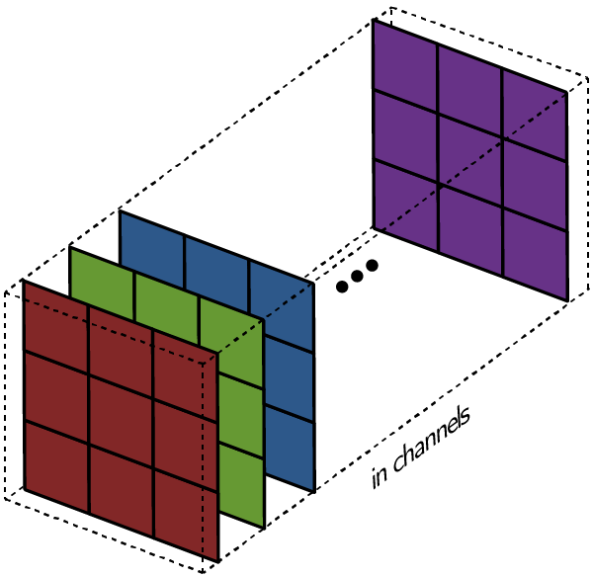


Рис. 4.15. Образование фильтра

В настоящее время для уменьшения выходной матрицы использование субдискритизирующих слоев определенного вида не является перспективной, поэтому используются различные комбинации чередующихся сверток.

Приведенные выше примеры сетей отлично подходят для черно-белых изображений, так как они используют только входной один канал. С помощью него формируются черный, белый, а также градации серого цвета. Но при работе с цветными изображениями, использующими RGB-модель, используются три канала, т.е. изображение разделяется на три канала: зеленый, красный и синий цвета. С каждым получившимся изображением в таком канале происходят операции, которые описывались ранее для одноканальной сети (рис. 4.15).

Для каждого из трех входных каналов есть свое ядро, отличающееся от ядер других каналов. Коллекция этих ядер впоследствии образует фильтр.

Для образования фильтра каждое ядро скользит по изображению своего канала. Их объединение и образует фильтр. Так же одним из главных аспектов при формировании фильтра, является вес ядра для определенного канала. Именно этот параметр будет контролировать размер весов для определенного цвета, а следовательно, станет доступной возможность контролировать возбудимость на определенные признаки в этом канале.

Затем каждая из обработанных в канале версий суммируется вместе для формирования одного канала. Ядра каждого фильтра генерируют одну версию каждого канала, а фильтр в целом создает один общий выходной канал:

Наконец, каждый выходной файл имеет свое смещение. Смещение добавляется к выходному каналу для создания конечного выходного канала.

Результат для любого количества фильтров идентичен: каждый фильтр обрабатывает вход со своим отличающимся от других набором ядер и скалярным смещением по описанному выше процессу, создавая один выходной канал. Затем они объединяются вместе для получения общего выхода, причем количество выходных каналов равно числу фильтров. При этом обычно применяется нелинейность перед передачей входа другому слою свертки, который затем повторяет этот процесс.

Далее рассмотрим пример классификации изображений цифр (рис 4.10) с помощью сверточных нейронных сетей.

```
from keras.layers import Convolution2D, MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.models import Sequential
from keras.utils import np_utils
from keras.datasets import mnist

# Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train.shape
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
#Линейная модель слоев.
model = Sequential()
# Слой двумерной свертки (например, пространственная свертка на изображениях).
model.add(Convolution2D(32, (3, 3), activation='relu', input_shape=(1,28,28),
data_format='channels_first'))
#усреднение значений фильтров. Обязателен после сверточных слоев;
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

```
#  Flatten – переход от многомерных слоев к одномерному
model.add(Flatten())
#еще один слой скрытый полносвязный (Dense) слой
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
#Обратите внимание, что конечный слой имеет выходной размер 10, со-
ответствующий 10 классам цифр.
model.add(Dense(10, activation='softmax'))
#собираем модель
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# training model
#verbose =0
model.fit(X_train, Y_train,
          batch_size=32, epochs=5, verbose=1)
#оценка модели
score = model.evaluate(X_test, Y_test, verbose=0)
model_json = model.to_json()
# Записываем модель в файл
json_file = open("model_01.json", "w")
model.save_weights("model_01_w.h5")
json_file.write(model_json)
json_file.close()

#тестирование
from keras.models import model_from_json
import glob
import numpy as np
from PIL import Image
from keras.utils import np_utils
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train.shape
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
X_train = X_train.astype('float32')
```

```
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
json_file = open("model_01.json", "r")
loaded_model_json = json_file.read()
json_file.close()
# Создаем модель на основе загруженных данных
loaded_model = model_from_json(loaded_model_json)
# Загружаем веса в модель
loaded_model.load_weights("model_01_w.h5")

loaded_model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])

score = loaded_model.evaluate(X_test, Y_test, verbose=0)
print("Точность модели на тестовых данных датасета: %.2f%%" %
(score[1] * 100))

# Проверяем модель на тестовых данных
result = []
data = []
start = True
#открытие всех изображений из папки
for filename in glob.glob('images/*.png'):
    img = Image.open(filename)
    result.append(filename.split("\\")[-1])
    #сохраняем только 28x28 пикселей массива в список
    data.append(np.array(img, dtype='uint8')[:,:,0])
# выделяем из названия цифру – выходное значение для нейронной сети
result = [int(i.split(".")[0][0]) for i in result]
# приводим значения к категориям
Y_test_cat = np_utils.to_categorical(result, 10)
# обрабатываем входные данные к тому же виду, что и в обучающем мно-
жестве
X_test = np.array(data, dtype='float32')
```

