МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

С.В. Аксенов, В.Б. Новосельцев

ОРГАНИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ НЕЙРОННЫХ СЕТЕЙ
(методы и технологии)

Томск – 2006

УДК 004.89
А 424

Аксенов С.В., Новосельцев В.Б. Организация и использование нейронных сетей (методы и технологии) / Под общ. ред. В.Б. Новосельцева. – Томск: Изд-во НТЛ, 2006. – 128 с.

А 424

ISBN 5-89503-285-0

Теория нейронных сетей является одним из разделов науки об искусственном интеллекте. Начиная с середины двадцатого века искусственные нейронные сети стали развиваться вместе с вычислительной техникой и появлением новых знаний о биологической структуре головного мозга. Среди всех подходов при создании искусственного интеллекта методы теории нейронных сетей отличаются тем, что они основаны на моделировании структуры головного мозга. Эти методы позволяют, в меру современного представления, использовать принципы обработки информации, свойственные человеческому мозгу. Любой ребенок способен говорить сам и понимать речь окружающих, узнавать лица родителей и друзей в различных ситуациях. Но не секрет, что подобные слабо формализуемые задачи из нашей жизни довольно сложно реализовать даже на современных вычислительных машинах. При помощи нейронных сетей становится реальным моделирование таких возможностей мозга, как обучаемость, ассоциативная память, способность к неосознанному управлению, помехоустойчивость и адаптивность.

В рамках данной монографии делается попытка кратко изложить современное состояние теории нейронных сетей и дать максимальное количество ссылок на доступные публикации для более углубленного изучения. Книга рассчитана на учащихся математических специальностей вузов и всех интересующихся теорией нейронных сетей.

УДК 004.89

Рецензенты: В.Г. Спицин, докт. техн. наук, профессор Томского политехнического университета
В.Т. Калайда, канд. техн. наук, доцент Томского университета систем управления и радиоэлектроники

ISBN 5-89503-285-0

Глава 1

ПРЕДЫСТОРИЯ ВОПРОСА

1.1. Биологический прототип

Представление о детальном устройстве головного мозга появилось только около ста лет назад. В 1888 г. испанский доктор Рамони Кайал экспериментально показал, что мозговая ткань состоит из большого числа связанных друг с другом однотипных узлов – нейронов.

Более поздние исследования при помощи электронного микроскопа показали, что все нейроны, независимо от типа, имеют схожую организационную структуру (рис. 1.1). От центральной части нейрона – сомы отходят древовидные отростки – дендриты. Они играют роль рецепторов – сигналов от других нейронов. Задача аксона, самого крупного отростка, заключается в том, чтобы передавать сигнал активности от сомы другим нейронам. Место соединения аксона с дендритами других нейронов разделено малым, порядка 200 нм, расстоянием. Этот промежуток принято называть синапсом.

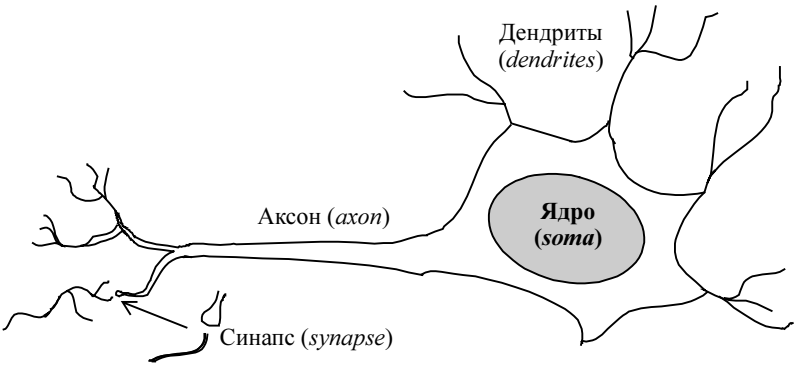


Рис. 1.1. Структура нейрона

Сигналы между нейронами передаются химическим и электрическим путем. В состоянии покоя протоплазма нейрона заряжена отрицательно, с потенциалом порядка 70 мВ. По мере поступления сигналов от дендритов заряд деполяризуется до значения потенциала 60 мВ, происходит диффузия в сому положительно заряженных ионов натрия Na+, нейрон «срабатывает» – его заряд резко повышается до положительного, и затем возбуждение распространяется через аксон на другие нейроны. Затем ядро постепенно приходит в свое исходное состояние. Нейроны головного мозга «срабатывают» несинхронно, подобно цифровым устройствам. Частота срабатывания для разных нейронов может варьироваться от 1 до 100 Гц. Скорость распространения сигнала лежит в диапазоне от 0,5 до 2 м/с. Помимо этого было замечено, что сила синаптической связи для разных нейронов различна и более того – не постоянна. Если нейрон «срабатывает» чаще остальных, то сила его синаптической связи возрастает. Подобная адаптивность синапса носит название правила Хебба (см. далее п. 2.4).

Общая структура головного мозга весьма сложна. Число нейронов можно оценить приблизительно как 10^{11}. Каждый нейрон в среднем имеет 10^4 синаптических связей, а всего мозг имеет 10^{15} соединений. Более подробно о биологических аспектах мозга можно узнать в [1, 2]. Из всего изложенного выше видно, что головной мозг – это очень сложная система с множеством параметров, связей и внешних сенсоров, и до конца понять, как происходит процесс мышления, мы пока не в состоянии.

1.2. История

В 1943 г. В. Маккаллох и В. Питтс предложили систему обработки информации в виде сети, состоящей из простых вычислителей, созданных по принципу биологического нейрона. Каждый такой элемент $i = 1, 2...n$ имеет входы и выходы, принимающие значения 0 или 1. Состояние отдельного нейрона определяется влиянием остальных как взвешенная линейная комбинация $\sum w_{ij} n_j$ их выходов n_j. К сумме затем применяется пороговая функция вида $g(\sum < 0) = 0$; $g(\sum \geq 0) = 1$. Маккаллох и Питтс показали, что такие сети могут производить произвольные вычисления, подобно известной машине Тьюринга. При этом единственной нерешенной проблемой оставался подбор весов w_{ij} – настраиваемых параметров для задачи. В 1962 г. Ф. Розенблатт для реше-

ния проблемы классификации символов предложил использовать особый тип искусственной нейронной сети, названный им персептроном [3]. В этой же работе им был предложен итеративный алгоритм получения весов w_{ij}, моделирующих силу синаптической связи. Значения весов получались по известным входным значениям и соответствующим желаемым выходам сети.

Однако уже в 1969 г. М. Минский и С. Паперт в своей известной работе [4] показали, что персептрон не может решать целый класс простых задач, таких как, например, реализация логической операции XOR (исключающее ИЛИ). Появление этой работы сыграло роковую роль для теории нейронных сетей, все исследования в этой области фактически были приостановлены вплоть до середины 80-х годов.

