Содержание

Предисловие

Для нас, членов команды AlphaGo, история этого алгоритма стала главным приключением всей жизни. Все началось, как это часто бывает, с небольшого шага – обучения простой сверточной нейронной сети на записях партий в го, сыгранных сильными игроками-людьми. Это привело к кардинальным прорывам в области машинного обучения, а также подарило нам несколько незабываемых событий, включая матчи против таких грозных профессиональных игроков, как Фань Хуэй, Ли Седоль и Ки Джи. Мы гордимся тем, что эти матчи не только повлияли на манеру игры в го по всему миру, но и привлекли внимание множества людей к теме искусственного интеллекта.

Но почему, спросите вы, нас должны интересовать игры? Подобно тому как дети используют игры для изучения тех или иных аспектов реального мира, исследователи в области машинного обучения используют их для подготовки программ-агентов. В этом смысле проект AlphaGo является частью стратегии компании DeepMind по использованию игр в качестве симулированных микрокосмов реальности. Это помогает нам развивать область искусственного интеллекта и тренировать обучающихся агентов, чтобы в будущем создавать интеллектуальные системы, способные решать самые сложные мировые проблемы.

Работа алгоритма AlphaGo напоминает два режима мышления, которые нобелевский лауреат Даниэль Канеман описал в своей книге «Думай медленно, решай быстро», посвященной вопросам человеческого познания. В случае с AlphaGo аналогом медленного режима мышления является алгоритм планирования, называемый *поиском по дереву методом Монте-Карло*, который просчитывает последовательность игровых состояний, начиная с заданной позиции, путем расширения дерева игры, включающего возможные будущие ходы и действия противника. Однако при наличии около 10^{170} (1 со 170 нулями) возможных игровых позиций просчет всех последовательностей оказывается невозможным. Чтобы обойти эту проблему и сократить пространство поиска, мы объединили алгоритм поиска по дереву методом Монте-Карло с компонентом *глубокого обучения*, состоящего из двух нейронных сетей, способных оценивать вероятность победы каждого из игроков и выбирать наиболее перспективные ходы.

В более поздней версии алгоритма, AlphaZero, используются принципы *обучения с подкреплением*, что позволяет программе играть против самой себя, не полагаясь на записи партий, сыгранных человеком. Этот алгоритм с нуля обучился игре в го (а также в шахматы и сёги), часто обнаруживая (и в дальнейшем отбрасывая) многие из стратегий, разработанных за сотни лет игроками-людьми, и создав множество собственных уникальных стратегий.

Авторы этой книги Макс Памперла и Кевин Фергюсон станут вашими проводниками в увлекательном путешествии от AlphaGo до более поздних версий этого алгоритма. В ходе чтения данного руководства вы не только реализуете движок для игры в го в стиле AlphaGo, но и получите отличное практическое понимание некоторых из основополагающих строительных блоков современных алгоритмов ИИ: поиска по дереву методом Монте-Карло, глубокого обучения и обуче-

ния с подкреплением. Авторы искусно объединили эти темы, используя игру го в качестве захватывающего и доступного для понимания примера. В дополнение к этому вы изучите основы одной из самых красивых и сложных игр, когда-либо изобретенных человечеством.

Кроме того, эта книга призывает вас с самого начала приступить к созданию работающего го-бота, который будет постепенно эволюционировать от алгоритма, выбирающего ходы совершенно случайным образом, до сложного самообучающегося ИИ для игры в го. Помимо исчерпывающих объяснений основополагающих концепций, авторы предоставили исполняемый код на языке Python. Кроме того, они не пренебрегли такими темами, как форматы данных, развертывание бота и облачные вычисления, необходимыми для обеспечения работоспособности программы для игры в го.

Таким образом, книга «Глубокое обучение и игра в го» представляет собой легко читаемое и увлекательное введение в тему искусственного интеллекта и машинного обучения. Объединяя в себе некоторые из самых захватывающих этапов развития области искусственного интеллекта, она превращается в интереснейший вводный курс по данному предмету. Любой читатель, прошедший этот путь от начала до конца, приобретет необходимые знания для понимания и создания современных систем ИИ, применяемых в ситуациях, требующих сочетания «быстрого» сопоставления образов с «медленным» планированием, которые являются аналогом двух режимов мышления, необходимых для осуществления базового процесса познания.

– *Тор Грепель*, научный сотрудник компании DeepMind,
от имени команды AlphaGo компании DeepMind

Введение

Когда в начале 2016 года о программе AlphaGo заговорили в новостях, мы были чрезвычайно взволнованы этим новаторским достижением в сфере компьютерных игр. В то время считалось, что до создания искусственного интеллекта для игры в го, способного играть на человеческом уровне, оставалось не менее 10 лет. Мы тщательно следили за играми и были готовы пожертвовать сном ради того, чтобы посмотреть трансляции матчей в прямом эфире. Но мы были в хорошей компании – миллионы людей по всему миру были зачарованы играми AlphaGo против Фань Хуэя, Ли Седоля, Ки Цжи и других профессиональных игроков.

Вскоре после появления этого алгоритма мы приступили к работе над небольшой библиотекой с открытым исходным кодом, которую назвали BetaGo (**github.com/maxpumperla/betago**), чтобы посмотреть, сможем ли мы самостоятельно реализовать некоторые базовые механизмы, лежащие в основе алгоритма AlphaGo. Идея BetaGo состояла в том, чтобы продемонстрировать интересующимся разработчикам некоторые из методов, использующихся в этой программе. Несмотря на то что наших ресурсов (времени, вычислительной мощности или интеллекта) было недостаточно, чтобы конкурировать с невероятным достижением компании DeepMind, мы получили огромное удовольствие в процессе создания собственного го-бота.

С тех пор нам много раз предоставлялась возможность рассказать об ИИ для игры в го. Поскольку мы являемся не только поклонниками этой игры, но и практиками машинного обучения, мы иногда забывали о том, как мало широкая публика могла вынести из событий, за которыми мы так пристально следили. Ирония заключалась в том, что наблюдающие за матчами миллионы людей, по-видимому, делились на две группы:

- те, кто понимает и любит игру го, но мало знает о машинном обучении;
- те, кто понимает и ценит машинное обучение, но практически незнаком с правилами игры в го.

Для далекого от этих тем человека обе сферы могут казаться одинаково туманными, сложными и трудными для освоения. Несмотря на то что в последние годы все большее количество разработчиков программного обеспечения задействует методы машинного обучения и, в частности, *глубокого обучения*, игра в го остается в значительной степени неизвестной многим жителям стран Запада. Мы считаем это весьма прискорбным и искренне надеемся, что данная книга позволит сблизить две вышеупомянутые группы людей.

Мы убеждены в том, что использованию принципов, лежащих в основе алгоритма AlphaGo, можно на практике обучить широкую аудиторию разработчиков программного обеспечения. Наслаждение игрой го и ее понимание приходит в процессе игры и экспериментов. То же самое можно сказать о машинном обучении и любой другой дисциплине.

Если в ходе изучения этой книги вы проникнетесь энтузиазмом в отношении игры го или машинного обучения (надеемся, и того, и другого!), мы будем считать, что выполнили свою задачу. Если, помимо этого, вы научитесь создавать и развертывать боты для игры в го, а также проводить собственные эксперименты, вам станет доступно множество других интересных ИИ-приложений. Наслаждайтесь путешествием!

Благодарности

Мы хотели бы поблагодарить всех сотрудников издательства Manning, сделавших публикацию этой книги возможной. В частности, мы благодарим наших неутомимых редакторов: Марину Майклз за то, что она помогла нам преодолеть первые 80 % пути, и Дженни Стаут за помощь в преодолении вторых 80 %. Выражаем благодарность нашему техническому редактору Чарльзу Федуке и техническому корректору Тане Вилке за проверку кода.

Благодарим всех рецензентов, предоставивших ценные отзывы: Александра Ерофеева, Алессандро Пузиелли, Алекса Орланди, Бурка Хуфнагеля, Крейга С. Коннелла, Даниэля Береца, Дениса Крайса, Доминго Салазара, Хельмута Хаушильда, Джеймса А. Худа, Джасбу Симпсона, Джин Лазароу, Мартина Мёллера, Скарбиникса Педерсена, Матиаса Поллигкайта, Ната Луенгнарюмитчая, Пьерлуиджи Рити, Сэма Де Костера, Шона Линдсея, Тайлера Коваллиса и Урсина Стосса.