```
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
X_test /= 255
#определяем выходное значение по новым данным
y_test = loaded_model.predict(X_test)
#возвращаем цифру класса
classes = np.argmax(y_test, axis=1)
# выводим результат и точность модели
print(y_test)
for i in zip(result,classes):
    print(i)
scores = loaded_model.evaluate(X_test, Y_test_cat, verbose=0)
print("Точность модели на тестовых данных: %.2f%%" % (scores[1] * 100))
```

4.4. РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) – популярные модели, используемые в обработке естественного языка (NLP). Обработкой естественного языка (NLP) называется активно развивающаяся научная дисциплина, занимающаяся поиском смысла и обучением на основании текстовых данных. Такие нейронные сети, во-первых, оценивают произвольные предложения на основе того, насколько часто они встречались в текстах. Это дает нам больше информации о грамматической и смысловой корректности. Во-вторых, языковые модели генерируют новый текст [5].

Идея RNN заключается в последовательном использовании информации. В традиционных нейронных сетях подразумевается, что все входы и выходы независимы. Но для многих задач это не подходит. Если вы хотите предсказать следующее слово в предложении, лучше учитывать предшествующие ему слова. RNN называются рекуррентными, потому что они выполняют одну и ту же задачу для каждого элемента последовательности, причем выход зависит от предыдущих вычислений (RNN схожа с рекуррентными функциями в языках программирования, которые вызывают сами себя и используют результаты предыдущих функций). Ещё одна интерпретация RNN: это сети, у которых есть «память», которая учитывает предшествующую информацию. Теоретически RNN могут использовать информацию в произвольных длинных последовательностях, но на практике они ограничены лишь несколькими шагами (подробнее об этом позже).

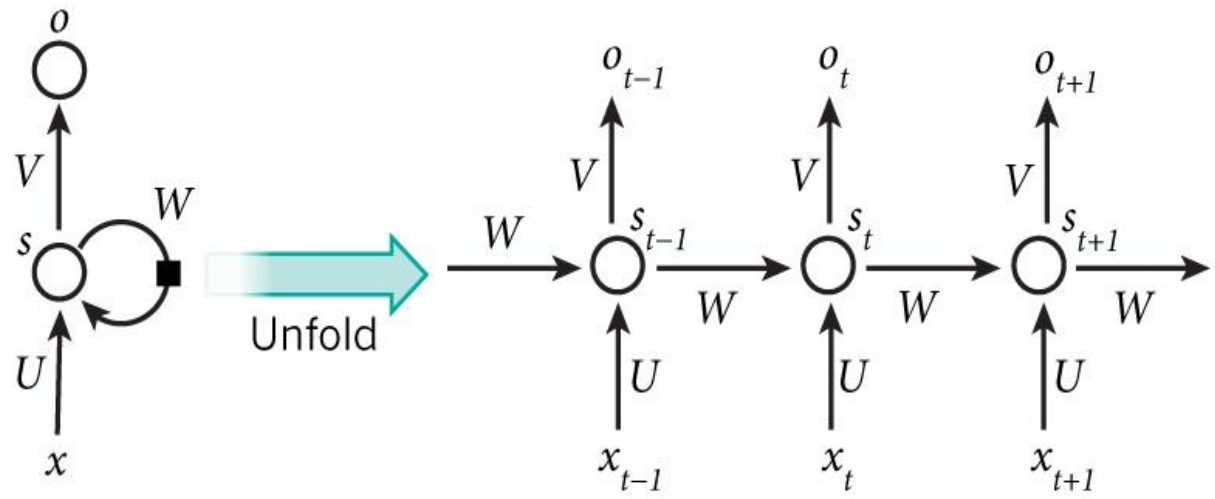


Рис. 4.16. Рекуррентная нейронная сеть и её развертка (unfolding)

На рисунке выше показано, что RNN разворачивается в полную сеть. При развертке сети мы проводим операцию «выписывания» сети для полной последовательности. Например, если последовательность представляет собой предложение из 5 слов, развертка будет состоять из 5 слоев, по слою на каждое слово. Формулы, задающие вычисления в RNN, следующие:

- x_t – входные данные на временном шаге t. Например x_1 может быть вектором с одним горячим состоянием (one-hot vector – примером может служить последовательность слов, закодированных в одном векторе), соответствующим второму слову предложения.
- s_t – это скрытое состояние на шаге t. Это «память» сети: s_t зависит, как функция, от предыдущих состояний и текущего входа x_t:

$$s\_t = f(Ux\_t + Ws\_\{t\text{-}1\})$$

- f – функция активации tanh или ReLU.
- o_t – выход на шаге t. Например, если мы хотим предсказать слово в предложении, выход может быть вектором вероятностей в нашем словаре. o_t = softmax(Vs_t)

Отметим несколько важных моментов.

- Можно интерпретировать s_t как память сети. s_t содержит информацию о том, что произошло на предыдущих шагах времени. Выход o_t вычисляется исключительно на основе «памяти» s_t. На практике все немного сложнее: s_t не может содержать информацию слишком большого количества предшествующих шагов;
- В отличие от традиционной глубокой нейронной сети, которая использует разные параметры на каждом слое, RNN имеет одинаковые (U, V, W) на всех этапах. Это отражает тот факт, что мы выполняем одну и

ту же задачу на каждом шаге, используя только разные входы. Это значительно уменьшает общее количество параметров, которые нам нужно подобрать;

- Схема на рисунке выше имеет выходы на каждом шаге, но, в зависимости от задачи, они могут не понадобиться. Например, при определении эмоциональной окраски предложения, целесообразно заботиться только о конечном результате, а не о окраске после каждого слова. Аналогично, нам может не потребоваться ввод данных на каждом шаге. Основной особенностью RNN является **скрытое состояние**, которое содержит некоторую информацию о последовательности.

Рекуррентные нейронные сети продемонстрировали большой успех во многих задачах обработки естественного языка. Ниболее часто используемым типом RNN являются LSTM, которые намного лучше захватывают (хранят) долгосрочные зависимости, чем RNN. LSTM – это, по сути, то же самое, что и RNN, но у них просто есть другой способ вычисления скрытого состояния.

Рассмотрим задачу генерации текста. Учитывая последовательность слов, мы хотим предсказать вероятность каждого слова (в словаре). Языковые модели позволяют нам измерить вероятность выбора, что является важным вкладом в машинный перевод (поскольку предложения с большой вероятностью правильны). Побочным эффектом такой способности является возможность генерировать новые тексты путем выбора из выходных вероятностей. Мы можем генерировать и другие вещи, в зависимости от того, что из себя представляют наши данные. В языковом моделировании наш вход обычно представляет последовательность слов (например, закодированных как вектор с одним горячим состоянием (one-hot), а выход – последовательность предсказанных слов. При обучении нейронной сети, мы подаем на вход следующему слою предыдущий выход o_t=x_{t+1}, поскольку хотим, чтобы результат на шаге t был следующим словом.

4.5. РЕГРЕССИОННЫЕ НЕЙРОННЫЕ СЕТИ

Два основных вида обучения с учителем (т.е. где при обучении имеются правильные варианты выходных данных) – классификация и регрессия. В обоих случаях модель обучается на данных с известными метками. В случае классификации метки – дискретные значения, например, жанр текста или категория изображения. В случае регрессии метками являются значения непрерывной величины, например, цены акций.

В большинстве рассмотренных выше примеров модели глубокого обучения использовались для классификации. Здесь же мы рассмотрим, как такая модель позволяет выполнить регрессию. Напомним, что в моделях классификации в конце имеется плотный слой с нелинейной функцией активации, число выходов которого равно числу классов в модели. Так, в модели классификации изображений из набора ImageNet размерность плотного выходного слоя равна 1000, т.е. он может предсказывать 1000 классов. Аналогично в модели анализа эмоциональной окраски плотный слой имеет два выхода, соответствующие положительной и отрицательной окраске.

В регрессионных моделях в конце тоже имеется плотный слой, но только с одним выходом и без нелинейной функции активации. Это значит, что плотный слой возвращает просто сумму откликов нейронов предыдущего слоя. Кроме того, в качестве функции потерь обычно используется **среднеквадратическая ошибка (СКО,** англ. MSE), но допустимо и несколько других функций. Среднеквадратичная ошибка – это усредненная сумма квадратов между предсказанными и истинными значениями.

Рассмотрим пример простейшей регрессионной нейронной сети, которая пытается предсказать некоторую операцию над входными значениями.