В 1986 г. ситуация изменилась – Д. Румельхарт, Г. Хинтон и Р. Вильямс предложили эффективный алгоритм для обучения более совершенного, так называемого многослойного, персептрона [5]. Алгоритм получил название алгоритма обратного распространения ошибки (он детально обсуждается в п. 2.2).

Начиная с конца 80-х и по настоящее время теория нейронных сетей испытывает настоящий бум по количеству научных публикаций. В современный период нейронные сети все чаще применяются для решения разнообразных практических задач. Более того, можно с уверенностью утверждать, что программно-аппаратные устройства, построенные на базе нейронных сетей, активно внедряются в нашу жизнь и все чаще используются в медицинской диагностике, промышленности, финансовой системе и в повседневной жизни – там, где необходимо решать задачи, до этого подвластные только человеку.

1.3. Формальный нейрон

В 1962 г. Розенблатт предложил следующую модель нейронной сети (персептрона – [3]).

Нейронная сеть (рис. 1.2) состоит из k нейронов, имеет d входов, k выходов и только один слой настраиваемых весов w_{ij}.

Нейрон имеет структуру, представленную на рис. 1.3. Каждый j-й нейрон сети вычисляет взвешенную сумму своих входов:

$$y_j(x) = \sum_{i=1}^{d} x_i w_{ij} + w_0 \ ,$$

где w_0 – порог нейрона.

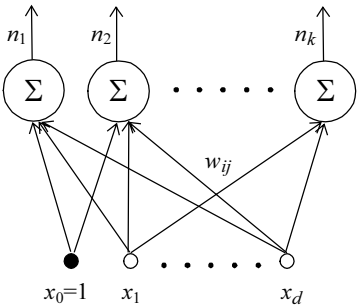


Рис. 1.2. Персептрон Розенблатта

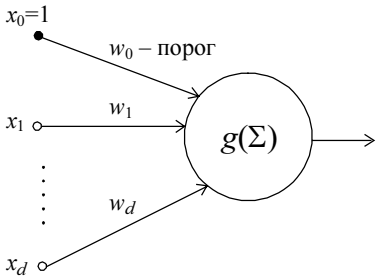


Рис. 1.3. Структура нейрона

Обычно для удобства входной вектор расширяется до $x = (1, x_0, ..., x_d)$ и порог w_0 вносится под знак суммы:

$$y_j(x) = \sum_{i=0}^{d} x_i w_{ij} \ . \tag{1.1}$$

Выходной сигнал нейрона определяется нелинейной пороговой функцией $g(a)$, изображенной на рис. 1.4:

$$g(a) = \begin{Bmatrix} -1, a < 0, \\ +1, a \geq 0. \end{Bmatrix} \tag{1.2}$$

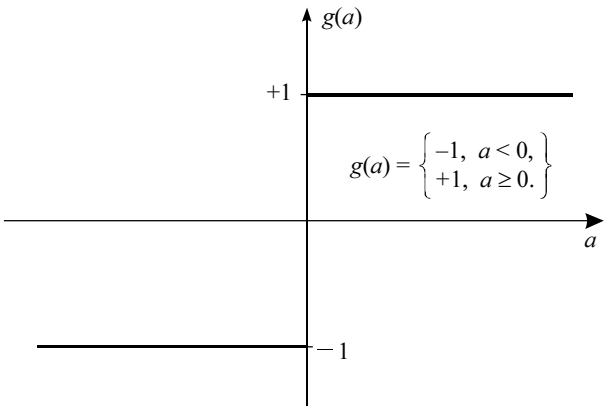


Рис. 1.4. Пороговая функция

Розенблатт предложил использовать персептрон для задач классификации. Рассмотрим простой случай и его геометрическую интерпретацию. Имеем обучающий набор 2-компонентных входных векторов вида $x = (x_1, x_2)$, каждый из которых принадлежит к одному из двух разных классов C_1 и C_2. Требуется: для любых новых векторов определить их принадлежность к классу. Для этого можно использовать персептрон всего с одним нейроном. Из уравнения (1.1) видно, что нейрон строит на плоскости прямую (в случае d-мерного пространства – гиперплоскость). Входные векторы соответственно будут попадать в ту или иную полуплоскость. Применение пороговой функции (1.2) позволяет определить полуплоскость, принявшую вектор. Вид прямой будет полностью определяться настраиваемыми весами w_{ij} и порогом w_0.

На рис. 1.5 сплошной линией обозначена прямая, которая неправильно разбивает векторы на классы (точки разных видов попадают в один класс), прямая с пунктирной линией будет правильно классифицировать обучающий набор и, с высокой вероятностью, новые векторы. Такую прямую назовем границей решения персептрона. Процесс определения весов задается определенным алгоритмом, который принято называть обучением персептрона.

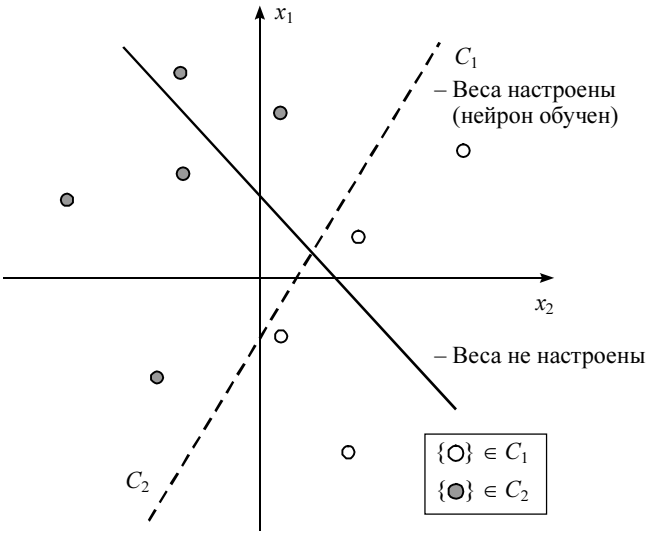


Рис. 1.5. Геометрическая интерпретация

Обучение персептрона основано на способе обучения «с учителем». Имеем набор обучающих входных векторов $\{x^n\}$ и соответствующий им набор значений желаемых выходов персептрона $\{t^n\}$. Будем полагать, что $t^n = +1$ если входной вектор x^n принадлежит классу C_1, и $t^n = -1$, если x^n принадлежит классу C_2. Из (1.1) и (1.2) видно, что для правильной классификации входных образов необходимо для x^n из C_1 ($t^n = +1$) иметь $W^T x > 0$, где $W^T x$ – матричная форма записи (1.1). Для x^n из C_2 ($t^n = -1$) необходимо $W^T x < 0$. В общем случае имеем $\forall x\ W^T(xt) > 0$. Отсюда следует, что для правильной работы персептрона необходимо минимизировать функцию ошибки, названную критерием персептрона:

$$E(w) = -\sum_{x^n \in M} W^T(xt)\,. \qquad (1.3)$$

Здесь M – набор неверно классифицированных данных для весов w.

