Глава 3

Генетические алгоритмы

§ 3.1. Введение

Предыдущая глава была посвящена искусственным нейронным сетям. В основе этих сетей, как мы уже обсуждали, лежит структура человеческого мозга: исследователи решили её скопировать (хотя бы приблизительно) и натолкнулись на настоящую золотую жилу. Результаты до сих пор применяются в самых разнообразных областях искусственного интеллекта, от компьютерных игр до распознавания текстов.

Но это не единственный случай, когда идеи алгоритмов машинного обучения были скопированы с природы. В этой главе речь пойдёт о *генетических алгоритмах* — аппарате, списанном с идей дарвиновской эволюции.

Как известно, основная идея дарвиновской эволюции заключается в *естественном отборе*: наименее приспособленные организмы умирают раньше и в больших количествах, наиболее приспособленные выживают и дают потомство. Их потомство уже оказывается в среднем более приспособленным к окружающей среде, но среди них опять выделяются наиболее приспособленные особи, и так далее.

В генетическом алгоритме происходит абсолютно то же самое. Есть некоторое пространство гипотез, из которых мы должны выбрать лучшую. Есть функция приспособленности Fitness, которая определяет, насколько хорошо тот или иной организм приспособлен к «окружающей среде». Есть набор генетических операций, при помощи которых на свет появляются новые особи. И, наконец, есть некоторое целевое значение $\text{Fitness}_{\max}$, к которому мы стремимся — как только мы его достигнем, работу алгоритма можно будет прекратить.[1]

На рис. 3.1 изображена общая схема генетического алгоритма. Рассмотрим её подробнее.

§ 3.2. Схема генетического алгоритма в деталях

1. *Инициализация.* Как говорил Чебурашка, прежде чем продать что-нибудь ненужное, нужно купить что-нибудь ненужное. Прежде чем отбросить часть

[1] Разумеется, можно предложить и другие условия остановки: запускать алгоритм на определённое число поколений, например.

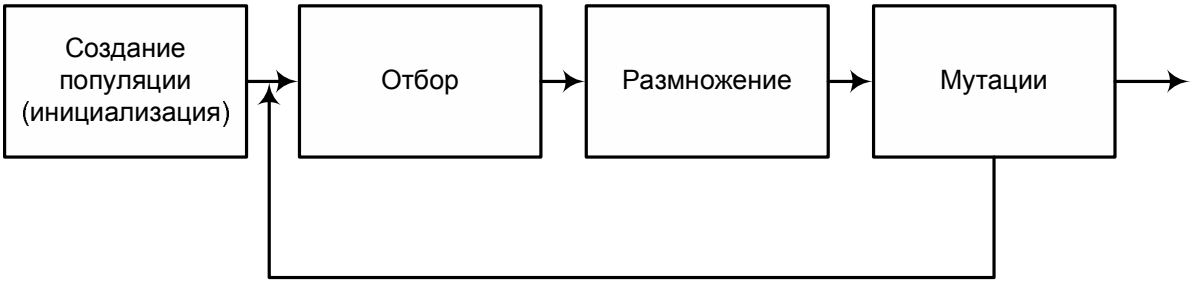

Рис. 3.1. Общая схема генетического алгоритма.

популяции, не выдержавшую естественного отбора, нужно её откуда-то породить. А перед первым шагом нужно случайным образом создать некую начальную популяцию; даже если она окажется совершенно неконкурентоспособной, генетический алгоритм всё равно достаточно быстро переведёт её в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию Fitness.[2] Итогом первого шага является популяция H, состоящая из N особей.

2. *Отбор*. На этапе отбора нужно из всей популяции выбрать определённую её долю, которая останется «в живых» на этом этапе эволюции. Есть разные способы проводить отбор — о них мы подробно поговорим в § 3.5, а пока упомянем очевидное: вероятность выживания особи h должна зависеть от значения функции приспособленности $\mathrm{Fitness}(h)$. Сама доля выживших s обычно является параметром генетического алгоритма, и её просто задают заранее. По итогам отбора из N особей популяции H должны остаться sN особей, которые войдут в итоговую популяцию H'. Остальные, увы, остаются на обочине жестокой жизни.
3. *Размножение*. Размножение в генетических алгоритмах обычно половое — чтобы произвести потомка, нужны несколько родителей; обычно, конечно,

[2] Не стоит забывать о том, что Чебурашка после этого добавлял: «. . . а у нас нет денег»; генетические операции, в том числе операцию инициализации, нужно реализовывать эффективно.