```
    from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, LSTM, RNN, Embedding, Input
from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np
# генерирование набора данных для регрессии
x_raw = []
y_raw = []
xscaler, yscaler = StandardScaler(),StandardScaler()
for i in range(1, 10001):
   x_raw.append([i])
   y_raw.append([i+50])
X = np.array(x_raw)
Y = np.array(y_raw)
X = xscaler.fit_transform(X)
Y = yscaler.fit_transform(Y)
means_ = yscaler.mean_
```

```
scale_ = yscaler.scale_
print(means_, scale_)
        # определяем модель
input_layer = Input(shape=(1,))
h_layer = Dense(15, activation='relu', kernel_initializer="glorot_uniform")(input_layer)
h_layer = Dense(100, activation='relu', kernel_initializer="glorot_uniform")(h_layer)
result_layer = Dense(1,  kernel_initializer="glorot_uniform")(h_layer)
model = Model(inputs=[input_layer], outputs=[result_layer])
model.compile(loss='mse', optimizer='adam')
model.fit(X, Y, epochs=25, verbose=1, validation_split=0.2)

# проверка модели
Xnew = np.array([[10], [20], [4.4], [9.6]])
y_predict = [i + 50 for i in Xnew]

# предсказание
ynew = model.predict(xscaler.transform(Xnew))

# сравнение полученных и «правильных» значений
for i in range(len(Xnew)):
   print("X=%s, Predicted=%s, Right=%s" % (Xnew[i], yscaler.inverse_transform(ynew[i]), y_predict[i]))
```

4.6. ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНЫЕ НЕЙРОННЫЕ СЕТИ

Генеративно-состязательная нейросеть (Generative adversarial network, GAN) – архитектура, состоящая из генератора и дискриминатора, настроенных на работу друг против друга. Отсюда GAN и получила свое название [5].

Потенциал GAN огромен, поскольку они имитируют любое распределение данных. GAN обучают создавать структуры, похожие на сущности из нашего мира в области изображений, музыки, речи, прозы. Генеративно-состязательные сети, в некотором смысле, роботы-художники, и результат их работы впечатляет.

Рис. 4.17. Реалистичные изображения несуществующих знаменитостей, созданные с помощью GAN

Рассмотрим, как работают алгоритмы дискриминатора и генератора.

Дискриминационные алгоритмы пытаются классифицировать входные данные. Учитывая особенности полученных данных, они стараются определить категорию, к которой они относятся.

К примеру, анализируя все слова в письме дискриминационный алгоритм может предсказать, является сообщение спамом или нет. Спам – это категория, а пакет слов, собранный из электронной почты – образы, которые составляют входные данные. Математически категории обозначают **y**, а образы обозначают **x**. Запись **p(y|x)** используется для обозначения «вероятности **y** при заданном **x**», которая обозначает «вероятность того, что электронное письмо является спамом при имеющемся наборе слов».

Итак, дискриминационные функции сопоставляют образы с категорией. Они заняты только этой задачей.

Генеративные алгоритмы заняты обратным. Вместо того, чтобы предсказывать категорию по имеющимся образам, они пытаются подобрать образы к данной категории.

В то время как дискриминационные алгоритмы волнует взаимосвязь между y и x, генеративные алгоритмы волнует «откуда берутся **x**». Они позволяют находить **p(x|y)**, вероятность **x** при данном **y** или вероятность образов при данном классе (генеративные алгоритмы также могут использоваться в качестве классификаторов. Они могут делать больше, чем классифицировать входные данные.)

Еще одно представление о работе генеративных алгоритмов можно получить, разделяя дискриминационные модели от генеративных таким образом:

- Дискриминационные модели изучают границу между классами;
- Генеративные модели определяют, как распределять отдельные классы.

Рассмотрим принцип работы GAN.

Одна нейронная сеть, называемая генератором, генерирует новые экземпляры данных, а другая – дискриминатор, оценивает их на подлинность; т.е. дискриминатор решает, относится ли каждый экземпляр данных, который он рассматривает, к набору тренировочных данных или нет.

В качестве примера сгенерируем рукописные цифры, подобные тем, что имеются в наборе данных MNIST. Цель дискриминатора – распознать подлинные экземпляры из набора.