Из определения ошибки (1.3) следует очень простая процедура обучения персептрона [6].

Процедура обучения персептрона

Ш0. На начальном этапе веса w_{ij} инициализируются случайными значениями.

Ш1. Подаем на вход персептрона вектор x^n из обучающей выборки.

Ш2. На выходе возможны три варианта:

1) если персептрон классифицировал входной вектор правильно, то ничего не делаем;

2) если неправильно и x^n должен принадлежать классу C_1 ($t^n = +1$), то прибавляем x^n к весу;

3) если неправильно и x^n должен принадлежать классу C_2 ($t^n = -1$), то отнимаем x^n от веса.

Кратко процесс изменения весов по вариантам 2 и 3 можно записать формулой

$$W^{\tau+1} = W^{\tau} + \eta(wt)\,, \qquad (1.4),$$

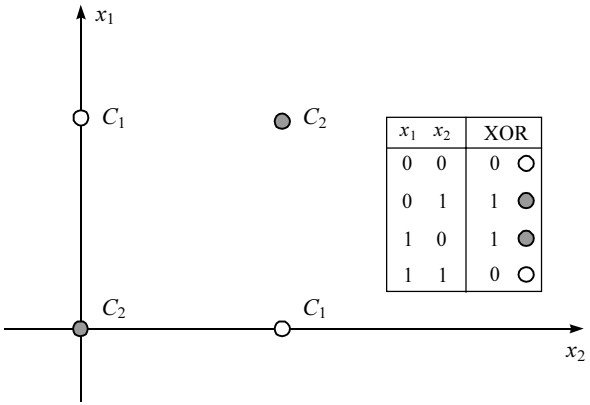
где $\eta > 0$ – коэффициент, задающий скорость обучения; $W^{\tau+1}$ – матрица новых значений весов.

Покажем, что при обучении по правилу (1.4) ошибка (1.3) будет уменьшаться. Так как $\eta > 0$ и $(xt)^2 > 0$, имеем

$$E^{\tau+1}(w) = -W^{\tau+1}(xt) = -W^{\tau}(xt) - \eta(xt)(xt) < W^{\tau}(w) = E^{\tau}(w)\,. \qquad (1.5)$$

Факт уменьшения ошибки еще не доказывает сходимость алгоритма. В 1973 г. Р. Дуда и П. Харт доказали теорему сходимости персептрона [7]. Для любых линейно разделяемых входных данных правило (1.4) находит решение за конечное число шагов.

Ключевым моментом в формулировке теоремы является факт линейной разделимости данных. Как отмечалось выше, Минский и Паперт в своей работе [4] показали, что персептрон не может решать целый класс простых задач. Из рис. 1.6. видно, что для XOR персептрон, в принципе, не сможет правильно разместить точки по соответствующим классам.



x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

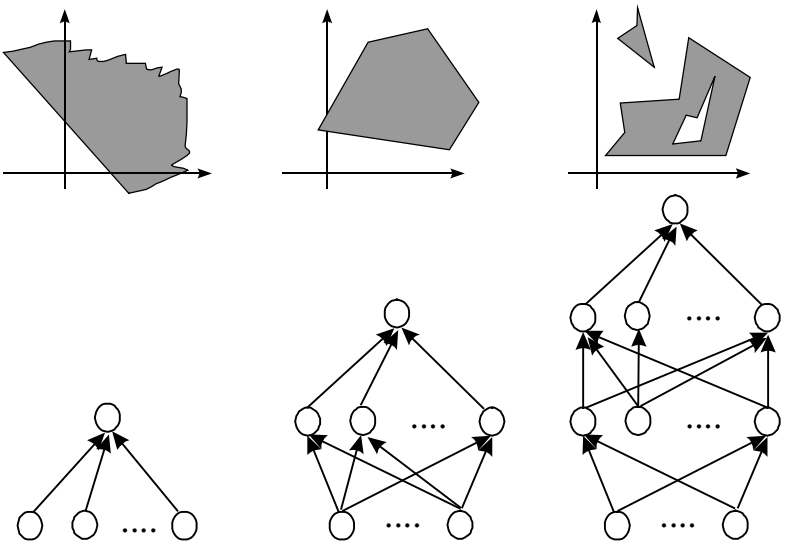
Рис. 1.6. Проблема XOR

Заметим, что добавление нейронов в персептрон не решает проблему в силу того, что это ведет только к увеличению числа классов, к которым можно отнести входные векторы. Как отмечалось выше, такое серьезное ограничение персептрона привело к длительному застою в теории нейронных сетей, пока в середине 80-х XX в. годов не была разработана процедура обучения многослойного персептрона.

1.4. Возможности многослойного персептрона

Недостатки персептрона можно преодолеть, если увеличить число слоев нейронов в персептроне. Было доказано, что уже для двух слоев настраиваемых весов граница решения принимает вид выпуклой комбинации (рис. 1.7, б). При трех и более слоях персептрона граница решения может состоять из несмежных областей, ограниченных гипер-

плоскостями (рис. 1.7, в). Таким образом, с помощью многослойного персептрона можно решать задачи произвольной сложности, а не только линейно разделяемые.



а) персептрон б) 2-уровневый персептрон в) 3-уровнеый персептрон

Рис. 1.7. Решающие границы персептрона

1.5. Классификация нейронных сетей

В настоящее время кроме многослойного персептрона существует множество способов задания нейроподобных структур. Все виды нейронных сетей можно условно разделить на сети прямого распространения и сети с обратными связями. Как следует из названия, в сетях первого типа сигналы от нейрона к нейрону распространяются в четко заданном направлении – от входов сети к ее выходам. В сетях второго типа выходные значения любого нейрона сети могут передаваться к его же входам. Это позволяет нейронной сети моделировать более сложные процессы, например временные, но делает выходы подобной сети нестабильными, зависящими от состояния сети на предыдущем цикле (см. далее п. 2.3). На рис. 1.8. собраны наиболее распространенные ти-

пы нейронных сетей. Разнообразие нейронных сетей увеличивается еще больше, благодаря огромному количеству алгоритмов и методик обучения, а также наличию нескольких видов пороговых функций.