Также спасибо всем, кто экспериментировал или участвовал в разработке нашего проекта BetaGo, особенно Эллиоту Герчаку и Кристоферу Мэлону.

Наконец, благодарим всех, кто когда-либо пытался научить компьютер играть в го и поделился результатами своих исследований.

Я хотел бы поблагодарить Карли за ее терпение и поддержку, а также папу и Джиллиан за то, что научили меня писать.

– Кевин Фергюсон

Особая благодарность Кевину за помощь в разъяснении материала, Андреасу за множество плодотворных дискуссий и Энн за ее постоянную поддержку.

– Макс Памперла

Об авторах

Макс Памперла является специалистом по работе с данными и инженером, занимающимся глубоким обучением в компании-разработчике ИИ-систем Skymind. Также он является сооснователем платформы глубокого обучения **aetros.com**.

Кевин Фергюсон на протяжении 18 лет работал в области создания распределенных систем и анализа данных. Он является специалистом по анализу данных в компании Honor и имеет опыт работы в таких компаниях, как Google и Meebo. Вместе Макс и Кевин разработали BetaGo, один из очень немногих го-ботов с открытым исходным кодом, созданных на языке Python.

Об иллюстрации на обложке

На обложке книги «Глубокое обучение и игра в го» изображен император Монтоку, правивший Японией с 850 по 858 год. Этот портрет был написан акварелью на шелке неизвестным художником. В 2006 году его репродукция была включена в раздел «Императоры и императрицы прошлого» японского исторического журнала *Bessatsu Rekishi Dokuhon*.

Подобные изображения напоминают нам об уникальности и индивидуальности древних городов и регионов мира. В то время по одежде можно было однозначно сказать, к какому из двух городов, разделенных несколькими десятками километров, принадлежит тот или иной человек.

С тех пор дресс-код изменился, и от былой самобытности разных уголков мира не осталось и следа. Сейчас жителей разных континентов бывает трудно отличить друг от друга. Возможно, мы променяли культурное и визуальное разнообразие на более разнообразную и интересную личную и интеллектуальную жизнь, или на более разнообразную жизнь в плане технологий. Мы, сотрудники издательства Manning, стремимся дополнить изобретательность, новаторство и увлеченность, пробуждаемые книгами по компьютерной тематике, обложками, отражающими богатый и разнообразный местный колорит прошлого.

Об этой книге

Книга «Глубокое обучение и игра в го» призвана познакомить читателя с современными концепциями машинного обучения на практическом примере создания искусственного интеллекта для игры в го. К концу главы 3 у вас будет готовая, хоть и очень примитивная программа, играющая в эту игру. В каждой следующей главе будет представлен новый метод улучшения ИИ вашего бота. В ходе экспериментов вы узнаете о преимуществах и недостатках каждого из этих методов. А в последних главах мы покажем, как алгоритмы AlphaGo и AlphaGo Zero позволяют интегрировать все представленные в книге методы в невероятно мощный ИИ.

Эта книга рассчитана на разработчиков программного обеспечения, желающих приступить к экспериментам в области машинного обучения и предпочитающих практический подход математическому. Мы предполагаем, что вы обладаете практическим опытом программирования на языке Python, однако описанные алгоритмы можно реализовать на любом современном языке. Мы не предполагаем, что вы знакомы с игрой го. Если вы предпочитаете шахматы или какую-либо другую игру подобного рода, то сможете адаптировать к ней большинство описанных методов. Если же вы сами являетесь игроком в го, то получите массу удовольствия, наблюдая за прогрессом вашего бота!

Книга состоит из трех частей, включающих 14 глав и 5 приложений.

Часть I «Основы» знакомит читателя с основными концепциями, подробно описанными в остальных частях книги.

- Глава 1 «На пути к глубокому обучению» предоставляет беглый высокоуровневый обзор таких дисциплин, как искусственный интеллект, машинное обучение и глубокое обучение. В этой главе мы объясняем, как они связаны между собой и что вы можете и не можете сделать с помощью методов, применяемых в этих областях.
- Глава 2 «Игра го как проблема машинного обучения» знакомит читателя с правилами игры го и объясняет, чему мы намерены обучить компьютер.
- В главе 3 «Реализация первого бота для игры в го» мы реализуем доску для игры в го, раскладываем камни и играем в игры с помощью средств языка Python. В конце этой главы вы сможете создать простейший ИИ для игры в го.

Часть II «Машинное обучение и игровой ИИ» посвящена техническим и методологическим основам создания мощного ИИ. В частности, там описаны три метода, которые очень эффективно используются в алгоритме AlphaGo: *поиск по дереву*, *нейронные сети* и *обучение с подкреплением*.

Поиск по дереву

- Глава 4 «Игры и поиск по дереву» содержит обзор алгоритмов, которые отвечают за поиск и оценку последовательностей игрового процесса. Мы начнем с простого поиска путем минимаксного перебора, а затем перейдем к таким сложным алгоритмам, как альфа-бета-отсечение и метод Монте-Карло.

Нейронные сети

- Глава 5 «Знакомство с нейронными сетями» представляет собой практическое введение в тему искусственных нейронных сетей. Вы научитесь предсказывать рукописные цифры, создав нейронную сеть с нуля средствами Python.
- Глава 6 «Проектирование нейронной сети для данных го» объясняет сходство данных го с данными изображения, а также представляет сверточные нейронные сети для прогнозирования ходов. В этой главе мы начнем использовать для построения моделей популярную библиотеку глубокого обучения Keras.
- В главе 7 «Глубокое обучение бота на основе данных» мы применим знания, полученные в предыдущих двух главах, чтобы создать бота для игры в го на основе глубоких нейронных сетей. Мы обучим этого бота на фактических игровых данных сильных любительских партий и поговорим об ограничениях данного подхода.
- В главе 8 «Использование ботов» вы узнаете о том, как обеспечить игру ботов с противниками-людьми через пользовательский интерфейс. Кроме того, вы узнаете, как организовать игру ботов с другими ботами локально и на сервере для игры в го.

Обучение с подкреплением

- Глава 9 «Обучение на практике: обучение с подкреплением» посвящена основам обучения с подкреплением и описанию способов его использования для игры в го с самим собой.
- Глава 10 «Обучение с подкреплением и градиенты политики» знакомит читателя с градиентами политики, жизненно важным методом для повышения эффективности прогнозирования ходов, о котором мы говорили в главе 7.
- Глава 11 «Обучение с подкреплением и методы на основе значений» демонстрирует процесс оценки состояний доски с помощью так называемых методов на основе значений, которые являются мощным инструментом в сочетании с поиском по дереву, описанным в главе 4.
- Глава 12 «Обучение с подкреплением и методы типа “актор–критик”» знакомит читателя с методами, позволяющими предсказать долгосрочное значение конкретной позиции на доске и конкретного следующего хода для более эффективного выбора ходов.

В последней части III «Больше, чем сумма всех частей» мы соберем все описанные ранее строительные блоки, чтобы создать приложение, принципом работы напоминающее программу AlphaGo.

- Глава 13 «AlphaGo: Собираем все вместе», по сути, является технической и математической кульминацией этой книги. В ней мы обсуждаем, как

тренировка нейронной сети на данных го (главы 5–7) и последующая игра с самим собой (главы 8–11) в сочетании с поиском по дереву (глава 4) позволяют создать бота для игры в го сверхчеловеческого уровня.

- Глава 14 «AlphaGo Zero: Интеграция поиска по дереву и обучения с подкреплением» посвящена описанию современного состояния ИИ для настольных игр. В ней мы подробно поговорим об инновационной комбинации поиска по дереву и обучения с подкреплением, лежащей в основе программы AlphaGo Zero.

В приложениях мы рассмотрим следующие темы:

- в приложении А «Математические основы» кратко изложены некоторые базовые концепции линейной алгебры и математического анализа и приведены способы представления некоторых структур линейной алгебры с помощью библиотеки Python NumPy;
- в приложении Б «Алгоритм обратного распространения ошибки» более подробно объясняется процедура обучения большинства нейронных сетей, которые мы начнем использовать в главе 5;
- в приложении В «Программы и серверы для игры го» перечислены некоторые ресурсы для читателей, желающих больше узнать о данной игре;
- приложение Г «Обучение и развертывание ботов с помощью Amazon Web Services» представляет собой краткое руководство по запуску вашего бота на облачном сервере Amazon;
- в приложении Д «Отправка бота на онлайн-сервер для игры в го» говорится о том, как подключить бота к популярному серверу для игры в го, где вы можете протестировать его в игре с игроками со всего мира.