Между тем, генератор создает новые изображения, которые он передает дискриминатору. Он делает это в надежде, что они будут приняты подлинными, хотя являются поддельными. Цель генератора состоит в том, чтобы генерировать рукописные цифры, которые будут пропущены дискриминатором. Цель дискриминатора – определить, является ли изображение подлинным.

Шаги, которые проходит GAN:

- Генератор получает некоторое число и возвращает изображение.
- Это сгенерированное изображение подается в дискриминатор наряду с потоком изображений, взятых из фактического набора данных.
- Дискриминатор принимает как реальные, так и поддельные изображения и возвращает вероятности истинности изображения – числа от 0 до 1. Единица представляет собой подлинное изображение, ноль – фальшивое.

Таким образом, у нас есть двойной цикл обратной связи:

- Дискриминатор находится в цикле с достоверными изображениями.
- Генератор находится в цикле вместе с дискриминатором.

Сеть дискриминаторов представляет собой стандартную сверточную сеть, которая может классифицировать изображения, подаваемые на нее с помощью классификатора, распознающего изображения как реальные

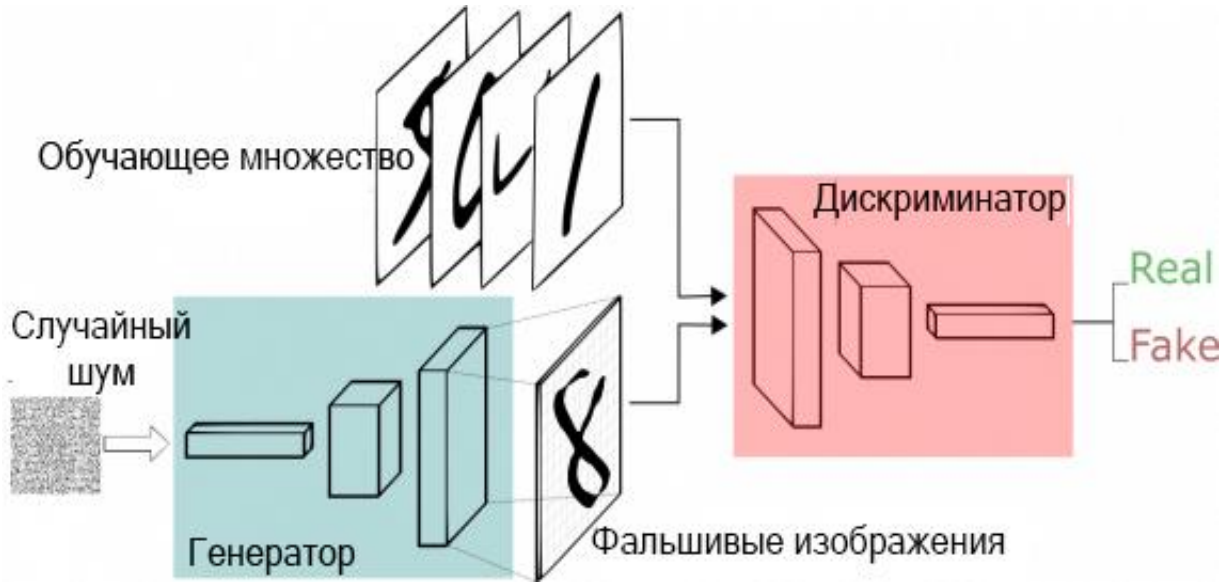


Рис. 4.18. Алгоритм генерации изображений

или как поддельные. Генератор в некотором смысле представляет собой обратную сверточную сеть: хотя стандартный сверточный классификатор принимает изображение и уменьшает его разрешение, чтобы получить вероятность, генератор принимает вектор случайного шума (вектор со случайным набором данных) и преобразует его в изображение. Первый отсеивает данные с помощью методов понижения дискретизации, таких как maxpooling, а второй генерирует новые данные (рис. 4.18).

Обе сети пытаются оптимизировать целевую функцию или функцию потерь. Когда дискриминатор меняет свое поведение, то и генератор меняет, и наоборот.

Полезно сравнить генеративные состязательные сети с другими нейронными сетями, такими как автокодеры (автоэнкодеры) и вариационные автокодеры.

Автокодеры кодируют входные данные в векторы. Они создают скрытое или сжатое представление необработанных данных (черный ящик). Они полезны при уменьшении размерности: вектор, служащий в качестве скрытого представления, сжимает необработанные данные в меньшее количество. Автокодеры могут быть сопряжены с так называемым декодером, который позволяет восстанавливать входные данные на основе их скрытого представления (рис. 4.19).

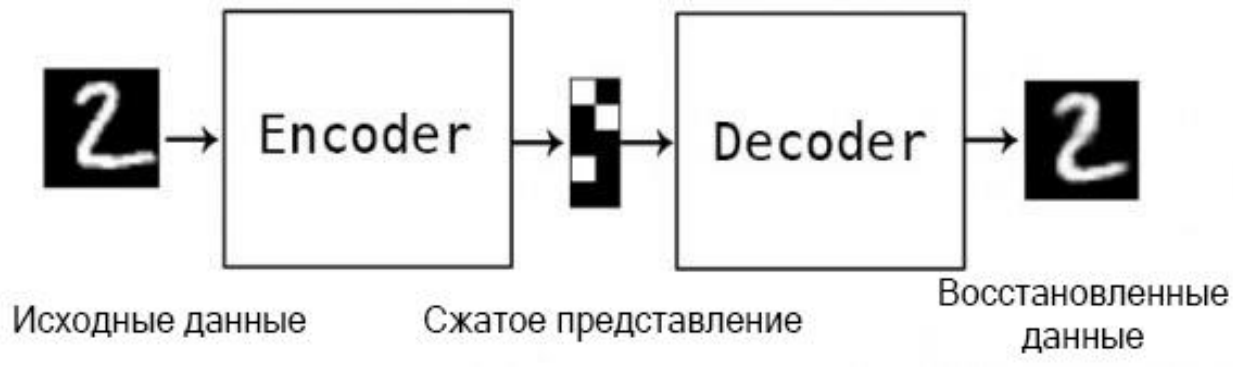


Рис. 4.19. Принцип работы автокодера

Вариационные автокодеры являются генеративным алгоритмом, который добавляет дополнительное ограничение для кодирования входных данных, а именно то, что скрытые представления нормализуются. Вариационные автокодеры способны сжимать данные как автокодеры и синтезировать данные подобно GAN. Однако, в то время как GAN генерируют данные детализовано, изображения, созданные VAE, бывают более размытыми.

Перед тем, как перейти к примеру GAN, отметим несколько важных моментов.

Когда вы тренируете дискриминатор, удерживайте значения генератора постоянными; и когда вы тренируете генератор, удерживайте дискриминатор на одном уровне. Каждый должен тренироваться против статичного противника. Например, генератору это позволит лучше считывать градиент, по которому он должен учиться. Для этого используется параметр модели *trainable = False.*

Каждая часть GAN может одолеть другую. Если дискриминатор слишком хорош, он будет возвращать значения очень близкие к 0 или к 1, так что генератор будет испытывать трудности в чтении градиента. Если генератор слишком хорош, он будет постоянно использовать недостатки дискриминатора, приводящие к неправильным негативам.

GAN требуют много времени на тренировку. На одном GPU тренировка может занимать часы, а на одном CPU – более одного дня.

Итак, перейдем к примеру GAN сети, используемой для генерации цифр, подобных набору MNIST. Напомним, что обучение данной сети может включать несколько сотен эпох до получения удовлетворительного результата.