нужны ровно два. Размножение в разных алгоритмах определяется по-разному — оно, конечно, зависит от представления данных. Главное требование к размножению — чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, «смешав» их каким-либо достаточно разумным способом. В § 3.3 мы подробно рассмотрим разные варианты операции размножения на битовых строках. Вообще говоря, для того чтобы провести операцию размножения, нужно выбрать $\frac{(1-s)p}{2}$ пар гипотез из H и провести с ними размножение, получив по два потомка от каждой пары (если размножение определено так, чтобы давать одного потомка, нужно выбрать $(1-s)p$ пар), и добавить этих потомков в H'. В результате H' будет состоять из N особей.

Почему особи для размножения обычно выбираются из всей популяции H, а не из выживших на первом шаге элементов H' (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный бич многих генетических алгоритмов — недостаток разнообразия (diversity) в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них — выбор для размножения не самых приспособленных, но вообще всех особей.

4. *Мутации.* К мутациям относится всё то же самое, что и к размножению: есть некоторая доля мутантов m, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать mN особей, а затем изменить их в соответствии с заранее определёнными операциями мутации.

О том, какими могут быть генетические операции на практике, мы поговорим в следующем параграфе.

§ 3.3. Генетические операции

Для начала давайте предположим, что элементы популяции зашифрованы в виде битовых строк (о том, как это сделать, мы поговорим в § 3.4). С битовыми строками можно делать практически всё, что угодно; давайте рассмотрим, как на строках работают стандартные генетические операции.

С инициализацией всё понятно: порождать случайные битовые строки несложно. Для отбора конкретная форма представления данных не очень важна: достаточно уметь вычислять функцию приспособленности Fitness.

При размножении особь должна унаследовать черты обоих предков. На битовых строках это можно реализовать достаточно естественным путём: собрать итоговую строку из частей строк–родителей. Такая операция называется *кроссовером*

(crossover). Чтобы сделать кроссовер двух строк, нужно выбрать определённую *маску*, а затем в соответствии с этой маской выбрать те или иные биты родителей.

Определение 3.1. *Для двух строк одинаковой длины* $x_1 \dots x_n$ *и* $y_1 \dots y_n$ *результатом* кроссовера с маской $m_1 \dots m_n$ *является строка* $z_1 \dots z_n$, *где*

$$z_i = \begin{cases} x_i, & \text{если } m_i = 1, \\ y_i, & \text{если } m_i = 0. \end{cases}$$

В зависимости от того, как выбирается маска, различают несколько видов кроссовера.

1. *Одноточечный кроссовер* (single-point crossover). Простейший вид кроссовера; в нём случайно выбирается одна позиция в строке, и маска состоит из единиц левее этой позиции и нулей — правее (т.е. левее выбранной позиции потомок совпадает с первым родителем, а правее — со вторым). Например:

Исходные строки	Маска	Результат
*10011*01011, 00101*01100*	1111100000	1001101100

2. *Двухточечный кроссовер* (double-point crossover). То же самое, что и в предыдущем случае, но случайно выбираются две позиции, и между этими двумя позициями берутся биты одного из родителей, а вне выбранных позиций — другого. Если по каким-то причинам желательно, чтобы в потомке битов родителей было поровну, можно выбирать только одну позицию, а затем отсчитывать от неё ровно половину общей длины строки. Например:

Исходные строки	Маска	Результат
100*1101*011, *001*0101*00*	0001111100	0010101100

3. *Однородный кроссовер* (uniform crossover). Здесь маска выбирается случайным образом, равномерно. Например:

Исходные строки	Маска	Результат
1*00*1*1*01*0*1, *0*01*0*1*0*10	0110100110	0000101010

Итак, мы рассмотрели различные виды кроссовера на битовых строках. Осталось понять, какими могут быть мутации. Наиболее распространённая мутация — это, конечно, инвертирование случайного бита строки. Её уже достаточно для многих приложений. Для того чтобы определить другие типы мутаций, нужно более явно знать структуру предметной области. Мы приведём пример более специфичной мутации в следующем параграфе, а пока отметим, что даже обычное инвертирование бита уже является очень мощным инструментом.