Структура книги схематически представлена на рисунке на следующей странице.

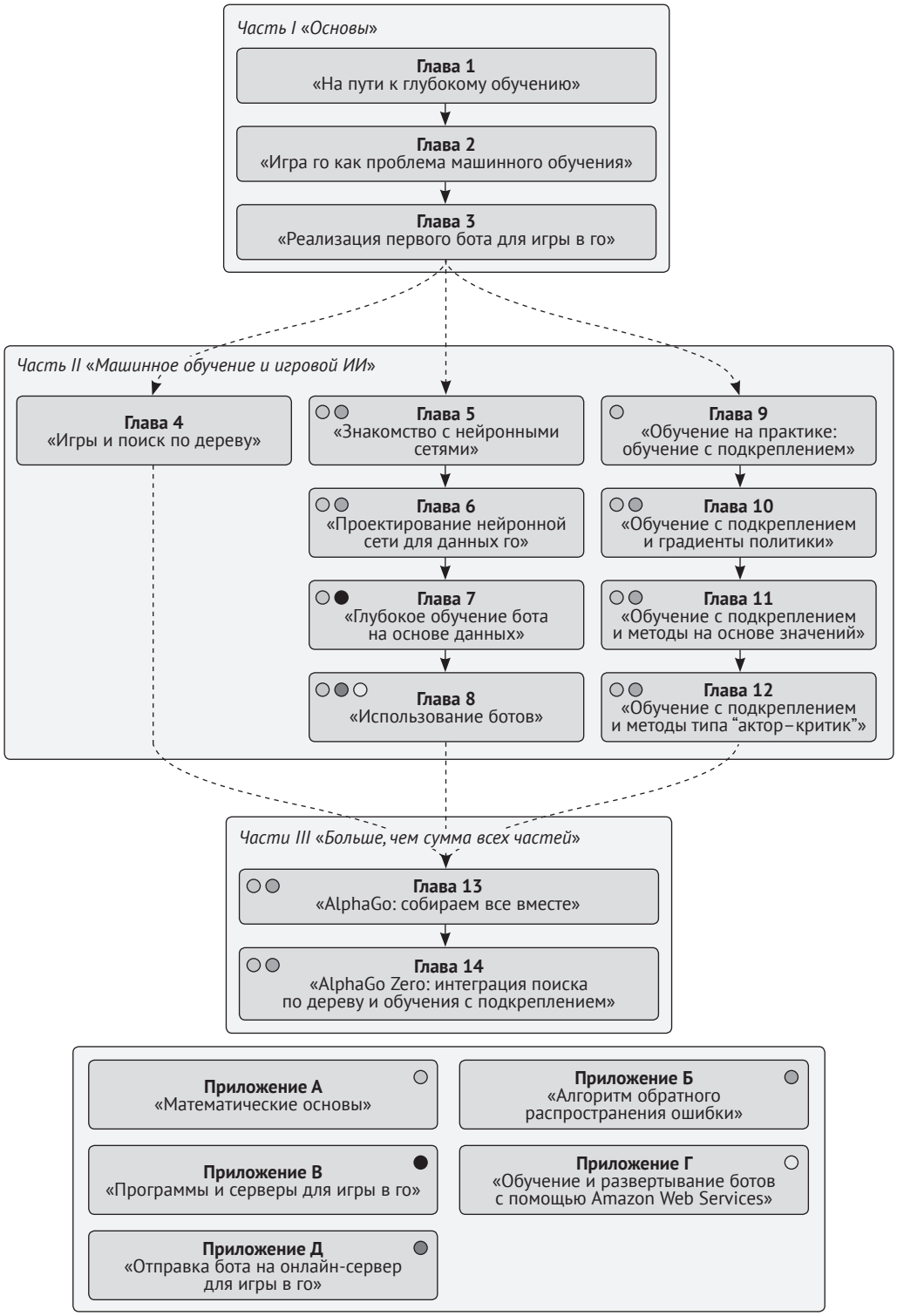
Эта книга содержит много примеров исходного кода как в листингах с пронумерованными строками, так и непосредственно в тексте. В обоих случаях код отформатирован моноширинным шрифтом, что позволяет отличить его от обычного текста. **Полужирным моноширинным** шрифтом выделяются изменения в коде, например новая функция, добавленная в уже существующую строку.

Во многих случаях мы переформатировали исходный код, добавив разрывы строк и изменив отступы с учетом доступного места на странице. В редких случаях даже этого оказалось недостаточно, поэтому мы включили в листинги символы продолжения строки (➥). Кроме того, мы убрали из листинга комментарии, относящиеся к коду, описанному в тексте. Многие листинги сопровождаются аннотациями, содержащими описание важных понятий.

Все примеры кода, а также некоторые дополнительные фрагменты связующего кода вы можете найти на сайте GitHub по адресу: **github.com/maxpumperla/deep_learning_and_the_game_of_go**.

Такая пиктограмма обозначает совет или рекомендацию.

Такая пиктограмма обозначает указание или примечание общего характера.


Часть I «Основы»
Глава 1
«На пути к глубокому обучению»
Глава 2
«Игра го как проблема машинного обучения»
Глава 3
«Реализация первого бота для игры в го»
Часть II «Машинное обучение и игровой ИИ»
Глава 4
«Игры и поиск по дереву»
Глава 5
«Знакомство с нейронными сетями»
Глава 6
«Проектирование нейронной сети для данных го»
Глава 7
«Глубокое обучение бота на основе данных»
Глава 8
«Использование ботов»
Глава 9
«Обучение на практике: обучение с подкреплением»
Глава 10
«Обучение с подкреплением и градиенты политики»
Глава 11
«Обучение с подкреплением и методы на основе значений»
Глава 12
«Обучение с подкреплением и методы типа "актор–критик"»
Части III «Больше, чем сумма всех частей»
Глава 13
«AlphaGo: собираем все вместе»
Глава 14
«AlphaGo Zero: интеграция поиска по дереву и обучения с подкреплением»
Приложение А
«Математические основы»
Приложение Б
«Алгоритм обратного распространения ошибки»
Приложение В
«Программы и серверы для игры в го»
Приложение Г
«Обучение и развертывание ботов с помощью Amazon Web Services»
Приложение Д
«Отправка бота на онлайн-сервер для игры в го»

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте **www.dmkpress.com** или **www.дмк.рф** на странице с описанием соответствующей книги.

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Часть I

ОСНОВЫ

Что такое машинное обучение? Что собой представляет игра в го, и почему она является важной вехой на пути развития игрового ИИ? Чем обучение компьютера игре в го отличается от его обучения игре в шахматы или шашки?

В этой части мы ответим на все эти вопросы, после чего вы создадите гибкую библиотеку игровой логики го, которая станет основой для упражнений, приведенных в остальной части книги.

Глава 1

На пути к глубокому обучению: введение в машинное обучение

В этой главе:

- машинное обучение и его отличия от традиционного программирования;
- проблемы, которые можно и нельзя решить с помощью машинного обучения;
- связь машинного обучения и искусственного интеллекта;
- структура системы машинного обучения;
- дисциплины, относящиеся к области машинного обучения.

На протяжении всего срока существования компьютеров программисты интересовались *искусственным интеллектом* (ИИ), который позволил бы реализовать человеческое поведение с помощью компьютера. С давних пор популярным предметом исследований в области ИИ являются игры. В эпоху персональных компьютеров ИИ побеждал людей в игре в шашки, нарды, шахматы и почти во все остальные классические настольные игры. Однако на протяжении многих десятилетий древняя стратегическая игра го оставалась недостижимой для компьютеров. В 2016 году ИИ AlphaGo компании Google DeepMind бросил вызов 14-кратному чемпиону мира Ли Седолю и одержал победу в четырех играх из пяти. Следующая версия AlphaGo оказалась совершенно недосягаемой для игроков-людей: она выиграла 60 игр подряд, победив практически всех известных игроков в го.