```
        import random
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

from keras.layers import Input,Convolution2D, MaxPooling2D, Reshape, Flat-
ten, Convolution1D
from keras.models import Model, Sequential
from keras.layers.core import Dense, Dropout
from keras.layers.advanced_activations import LeakyReLU
from keras.datasets import mnist
from keras.optimizers import Adam
from keras import initializers
np.random.seed(10)
# Размерность вектора случайного шума.
random_dim = 1

def load_minst_data():
   # загрузка данных
   (x_train, y_train), (x_test, y_test) = mnist.load_data()
   # нормализация входных данных на интревале[-1, 1]
   x_train = (x_train.astype(np.float32) – 127.5)/127.5
   # преобразуем x_train с формой (60000, 28, 28) в (60000, 784),
         x_train = x_train.reshape(60000, 28,28)
   return (x_train, y_train, x_test, y_test)

def get_optimizer():
   return Adam(lr=0.0002, beta_1=0.5)

        #функция создания модели генератора
def get_generator(optimizer):
   generator = Sequential()
   generator.add(Dense(100, input_dim=random_dim, kernel_initializer=ini-
tializers.RandomNormal (stddev=0.02)))
   generator.add(LeakyReLU(0.2))
   generator.add(Dense(784, activation='tanh'))
   generator.add(Reshape(( 28, 28)))
```

```
    generator.compile(loss='binary_crossentropy', optimizer=optimizer)
    return generator

#функция создания модели дискриминатора
def get_discriminator(optimizer):
    discriminator = Sequential()
    discriminator.add(Convolution1D(32, 3, activation='relu', input_shape=(28,
28)))
    discriminator.add(Dense(200, activation='tanh'))
    discriminator.add(Flatten())
    discriminator.add(Dense(1, activation='sigmoid'))
    discriminator.compile(loss='binary_crossentropy', optimizer=optimizer)
    discriminator.summary()
    return discriminator

        #функция создания модели GAN
def get_gan_network(discriminator, random_dim, generator, optimizer):
    discriminator.trainable = False
    gan_input = Input(shape=(random_dim,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss='binary_crossentropy', optimizer=optimizer)
    return gan