Рис. 1.8. Классификация нейронных сетей

1.6. Пороговые функции

На практике пороговую функцию вида (1.2) очень часто заменяют похожими по форме, но обладающими лучшими свойствами функциями: логическим сигмоидом (1.6) (рис. 1.9) и гиперболическим тангенсом (рис. 1.10). Эти функции фактически реализуют пороговую функцию, но при этом являются гладкими, дифференцируемыми, что является принципиально важным для реализации многих алгоритмов обучения:

$$g(a) = \frac{1}{1 - e^{-a}}; \tag{1.6}$$

$$g(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}. \tag{1.7}$$

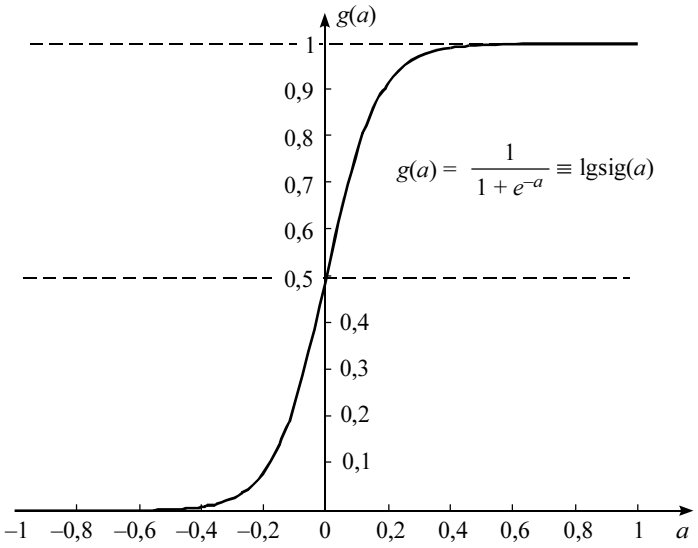


Рис. 1.9. Логический сигмоид

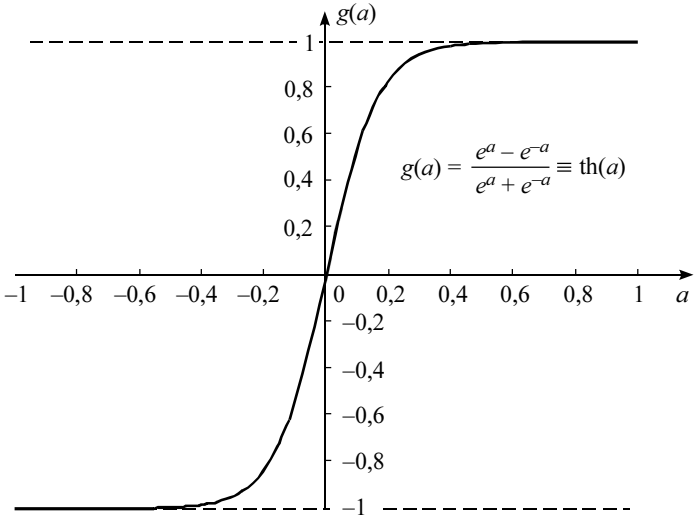


Рис. 1.10. Гиперболический тангенс

1.7. Обучение нейронных сетей

Настройка нейронных сетей производится при помощи трех парадигм обучения:

- обучение «с учителем». При данном подходе считается, что для каждого примера x^n из обучающей выборки $\{x^n\}$ заранее известен желаемый результат $t \in \{t^n\}$ работы нейронной сети. Это позволяет эффективно корректировать веса сети. Примером обучения «с учителем» может служить процедура обучения персептрона (п. 1.3). Очевидным недостатком подобного подхода является то, что не всегда имеется достаточное количество примеров «с ответами», а порой их и вовсе невозможно получить. В этом случае используется:

- обучение «без учителя». Этот тип обучения предполагает, что желаемые выходы сети не определены, и алгоритм обучения нейронной сети подстраивает веса по своему усмотрению. Фактически при этом нейронная сеть ищет закономерности в обучающих данных и выполняет группирование схожих входных векторов по неявным признакам. Чаще всего этот метод используется для задач классификации. Примером такого типа обучения служит алгоритм обучения Кохонена (см. далее п. 3.4);

- обучение методом критики. Данный подход является фактически промежуточным между первыми двумя. Предполагается, что имеется возможность только оценивать правильность работы сети и указывать желаемое направление обучения. Подобная ситуация часто встречается в системах, связанных с оптимальным управлением. Данный подход, например, используется при стратегии «кнута и пряника», предложенной Г. Барто в 1985 г. [8]. Подобная нейронная сеть состоит из так называемых эгоистичных нейронов. Процедура обучения таких нейронов поощряет каждый отдельный нейрон к увеличению только собственной «награды», а не производительности всей сети, как реализовано в методе обратного распространения ошибки (см. далее п. 2.2).

Глава 2

СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

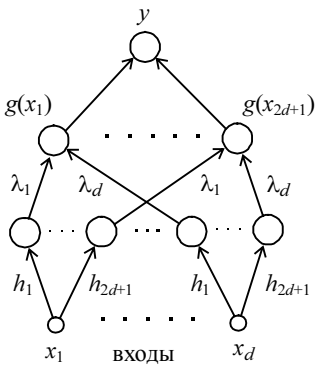
2.1. Теорема Колмогорова

В разд. 1.4 было показано, что задачи, с которыми не в состоянии работать персептрон, можно решать при помощи многослойного персептрона. Однако формального математического доказательства не было приведено. В 1957 г. Колмогоровым была доказана теорема, позволяющая говорить о том, что для решения любой задачи возможно построить нейронную сеть.

Теорема Колмогорова. Каждая непрерывная функция d переменных, заданная на единичном кубе d-мерного пространства, представима в виде

$$f(x_1,...,x_d) = \sum_{q=1}^{2d+1} h_q \left\{ \sum_{p=1}^{n} \varphi_q^p(x_p) \right\}, \qquad (2.1)$$

где h_q– непрерывные негладкие функции; $\varphi_q^p(x_p)$ – стандартные функции, не зависящие от вида функции f. Доказательство этой теоремы и ее применимость к теории нейронных сетей приведено в [9].



В терминах нейронных сетей теорему можно перефразировать следующим образом [6]:

Любое отображение входов нейронной сети в ее выходы может быть реализовано трехслойной нейронной сетью прямого распространения с $d(2d+1)$ нейронов на первом и $2d+1$ на втором слое (рис. 2.1).

Рис. 2.1. Нейронная сеть по теореме Колмогорова: $g(x)$ – функции, зависящие от вида f

К сожалению, практического применения для построения нейронных сетей теорема Колмогорова не имеет [6]. Она только гарантирует существование такой сети, но не определяет алгоритм ее построения. Вдобавок к этому, по условию, функции g обязательно должны быть негладкие и их вид зависит от f. А в нейронных сетях вид пороговой функции обычно фиксируется, оставляя для задачи только подстройку весов. Тем не менее важность этой теоремы трудно переоценить, так как она теоретически обосновывает всю мощь нейронных сетей. Более общий результат, теоретически обосновывающий эффективность нейронных сетей, приведен в [10].

2.2. Алгоритм обратного распространения ошибки

Как показано в п. 1.4, многослойный персептрон в состоянии решать произвольные задачи. Но долгое время его использование было затруднительно из-за отсутствия эффективного алгоритма обучения. При нескольких слоях настраиваемых весов становится непонятным, какие именно веса подстраивать в зависимости от ошибки сети на выходе. Процедура обучения персептрона (п. 1.3) при этом переставала работать.

