Глава 3

Вопросы использования алгоритмов

В ЭТОЙ ГЛАВЕ...

- Роль алгоритмов в искусственном интеллекте
- Победа в играх с поиском в пространстве состояний и мини-максом
- Анализ работы экспертных систем
- Машинное обучение и глубокое обучение как часть искусственного интеллекта

Данные — это решающий фактор искусственного интеллекта. Недавние авансы по поводу искусственного интеллекта намекают, что для решения некоторых проблем правильный объем данных важнее правильного алгоритма. Например, в 2001 году два исследователя из Microsoft, Мишель Банко (Banko) и Эрик Брил (Brill), опубликовали незабываемую статью "Scaling to Very Very Large Corpora for Natural Language Disambiguation" (`http://www.aclweb.org/anthology/P01-1005`), в которой утверждали, что если вы хотите, чтобы компьютер создал модель языка, то вам не нужен самый умный алгоритм в городе. После ввода более чем одного миллиарда слов в пределах контекста любые алгоритмы начнут работать невероятно хорошо. Эта

глава поможет вам понять отношения между алгоритмами и данными, а также как они заставляют их выполнять полезную работу.

Однако, независимо от количества имеющихся данных, чтобы сделать их полезными, необходим алгоритм. Кроме того, чтобы данные правильно работали с выбранными алгоритмами, следует выполнить *анализ данных* (набор определенных этапов). Ничего пропускать здесь нельзя. Даже при том, что искусственный интеллект — это интеллектуальная автоматизация, иногда автоматизация должна держаться в тени анализа. Обучаемые машины находятся в далеком будущем. Сейчас вы не найдете машин, обладающих самодостаточностью и способных полностью избежать человеческого вмешательства. Вторая половина этой главы поможет понять роль экспертных систем, машинного обучения, глубокого обучения и таких приложений, как AlphaGo, для приближения будущих возможностей к современной действительности.

Понятие роли алгоритмов

Люди обычно распознают искусственный интеллект, когда инструмент представляет новый подход и взаимодействует с пользователем способом, подобным человеческому. Примерами являются такие цифровые помощники, как Сири (Siri), Алекса (Alexa) и Кортана (Cortana). Но некоторые другие общепринятые инструментальные средства, например навигатор GPS и специализированные планировщики (такие, как позволяющие избежать столкновений автомобилей, автопилоты в самолетах и составители рабочих планов), даже не выглядят, как искусственный интеллект, поскольку слишком распространены и считаются само собой разумеющимися, ведь они действуют на заднем плане приложений.

Это, безусловно, *эффект искусственного интеллекта* (AI effect), описанный Памелой Мак-Кордак (Pamela McCorduck), американской писательницей, автором известной истории искусственного интеллекта, в 1979 году. Эффект искусственного интеллекта заключается в том, что успешные интеллектуальные компьютерные программы быстро теряют благодарность людей и становятся тихими исполнителями, в то время как внимание смещается к тем проблемам искусственного интеллекта, которые все еще требуют решения. Люди перестают осознавать важность классических алгоритмов для искусственного интеллекта и начинают фантазировать об интеллекте, созданном по тайной технологии, или сравнивать его с последними футуристическими анонсами, таким как машинное и глубокое обучение.

Алгоритм (algorithm) — это процедура, представляющая собой последовательность операций (обычно выполняемых компьютером) и гарантирующая

нахождение правильного решения задачи за конечный промежуток времени или уведомляющая об отсутствии решения. Даже при том, что люди использовали алгоритмы вручную на протяжении буквально тысяч лет, в зависимости от сложности решаемой задачи они могут потребовать огромных периодов времени и множества числовых вычислений. Чем быстрее и проще алгоритмы находят решение, тем лучше. Алгоритмы являются продуктом интеллекта создавших их людей, поэтому любая работающая на алгоритмах машина не может не отражать человеческий интеллект, встроенный в такие алгоритмические процедуры.

Что означает алгоритм

Алгоритм всегда представляет последовательность этапов, но для решения проблемы не обязательно выполнять все этапы. Возможности алгоритмов невероятно велики. Операции могут задействовать хранение данных, их исследование и упорядочение в структуры данных. Вы можете найти алгоритмы, решающие задачи в науке, медицине, финансах, промышленном производстве и телекоммуникациях.

Все алгоритмы — это последовательности операций по нахождению правильного решения задачи за разумное время (или оповещению об отсутствии решения, если его нет). Алгоритмы искусственного интеллекта отличаются от обобщенных алгоритмов решения задач, являющихся обычно (или даже исключительно) результатом действия человеческого интеллекта. Алгоритмы искусственного интеллекта обычно имеют дело со сложными проблемами, зачастую являющимися частью NP-полной задачи (где NP — это недетерминированное полиномиальное время), а люди обычно имеют дело с комбинацией рационального подхода и интуиции. Вот лишь несколько примеров.

- Планирование и распределение при недостатке ресурсов.
- Поиск маршрутов в сложных физических или фигуративных пространствах.
- Распознавание шаблонов в видимом изображении (не путать с восстановлением или обработкой изображения) или слышимом звуке.
- Обработка языка (понимание текста и перевод на другой язык).
- Игра (и победа) в соревновательных играх.