§ 3.4. Представление данных

В предыдущем параграфе мы предполагали, что элементы популяции генетического алгоритма — это строки битов. Действительно, в таком случае все генетические

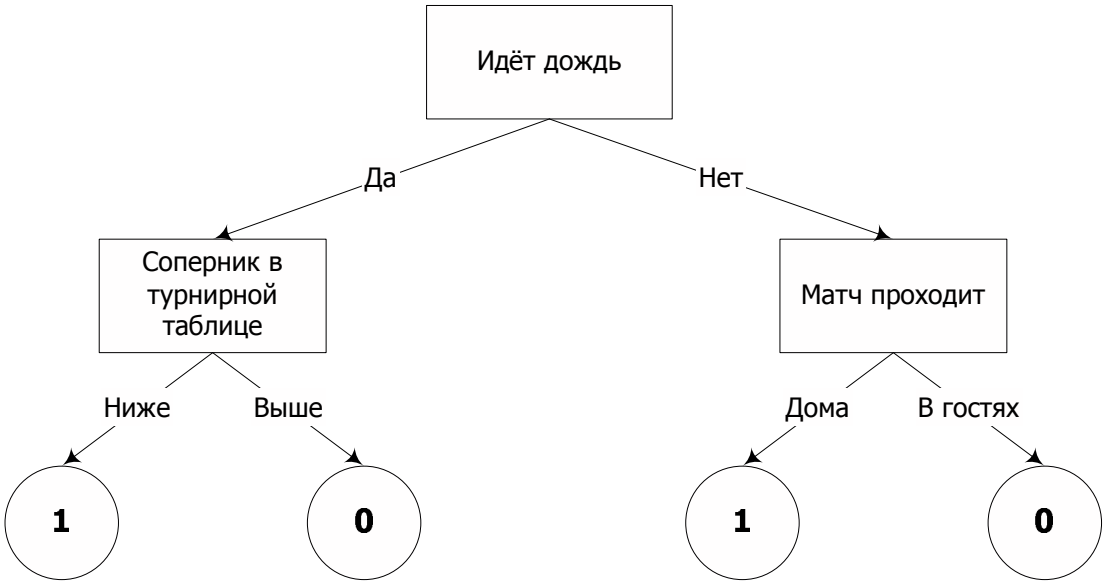

Рис. 3.2. Дерево принятия решений для игр «Зенита».

операции легко описываются и применяются, давая разумные результаты. Теоретически этого вполне достаточно: понятно, что любую, сколь угодно сложную структуру можно отобразить в слова алфавита из двух символов. Но как на практике зашифровать имеющиеся достаточно сложные структуры данных в строки битов?

Давайте рассмотрим задачу классификации — пример, которому мы неотступно следуем ещё с Главы 1. Пусть результаты игр «Зенита» находятся в зависимости от четырёх параметров (см. Главу 1, где это описано более подробно). Тогда классификация, представленная в виде дерева, может быть записана в виде набора гипотез, соответствующих листьям дерева (точнее говоря, путям от корня к каждому из листьев). Например, крайняя правая ветка дерева с рис. 3.2 представляет собой гипотезу

$$(\text{Дождь} = \text{Нет}) \wedge (\text{Играем} = \text{В гостях}) \Rightarrow (\text{Победа} = 0).$$

Как закодировать гипотезу такого рода в виде строки битов? Первая мысль — просто выделить по одному биту на каждый (бинарный) атрибут и на целевую функцию. Например, в примере с играми «Зенита» четыре атрибута — турнирное положение соперника, место проведения матча, игра лидеров и погода, и строка 00101 соответствовала бы гипотезе

$$(\text{Соперник} = \text{Ниже}) \wedge (\text{Играем} = \text{В гостях}) \wedge$$
$$\wedge (\text{Лидеры} = \text{На месте}) \wedge (\text{Дождь} = \text{Нет}) \Rightarrow (\text{Победа} = \text{Да}).$$