Прорыв AlphaGo заключался в расширении классических алгоритмов ИИ с помощью машинного обучения. В частности, программа AlphaGo использовала современные методы *глубокого обучения* – алгоритмы, способные организовывать необработанные данные в полезные уровни абстракции. Применение этих методов не ограничивается играми. Глубокое обучение используется в приложениях для распознавания изображений и речи, системах машинного перевода и управления роботами. Изучение основ глубокого обучения поможет вам разобраться в принципах работы всех этих приложений.

Зачем посвящать целую книгу компьютеру, играющему в го? Авторы не сошли с ума, дело в том, что, в отличие от ИИ для игры в шахматы или нарды, мощный ИИ для игры в го требует применения методов глубокого обучения. Первоклассный шахматный движок вроде Stockfish содержит обширные знания о логике

игры в шахматы. Для создания чего-то подобного вы должны обладать определенными знаниями об игре. Глубокое обучение позволяет научить компьютер подражать сильным игрокам в го, даже если вы не понимаете логику их действий. И эта мощная техника позволяет создавать всевозможные виды приложений как в игровом, так и в реальном мире.

ИИ для игры в шахматы и шашки призваны более точно прогнозировать игру по сравнению с людьми. Применение данного подхода к игре в го имеет две сложности. Во-первых, вы не можете заглядывать достаточно далеко вперед из-за слишком большого количества возможных ходов. Во-вторых, даже если бы вы могли прогнозировать игру на много ходов вперед, вы не смогли бы оценить результат. Ключом к решению обеих проблем является глубокое обучение.

Эта книга позволяет познакомиться с темой глубокого обучения путем рассмотрения методов, лежащих в основе алгоритма AlphaGo. Вам не придется подробно изучать игру в го, вместо этого вы познакомитесь с общими принципами обучения компьютера. Первая глава посвящена машинному обучению и проблемам, которые можно (и нельзя) решить с его помощью. Мы рассмотрим примеры, иллюстрирующие основные направления в сфере машинного обучения, и увидим, как глубокое обучение расширило область применения машинного обучения.

Рассмотрим задачу распознавания фотографии друга. Для большинства людей это не является проблемой, даже если фотография отличается плохим освещением, друг постригся или надел другую рубашку. Но что, если вы хотите запрограммировать компьютер для решения этой же задачи? С чего бы вы начали? Это та проблема, которую можно решить с помощью машинного обучения.

Традиционное компьютерное программирование подразумевает применение четких правил к структурированным данным. Разработчик-человек программирует компьютер с целью применения к данным набора инструкций для получения желаемого результата (см. рис. 1.1). Представьте себе налоговую декларацию: каждое поле имеет четко определенное значение, а его заполнение подчиняется конкретным правилам. Сложность этих правил зависит от того, где вы живете. При заполнении человек может легко допустить ошибку, однако компьютерные программы превосходно справляются с этой задачей.

В отличие от традиционной парадигмы программирования, *машинное обучение* представляет собой семейство методов, позволяющих вывести программу или алгоритм из некоторых данных вместо их непосредственной реализации. Таким образом, при машинном обучении вы, как и раньше, предоставляете данные своему компьютеру, но вместо навязывания конкретных инструкций и ожидания результата *вы предоставляете ожидаемый результат и позволяете машине самостоятельно найти алгоритм*.

Чтобы создать компьютерную программу для распознавания лица на фотографии, вы можете применить алгоритм, который анализирует большую коллекцию изображений вашего друга и генерирует соответствующую им функцию. Если вы все сделаете правильно, то сгенерированная функция также будет соответствовать новым фотографиям, которые вы еще не видели. Разумеется, программа не будет знать о своем назначении. Она способна лишь идентифицировать изображения, похожие на предоставленные ей оригиналы.

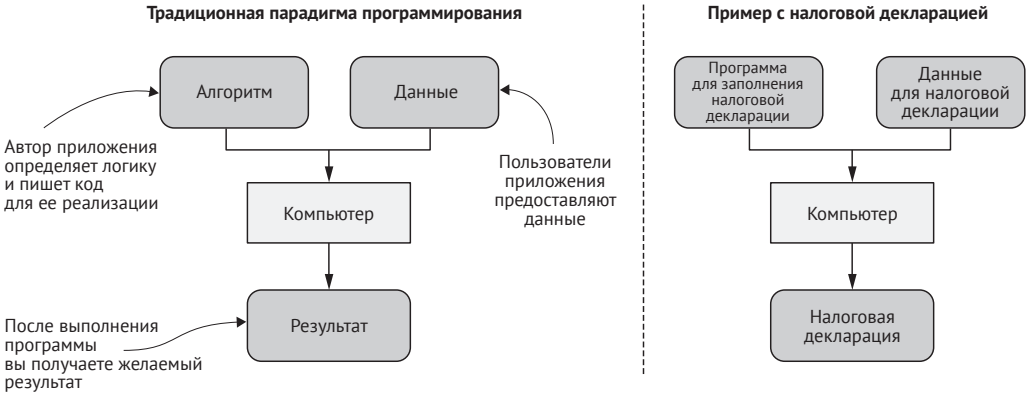


Рис. 1.1 ❖ Стандартная парадигма программирования, знакомая большинству разработчиков программного обеспечения. Разработчик идентифицирует алгоритм и реализует код, а пользователи предоставляют данные

В данном случае изображения, предоставляемые программе, называются *обучающими данными*, а имена людей на изображениях – *метками*. После *обучения* алгоритма вы сможете использовать его для *предсказания* меток на новых данных с целью его тестирования. На рис. 1.2 показан этот пример и схематически представлена парадигма машинного обучения.

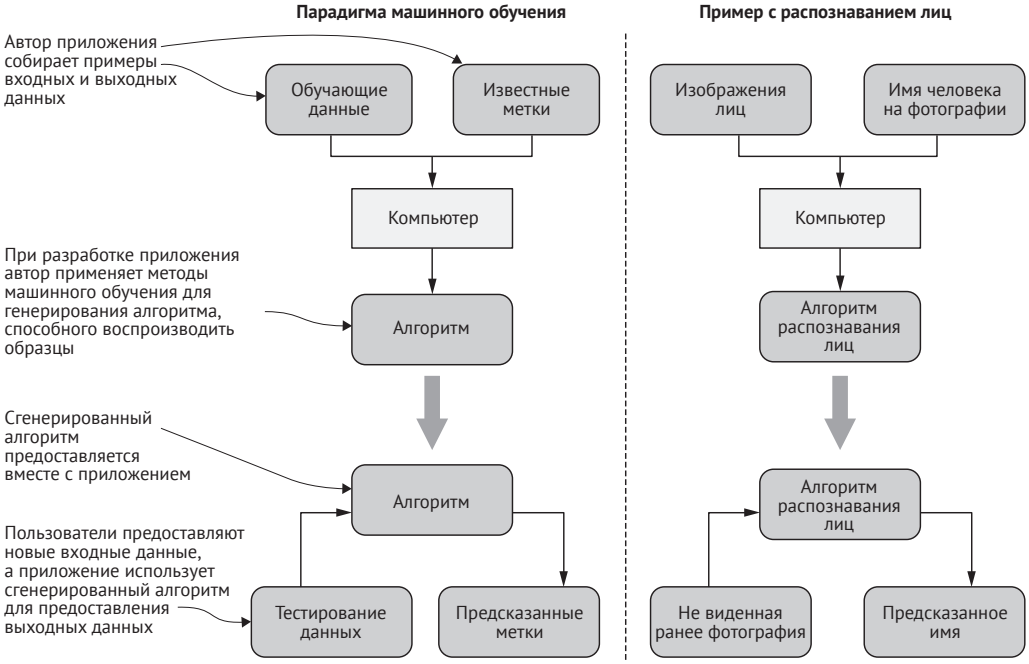


Рис. 1.2 ❖ Парадигма машинного обучения: в процессе разработки вы генерируете алгоритм на основании набора данных, а затем включаете его в итоговое приложение

Машинное обучение необходимо в ситуациях, когда правила не ясны. Оно может использоваться для решения проблем вроде «я узнаю это, когда увижу». Вместо непосредственного программирования функции вы предоставляете данные, указывающие, что эта функция должна делать, а затем методично генерируете функцию, соответствующую вашим данным.

На практике машинное обучение обычно комбинируется с традиционным программированием с целью создания полезной программы. В случае нашего приложения для распознавания лиц, прежде чем применять алгоритм машинного обучения, вы должны проинструктировать компьютер о том, как находить, загружать и преобразовывать примеры изображений. Кроме того, вы можете использовать эвристические алгоритмы, чтобы отделить изображения лиц от фотографий закатов и латте-арт, после чего применить методы машинного обучения для сопоставления имен и лиц. Часто сочетание традиционных методов программирования и продвинутых алгоритмов машинного обучения обеспечивает лучший результат, чем применение только одного из них.