СОВЕТ

NP-полные задачи отличаются от других алгоритмических задач, поскольку поиск их решения за разумный период времени все еще невозможен. NP-полные задачи — это не те задачи, которые можно решить перебором всех возможностей и их комбинаций. Даже если бы были компьютеры, куда более мощные, чем современные, поиск

решения затянулся бы почти навсегда. Подобной проблемой в области искусственного интеллекта является *полный искусственный интеллект* (AI-complete).

Планирование и ветвление

Планирование помогает определить последовательность действий для достижения определенной цели. Это классическая задача искусственного интеллекта, ее примеры можно найти при планировании в промышленном производстве, распределении ресурсов и перемещениях робота в помещении. Искусственный интеллект определяет все возможные действия начиная с текущего состояния. Технически он *развивает* (expand) текущее состояние во многие будущие состояния. Затем он развивает все будущие состояния уже в их собственные будущие состояния и т.д. Когда дальнейшее развитие становится невозможным, искусственный интеллект останавливает процесс, создав *пространство состояний* (state space), состоящее из того, что могло бы случиться в будущем. Искусственный интеллект может использовать пространство состояний совсем не так, как возможный прогноз (фактически он прогнозирует все, хотя одни будущие состояния более вероятны, чем другие). Он может использовать пространство состояний для исследования решений, которые он может принять, чтобы достигнуть своей цели наилучшим способом. Это *поиск в пространстве состояний* (state-space search).

Работа с пространством состояний требует использования и специфических структур данных и алгоритмов. Общепринятыми структурами для базовых данных являются деревья и графы. Для эффективного исследования графов применяются такие популярные алгоритмы, как *поиск в ширину* (breadth-first search) и *поиск в глубину* (deep-first search).

Дерево данных очень похоже на настоящее дерево. Каждый добавляемый элемент является *узлом* (node). Узлы соединяются между собой связями. Комбинация узлов и связей формирует структуру, выглядящую, как дерево (рис. 3.1).

ЗАПОМНИ!

Деревья имеют один корневой узел, точно так же, как и физическое дерево. *Корневой узел* (root node) — это отправная точка для выполняемой обработки. С корнем соединены ветви или листья. *Узел листа* (leaf node) — это конечная точка дерева. *Узлы ветвей* (branch node) поддерживают другие ветви или листья. Дерево на рис. 3.1 является бинарным, поскольку у каждого узла есть по меньшей мере два соединения (но у деревьев, представляющих пространства состояний, может быть какое угодно количество ветвей).

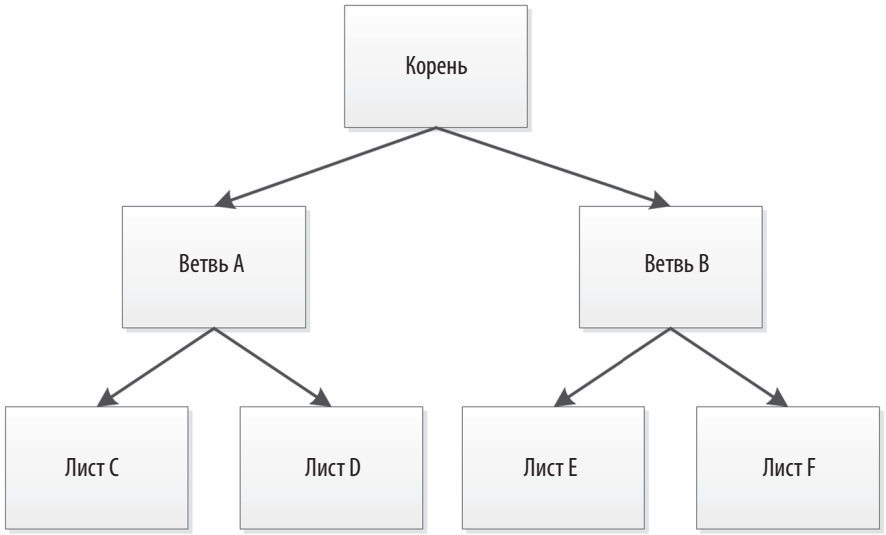

Рис. 3.1. Дерево может выглядеть как его физический прототип, а может располагаться корнем вверх

Если рассмотреть дерево, то ветвь B является *потомком* (child) корневого узла. Поэтому корневой узел является первым в списке. Листья E и F оба являются потомками ветви B, что делает ветвь B *предком* (parent) листьев E и F. Отношения между узлами важны, поскольку обсуждения деревьев нередко рассматривают родительски/дочерние отношения между узлами. Без этих терминов обсуждение деревьев может стать весьма невразумительным.

Граф (graph) — это своего рода модификация дерева. Подобно деревьям, здесь есть соединенные между собой узлы, формирующие отношения. Однако в отличие от бинарных деревьев, у узла графа может быть больше одной или двух связей. Фактически у узлов графа обычно есть множество соединений, а самые важные узлы могут иметь связи в любом направлении, а не только от предка к потомку. В простейшем виде граф выглядит как на рис. 3.2.