        #функция отрисовки и сохранения сгенерированных изображений
def plot_generated_images(value, epoch, generator, dim=(1, 1), figsize=(28,
28)):
    noise = value
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(1, 28, 28)
    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
```

```
    plt.savefig('gan_generated_'+str(value)+'_'+str(epoch)+'.png')

def train(epochs=1, batch_size=128):
    # Получим данные обучения и тестирования
    x_train, y_train, x_test, y_test = load_minst_data()
    # Разделим тренировочные данные на партии размером 128
        batch_count = int(x_train.shape[0] / batch_size)
    # Создим сеть GAN
            adam = get_optimizer()
    generator = get_generator(adam)
    discriminator = get_discriminator(adam)
    gan = get_gan_network(discriminator, random_dim, generator, adam)
    for e in range(1, epochs+1):
        print('-'*15, 'Epoch %d' % e, '-'*15)
        for _ in range(batch_count):
            # Получим случайный набор входного шума и изображений
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            image_batch = x_train[np.random.randint(0, x_train.shape[0],
size=batch_size)]
            # Сгенерируем ненастоящие изображения MNIST
            generated_images = generator.predict(noise)
            #print(image_batch.shape,generated_images.shape )
            X = np.concatenate([image_batch, generated_images])
            # Метки для сгенерированных и реальных данных
            y_dis = np.zeros(2*batch_size)
            # Одностороннее сглаживание
            y_dis[:batch_size] = 0.9
            #print(X.shape, y_dis.shape)

            # Дискриминатор
            discriminator.trainable = True
            discriminator.train_on_batch(X, y_dis)

            # Генератор
            noise = np.random.normal(0, 1, size=[batch_size, random_dim])
            y_gen = np.ones(batch_size)
            discriminator.trainable = False
```