В 1986 г. Румельхарт, Хинтон и Вильяме предложили так называемый алгоритм обратного распространения ошибки. Обучение многослойного персептрона основано на минимизации функции ошибки сети $E(w)$. Ошибка определяет отклонение желаемых выходов сети t^n от получившихся y^n. Обычно функция $E(w)$ определяется методом наименьших квадратов:

$$E(w) = \frac{1}{2}\sum_n (t^n - y^n)^2 \ . \qquad (2.2)$$

Функция ошибки (2.2) минимизируется методом градиентного спуска, известным из теории численных методов. Суть метода заключается в том, что двигаясь в направлении, противоположном градиенту функции Δw_{ji}, в конце концов приближаемся к минимуму, возможно локальному, функции $E(w)$:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} . \qquad (2.3)$$

Здесь $0 < \eta < 1$ задает скорость обучения; w_{ji} – коэффициент связи i нейрона слоя $n-1$ с j нейроном слоя n. Найдем способ вычисления гради-

ента $\partial E/\partial w$. Запишем формально схему работы многослойного персептрона. Каждый нейрон рассчитывает взвешенную сумму своих входов:

$$a_i = \sum_n w_{ji} z_i \,. \tag{2.4}$$

Здесь z_i – это входы нейрона и соответственно выходы предыдущего слоя нейронов (рис. 2.2, *а*). Выход нейрона j – это преобразование суммы a_i пороговой функцией g:

$$z_j = g(a_j) \,. \tag{2.5}$$

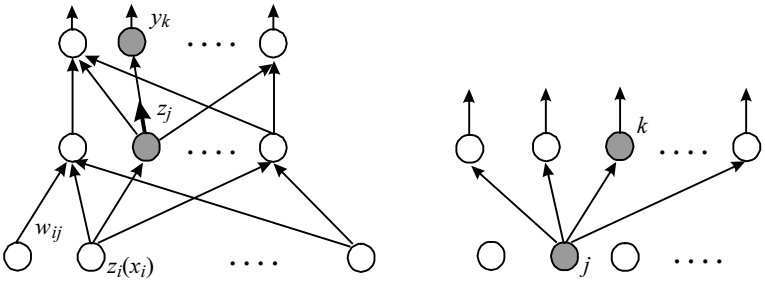


Рис. 2.2. Схема нейронной сети для алгоритма обратного распространения

Вернемся к $\partial E/\partial w_{ji}$ и заметим, что $E(w)$ является сложной функцией, для которой $w = w(a_j)$ является функцией от a_j. Отсюда по правилу вычисления производной сложной функции имеем

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \,.$$

Используя (2.4), получаем

$$\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial \sum w_{ji} z_i}{\partial w_{ji}} = z_i \,.$$

Выражение $\delta_j = \partial E/\partial a_j$ назовем ошибкой на слое j. В итоге можно заметить, что для вычисления градиента на каждом слое i вычисляется произведение величины ошибки предыдущего i, находящегося сверху слоя j, на входное значение слоя:

$$\frac{\partial E}{\partial w_{ji}} = \delta_j z_i \,. \tag{2.6}$$

Найдем теперь значение ошибки δ_j для каждого слоя. Для выходного слоя δ_k можно получить довольно просто:

$$\delta_k = \frac{\partial E}{\partial a_k} = \left| \begin{matrix} a_k = \sum_j w_{jk} z_j \\ y_k = g(a_k) \end{matrix} \right| = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial a_k} \tag{2.7}$$

Заметим, что $\partial y_k / \partial a_k$ есть ни что иное, как производная пороговой функции $\partial y_k / \partial a_k = g'(a)$. Для логического сигмоида (1.6) такая производная будет иметь довольно простой вид: $g'(a) = g(a)(1 - g(a))$. Второй сомножитель $\partial E / \partial y_k$ вычисляется по формуле (2.2) и приводится к простому выражению: $\partial E / \partial y_k = (y_k - t_k)$. В итоге формула вычисления ошибки на выходном слое имеет вид

$$\delta_k = g'(y_k - t_k)\,. \tag{2.8}$$

Для промежуточного слоя δ_j получается аналогичным способом:

$$\delta_j = \frac{\partial E}{\partial a_j} = \left| \begin{matrix} a_k = \sum_j w_{ji} z_j \\ z_j = g(a_j) \end{matrix} \right| = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial a_j}\,. \tag{2.9}$$

Суммирование происходит по всем k, к которым нейрон j посылает сигнал (рис. 2.2, б). Здесь $\partial E / \partial y_k$ есть не что иное, как δ_k, а $\partial z_j / \partial a_j$ – это производная пороговой функции $g'(a_j)$. От множителя $\partial a_k / \partial z_j$ остается просто вес w_{jk}. В итоге получаем

$$\delta_j = g'(a_j) \sum_k \delta_k w_{jk}\,. \tag{2.10}$$

Таким образом, ошибка на каждом слое вычисляется рекурсивно через значения ошибки на предыдущих слоях: коррекция ошибки как бы распространяется обратно по нейронной сети.

Алгоритм обратного распространения ошибки

Ш0. Перед началом работы алгоритма веса w_{ji} инициализируем случайными значениями.

Ш1. Подаем на вход персептрона вектор x^n из обучающей выборки и получаем значение y^n на выходе по формулам (2.5) и (2.4).

Ш2. Вычисляем ошибки δ_k для выходов сети по (2.8).

Ш3. Вычисляем ошибки δ_j для всех скрытых слоев по (2.10).

Ш4. Находим значение градиента по (2.6).

Ш5. Корректируем значения синаптических весов по (2.3).

Ш6. Вычисляем значение ошибки (2.2). Если величина ошибки не устраивает, то идем на Ш1.

Один прогон распространения ошибки принято называть эпохой. Трудоемкость алгоритма обратного распространения ошибки определяется, как величина порядка $O(W)N$. Здесь W – число настраиваемых весов сети; N – размер обучающей выборки.

Функция ошибки $E(w)$, благодаря тому, что в нейронной сети используются нелинейные пороговые функции, представляет собой довольно сложную «овражистую» поверхность с большим числом локальных минимумов. Если при градиентном спуске попасть в такой минимум, то, очевидно, сеть не будет настроена на оптимальную производительность. К счастью, способы оптимизации поиска минимума функций подробно исследованы в теории численных методов, и некоторые из них перенесены на алгоритм обратного распространения.