Графы — это структуры, представляющие множество узлов (или *вершин* (vertex)), соединенных множеством *ребер* (edge) или *дуг* (arc) в зависимости от представления. Граф можно считать структурой, подобной карте, на которой каждое место является узлом, а улицы — ребрами. Это отличается от дерева, на котором каждый путь завершается листом. Граф представлен на рис. 3.2. Графы особенно полезны при изучении состояний, представляющих своего рода физическое пространство. Например, системы GPS используют граф для представления мест и улиц.

Графы имеют также несколько новых подвохов, которые вы, возможно, не заметили. Например, граф может иметь концепцию *направленности*

(directionality). В отличие от дерева, у которого есть родительские/дочерние отношения, узел графа может соединяться с любым другим узлом в определенном направлении. Считайте их улицами в городе. Большинство улиц — с двухсторонним движением, но некоторые — с односторонним, по которым можно двигаться только в одном направлении.

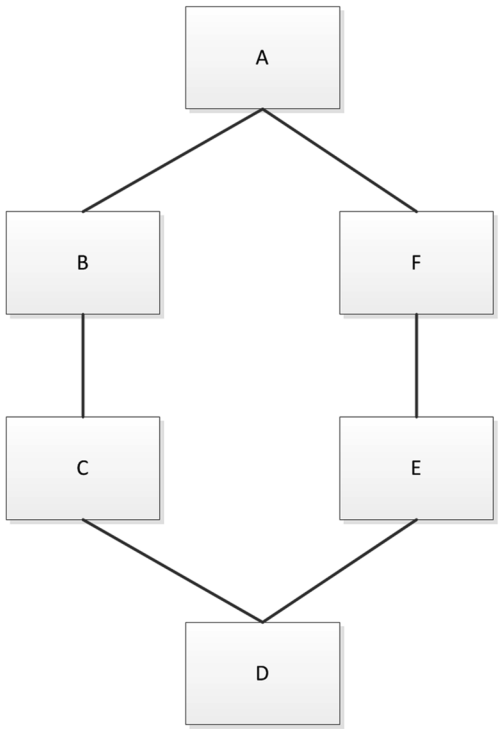

Рис. 3.2. Узлы графа могут соединяться между собой по-разному

Представление соединений графа может и не отражать реалии графа. Граф может определять коэффициенты для соединений. Коэффициент может определять дистанцию между двумя пунктами, время для прохождения маршрута или другие виды информации.

СОВЕТ

Дерево — это не более чем граф, в котором любые две вершины соединяются только одной связью, и это не допускает циклов (чтобы можно было вернуться к предку от любого потомка). Многие алгоритмы графов применимы только к деревьям.

Пересечение графа означает поиск (посещение) каждой вершины (узла) в определенном порядке. Процесс посещения вершины может включать его чтение и модификацию. По мере пересечения графа вы обнаруживаете вершины,

которые еще не были посещены. Вершина становится обнаруженной (поскольку вы ее только что посетили) или обработанной (поскольку алгоритм опробовал все следующие из нее ребра) после поиска. Порядок поиска определяет его вид: неосведомленный (поиск вслепую) и осведомленный (эвристика). При *неосведомленной* (uninformed) стратегии искусственный интеллект исследует пространство состояний без дополнительной информации, кроме структуры графа, которую он выясняет по мере его пересечения. В следующих разделах рассматриваются два популярных алгоритма поиска вслепую: поиск в ширину и поиск в глубину.

Поиск в ширину (Breadth-First Search — BFS) начинается с корня графа и исследует каждый присоединенный к нему узел. Затем поиск переходит на следующий уровень и далее по очереди, пока не будет достигнут конец. Следовательно, в примере графа поиск распространится от вершины A к B и C прежде, чем дойдет до вершины D. Поиск в ширину исследует граф систематически, просматривая вершины вокруг начальной вершины. Он начинается с посещения всех вершин от стартовой вершины, а затем переходит к следующей и т.д.

Поиск в глубину (Depth-First Search — DFS) начинается с корня графа, а затем переходит к каждому следующему узлу от корня вниз и до конца. Затем он следует в обратном порядке и начинает исследовать путь, отличный от предыдущего, пока снова не достигнет корня. В этот момент, если доступны другие пути от корня, алгоритм выбирает следующий и начинает тот же самый поиск снова. Идея в том, чтобы исследовать каждый путь полностью прежде, чем исследовать любой другой путь.

Состязательные игры

Самое интересное в поиске в пространстве состояний — это то, что он демонстрирует и нынешние и будущие возможности искусственного интеллекта. Так происходит в случае *состязательных игр* (игр, в которых один побеждает, а остальные проигрывают) или любой подобной ситуации, в которой участники преследуют цель, находящуюся в противоречии с целями других. Простая игра в крестики-нолики представляет собой совершенный пример игры поиска в пространстве, в которую вы, возможно, уже играли с искусственным интеллектом. В фильме *Военные игры* (WarGames), вышедшем в 1983 году, суперкомпьютер WOPR (War Operation Plan Response) играет против себя с потрясающей скоростью, но все же не может победить, поскольку игра действительно проста, и если использовать поиск в пространстве состояний, то проиграть невозможно.