1.1.1. Связь машинного обучения и искусственного интеллекта

Термин *искусственный интеллект* в широком смысле слова может быть применен к любому методу, позволяющему компьютерам имитировать человеческое поведение. ИИ предусматривает огромный спектр методов, включая следующие:

- логические продукционные системы, применяющие формальную логику для оценки утверждений;
- экспертные системы, в которых программисты пытаются закодировать человеческие знания непосредственно в программное обеспечение;
- нечеткая логика, определяющая алгоритмы, помогающие компьютерам обрабатывать неточные высказывания.

Такие методы, основанные на правилах, иногда называются *классическим ИИ*, или *GOFAI* (Good Old-Fashioned Artificial Intelligence, старый добрый искусственный интеллект).

Машинное обучение – это лишь одно из многих направлений в области ИИ, но, пожалуй, наиболее успешное на сегодняшний день. В частности, глубокое обучение позволило совершить некоторые из самых захватывающих прорывов в области искусственного интеллекта и решить задачи, к которым на протяжении многих десятилетий исследователи не могли подступиться. В случае классического ИИ исследователи изучают поведение человека и пытаются закодировать соответствующие ему правила. Машинное обучение и глубокое обучение ставят все с ног на голову: теперь вы собираете примеры человеческого поведения и выводите его правила с помощью математических и статистических методов.

Глубокое обучение используется повсеместно, поэтому для некоторых участников сообщества термины *ИИ* и *глубокое обучение* являются взаимозаменяемыми. Для ясности мы будем использовать понятие *ИИ*, говоря об общей проблеме имитации человеческого поведения с помощью компьютеров, а понятия *машинного обучения* или *глубокого обучения* – говоря о математических методах выведения алгоритмов из примеров.

1.1.2. Что можно и чего нельзя сделать с помощью машинного обучения

Машинное обучение представляет собой специализированную технику. Вы не будете использовать машинное обучение для обновления записей базы данных или визуализации пользовательского интерфейса. В следующих ситуациях предпочтительным является традиционное программирование:

- **проблему можно решить с помощью традиционных алгоритмов**: если для решения проблемы можно написать код, то его будет проще понимать, поддерживать, тестировать и отлаживать;
- **вы ожидаете идеальной точности**: все сложные программы содержат ошибки. Однако при традиционном подходе к разработке программного обеспечения вы методически выявляете и исправляете их. Это не всегда возможно при использовании машинного обучения. Вы можете улучшить систему машинного обучения, однако слишком большое внимание, уделенное конкретной ошибке, часто приводит к ухудшению системы в целом;
- **хорошо работают простые эвристические алгоритмы**: если вы можете реализовать достаточно хорошее правило с помощью нескольких строк кода, сделайте это и будьте счастливы. Простую четко реализованную эвристику будет легко понять и поддерживать. Функции, реализуемые с помощью машинного обучения, не ясны, а для их обновления требуется отдельный процесс обучения. (С другой стороны, если вам приходится поддерживать сложную последовательность эвристических алгоритмов, то их можно заменить машинным обучением.)

Часто существует тонкая грань между проблемами, которые можно решить с помощью традиционного программирования, и проблемами, которые практически невозможно решить даже с помощью машинного обучения. Распознавание лиц на изображениях и их сопоставление с именами – это только один пример. Другим примером является определение языка, на котором написан текст, и перевод текста на конкретный язык.

Мы часто выбираем традиционное программирование, когда машинное обучение могло бы помочь, например при чрезвычайно высокой сложности проблемы. Столкнувшись с очень сложными, насыщенными информацией сценариями, люди склонны прибегать к эмпирическим правилам и изложению фактов, например когда речь идет о макроэкономике, фондовом рынке или политике. Менеджеры процессов и так называемые эксперты часто могут извлечь большую пользу из идей, полученных в результате машинного обучения. Зачастую данные из реального мира оказываются более структурированными, чем это предполагалось, и мы только начинаем использовать преимущества автоматизации и усиления интеллекта во многих из этих областей.

Цель машинного обучения заключается в создании функции, которую было бы сложно реализовать напрямую. Для этого вы выбираете *модель*, большое семейство родовых функций. Затем вам требуется процедура для выбора из этого се-

мейства функции, соответствующей вашей цели. Этот процесс называется *обучением*, или *подбором*, модели. Разберем это на простом примере.

Допустим, вы собираете значения роста и веса нескольких людей и наносите их на график. На рис. 1.3 показаны данные, взятые из реестра членов профессиональной футбольной команды.

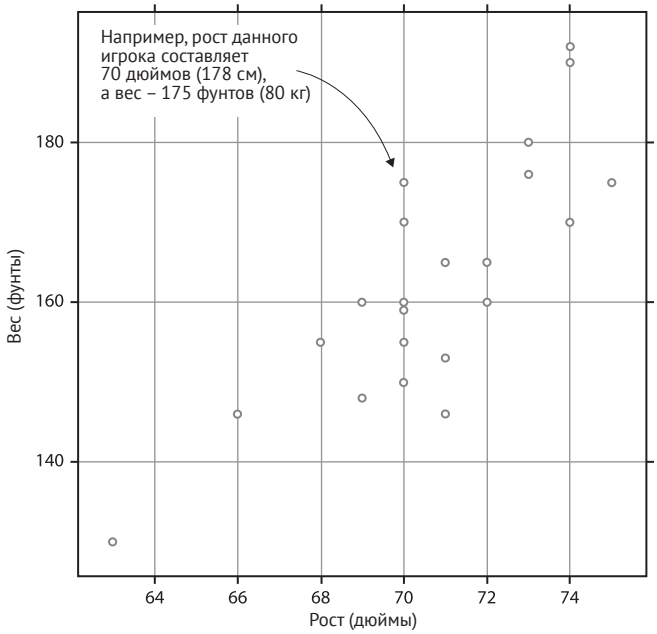


Рис. 1.3 ❖ Простой пример набора данных. Каждая точка на графике соответствует росту и весу футболиста. Ваша цель заключается в подборе модели, соответствующей этим точкам

Допустим, вы хотите описать эти точки с помощью математической функции. Во-первых, обратите внимание на то, что точки образуют почти прямую линию, направленную вверх и вправо. Из школьного курса алгебры вам известно, что прямые линии описываются функцией вида $f(x) = ax + b$. Вы можете предположить возможность нахождения таких значений a и b, при которых выражение $ax + b$ будет достаточно близко соответствовать вашим точкам данных. Значения a и b – это *параметры*, или *веса*, которые вам необходимо выяснить. Это ваша модель. Вы можете написать код на языке Python, способный генерировать любую функцию из этого семейства:

```
class GenericLinearFunction:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def evaluate(self, x):
        return self.a * x + self.b
```

Как узнать правильные значения *a* и *b*? Для этого можно использовать строгие алгоритмы, однако для получения быстрого и приблизительного решения вы можете просто начертить на графике линию с помощью линейки и попытаться вывести ее формулу. На рис. 1.4 показана линия, которая соответствует закономерности распределения точек данных.