Усовершенствования алгоритма обратного распространения

1) Простейший способ – это использование переменной скорости обучения η. В начале работы алгоритма ее величина представляет собой большое значение, близкое к 1, по мере сходимости η последовательно уменьшается. Это позволяет быстро подойти к минимуму, а затем точно попасть в него без лишнего «стробирования»;

2) «Овражный» метод. Учитываются тенденции в поверхности добавлением момента инерции μ: $\Delta W_n = -\eta \nabla E + \mu W_{n-1}$. Здесь W_n – матрица значений новых весов; W_{n-1} – матрица значений весов на предыдущей эпохе; $\nabla E \equiv \partial E / \partial W_n$. Идея заключается в скачке через резкие «провалы» в поверхности ошибки – локальные минимумы;

3) Метод сопряженных градиентов. Флетчер и Ривс предложили выбирать направление, сопряженное градиенту, более точно указывающее именно на минимум функции:

$$\Delta W_n = -\eta \nabla E_n + \beta W_{n-1}, \quad \text{при } \beta = \frac{|\nabla E_n|^2}{|\nabla E_{n-1}|^2}; \qquad (2.11)$$

4) Наиболее точное решение – это решение, которое позволяют получить так называемые методы второго порядка. Общий принцип работы основан на использовании матрицы вторых производных – гессиана $H = \nabla^2 E$. Градиент не всегда может указывать в сторону, противоположную минимуму, а величина $-H^{-1}\nabla E$, направление Ньютона, всегда будет направлена в сторону минимума (рис. 2.3).

Отсюда изменение весов происходит по правилу $\Delta W = -\eta H^{-1} \nabla E$.

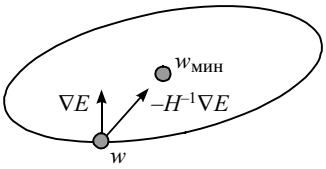


Рис. 2.3. Гессиан

Но сам гессиан довольно сложен для вычисления, поэтому используются методы, где значение H считается приближенно. Лучшим по производительности является метод Левенберга – Маркварта, в котором гессиан аппроксимируется как $H = J^T J + \alpha J$; J – это якобиан, который вычислять намного легче. Процедура коррекции весов приобретает вид $\Delta W = -\eta(J^T J)^{-1}(J^T)$. Трудоемкость данного метода – величина порядка $O(W^2)N$. Метод является очень эффективным и для нейронной сети среднего размера – минимум ошибки, как правило, находится за несколько эпох. Сравнение производительности этих методов обучения приводится в [11].

Многослойный персептрон является на сегодняшний день наиболее распространенным и исследованным типом нейронных сетей. Он широко применяется для задач классификации, построения экспертных сетей. Одной из нерешенных проблем многослойного персептрона, как и большинства классов нейронных сетей, является неопределенность выбора топологии (числа слоев, нейронов в слоях) (см. далее п. 4.3).

Пример использования многослойного персептрона

З а д а ч а: Имеется зашифрованное сообщение. Известен шифр: каждой букве ставилась в соответствие предыдущая по алфавиту буква. Каждая буква находится в области размером 6 × 6 пикселей. Требуется расшифровать сообщение.

Обучающие входные и выходные векторы создадим следующим образом. Если некоторый пиксель буквы закрашен, то соответствующий ему компонент входного или выходного вектора принимает значение 1, в противном случае – 0. Примеры букв представлены на рис. 2.4.

Так как входной и выходной векторы кодируются 36 битами, то сеть имеет 36 входных и 36 выходных нейронов.

Для обучения используем алгоритм обратного распространения с последующей редукцией методом OBD (подробнее в п. 4.4.6). Пусть в скрытом слое имеется 16 нейронов. После обучения сети следует процесс исключения из нее весов и нейронов, оказывающих наименьшее влияние на классификацию образов, для увеличения обобщающей способности сети. В результате обучения в скрытом слое сети остается 8 нейронов, таким образом минимизируется влияние шумов на результат классификации.

Примеры работы обученной сети на некоторых зашумленных символах представлены на рис. 2.5.

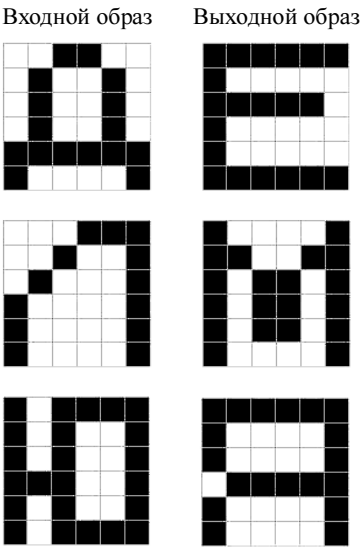


Рис. 2.4. Примеры из обучающего набора

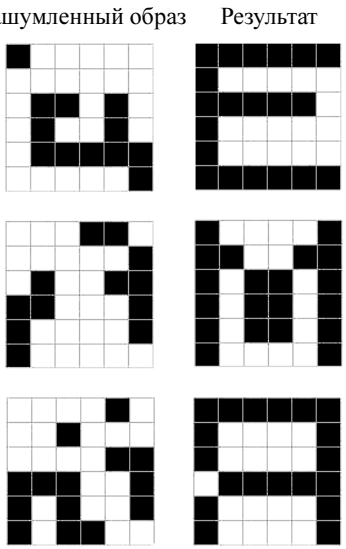


Рис. 2.5. Результаты расшифрования

2.3. Сети радиально-базисных функций

Сети радиально-базисных функций – RBF-сети кардинально отличаются от многослойного персептрона. Многослойный персептрон решает задачу с помощью нейронов, производящих нелинейные преобразования своих взвешенных входов, алгоритм его обучения сложен и трудоемок. В RBF-сети активизация нейронов задается дистанцией между входным вектором и заданным в процессе обучения вектором-прототипом, а обучение происходит быстро и носит элементы как обучения «с учителем», так и «без учителя». RBF-сети берут свое начало от теории точного приближения функций, предложенной Пауэлом в 1987 г.

Пусть задан набор из N входных векторов x^n с соответствующими выходами t^n. Задача точного приближения функций – найти такую функцию h, чтобы $h(x^n) = t^n$, $n = 1,...,N$. Для этого Пауэл предложил

использовать набор базисных функций вида $\Phi(\left|x - x^n\right|)$, тогда получаем

$$h(x) = \sum_n w_n \Phi(\left|x - x^n\right|) = t^n,\ n = 1,...,N\ . \qquad (2.12)$$

Здесь w_n – свободно настраиваемые параметры. Обычно в качестве базисной берут экспоненциальную функцию $\Phi(x) = e^{-x^2/(2\sigma^2)}$, где σ – регулирующий параметр. Такая функция обладает необходимым свойством базисной функции $\lim_{x\to\infty}(\Phi(x)) \to 0$.

Но такое точное приближение дает плохие результаты для зашумленных данных (рис. 2.6, *а*), поэтому в 1988 г. Д. Брумхеад и Д. Лоув предложили модель RBF-сети.

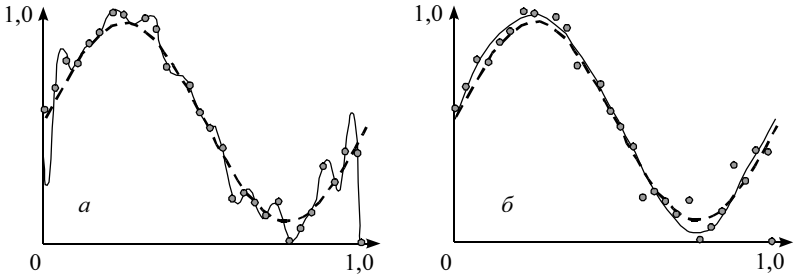


Рис. 2.6. Аппроксимация функции RBF-сетью

Правила задания RBF-сети

1. Число M базисных функций выбирается много меньше числа обучающих данных: $M << N$.