В игре девять клеток, в которые каждый игрок записывает крестик или нолик. Первый, кто соберет метки в ряд (горизонтальный, вертикальный или диагональный), победил. При создании дерева пространства состояний каждый

уровень дерева представляет ход в игре. Конечные узлы представляют финальные состояния доски и определяют победу, ничью или поражение искусственного интеллекта. У каждого листа есть бал, у победного — высокий, у ничейного — средний, а у проигрышного — низкий или даже отрицательный. Используя суммирование, искусственный интеллект просчитывает балы к верхним узлам и ветвям до достижения стартового узла. Стартовый узел представляет текущую ситуацию. Для пересечения дерева используется простая стратегия: когда ход искусственного интеллекта и вы должны рассмотреть значения многих узлов, вы суммируете максимальное значение (очевидно, потому, что искусственный интеллект должен добиться максимального счета в игре); когда наступает очередь противника, вы суммируете вместо этого минимальное значение. В результате вы получаете дерево, ветви которого квалифицируются балами. Когда наступает очередь искусственного интеллекта, он выбирает свой переход на основании ветви с самым высоким значением, поскольку это подразумевает путь по узлам с самой высокой возможностью победить. Пример этой стратегии приведен на рис. 3.3.

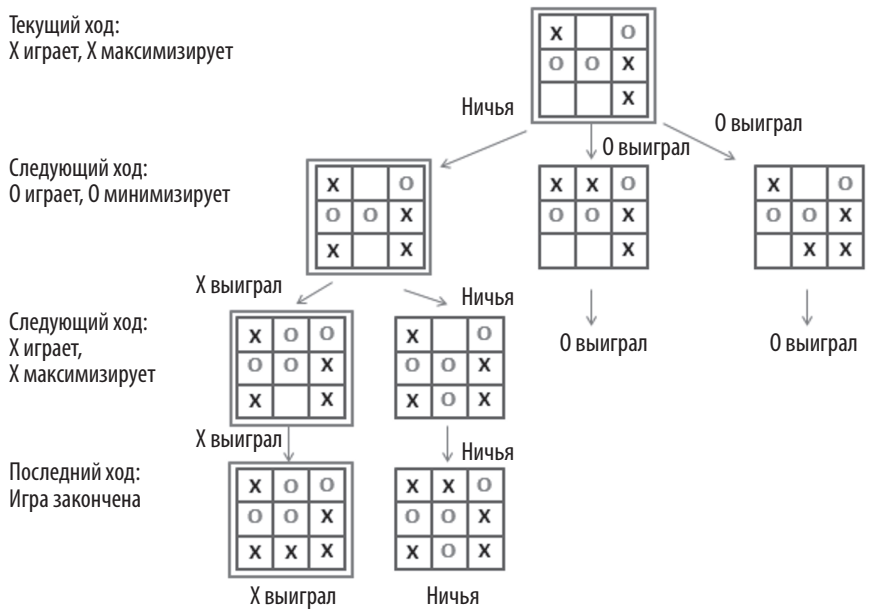

Рис. 3.3. Приближение к мини-максу в игре "крестики-нолики"

Это подход *приближения к мини-максу* (min-max approximation). Рональд Линн Ривест (Ronald Rivest) из лаборатории информатики Массачусетсского технологического института опубликовал его в 1987 году (его статью можно прочитать по адресу `https://people.csail.mit.edu/rivest/pubs/Riv87c.pdf`). С тех пор этот алгоритм и его варианты легли в основу множества

соревновательных игр, включая недавно анонсированную игру AlphaGo от Google DeepMind, в которой используется подход, напоминающий приближение к мини-максу (в фильме *Военные игры* 1983 года представлен тот же самый подход).


СОВЕТ

Иногда в контексте приближения к мини-максу можно услышать термин *альфа-бета-отсечение* (alpha-beta pruning). Это интеллектуальный способ сокращения количества рассматриваемых узлов в древовидной иерархии сложных пространств состояний, существенно сокращающий количество вычислений. Не все игры имеют компактные деревья пространства состояний, но когда количества ветвей исчисляются миллионами, их отсечение необходимо для сокращения объемов вычислений.

Использование локального поиска и эвристики

Многое изменилось со времени появления подхода поиска в пространстве состояний. В конце концов, никакая машина независимо от ее мощности не сможет перебрать все возможности в любой ситуации. В этом разделе продолжается рассмотрение игр, поскольку они предсказуемы и имеют фиксированные правила, тогда как большинство реальных ситуаций непредсказуемы и никаких правил не имеют, что делает игры оптимальным выбором.

В шашках, относительно простой игре по сравнению с шахматами или Го, возможно 500 миллиардов миллиардов (500 000 000 000 000 000 000) позиций на доске. Это значение вычислено математиками из Университета Гавайев и соответствует количеству всех песчинок на Земле. Конечно, во время игры в шашки делается куда меньше ходов, но все же количество потенциальных вариантов при каждом ходе слишком велико для вычислений. Чтобы вычислить все 500 миллиардов миллиардов возможных ходов, мощным компьютерам потребуется 18 лет (`http://sciencenetlinks.com/science-news/science-updates/checkers-solved/`). Только представьте, как долго компьютер потребительского класса будет рассчитывать даже куда меньшее количество ходов. Чтобы процесс остался подконтрольным, это должно быть очень маленькое подмножество всех возможных ходов.