Это кодирование, очевидно, является наиболее эффективным, и, что также способствует успеху генетического алгоритма, все строки имеют одинаковую длину, и каждая строка (заданной длины) имеет смысл (т.е. описывает реальную гипотезу). Однако его недостатки также видны невооружённым взглядом. Как представить гипотезу, в которой не все значения атрибутов специфицированы, а, наоборот, от некоторых из них ничего не зависит? Если это невозможно (а в нашем текущем представлении это невозможно), то любое дерево придётся «разворачивать» до полного набора гипотез (любую ДНФ придётся преобразовывать в СДНФ), что явно неэффективно. Можно было бы представлять гипотезы с атрибутами без фиксированных значений строками меньшей длины, но тогда было бы не понятно, какие атрибуты заданы, а какие — нет.

В данном случае общепринятым решением возникшей проблемы будет следующая конструкция. Для каждого атрибута нужно выделить столько битов, сколько у него возможных значений (мы до сих пор рассматривали только бинарные атрибуты, но ничто не помешает рассмотреть и атрибуты с бóльшим количеством значений). Ноль в той или иной позиции означает, что данное значение *не* встречается в посылке правила, а единица — что встречается. В такой нотации, если все биты, соответствующие значениям данного атрибута, равны 1, это означает, что значение его (атрибута) не играет никакой роли для применения этого правила. Для значения функции мы оставим один бит — правила, где целевая функция не определена или разрешено любое её значение, лишены смысла. Таким образом, в случае примера с играми «Зенита» четыре бинарных атрибута дадут восемь бит посылки плюс один бит значения целевой функции. Например, строка

$$01\ 10\ 11\ 11\ 1$$

эквивалентна правилу

$$(\text{Соперник} = \text{Выше}) \wedge (\text{Играем} = \text{В гостях}) \Rightarrow (\text{Победа} = \text{Да}).$$

У этой системы кодирования осталось только одно тонкое место: что делать, если в результате кроссовера или мутации в гипотезе получится набор из символов 00. Такая гипотеза не может быть применена ни разу — её условия соответствуют пустому множеству примеров. Здесь есть два пути. Первый — просто запретить появление таких гипотез: повторять кроссовер или мутацию до тех пор, пока результат не будет разумным. Второй — оставлять такие гипотезы в популяции, но присваивать им нулевое значение функции Fitness; тогда эти гипотезы наверняка будут «выполоты» на следующем шаге отбора. Оба подхода имеют право на существование; позволим себе рекомендовать второй подход как более идейный: отбраковка гипотез должна производиться посредством естественного отбора, а не искусственных ограничений.

В § 3.3 мы обещали привести пример более специфичной мутации, зависящей от представления данных. В случае задачи классификации весьма успешной мутацией оказывается удаление из посылки правила случайно выбранного атрибута (замена любой строки, соответствующей этому атрибуту, на строку из единиц). Такая мутация соответствует обобщению полученных гипотез и зачастую приводит к хорошим результатам.

Впрочем, пока что это всё ещё не совсем то, что нужно для решения задачи. Ведь мы до сих пор описывали отдельные правила, т.е. отдельные ветки дерева принятия решений. Но одна гипотеза — это не одна ветка, а несколько (все ветви дерева).

Как закодировать несколько правил? Здесь, к счастью, ответ прост: при фиксированном наборе атрибутов все правила будут иметь постоянную длину, равную суммарному количеству возможных значений атрибутов плюс один бит на значение целевой функции (или больше, если целевая функция может принимать больше двух значений). Поэтому можно просто записывать правила подряд — никакой многозначности это не внесёт.

Гораздо более серьёзная проблема происходит от того, что в разных деревьях разное количество веток и, следовательно, в разных гипотезах будет разное количество правил.

Как же делать кроссовер со строками переменной длины? Нам ведь нужно не только суметь скрестить две строки длиной $l_1 n$ и $l_2 n$ каждая, где n — длина правила, а l_i — количество правил в каждой гипотезе, но и получить в результате строку длиной ln — будет неприятно получить гипотезу, в которой дробное количество правил. Причём желательно, чтобы это l выбиралось более или менее случайно из промежутка от 1 до $l_1 + l_2$, а не всегда равнялось, скажем, $\max\{l_1, l_2\}$.