```
            gan.train_on_batch(noise, y_gen)
        if e == 1 or e % 10 == 0:
            plot_generated_images([random.randint(0,10)],e, generator)
if __name__ == '__main__':
    train(40, 128)
```

Вопросы для закрепления

1. Для чего применяется библиотека scikit-learn?
2. Для чего применяется библиотека NumPy?
3. Для чего применяется библиотека pandas?
4. Опишите процесс классификации объектов по признакам с использованием библиотеки scikit-learn.
5. Для чего применяется библиотека Keras?
6. Что такое регуляризация?
7. Как работают сверточные нейронные сети?
8. Как работают рекуррентные нейронные сети?
9. Что такое регрессионные нейронные сети?
10. Опишите генеративно-состязательные нейронные сети.

5. РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Разработка интеллектуальных систем в настоящее время применяется как в научно-исследовательских целях, так и коммерческих. Если первая категория отличается многоэтапностью, цикличностью, творческим подходом, направленным скорее на процесс анализа и обработки данных, чем на получение конкретного результата, то во второй категории требуется получить законченное решение, обеспечивающее положительный экономический эффект.

Для коммерческих экспертных и интеллектуальных систем можно выделить следующие основные этапы.

Этап 1: Анализ проблем

На данном этапе проводятся необходимые мероприятия по анализу и изучению предметной области, изучению имеющихся в ней задач и проблем, поиску экспертной группой, определению набора возможных решений проблемы, подготовке плана разработки системы.

Среди этих мероприятий выбор проблемы, для решения которой планируется разработка интеллектуальной системы, является ключевым. Во-первых, необходимо обеспечить наличие эксперта, способного объяснить как эту проблему возможно решить. При его отсутствии разработчикам будет затруднительно определить направление решения проблемы. Кроме того, решая не те проблемы, разработчики изначально увеличивают стоимость системы и время её разработки, приходя в конце разработки к осознанию зря потраченного времени и сил.

На данном этапе нужно четко понимать, требуется ли интеллектуальная система для решения задачи или классический алгоритмический подход также подходит. В последнем случае решение будет получено более точно, быстро.

Разработка интеллектуальной системы не исключает необходимости в создании и наполнении баз данных, программном обеспечении по сбору, хранению, обработке и передаче данных. Интеллектуальная система без соответствующего окружения может быть просто непригодна для использования конечным пользователем. С другой стороны, всегда нужно правильно оценить её необходимость в рамках общей информационной среды. Если решение задачи зависит от правильной интерпретации знаний или фактов, отличается субъективизмом, то появляется и потребность в интеллектуальных системах.

Для упрощения можно сформулировать некоторые признаки, приводящие к необходимости реализации интеллектуальных или экспертных систем:

- нехватка специалистов или их высокая загруженность;
- невозможность освоения одним специалистов всего объёма знаний, потребность в большом количестве узкоспециализированных экспертов;
- низкая производительность эксперта из-за необходимости анализа большего объёма данных, что невозможно в ходе рабочего процесса;
- значительные расхождения в оценках и мнениях специалистов;
- острая конкурентная ситуация, требующая перехода к следующему уровню анализа и обработки данных.

Следующие проблемы и задачи могут быть решены в интеллектуальных системах:

- узкоспециализированные;
- не зависят от общепризнанных знаний и соображений, не выводятся логически;
- имеют для эксперта среднюю сложность;
- условия задачи заданы пользователем;
- решение задачи можно оценить.

Разработка любой экспертной системы включает сбор данных от эксперта. Обычно эксперт и разработчик (инженер по знаниям) работают в паре: первый предоставляет знания, второй – обрабатывает и вносит их в систему. Важно на данном этапе согласовать общую форму представления знаний, их структуру, наладить взаимодействие, т.к. работа между экспертом и инженером может продолжаться не один месяц.

Зачастую на данном этапе осуществляется поиск и анализ существующих альтернатив, возможность применения уже разработанных экспертных систем. Это позволяет дополнительно оценить целесообразность разработки новой системы, её коммерческую оправданность. Если аналогов не найдено или они не удовлетворяют поставленным условиям, то осуществляется анализ имеющихся инструментов и средств для разработки новой системы. Рассчитываются все сопутствующие расходы, например, стоимость оплаты труда разработчиков и экспертов, закупаемое оборудование и т.д.

Помимо расходов на данном этапе оценивается прибыль от внедрения и использования экспертной системы, тот положительный эффект, что планируется получить после её разработки.

Итак, инженер по знаниям и команда разработчиков в итоге должны убедиться, что задача решаема с помощью интеллектуальной системы, определены необходимые инструменты для её реализации, найден эксперт в данной предметной области, разработка системы экономически выгодна. Если все условия выполнены, составляется план разработки, включающий все этапы, расходы, а также ожидаемые результаты.

Этап 2: Разработка прототипа

На следующем этапе формируется прототип экспертной системы, такая её версия, обеспечивающая возможность проверки правильности анализа, обработки и вывода знаний. Прототип ограничен функционально, может не содержать весь массив знаний. Он используется в том числе для того, чтобы привлечь эксперта к реализации системы и проверке её работоспособности.

В разработке прототипа можно выделить 6 ключевых стадий, представленных на рис. 5.1.

Идентификация проблемы. На данной стадии происходит уточнение проблемы, планирование, определение необходимых ресурсов, источников знаний, анализ существующих экспертных систем, определение подзадач и возможных инструментов их решения.

Извлечение знаний. Стадия включает перенос знаний от экспертов к инженерам по знаниям путем диалогов, дискуссий, лекций. Результатом стадии является получение инженерами комплексного понимания протекающих в предметной области процессов и способов принятия решений в них.

Структурирование. Определение структуры знаний, терминологии, закономерностей и соотношений между понятиями, объектами, выявление структуры процессов, входной и выходной информации. Таким образом, в ходе данной стадии разрабатывается неформальное описание знаний.

Формализация. На основе полученного на прошлой стадии описания строится формализованное представление знаний с использованием логических методов и различных способов представления знаний (фрейм, продукционные модели, семантические сети, объектно-ориентированные модели и т.д.). В результате формируется формализованная база знаний, советующая реальным процессам и позволяющая на её основе получить программный продукт.

Реализация. Стадия включает программную реализацию прототипа экспертной системы на основе полученной базы знаний на выбранном языке программирования.

Рис. 5.1. Этапы разработки прототипа

Тестирование. Разработанный прототип проверяется на соответствие тем задачам и требованиям, определенным ранее. Определяется адекватность и корректность работы прототипа и базы знаний.

Этап 3: Развитие прототипа до промышленной экспертной системы

Разработанный и протестированный прототип оценивается экспертом и инженером по знаниям. Найденные неточности, ошибки в функционировании исправляются, в архитектуру вносятся необходимые коррективы.

5.1. Переход от прототипа к промышленной экспертной системе

Демонстрационный прототип	Включает несколько десятков правил или понятий
Исследовательский прототип	Содержит несколько сотен правил или понятий
Действующий прототип	Система работоспособна и надежно решает поставленные задачи, но не отвечает всем требованиям по производительности
Промышленная система	Система работоспособна и отвечает требованиям по производительности и качеству