Оптимизация с использованием локального поиска и эвристики позволяет ограничить количество возможных вычислений, как при альфа-бета-отсечении, когда некоторые вычисления отбрасываются, поскольку ничего не дают для успеха. *Локальный поиск* (local search) — это общий подход к решению задач, реализуемый множеством алгоритмов, позволяющих избежать экспоненциального роста сложности многих NP-задач. Локальный поиск начинается с текущей ситуации или несовершенного решения задачи и перемещается от

нее шаг за шагом. Локальный поиск выясняет пригодность соседних решений, потенциально приводя к более совершенному решению на основании случайного выбора или осмысленной эвристики (а значит, никакого точного метода решения нет).

ЗАПОМНИ!

Эвристика (heuristic) — это обоснованное предположение о решении, такое как эмпирическое правило, которое указывает направление к желаемому результату, но не способное подсказать точно, как его достичь. Это как будто, заблудившись в незнакомом городе, спрашивать людей, как добраться до своей гостиницы. Вам укажут направление, но без точных инструкций и примерного расстояния.

Алгоритмы локального поиска поэтапно улучшают состояние начиная от стартового и переходя шаг за шагом к соседним решениям в пространстве состояний, пока дальнейшее улучшение не окажется невозможным. Поскольку алгоритмы локального поиска просты и интуитивно понятны, разработать подход локального поиска для решения алгоритмической задачи нетрудно, а вот сделать его эффективным обычно более чем трудно. Главное — определить правильную процедуру.

1. Начните с существующей ситуации (это может быть текущая ситуация, случайное или известное решение).
2. Найдите набор новых возможных решений в пределах окрестностей текущего решения; они составят список кандидатов.
3. Определите решение, используемое вместо текущего на основании вывода эвристики, получившей на входе список кандидатов.
4. Продолжите выполнять пп. 2 и 3 до тех пор, пока не прекратится дальнейшее улучшение, означающее нахождение лучшего из решений.

Хотя разработка локального поиска проста, она зачастую не позволяет найти решение за разумное время (вы можете остановить процесс и использовать текущее решение) или предлагает решение, незначительно лучшее. Нет никакой гарантии, что локальный поиск найдет решение задачи, но есть шанс действительно улучшить первоначальное решение, если предоставить ему достаточно времени. Он остановится только после того, как не сможет найти дальнейший способ улучшить решение. Секрет в том, чтобы исследовать правильную окрестность. Если вы исследуете все, то скатитесь к полному перебору, подразумевающему взрывообразный рост вариантов для исследования и проверки.

Доверие к эвристике ограничивает, когда нужна эмпирика. Иногда эвристика — это случайность, и такое решение, несмотря на не интеллектуальность подхода, вполне может прекрасно работать. Немногие знают, что автономный робот-пылесос Roomba, созданный тремя специалистами из Массачусетсского

технологического института, первоначально не планировал свою траекторию, он просто двигался в случайных направлениях. И все же владельцы считали его разумным устройством, ведь свою работу по уборке он выполнял превосходно. (Фактически интеллект заключался в идее использовать случайность для решения задачи, которая в противном случае оказалась бы слишком сложной).

Случайный выбор — не единственная доступная эвристика. При исследовании локальный поиск способен полагаться и на более осмысленные решения с использованием лучше обоснованной эвристики для получения направления поиска, такие как *оптимизация восхождением к вершине* (hill-climbing optimization) или *алгоритм возврата* (twiddle), и избежать ловушки принятия посредственных решений, как при *имитации отжига* (simulated annealing) и *поиске с запретами* (tabu search). Оптимизация восхождением к вершине, алгоритм возврата, имитация отжига и поиск с запретами — это все алгоритмы поиска, эффективно используемые эвристикой для получения направления.

Алгоритм восхождения к вершине сродни действию силы гравитации. Представьте, что, когда мяч катится по склону, он выбирает самый крутой спуск, а когда его нужно закатить на холм, имеет смысл выбрать кратчайший путь достижения вершины, которым также является самый крутой склон. Таким образом, задача искусственного интеллекта — спуститься во впадину или подняться на вершину, а эвристика подскажет направление, указав самый крутой подъем или спуск из возможных в пространстве состояний. Это весьма эффективный алгоритм, хотя в некоторых ситуациях, известных как плато (промежуточные точки минимума) и пики (локальные максимальные точки), он ошибается.

Алгоритм возврата или *покоординатного спуска* (coordinate descent) подобен алгоритму восхождения к вершине. Эвристический алгоритм возврата подразумевает исследование всех возможных направлений, но поиск концентрируется в направлении лучших окрестностей. По мере продвижения он калибрует свои шаги, замедляясь по мере приближения к трудным для поиска участкам, чтобы находить лучшие решения, пока процесс не достигнет остановки.