Одно из возможных решений таково: будем использовать двухточечный кроссовер так, чтобы сохранять постоянное расстояние до краёв правил. Тогда при замене участка первой гипотезы на участок второй, хотя эти участки могут быть разной длины, их длина по модулю n будет постоянной. Этот принцип проще всего объяснить на примере.

Пример 3.1. Кроссовер на строках переменной длины.

Предположим, что в нашем примере участвуют правила длины 5, и мы случайно выбрали две точки из первой гипотезы:

0[0101 110]10.

Тогда во второй гипотезе нужно выбирать такие точки, чтобы расстояние от левой точки до левого края правила и от правой точки до правого края правила были теми же. Например,

в правиле длины 15 могут быть варианты:

1[10]11 01010 01110	1[1011 010]10 01110
1[1011 01010 011]10	11011 0[10]10 01110
11011 0[1010 011]10	11011 01010 0[11]10

Теперь кроссовер будет порождать корректные гипотезы:

Исходные строки	Результат
0[0101 110]10	01010
1[10]11 01010 01110	10101 11011 01010 01110

§ 3.5. Отбор

Мы уже рассмотрели все элементы типичного генетического алгоритма на примере задачи классификации, кроме одного — собственно естественного отбора. Более того, мы даже не определили функцию приспособленности Fitness. Эту функцию мы будем определять следующим образом:

$$\mathrm{Fitness}(h) = \left(\frac{\mathrm{Correct}(h)}{\mathrm{TotalExamples}}\right)^2,$$

где $\mathrm{Correct}(h)$ — количество примеров, верно расклассифицированных гипотезой h, TotalExamples — общее количество примеров. Отметим, что такая функция приспособленности прекрасно справляется с проблемой гипотез, не соответствующих никаким конкретным примерам — они не смогут классифицировать ни одного примера, и их приспособленность будет равна нулю.

Как же теперь выбрать наиболее приспособленные особи (нам нужно отобрать sN особей из популяции размера N)? Не забудем то, о чём мы уже упоминали в § 3.2: главная беда генетических алгоритмов — недостаточная гибкость, которая приводит к тому, что популяция становится слишком однородной и не может выбраться из локального максимума. Поэтому просто отбирать верхние sN особей по функции Fitness — не самая хорошая идея.

Обычно используют один из двух наиболее популярных методов. Первый из них — так называемый *метод рулетки* (roulette wheel selection). В этом методе у каждой гипотезы с ненулевой функцией приспособленности есть шанс быть выбранной, и вероятность её выживания пропорциональна её функции приспособленности: у каждой гипотезы h_i вероятность быть выбранной

$$\Pr(h_i) = \frac{\mathrm{Fitness}(h_i)}{\sum_{j=1}^{N} \mathrm{Fitness}(h_j)}.$$

При этом одна и та же особь может быть выбрана несколько раз; алгоритм просто проводит выбор с заданной вероятностью в течение нужного количества итераций

$\mathrm{Genetic}(N, p, s, m, \mathrm{Fitness}, \mathrm{Fitness}_{\max})$

1. Создать N случайных гипотез $H = \{h_1, \ldots, h_n\}$.
2. Для каждой гипотезы $h \in H$ вычислить $\mathrm{Fitness}(h)$.
3. Пока $\max_h \mathrm{Fitness}(h) < \mathrm{Fitness}_{\max}$:
   а) $H' = \emptyset$.
   б) Случайно выбрать sN гипотез из H и добавить их в H'. Вероятность выбрать гипотезу h_i $\Pr(h_i) = \frac{\mathrm{Fitness}(h_i)}{\sum_{j=1}^{N} \mathrm{Fitness}(h_j)}$.
   в) Случайно выбрать $\frac{(1-s)p}{2}$ пар гипотез из H с теми же вероятностями. Для каждой пары (h_i, h_j) запустить операцию кроссовера и добавить её результат в H'.
   г) Равномерно выбрать mN случайных гипотез из H' и в каждой из них инвертировать случайный бит.
   д) $H = H'$.
   е) Для каждой гипотезы $h \in H$ вычислить $\mathrm{Fitness}(h)$.
4. Выдать $\mathrm{argmax}_h \mathrm{Fitness}(h)$.