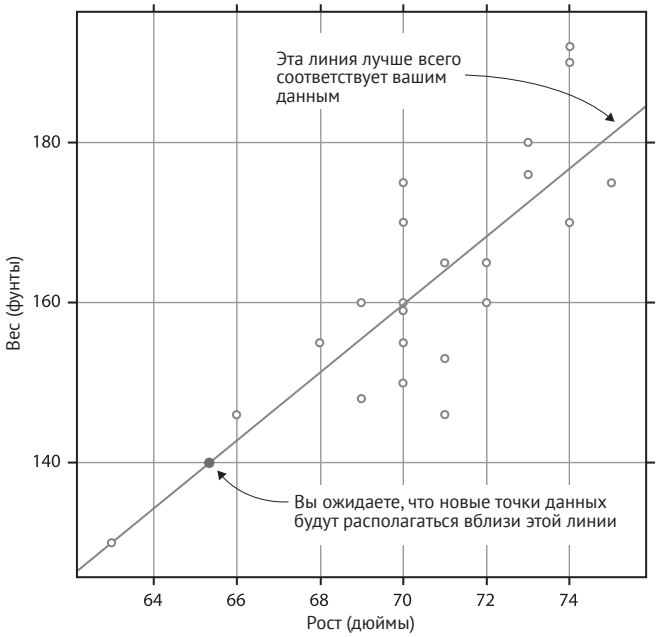


Рис. 1.4 ❖ Сначала вы замечаете, что точки данных распределены вдоль некоторой линии, а затем выводите формулу этой линии

Если вы посмотрите на пару точек, через которые проходит линия, то сможете вывести для этой линии формулу. В результате вы получите что-то вроде $f(x) = 4{,}2x - 137$. Теперь у вас есть конкретная функция, соответствующая вашим данным. Если вы измерите рост нового человека, то сможете использовать свою формулу для приблизительной оценки веса этого человека. Полученное значение будет не точным, но достаточно близким для того, чтобы оказаться полезным. Вы можете превратить `GenericLinearFunction` в конкретную функцию:

```
height_to_weight = GenericLinearFunction(a=4.2, b=-137)
height_of_new_person = 73
estimated_weight = height_to_weight.evaluate(height_of_new_person)
```

Эта функция позволит вам получить довольно хорошую оценку, при условии что ваш новый человек также будет являться профессиональным футболистом. Все данные в вашем наборе относятся к взрослым мужчинам в довольно узком возрастном диапазоне, которые каждый день занимаются одним и тем же видом спорта. Если вы попытаетесь применить свою функцию к женщинам-футболисткам, олимпийским тяжелоатлетам или детям, то получите крайне неточные результаты. Ваша функция соответствует только вашим обучающим данным.

Так выглядит простейший процесс машинного обучения. В данном случае вашей моделью является семейство всех функций вида $f(x) = ax + b$. На самом деле даже такие простые модели оказываются полезными, поэтому статистики используют их постоянно. При решении более сложных проблем применяются более сложные модели и более продвинутые методы обучения. Однако основная идея остается прежней: сначала описывается большое семейство возможных функций, среди которых затем выбирается лучшая.

Python и машинное обучение

Все примеры кода, приведенные в этой книге, написаны на языке программирования Python. Почему именно на нем? Во-первых, Python представляет собой выразительный высокоуровневый язык, предназначенный для разработки приложений. Кроме того, Python является одним из самых популярных языков для машинного обучения и математического программирования. Эта комбинация делает Python естественным выбором для создания приложения с элементами машинного обучения.

Язык Python широко применяется в сфере машинного обучения благодаря своей удивительной коллекции пакетов для числовых вычислений. В этой книге мы используем следующие из них:

- **NumPy** – эта библиотека предусматривает эффективные структуры данных для представления числовых векторов и массивов, а также обширную библиотеку быстрых математических операций. NumPy является ядром вычислительной экосистемы Python: все известные библиотеки для машинного обучения или статистики интегрированы с NumPy;
- **TensorFlow и Theano** – это две библиотеки для построения графа вычислений (*граф* – это сеть связанных между собой вершин). Они позволяют определять сложные последовательности математических операций, а затем генерировать высоко оптимизированные реализации;
- **Keras** – это высокоуровневая библиотека для глубокого обучения. Она обеспечивает удобный способ создания нейронных сетей, а для реализации вычислений использует библиотеки TensorFlow или Theano.

Примеры кода, приведенные в этой книге, написаны с учетом Keras 2.2 и TensorFlow 1.8. Внеся минимальные изменения, вы сможете использовать любую версию библиотеки Keras серии 2.x.

1.2.1. Использование машинного обучения в приложениях

В предыдущем разделе мы рассмотрели чисто математическую модель. Как можно применить машинное обучение к реальной программе?

Предположим, вы работаете над приложением для обмена фотографиями, в которое пользователи загрузили миллионы изображений с тегами. Вы хотите добавить функцию, которая предлагает тег для новой фотографии. Для создания этой функции идеально подойдут методы машинного обучения.

Во-первых, вы должны четко указать функцию, которую намерены обучить. Допустим, у вас была такая функция:

```
def suggest_tags(image_data):
    """Предлагает теги для изображения.

    Входные данные: image_data - это фотография в растровом формате

    Возвращает: ранжированный список рекомендуемых тегов
    """
```

Дальше все довольно просто. Однако не совсем ясно, с чего следует начинать реализацию такой функции, как `offer_tags`. И тут в игру вступает машинное обучение.

Обычная функция Python принимала бы в качестве входных данных некоторый объект `Image`, а в качестве результата, вероятно, возвращала бы список строк. Алгоритмы машинного обучения не так гибки в отношении своих входных и выходных данных, обычно они работают с векторами и матрицами. Поэтому сначала вы должны математически представить свои входные и выходные данные.

Если вы изменяете размер входной фотографии до фиксированной величины, скажем, 128×128 пикселов, то можете закодировать ее как матрицу, состоящую из 128 строк и 128 столбцов, – по одному значению с плавающей запятой на пиксел. А как насчет выходных данных? Одним из вариантов является ограничение набора идентифицируемых вами тегов: вы можете выбрать, например, 1000 самых популярных тегов в приложении. Выходные данные могли бы представлять собой вектор из 1000 элементов, каждый из которых соответствует определенному тегу. Позволив выходным значениям варьироваться в диапазоне от 0 до 1, вы сможете создать ранжированные списки предлагаемых тегов. На рис. 1.5 показаны концепции, лежащие в основе вашего приложения, и соответствующие им математические структуры.

Только что выполненный вами шаг предварительной обработки данных является неотъемлемой частью любой процедуры машинного обучения. Обычно вы загружаете необработанные данные и выполняете их предварительную обработку для создания *признаков* – входных данных, подаваемых алгоритму машинного обучения.

1.2.2. Обучение с учителем

Теперь вам нужен алгоритм для обучения вашей модели. В данном случае у вас уже есть миллионы правильных примеров – все фотографии, которые пользователи уже загрузили и вручную отметили в вашем приложении. Вы можете вывести функцию, максимально точно соответствующую этим примерам, в надежде, что она будет достаточно хорошо соответствовать и новым фотографиям. Этот метод называется *обучением с учителем*, поскольку в процессе обучения руководством служат *метки*, присвоенные человеком.

После завершения обучения вы сможете интегрировать окончательную функцию в ваше приложение. Каждый раз, когда пользователь загружает новую фотографию, вы передаете ее обученной функции и получаете вектор. Вы можете сопоставить каждое значение в векторе с соответствующим тегом, а затем выбрать теги с самыми большими значениями и показать их пользователю. Только что описанная процедура схематически представлена на рис. 1.6.