Термин *имитация отжига* произошел от металлургической технологии, в ходе которой металл нагревают, а затем медленно охлаждают, чтобы сделать его мягче для обработки в холодном состоянии, а также для устранения межкристаллических напряжений (см. `http://www.brighthubengineering.com/manufacturing-technology/30476-what-is-heat-treatment/`). Локальный поиск воспроизводит эту технологию при поиске решения, как будто меняется атомная структура, чтобы улучшить работоспособность. Температура — решающий фактор в процессе оптимизации. Подобно тому, как высокая температура ослабляет структуру материала (твердое плавится, а жидкое испаряется), высокая температура в алгоритме локального поиска снижает *критерий отбора* (objective function), позволяя предпочесть худшие решения лучшим.

Имитация отжига изменяет процедуру восхождения к вершине, сохраняя критерий отбора для вычисления соседнего решения и позволяя осуществлять выбор решения для поиска различными способами.

Поиск с запретами подразумевает запоминание подлежащих исследованию частей окрестности. Когда решение кажется найденным, предпринимается попытка вернуться к другим возможным путям, которые еще не опробовались, чтобы выявить лучшее из решений.

Использование направления (вверх или вниз), температуры (контролируемой случайности) или просто запрета или выделения части в искомом фактически является способом избежать перебора всех возможностей и сконцентрироваться на перспективных решениях. Рассмотрим, например, перемещение робота. Задача — провести робота по неизвестной окружающей обстановке, избегая препятствий, и достичь заданного места. Это сложная фундаментальная задача искусственного интеллекта. Для ориентации на местности роботы могут полагаться на лазерный инфракрасный дальномер (*лидар* — LIDAR) или *сонар* (использует звук для наблюдения окружающей обстановки). Но несмотря на техническое оснащение, роботы все еще нуждаются в надлежащих алгоритмах для следующего.

- Поиск кратчайшего (или по крайней мере разумно короткого) пути к месту назначения.
- Избегание препятствий на пути.
- Выполнение специальных требований, таких как минимум поворотов или торможений.

Алгоритм поиска пути позволит роботу начать движение в одном месте и достичь места назначения по кратчайшему пути между этими двумя точками, избежав препятствий. (Реакции после столкновения со стеной недостаточно.) Поиск пути используется также при перемещении любого объекта к цели в пространстве, даже виртуальном, как в видеоигре или на веб-странице. Робот при поиске пути воспринимает пройденный путь как последовательность пространств состояний в границах его сенсоров. Если цель находится вне пределов их досягаемости, робот не будет знать, куда идти. Эвристика может направить его в правильную сторону (например, знание того, что цель находится в северном направлении), позволив избежать ненужных затруднений, связанных с необходимостью рассматривать все остальные возможные пути.

Понятие обучения машины

Все рассмотренные до сих пор примеры алгоритмов связаны с искусственным интеллектом, поскольку все они — интеллектуальные решения повторяющихся, жестко разграниченных, сложных задач, требующих интеллекта. Они требуют, чтобы архитектор изучил задачу и выбрал правильный алгоритм для ее решения. Однако смена задачи, ее коррекция или непредвиденные факторы могут стать реальной проблемой для успешного применения алгоритма. Это связано с тем, что изучение задачи и ее решение осуществляются раз и навсегда, пока алгоритм используется в программном обеспечении. Например, вы вполне можете создать программу с искусственным интеллектом для решения судоку (популярная игра, требующая вписать числа в клетки поля согласно правилам: `https://www.learn-sudoku.com/what-is-sudoku.html`). Вы даже можете обеспечить алгоритму некую гибкость, которая позволит учесть больше правил или больший размер игрового поля. Питер Норвиг (Peter Norvig), директор по исследованиям корпорации Google, написал чрезвычайно интересное эссе на эту тему (`http://norvig.com/sudoku.html`), в котором демонстрирует, как разумное использование поиска в глубину позволяет сократить количество вычислений (в противном случае вычисления могут длиться вечно). Применение ограничителей позволяет вначале исследовать меньшие ветви, что делает решение судоку вполне возможным.

К сожалению, не все проблемы могут полагаться на решения, подобные судоку. Проблемы реального мира никогда не возникают в простых мирах совершенной информации и четких действий. Рассмотрим задачу поиска страхового мошенника или задачу диагностики заболеваний.

- **Большой набор правил и возможностей.** Количество возможных мошенничеств невероятно велико; у многих болезней схожие симптомы.
- **Недостаток информации.** Мошенники могут скрывать информацию; врачи зачастую полагаются на неполную информацию (анализы могут еще отсутствовать).
- **Правила могут изменяться.** Мошенники изобретают все новые и новые способы надувательства; новые болезни возникают и обнаруживаются.

Для решения таких задач нельзя использовать предопределенный подход. Чтобы справляться с любыми новыми ситуациями, подход должен быть гибким, необходимо накопление полезных знаний. Другими словами, для адаптации к изменениям в сложной окружающей обстановке необходимо продолжать учиться, как люди учатся весь свой век.

Работа экспертных систем

Экспертные системы были первой попыткой избежать жестко заданных алгоритмов и выработать более гибкие и интеллектуальные способы решения реальных задач. Лежащие в основе экспертных систем идеи были простыми и вполне подходящими на те времена, когда хранение и доступ к большим количествам данных в машинной памяти были дорогостоящими. Сегодня это может показаться странным, но в 1970-х годах такие ученые в области искусственного интеллекта, как Росс Квиллиан (Ross Quillian), вынуждены были демонстрировать построение рабочих языковых моделей на базе словаря только из 20 слов, поскольку машинная память тех времен могла содержаться лишь столько слов. Немного возможностей доступно, если компьютер не может содержать все данные, и решение заключалось в работе только с ключевой информацией по проблеме и получению ее от людей, разбирающихся в ней лучше всех.