2. Центры базисных функций μ_j не опираются на точки входных данных. Определение центров функций становится частью процесса обучения.

3. Для каждой из M базисных функций задается свой регулирующий параметр σ_j, который также определяется в процессе обучения RBF-сети.

4. В сумму (2.12) добавляется константа w_{k0} – порог нейрона. В итоге RBF-сеть будет описываться формулой

$$y_k(x) = \sum_{j=1}^{M} w_{kj}\Phi_j(x) + w_{k0} = \sum_{j=0}^{M} w_{kj}\Phi_j(x)\,. \qquad (2.13)$$

Здесь базисные функции: $\Phi_j(x) = e^{-\frac{|x-\mu_j|^2}{2\sigma_j^2}}$. Для удобства представления добавлена функция $\Phi_0(x) = 1$.

В 1993 г. Д. Парк и И. Сандберг показали, что RBF-сети, построенные подобным образом, обладают свойством аппроксимации (рис. 2.6, *б*) любой произвольно гладкой функции при условии достаточного количества базисных функций [6]. На рис. 2.7. приведена архитектура RBF-сети. Каждая базисная функция выполняет роль нейрона.

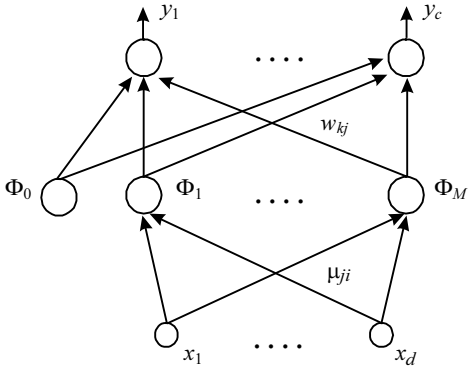


Рис. 2.7. Сеть радиально-базисных функций

Обучение RBF-сети

Имеем обучающий набор: $\{x^n\} \rightarrow \{t^n\}$. Обучение происходит в два этапа:

1) определяются параметры базисных функций: μ_j, σ_j. Причем, как правило, используются только входные векторы $\{x^n\}$, т.е. обучение происходит по схеме «без учителя» (п. 1.7).

Для определения σ_j используется алгоритм «ближайшего соседа». Выбирается количество центров (базисов) M. Алгоритм ищет разбиение множества $\{x^n\}$ на M несмежных подмножеств S_j таким образом, чтобы минимизировать функцию J:

$$J = \sum_{j=1}^{M} \sum_{n \in S_j} \left| x^n - \mu_j \right|^2, \text{ где } \mu_j = \frac{1}{N_j} \sum_{n \in S_j} x^n . \qquad (2.14)$$

Здесь μ_j фактически является средней точкой S_j. Первоначальное раз-

биение производится случайным образом. Вычисляется μ_j. Затем разбиение изменяется, пока существует возможность уменьшить J.

Параметр σ_j – выбирается эвристическим путем. Как правило, σ_j делают чуть большим расстояния между центрами соответствующих базисных функций μ_j;

2) на втором этапе фиксируются базисные функции. RBF-сеть получается равной однослойной нейронной сети. Затем обучение происходит по правилу обучения «с учителем». Как обычно, минимизируется ошибка $E(w) = \frac{1}{2}\sum_n \sum_k (w_{kj}\Phi_j - t_k^n)^2$. Так как E является квадратической функцией от весов w, то минимум E может быть найден решением системы линейных уравнений

$$\frac{\partial E}{\partial w} = \sum_n \left\{ \sum_k (w_{kj}\Phi_j - t_k^n) \right\} \Phi_j = 0 . \qquad (2.15)$$

Решение системы (2.15) находится крайне быстро. Следовательно, алгоритм обучения RBF-сетей является очень эффективным.

В заключение приведем краткую таблицу сравнительных характеристик RBF-сетей и многослойного персептрона:

Многослойный персептрон	RBF-сети
Граница решения представляет собой пересечение гиперплоскостей (п. 1.4)	Граница решения – это пересечение гиперсфер, что задает границу более сложной формы
Сложная топология связей нейронов и слоев	Простая 2-слойная нейронная сеть
Сложный и медленно сходящийся алгоритм обучения (п. 2.2)	Быстрая процедура обучения: решение системы уравнений + кластеризация
Работает на небольшой обучающей выборке	Требуется значительное число обучающих данных для приемлемого результата
Универсальность применения: кластеризация, аппроксимация, управление и проч.	Как правило, только аппроксимация функций и кластеризация

2.4. Обучение без учителя. Правило Хебба

Можно утверждать, что большая часть информации из окружающего мира усваивается головным мозгом по процедуре обучения «без учителя» (п. 1.7), когда явно правильный «ответ» не подсказывается. Основываясь на этом соображении и исследованиях в области структуры

мозга (п. 1.1), еще в 1949 г. Д. Хебб предложил следующую процедуру обучения: увеличивать вес связи между двумя нейронами, если они возбуждаются чаще других нейронов сети. Правило Хебба может применяться для обучения нейронных сетей прямого распространения в задаче классификации. Простейшая реализация правила Хебба называется сигнальным методом:

$$\Delta w_{ij} = \eta z_i^{n-1} z_j^n . \qquad (2.16)$$

Здесь Δw_{ij} – приращение веса, соединяющего выход i-го нейрона предыдущего слоя $n-1$ и нейрона j текущего слоя n; z_i^{n-1} и z_j^n – соответственно выходные значения нейронов i и j; величина $\eta > 0$ задает скорость обучения. Более совершенной вариацией правила Хебба является дифференциальный метод:

$$\Delta w_{ij} = \eta(z_i^{n-1} - z_i^{*n-1})(z_j^n - z_j^{*n}) . \qquad (2.17)$$

Здесь добавляется z_i^{*n-1} и z_j^{*n} – выходные значения нейронов i и j на предыдущей итерации обучения. Таким образом, дифференциальный метод усиливает сильнее всего те веса, которые связывают нейроны, выходы которых увеличиваются наиболее динамично.

Существует третий метод обучения – обучение с забыванием. Введем понятие коэффициента забывания γ – часть синаптического веса, которая «забывается» каждым нейроном на каждой итерации [12]. Тогда обучение выполняется согласно выражению

$$w_{ij}(t+1) = (1-\gamma) \cdot w_{ij}(t) + \eta \cdot z_i^{n-1} z_j^n . \qquad (2.18)$$

Коэффициент γ выбирается из расчета $< 0{,}1$, таким образом при обучении нейрон сохраняет большую часть информации.