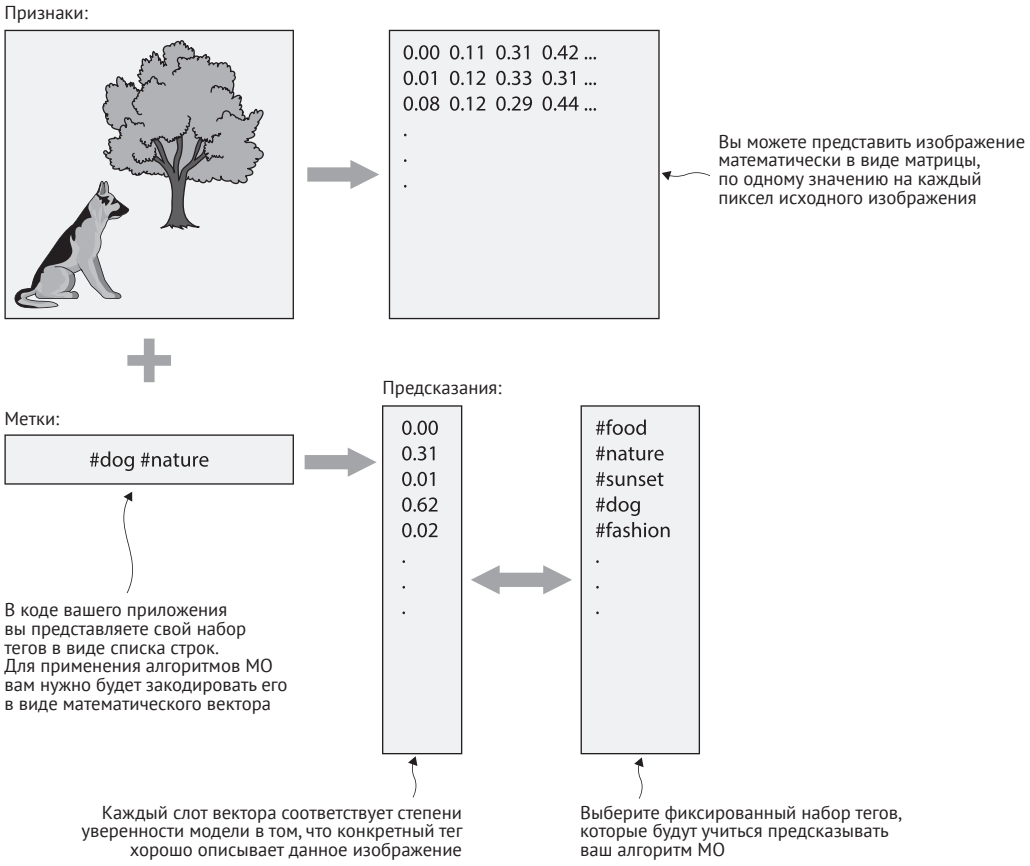


Рис. 1.5 ❖ Алгоритмы машинного обучения применяются к таким математическим структурам, как векторы и матрицы. Ваши теги для фотографий хранятся в стандартной компьютерной структуре данных – в списке строк. Это одна из возможных схем кодирования списка в виде математического вектора

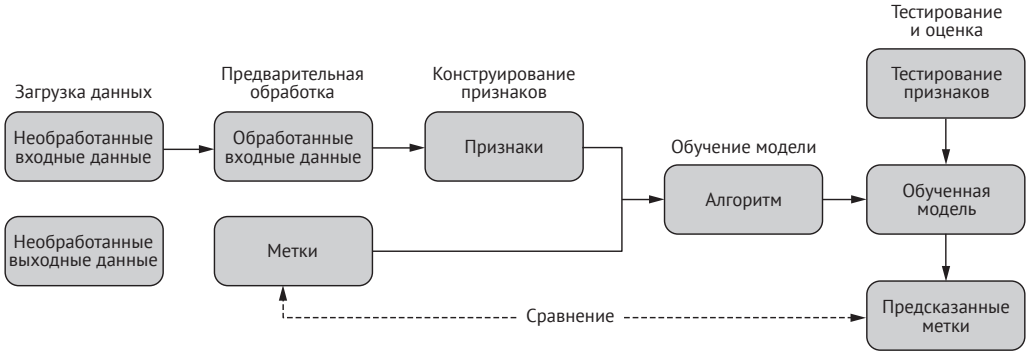


Рис. 1.6 ❖ Конвейер машинного обучения при обучении с учителем

Как протестировать обученную модель? Стандартная практика предполагает использование для этой цели некоторого количества исходных помеченных данных. До начала обучения вы можете выделить часть своих данных, например 10 %, в качестве *контрольного набора*. Этот контрольный набор *не включается* в набор данных, используемых для обучения. Затем вы можете применить свою обученную модель к изображениям из контрольного набора и сравнить предложенные теги с уже известными подходящими тегами. Это позволяет определить степень точности вашей модели. Если вы захотите поэкспериментировать с различными моделями, у вас будет единая метрика для определения лучшей из них.

В случае игрового ИИ вы можете извлечь помеченные данные для обучения из записей партий, сыгранных людьми. Отличным ресурсом данных для машинного обучения являются онлайн-игры, поскольку игровой сервер может хранить записи сыгранных пользователями партий. Обучение с учителем может быть применено к играм следующим образом:

- при наличии набора полных записей шахматных партий представьте игровое состояние в векторной или матричной форме и научитесь предсказывать следующий ход, исходя из имеющихся данных;
- научитесь прогнозировать вероятность выигрыша, исходя из текущего состояния доски.

1.2.3. Обучение без учителя

В отличие от обучения с учителем, *обучение без учителя* не предусматривает использования каких-либо меток, которые направляли бы процесс обучения. При обучении без учителя алгоритм должен научиться самостоятельно находить закономерности во входных данных. Единственное отличие от схемы на рис. 1.6 заключается в отсутствии меток, поэтому вы не можете оценить свои прогнозы, как раньше. Все остальные компоненты остаются прежними.

Примером может являться *обнаружение выбросов* – выявление точек данных, которые не согласуются с общей тенденцией набора. В наборе данных о футболистах выбросы указывают на игроков, телосложение которых не соответствует типичным показателям их товарищей по команде. Например, вы можете придумать алгоритм, измеряющий расстояние от точки рост–вес до намеченной линии. Если это расстояние превышает определенное значение, вы объявляете данную точку выбросом.

В ИИ для настольных игр естественным является вопрос о том, какие фигуры на доске образуют группу. В следующей главе вы узнаете, какое значение это имеет для игры в го. Нахождение групп связанных между собой элементов иногда называется *кластеризацией*, или *фрагментацией*. На рис. 1.7 это показано на примере шахмат.

1.2.4. Обучение с подкреплением

Обучение с учителем представляет собой мощный механизм, однако при его использовании серьезную проблему может представлять поиск качественных обучающих данных. Предположим, вы создаете робота для домашней уборки. Робот имеет различные датчики для детектирования находящихся рядом препятствий и моторы, позволяющие ему перемещаться по полу и поворачивать влево или вправо. Вам нужна система управления: функция, которая способна анализировать данные, полученные от датчика, и принимать решение относительно

дальнейшего движения робота. В данной ситуации обучение с учителем неприменимо. У вас нет примеров, которые можно было бы использовать в качестве обучающих данных, – ваш робот еще даже не существует.

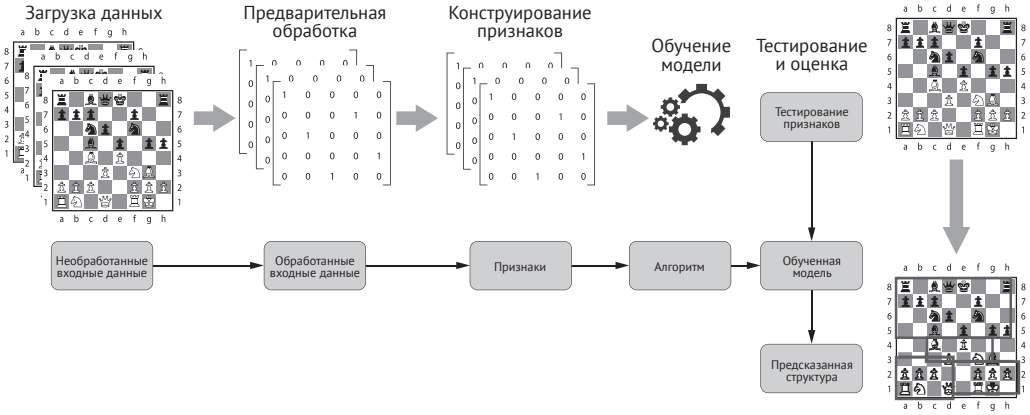


Рис. 1.7 ❖ Конвейер машинного обучения при обучении без учителя для нахождения кластеров шахматных фигур

Вместо этого вы можете воспользоваться своего рода методом проб и ошибок под названием *обучение с подкреплением*. Вы начинаете с неэффективной или неточной системы управления, а затем позволяете роботу выполнить свою задачу. Во время выполнения задачи вы записываете все входные данные, получаемые вашей системой управления, и решения, которые она принимает. После этого вам нужно будет оценить качество проделанной работы, например определить процент площади пола, который пропылесосил робот, и степень разряда батареи. В результате у вас появится небольшой объем обучающих данных, который вы сможете использовать для улучшения системы управления. Многократное повторение этого процесса позволит вам, в конце концов, вывести эффективную функцию управления. На рис. 1.8 этот процесс представлен в виде блок-схемы.