Рис. 3.3. Пример генетического алгоритма.

(sN). Этот метод — один из самых классических, но на практике используется не так уж часто. Во-первых, такой сэмплинг достаточно дорог вычислительно, а во-вторых, опять же, этому методу зачастую не хватает разнообразия в получающихся гипотезах. Вторую проблему можно отчасти решить, перейдя от метода рулетки к *ранговому методу*. В ранговом методе гипотезы (особи) сначала сортируются по приспособленности, а затем используется такой же сэмплинг, но вероятность быть выбранной у гипотезы прямо пропорциональна не абсолютному значению её приспособленности, а её рангу. Даже если гипотеза далеко отрывается от своих конкурентов по значению Fitness, ранговый метод всего лишь поставит её на первое место, девальвируя разницу в абсолютных значениях.

Другой часто используемый метод — *турнирный метод* (tournament selection). Сначала случайно выберем две гипотезы. Затем с некоторой фиксированной вероятностью p (обычно, конечно, $p > \frac{1}{2}$) выживает более приспособленная, с вероятностью $(1 - p)$ — менее приспособленная гипотеза.

Давайте попробуем объединить всё то, что мы до сих пор изучили, в единую схему алгоритма. На рис. 3.3 изображён пример генетического алгоритма. В нём мы избрали метод рулетки в качестве метода естественного отбора и инвертирование случайного бита — в качестве единственного типа мутаций.

§ 3.6. Дарвин, Ламарк и Болдуин

Как известно, Дарвин был не единственным учёным, предложившим схему эволюции. Жан Батист Ламарк в своей опубликованной в 1809 году (за полстолетия до Дарвина) книге «Философия зоологии» уже высказывал основные идеи эволюционного развития видов. При жизни Ламарка его мысли не получили поддержки среди научного сообщества. Вспомнили о нём только после выхода в свет книги Дарвина, когда реального научного значения его идеи уже не имели — все мыслимые подтверждения указывали на то, что прав был Дарвин, а не Ламарк.

Основной идеей Ламарка было то, что организмы изменяются под воздействием окружающей среды и условий их жизнедеятельности. Главное отличие от дарвиновской теории состояло в том, что по Ламарку виды могут изменяться в течение своей жизни, а не только на генетическом уровне. Грубо говоря, киплинговский слонёнок даже без помощи крокодила действительно мог бы за свою жизнь чуть-чуть вытянуть хобот, это бы передалось его детям, те вытянули бы хоботы ещё чуть-чуть, и рано или поздно они бы эволюционировали в современных слонов.

Хотя теория Ламарка не выдерживает биологической критики, кто сказал, что она совсем бесполезна для искусственного интеллекта? Фактически, все программы машинного обучения, которые мы рассматривали в предыдущих главах — это ламаркианские виды, которые обучаются по ходу жизни, а не только посредством естественного отбора. И в рамках общей парадигмы генетических алгоритмов мы тоже можем совместить естественный отбор (его идею Ламарк, кстати, тоже выдвинул — выживают те слонята, которые сумели вытянуть хобот максимально далеко) с обучением «on-the-fly».

Для этого нужно будет в качестве гипотез (особей популяции) использовать какой-нибудь из аппаратов машинного обучения. А затем обучать их на тестовых примерах. Обычно в качестве такого аппарата применяют нейронные сети: помимо прочего, их ещё и легко обучить «чуть-чуть», так, чтобы «генетический код» не потерялся от обучения; а, например, деревья принятия решений после работы алгоритма ID3 все были бы на одно лицо.