Коммерческая система	Промышленная система, включающая документацию, справочную систему, готовая для широкого коммерческого потребления

Иногда для лучшего контроля процесса разработки экспертной системы добавляются дополнительные этапы между прототипом и промышленной системой, например, демонстрационный прототип или действующий прототип (табл. 5.1).

Данный этап включает реализацию и адаптацию интерфейса для последующего взаимодействия с пользователями и экспертом. Реализация такого интерфейса позволяет эксперту самостоятельно вносить данные в систему после предварительного ознакомления. Интерфейс должен содержать подсказки и инструкции, а также помогать пользователю, указывая на неправильные действия или ошибочно введенные данные.

Этап 4: Оценка системы

Разработанная промышленная система проверяется на соответствие выбранным критериям эффективности, точности, качества, полезности. Правильность работы базы знаний контролируется дополнительно привлеченными экспертами.

Пользователи могут оценивать систему с позиции понятности и удобства работы с интерфейсом. Эксперты рассматривают систему с позиции соответствия принятых системой решений собственной точке зрения, особенностям предметной области. Разработчики тестируют производительность, надежность, защищенность.

Этап 5: Интеграция системы

Успешно реализованная система в дальнейшем должна быть интегрирована с общей информационной средой предприятия или организации. В

ходе такой интеграции реализуются связи со сторонними модулями и системами с привлечением инженера по знаниям и команд разработчиков, задействованных в реализации информационной среды.

После интеграции системы инженер по знаниям контролирует процесс обучения персонала правильному взаимодействию с системой, её эксплуатации и поддержке.

Этап 6: Поддержка системы

Экспертная система должна поддерживаться и развиваться, а база знаний – быть актуальной. Появление новых технологий и инструментов, позволяющих повысить эффективность работы экспертной системы, также может привести к её модернизации и реконструкции, если это коммерчески выгодно.

Вопросы для закрепления

1. Назовите основные этапы разработки интеллектуальных систем.
2. Какие действия входят в этап анализа проблем?
3. Опишите основные стадии разработки прототипа системы.
4. Опишите суть перехода от прототипа до промышленной системы
5. Назовите типы прототипов систем.
6. Каким способом можно оценить промышленную экспертную систему?
7. В чем заключается интеграция системы?
8. Что является завершающим этапом разработки интеллектуальных систем.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО МАШИННОМУ ОБУЧЕНИЮ

Лабораторная работа № 1. Классификация численных данных с использованием scikit-learn

1) Сформировать набор исходных данных вида: вектор входных параметров (2 – 5 значений признаков), выходное значение класса (численное обозначение).

2) Осуществить загрузку и обработку исходных данных.

3) Реализовать алгоритм машинного обучения для классификации исходных данных.

4) Осуществить проверку правильности классификации на контрольных данных, не присутствующих в обучающем наборе.

Дополнительное задание. Осуществить визуализацию разделения объектов по классам.

Лабораторная работа № 2. Классификация численных данных с использованием Keras

1) Использовать исходные данные лабораторной работы № 1.

2) Осуществить загрузку и обработку исходных данных.

3) Реализовать алгоритм машинного обучения для классификации исходных данных.

4) Осуществить проверку правильности классификации на контрольных данных, не присутствующих в обучающем наборе.

Дополнительное задание. Осуществить сравнение точности алгоритмов машинного обучения лабораторных работ № 1 и № 2.

Лабораторная работа № 3. Классификация изображений с использованием Keras

1) Подготовить исходных набор изображений для классификации (разрешается использовать существующие базы данных изображений).

2) Осуществить загрузку и обработку исходных данных.

3) Реализовать алгоритм машинного обучения для классификации исходных данных.

4) Осуществить проверку правильности классификации на контрольных данных, не присутствующих в обучающем наборе.

Дополнительное задание. Осуществить сравнение точности разработанной модели нейронной сети с существующими (VGG16, VGG19, GoogleNet, ResNet и т.д.).

ЗАКЛЮЧЕНИЕ

В рамках данного учебного пособия рассмотрены основные подходы к системному анализу и обработке информации в интеллектуальных системах. В настоящее время наиболее перспективными методами обработки данных являются применение алгоритмов машинного обучения и нейронных сетей.

В пособии рассмотрены основные типы интеллектуальных систем, история их развития, способы классификации. Представлены способы представления знаний в экспертных системах с использованием баз знаний, семантических сетей, фреймов и продукционных моделей.

Современные и эффективные интеллектуальные системы базируются на применении методов машинного обучения, поэтому дальше рассмотрены теоретические и практические основы их применения в интеллектуальных системах. Даны основные понятия в теории нейронных сетей, а также принципы их функционирования.

На примере двух популярных библиотек Python рассмотрены задачи классификации и обработки данных. На первом этапе мы используем библиотеку машинного обучения scikit-learn для классификации данных по набору признаков. Далее рассматривается библиотека Keras, в рамках которой рассматриваются многослойные, сверточные, регрессионные и генеративные состязательные нейронные сети. Благодаря представленным примерам программного кода рассмотренные теоретические аспекты применения нейронных сетей могут быть сразу же проверены на практике.

Предлагаемый в пособии лабораторный практикум позволит выработать необходимый набор навыков и умений по решению задач обработки данных с использованием технологий машинного обучения.

Пособие будет полезно как студентам и магистрантам направления «Информатика и вычислительная техника», так и специалистам в области информационных технологий, так как позволяет сформировать базовые знания в области интеллектуальных систем, искусственного интеллекта, нейронных сетей и их применения для решения задач обработки данных.

СПИСОК ЛИТЕРАТУРЫ

1. Дошина, А. Д. Экспертная система. Классификация. Обзор существующих экспертных систем // Молодой ученый. – 2016. – № 21. – С. 756 – 758.
2. Мюллер, А. Введение в машинное обучение с помощью Python // Руководство для специалистов по работе с данными / А. Мюллер, С. Гвидо. – СПб. : ООО Альфакнига, 2017.
3. Джулли А., Пал С. Библиотека Keras-инструмент глубокого обучения / А. Джулли, С. Пал. – М. : ДМК Пресс, 2018.
4. Барсегян, А. А. Методы и модели анализа данных: OLAP И Data Mining / А. А. Барсегян, М. С. Куприянов, В. В. Степаненко, И. И. Холод. – СПб. : BHV, 2004. – 331 с.
5. Янбекова, Г. Б. Эволюция машинного перевода / Г. Б. Янбекова, З. В. Галимзянова, Ф. Б. Ситдикова // Научные революции: сущность и роль в развитии науки и техники. – 2018. – С. 53.
6. Краснянский, М. Н. Сравнительный анализ методов машинного обучения для решения задачи классификации документов научно-образовательного учреждения / М. Н. Краснянский, А. Д. Обухов, Е. М. Соломатина, А. А. Воякина // Вестник ВГУ, Серия: Системный анализ и информационные технологии. – 2018. – № 3. – С. 173 – 182.
7. Miller S. Mind: How to Build a Neural Network (Part One) // Retrieved from. – 2015.
8. Van Veen, F. & Leijnen, S. (2019). The Neural Network Zoo. – URL : http://www.asimovinstitute.org/neural-network-zoo

ОГЛАВЛЕНИЕ

Учебное электронное издание

ОБУХОВ Артем Дмитриевич
КОРОБОВА Ирина Львовна

СИСТЕМНЫЙ АНАЛИЗ И ОБРАБОТКА ИНФОРМАЦИИ В ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ

Учебное пособие

Редактирование Е. С. Мордасовой
Инженер по компьютерному макетированию И. В. Евсеева

ISBN 978-5-8265-2217-2

Подписано к использованию 12.03.2020.
Тираж 50 экз. Заказ № 29

Издательский центр ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 14
Тел. 8(4752) 63-81-08
E-mail: izdatelstvo@tstu.ru