ЗАПОМНИ!

Экспертные системы были экспертными не потому, что их знания базировались на результате собственного обучения, а скорее потому, что они получали свои знания от экспертов-людей, которые предоставляли предварительно подготовленную систему ключевой информации, полученной ими из книг либо от других экспертов или изобретенной самостоятельно. В основном это был интеллектуальный способ воплощения знаний в машине.

Примером одной из первых таких систем является MYCIN; она диагностировала болезни, связанные со свертываемостью крови, и такие бактериальные инфекции крови, как менингит (воспаление мембран, защищающих головной и спинной мозг). Система MYCIN рекомендовала правильную дозировку антибиотиков на основании более 500 правил, а при необходимости вызывала использующего систему врача. Когда информации не хватало, например отсутствовали результаты лабораторных анализов, система MYCIN начинала консультативный диалог, задавая корректные вопросы по существу, чтобы удостовериться в правильности диагноза и терапии.

Написанная на языке LisP в качестве докторской диссертации Эдвардом Хансом Шортлифом (Edward Shortliffe) из Стэнфордского университета, система MYCIN потребовала пяти лет труда, а по завершении работала лучше, чем любой начинающий врач, с перспективой достичь точности диагностики опытного врача. За несколько лет до этого в той же лаборатории была разработана система DENDRAL — первая из когда-либо созданных экспертных систем. Она была весьма сложным приложением, специализировавшимся на органической химии с алгоритмами, решавшими задачи “в лоб” и оказывающимися бесполезными при встрече с человеческой эвристикой на базе практического опыта.

Что касается успеха MYCIN, то возникли некоторые проблемы. Во-первых, были не ясны условия ответственности. (Если система поставила неправильный диагноз, то кто несет ответственность?) Во-вторых, у MYCIN была проблема удобства и простоты использования, поскольку во времена младенчества Интернета врач мог подключиться к MYCIN, только используя дистанционный терминал для мэйнфрейма в Стэнфорде, что было весьма трудно и медленно. Но все же система MYCIN доказала свою эффективность и полноценность в поддержке решений человека, и это проложило путь многим другим экспертным системам, которые распространились чуть позже, в 1970- и 1980-х годах.

Вообще, экспертные системы того времени состояли из двух великолепных компонентов: базы знаний и механизма логического вывода. *База знаний* (knowledge base) хранит знания как коллекцию правил в форме операторов `if...then` (где часть `if` включает одно или несколько условий, а часть `then` — выдаваемые рекомендации). Эти символы операторов, разные у разных систем, способны проверять одиночные события или факты, классы и производные классы, позволяя манипулировать ими, используя булеву логику или сложную логику первого порядка, которая учитывает все возможные операции.


СОВЕТ

Логика первого порядка (first-order logic) — это набор операций, идущих куда далее связанных с простыми комбинациями значений TRUE (истина) и FALSE (ложь). Например, здесь есть такие концепции, как FOR ALL (для всех) и THERE EXIST (существует), позволяющие иметь дело с операторами, которые могут быть истиной, но не могут быть подтверждены доказательством на данный момент. Куда больше об этой форме логики можно узнать из статьи `http://whatis.techtarget.com/definition/first-order-logic`.

Механизм логического вывода (inference engine) — это набор инструкций, указывающих системе, как манипулировать условиями на основании таких операторов булевой логики, как AND, OR или NOT. Этот логический набор, использующий TRUE (соответствующее или, технически, “сработавшее” правило) и FALSE (не соответствующее правило) как символьные условия, способен объединять их в сложные рассуждения.

Поскольку система была создана на базе последовательности операторов `if` (условия) и `then` (выводы) и имела вложенный структурированный по уровням характер, полученная вначале информация позволяла сразу исключить некоторые заключения, а также помочь системе, взаимодействуя с пользователем, выяснить информацию, способную привести к правильному ответу. При использовании с механизмом логического вывода для экспертных систем было характерно следующее.

- **Прямая цепочка рассуждений.** Доступное доказательство делало на каждом этапе серию правил подходящими, а другие исключало. Система первоначально концентрировалась на правилах, которые могли бы быть подходящими и привести к заключению. Это явно подход, управляемый данными.
- **Обратная цепочка рассуждений.** Система вычисляет каждое возможное заключение и пытается доказать каждое из них на основании доступного доказательства. Этот подход, управляемый целями, позволяет определить, какие вопросы стоит задать, и исключает целые наборы задач. Система MYCIN использовала обратную цепочку рассуждений — общепринятую стратегию развития от гипотезы назад, к доказанному медицинскому диагнозу.
- **Разрешение конфликтов.** Если система сделала больше одного вывода за раз, она предпочитает заключение с определенными характеристиками (важность, риск или другие факторы). Иногда система консультируется с пользователем, и решение принимается на основании оценки пользователя. Например, система MYCIN использовала коэффициент уверенности, оценивавший вероятную точность диагноза.