Другая интересная мысль, также пришедшая в искусственный интеллект из биологии — это так называемый *эффект Болдуина*.[3] Он связан со способностью организма обучаться в течение жизни. Как повлияет такая способность на эволюцию? Она сможет «сгладить» зависимость приспособленности от генотипа: чем лучше обучается особь, тем меньше она зависит от генотипа. Например, человек в процессе своей жизни обучается стольким разным вещам, что в итоге от генотипа конкретного человека зависит в его жизни очень мало, и совершенно не понятно, какие мутации

[3] На самом деле его независимо и практически одновременно — в 1896 году — открыли Болдуин (Baldwin), Морган (Morgan) и Осборн (Osborn).

могут оказаться для человека благоприятными и быть закреплёнными естественным отбором.

Болдуиновский эффект не является, как может показаться, развитием ламаркианских идей. Обученные навыки не передаются по наследству. Слонята учатся вытягивать хобот, чтобы достать еду, но от этого их хоботы не становятся длиннее. Однако обучение делает их более приспособленными к окружающей среде и даёт популяции слонов время для того, чтобы развить у себя длинный хобот сугубо дарвиновскими методами [4].

Применительно к искусственному интеллекту болдуиновский эффект выражается в том, что в естественном отборе участвуют обучающиеся особи, причём их способность к обучению также является предметом естественного отбора.

Пример 3.2. Генетический алгоритм, реализующий эффект Болдуина.

Опишем конкретную (впрочем, опишем мы её в достаточно общем виде, без элементов конкретной реализации) схему построения генетического алгоритма (основных его операций) на нейронных сетях, в основном следуя [7].

Каждый элемент популяции — нейронная сеть глубины 2 с N входами, M нейронами на скрытом уровне и N нейронами на выходе. Таким образом, у такой особи образуются $(N+1)M+(M+1)N$ генов, соответствующих весам нейронной сети («плюс единицы» образуются вследствие того, что у каждого нейрона есть ещё вес w_0, добавляющий константу к взвешенной сумме входов — см. Главу 2). Более того, у каждой особи есть ещё столько же бинарных генов (*генов пластичности*), которые определяют, может ли соответствующий вес изменяться в процессе обучения или нет.

Каждая итерация алгоритма состоит из двух частей. В течение первой части имеющиеся особи обучаются на тестовых примерах; при этом обучение учитывает также гены пластичности: для веса w обучение происходит по формуле

$$\Delta w = -\eta p \sum_{v \in V} \frac{\partial E_v(w)}{\partial w},$$

где η — скорость обучения, V — тестовые примеры, E_v — функция ошибки, а p — это как раз соответствующий весу w ген пластичности. Таким образом, если этот ген равен 1, то вес может изменяться в процессе обучения, а если он равен 0, то вес на протяжении обучения останется постоянным.

На втором этапе обученные нейронные сети участвуют в генетических операциях. Эти операции производятся с функцией приспособленности

$$\text{Fitness} = 1.0 - \frac{1}{N2^N} \sum_{v \in V} \sum_{i=0}^{N-1} (\text{Out}_{v,i} - \text{Target}_{v,i})^2,$$

где $\text{Target}_{v,i}$ — значение целевой функции на i-м выходе сети, а $\text{Out}_{v,i}$ — реальный выход i-го выходного нейрона сети.

Отбор и размножение в [7] велись очень простым способом: наихудшая особь популяции заменялась копией наилучшей особи. Можно было бы заменять несколько худших на несколько лучших или всё-таки использовать более традиционные методы; с другой стороны, большая вычислительная стоимость каждой итерации (нужно обучить каждую особь) приводит к тому, что в популяции должно быть не так уж много особей. Возможно, с учётом этого обстоятельства выбор авторов [7] действительно оптимален.

Главная изменчивость в этом примере достигается за счёт мутаций. Мутации бывают двух видов. Мутации весов нейронной сети изменяют случайно выбранный вес на случайное значение из заданного интервала $[-\mathrm{d}, \mathrm{d}]$. Мутации генов пластичности — обычное инвертирование случайного бита.

На практике оказывается, что ламаркианский подход даёт хорошие сети достаточно быстро, в то время как болдуиновский подход позволяет получить сети более высокого качества, но медленнее. Если у задачи нет большого количества локальных минимумов, и результат можно получить относительно быстро, то лучше использовать ламаркианскую стратегию; а в общем случае с задачей лучше справляется подход, основанный на эффекте Болдуина [2].