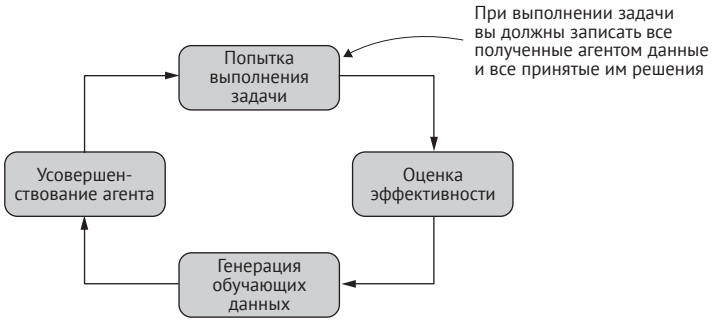


Рис. 1.8 ❖ При обучении с подкреплением агенты учатся взаимодействовать со своей средой методом проб и ошибок. Ваш агент многократно пытается выполнить свою задачу, собирая данные, на которых он может учиться. Каждый цикл позволяет вносить постепенные улучшения

Текст этой книги состоит из предложений. Предложения состоят из слов, слова – из букв, буквы – из прямых и кривых линий, а эти линии, в свою очередь, – из крошечных чернильных точек. Обучение ребенка чтению начинается с самых маленьких фрагментов: сначала буквы, потом слова, затем предложения и, наконец, целые книги. (Обычно дети учатся распознавать линии самостоятельно.) Такая иерархия представляет собой естественный процесс изучения людьми сложных концепций. На каждом уровне игнорируются некоторые детали, и концепции становятся более абстрактными.

Глубокое обучение предполагает применение этой идеи к сфере машинного обучения. Глубокое обучение – это направление, в рамках которого используется определенное семейство моделей: последовательностей, состоящих из простых функций. Эти цепочки функций называются *нейронными сетями*, поскольку они несколько напоминают структуру естественного мозга. Основная идея глубокого обучения заключается в том, что эти последовательности функций могут проанализировать сложную концепцию как иерархию более простых. Первый слой глубокой модели может научиться принимать необработанные данные и организовывать их простыми способами, например группировать точки в линии. Каждый последующий слой организует предыдущий в более сложные и абстрактные концепции. Процесс изучения этих абстрактных понятий называется *репрезентативным обучением*.

Удивительным в глубоком обучении является то, что вам не нужно заранее знать, каковы эти промежуточные концепции. Если вы выберете модель с достаточным количеством слоев и предоставите достаточно обучающих данных, то в процессе обучения необработанные данные будут постепенно организованы в концепции все более высокого уровня. Откуда алгоритм обучения знает, какие концепции следует использовать? Он не знает, он просто организует входные данные таким способом, который позволяет обеспечить наилучшее соответствие обучающим примерам. Нет никаких гарантий, что это представление будет совпадать с человеческим способом обработки этих данных. На рис. 1.9 показано, как репрезентативное обучение вписывается в процесс обучения с учителем.

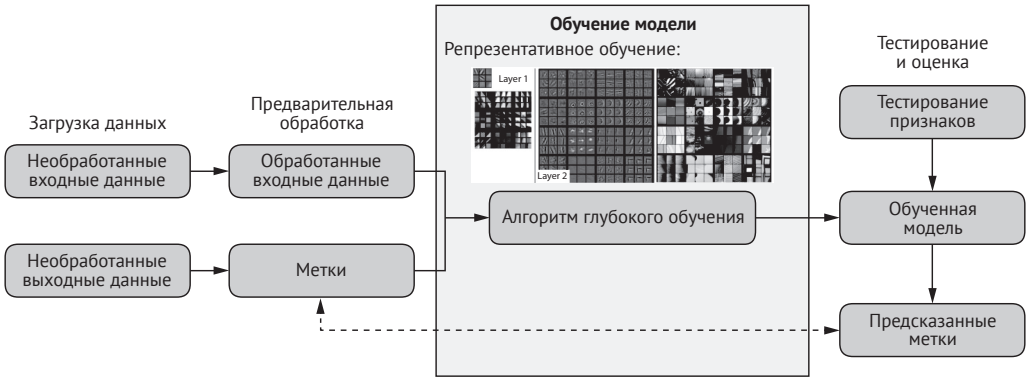


Рис. 1.9 ❖ Глубокое обучение и репрезентативное обучение

Вся эта мощь имеет свои издержки. Глубокие модели должны изучить огромное количество весов. Простой модели $ax + b$, которую вы применяли к набору данных о росте и весе футболистов, было достаточно двух весов. Глубокой модели, предназначенной для приложения с функцией тегирования изображений, может потребоваться миллион. Таким образом, глубокое обучение требует больших наборов данных, большей вычислительной мощности и более практичного подхода к обучению. Оба метода имеют свои сферы применения. Глубокое обучение является хорошим вариантом при следующих обстоятельствах:

- **ваши данные не структурированы**: изображения, аудиозаписи и тексты являются хорошими кандидатами на применение глубокого обучения. К таким данным можно применить простые модели, однако при этом обычно требуется сложная предварительная обработка;
- **вы обладаете большими объемами данных или знаете, как их получить**: как правило, чем сложнее модель, тем больше данных требуется для ее обучения;
- **у вас большие вычислительные мощности или много времени**. Глубокие модели требуют большого количества вычислений как в процессе обучения, так и в процессе оценки.

Традиционные модели с меньшим количеством параметров являются предпочтительными в следующих случаях:

- **ваши данные структурированы**: если ваши входные данные больше похожи на записи базы данных, вы зачастую можете применить простые модели напрямую;
- **вам нужна описательная модель**: в случае с простыми моделями вы можете посмотреть на выведенную в процессе обучения функцию и узнать, как отдельные входные данные влияют на результат. Это позволяет вам получить представление о том, как на самом деле работает изучаемая вами система. В случае с глубокими моделями связь между определенными входными данными и результатом является гораздо менее очевидной. Такую модель трудно интерпретировать.

Поскольку *глубокое обучение* относится к типу используемой модели, вы можете применить его к любому из основных направлений машинного обучения. Например, вы можете проводить обучение с учителем, используя глубокую или простую модель, в зависимости от типа имеющихся у вас обучающих данных.

Эта книга представляет собой практическое введение в тему глубокого обучения и обучения с подкреплением. Чтобы получить максимальную пользу от этой книги, вы должны обладать навыком чтения и написания кода на языке Python, а также иметь некоторое представление о линейной алгебре и математическом анализе. Из этой книги вы узнаете, как:

- проектировать, обучать и тестировать нейронные сети с помощью библиотеки глубокого обучения Keras;
- осуществлять глубокое обучение с учителем;
- осуществлять обучение с подкреплением;
- интегрировать глубокое обучение в полезное приложение.

На протяжении всей книги мы будем создавать ИИ для игры в го. Наш бот будет сочетать в себе глубокое обучение со стандартными компьютерными алгоритмами. Мы будем использовать простой код Python для обеспечения соблюдения правил игры, отслеживания игрового состояния и прогнозирования последовательностей ходов. Глубокое обучение поможет боту определить перспективные ходы и лидера игры. На каждом этапе вы можете играть против своего бота и наблюдать за его прогрессом по мере применения все более изощренных приемов.

Если вас интересует сама игра го, то вы можете использовать созданного бота в качестве отправной точки для собственных экспериментов. Вы можете адаптировать описанные приемы к другим играм, а также добавлять функции, основанные на глубоком обучении, в другие приложения, не имеющие отношения к играм.

- Машинное обучение – это семейство методов, позволяющих генерировать функции на основе данных вместо их непосредственного написания. Вы можете применить машинное обучение к проблемам, которые являются слишком неоднозначными для непосредственного решения.
- Как правило, процесс машинного обучения начинается с выбора *модели* – семейства математических функций. Затем вы *обучаете* эту модель, т. е. применяете алгоритм для нахождения лучшей функции в этом семействе. По большей части машинное обучение сводится к выбору подходящей модели и преобразованию конкретного набора данных для работы с ней.
- Тремя основными направлениями в сфере машинного обучения являются: обучение с учителем, обучение без учителя и обучение с подкреплением.
- Обучение с учителем предполагает выведение функции из заведомо верных примеров. При наличии примеров человеческого поведения или знаний вы можете имитировать их на компьютере с помощью методов обучения с учителем.
- Обучение без учителя предполагает извлечение из данных неизвестной заранее структуры. Обычно оно применяется для разбиения набора данных на логические группы.
- Обучение с подкреплением предполагает выведение функции методом проб и ошибок. Если вы способны написать код для оценки эффективности программы, то можете применить обучение с подкреплением для постепенного улучшения программы путем многократных испытаний.
- Глубокое обучение – это машинное обучение с использованием модели определенного типа, которая хорошо работает с такими неструктурированными входными данными, как изображения или текст. В настоящее время это одна из самых захватывающих областей информатики, которая постоянно расширяет наши представления о том, на что способны компьютеры.