Одним из главных преимуществ таких систем было представление знания в удобочитаемой для людей форме, процесс принятия решения был понятен и изменяем. Если система пришла к выводу, она оповещает об использованных при этом правилах. Пользователь может систематически наблюдать за работой системы и оценивать достоверность или ошибочность выводов. Кроме того, экспертные системы были просты в реализации на таких языках программирования, как LisP или ALGOL. Пользователи улучшали экспертные системы в течение долгого времени, добавляя новые или изменяя существующие правила. Они могли даже заставить ее работать в условиях неуверенности за счет применения *нечеткой логики* (fuzzy logic), своего рода многозадачной логики, в которой значение может содержать нечто среднее между 0, абсолютной ложью и 1, абсолютной истинной. Нечеткая логика избегает резких движений, когда правила выбираются на основании пороговых значений. Например, если утверждается, что правило применимо, если в комнате жарко, оно не будет применено при точном задании температуры, если она чуть-чуть не дотягивает до порогового значения. Закат экспертных систем наступил в конце 1980-х, их разработка остановилась главным образом по следующим причинам.

- Логика и символика таких систем оказались ограниченными выражением правил, лежащих в основе решения, что привело к созданию индивидуальных систем, сведя все снова к жестко заданным правилам и классическим алгоритмам.

- Экспертные системы для многих сложных задач стали настолько сложными и запутанными, что потеряли свою привлекательность с точки зрения реализуемости и цены.
- Поскольку данные становятся все более распространенными и доступными, нет особого смысла тщательно опрашивать опытных специалистов, накапливать и систематизировать ценные знания, когда то же самое (или даже лучшее) можно просто найти в открытых данных.

Экспертные системы все еще существуют. Вы можете найти их используемыми при выдаче кредита, обнаружении мошенничества и в других областях, в которых ответ предоставляется наравне с правилами, лежащими в основе решения, и таким способом, которым пользователь системы находит приемлемым (как сделал бы настоящий эксперт).

Введение в машинное обучение

Решения, способные к самостоятельному обучению непосредственно на данных без их предварительного представления в качестве символов, возникли за несколько десятилетий до экспертных систем. Одни были по природе статистическими; другие по-разному подражали природе; а третьи пытались создать автономную символическую логику в форме правил для необработанной информации. Все эти выработанные различными школами решения сегодня известны под различными названиями, включая *машинное обучение* (machine learning). Машинное обучение — это целый раздел алгоритмов, хотя в отличие от многих других обсуждавшихся до сих пор алгоритмов они не являются набором предопределенных этапов, предназначенных для решения задачи. Как правило, машинное обучение имеет дело с задачами, которые люди не умеют подразделять на этапы, но сами их, естественно, решают. Примером таких задач является распознавание лиц на изображениях или слов в разговорной речи. Машинное обучение упоминается почти в каждой главе этой книги, но работе его основных алгоритмов посвящены только главы 9–11, включая глубокое обучение, вызвавшее новую волну технологий с применением искусственного интеллекта, заголовками о которых пестрят новости почти каждый день.

Достижение новых высот

Роль машинного обучения в новой волне алгоритмов искусственного интеллекта — частично заменить, а частично дополнить существующие алгоритмы, демонстрируя действия, требующие интеллекта, с человеческой точки зрения, поскольку их непросто формализовать в точную последовательность этапов. Хороший пример такой роли — демонстрация мастерства игры в Го, в которой

мастер сразу понимает угрозы и возможности обстановки на доске и интуитивно находит правильные ходы. (Читайте историю игры Го на `http://www.usgo.org/brief-history-go`.)

Го — невероятно сложная игра для искусственного интеллекта. В шахматах приходится просчитывать порядка 35 возможных ходов на доске, и игра обычно длится не более 80 ходов, в то время как в Го возможных ходов бывает до 140 и длится игра обычно более 240 ходов. Никакой вычислительной мощи в современном мире не хватит, чтобы создать полное пространство состояний для игры Го. Группа Google DeepMind в Лондоне разработала программу AlphaGo, победившую многих знаменитых мастеров Го (см. `https://deepmind.com/research/alphago/`). Программа не полагается на алгоритмический подход на базе поиска в огромном пространстве состояний, она использует следующее.

- Метод интеллектуального поиска на основании проверки случайного возможного хода. Искусственный интеллект многократно применяет поиск в глубину, чтобы выяснить, является ли первый найденный результат лучшим или худшим (в неполном или частичном пространстве состояний).
- Алгоритм глубокого обучения обрабатывает изображение доски (сразу) и получает как наилучший возможный ход в этой ситуации (алгоритм — *стратегическая сеть* (policy network)), так и оценку вероятности выигрыша искусственного интеллекта после данного хода (алгоритм — *оценочная сеть* (value network)).
- Возможность учиться на примерах игр, сыгранных экспертами Го в прошлом, а также игра против себя, как WOPR в фильме *Военные игры* 1983 года. Последняя версия программы, AlphaGo Zero, способна учиться совершенно автономно, без человеческих примеров (см. `https://deepmind.com/blog/alphago-zero-learning-scratch/`). Эта способность известна как *обучение с подкреплением* (reinforcement learning).