Алгоритм обучения нейронной сети по правилу Хебба

В начале работы алгоритма имеем: многослойный персептрон (п. 1.4); набор входных образов $\{x^n\}$, которые необходимо классифицировать (т.е. разбить на классы по подобию).

Ш0. Инициируем веса w_{ij} случайными величинами малой величины.

Ш1. Подаем на входы сети все векторы из $\{x^n\}$ по очереди и получаем выходные значения z для всех нейронов сети.

Ш2. На основании полученных значений z по формулам (2.16) или (2.17), или (2.18) производится изменение весов w_{ij} всей сети.

Ш3. Переходим на Ш1 до тех пор, пока выходные значения сети не стабилизируются с заданной точностью. Так как нейронная сеть – сеть

прямого распространения, то она рано или поздно придет в устойчивое состояние.

Следует заметить, что для обучения по правилу Хебба значения выходов сети для каждого класса входных образов заранее неизвестны. В основном это будет зависеть от случайного распределения весов на Ш0. Обычно соответствие выходов сети классам векторов определяется на этапе тестирования обученной сети.

Иногда все-таки требуется, чтобы выходы сети были четко определены – $\{t^n\}$. Тогда на основе сетей, обучающихся «без учителя», строят сети встречного распространения [13]. Для этого к последнему слою сети, функционирующей «без учителя», добавляют один слой нейронов и по любому алгоритму обучения «с учителем», например критерию персептрона, переводят полученные выходы $\{y^n\}$ для класса входных векторов в желаемые $\{t^n\}$. Подробнее обучение Хебба рассмотрено в [13, 14].

Рассмотрим пример обучения однослойной нейронной сети по модифицированному алгоритму Хебба с забыванием.

Сеть имеет 4 нейрона, размерности входного и выходного вектора равны также 4. Проинициализируем веса сети следующим образом:

$$W = \begin{bmatrix} 0 & 0 & 0 & 0{,}5 \\ 0 & 0{,}5 & 0 & 0 \\ 0 & 0 & 0{,}5 & 0 \\ 0{,}5 & 0 & 0 & 0 \end{bmatrix},$$

где числа в строке являются значениями синаптических весов.

Сеть будет обучаться на следующих векторах:

$$x_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix},\ x_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},\ x_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Обучение производилось при $\eta = 0{,}1$ и $\gamma = 0{,}025$. После двадцати итерации матрица весов приобрела вид

$$W = \begin{bmatrix} 0{,}592 & 0 & 0 & 1{,}343 \\ 0 & 0{,}3 & 0 & 0 \\ 0 & 0 & 0{,}541 & 0 \\ 0{,}752 & 0 & 0 & 0{,}586 \end{bmatrix}.$$

Сеть значительно усилила связи первого и четвертого нейронов с первыми и четвертыми входами.

2.5. Сети Кохонена. SOM

Сети Кохонена – еще один вид сетей, обучающихся «без учителя». В 1987 г. Т. Кохонен предложил нейронную сеть следующего вида (рис. 2.8). Сеть Кохонена состоит из одного слоя настраиваемых весов и функционирует в духе стратегии, согласно которой победитель забирает все, т.е. только один нейрон возбуждается, остальные выходы слоя подавляются, выходы z_j сети при этом:

$$z_j = \sum_i w_{ij} x_i \quad \exists k, z_k = 1, z_{j \neq k} = 0 . \tag{2.19}$$

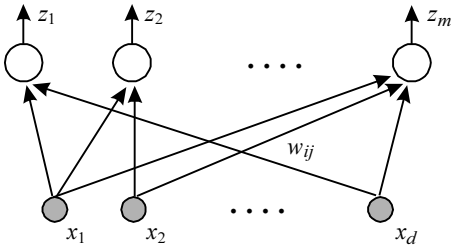


Рис. 2.8. Сеть Кохонена

Сеть Кохонена осуществляет классификацию входных векторов в группы схожих, подстраивая веса таким образом, что входные образы, принадлежащие одному классу, будут активизировать один и тот же выходной нейрон z_j.

Обучение нейронной сети Кохонена

Для работы алгоритма важно, чтобы все входные векторы $\{x^n\}$ были нормализованы, т.е. представляли собой единичный вектор в пространстве:

$$x_i' = x_i \Big/ \sqrt{\sum_i x_i^2} . \tag{2.20}$$

Ш0. Инициируются начальные значения w_{ij} (см. ниже).

Ш1. Вариант 1. На вход подается нормализованный вектор x_i и вычисляется его скалярное произведение с вектором весов каждого нейрона $w_j = (w_{1j}, w_{2j}, ..., w_{dj})$:

$$wx = |x||w|\cos(\hat{x w}) . \tag{2.21}$$

Выбирается вектор весов, дающий максимальное скалярное произведение. Напомним, что cos(0°) = 1, отсюда максимальное значение скалярного произведения будет давать наиболее сходный с входным вектором вектор весов.

Вариант 2. На вход также подается нормализованный вектор x_i. Вычисляется расстояние между векторами w_j и x в пространстве и выбирается вектор весов, расстояние до которого меньше всего:

$$D_j = \sqrt{(x - w_j)^2} \ . \qquad (2.22)$$

Ш2. Проводится аккредитация – настройка весов только выбранного на Ш1 нейрона по правилу

$$w^{t+1} = w^t + \eta(x - w^t), \text{ где } w^{t+1} \text{ – новое значение веса.} \qquad (2.23)$$

Подстройка весов подобным образом сводится к минимизации разницы между входным вектором и векторами весов выбранного нейрона. Существует простая геометрическая интерпретация для двумерного случая (рис. 2.9). Находится вектор $(x - w^t)$ и уменьшается на величину η, задающую скорость обучения $\eta \approx 0{,}7$.

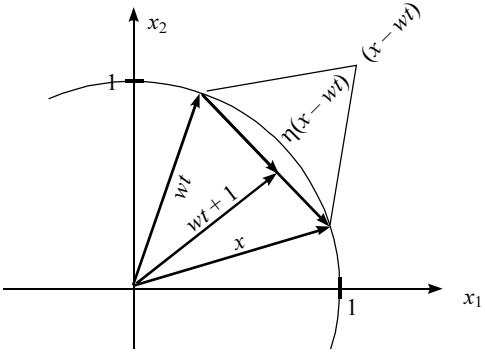


Рис. 2.9. Подстройка весов нейрона

Таким образом, обучение сводится к вращению вектора весов нейрона в направлении входного вектора без существенного изменения его длины.

Ш3. Повторяются шаги Ш1 и Ш2 до тех пор, пока выходные значения сети не стабилизируются с заданной точностью.

Выбор начальных значений на Ш0

Обычно перед началом работы обучающего алгоритма нейронной сети веса w_{ij} инициируются случайными величинами. В случае сети Кохонена получаются векторы весов выходных нейронов, равномерно распределенные по поверхности гиперсферы. Но, как правило, входные векторы x_i распределены неравномерно и группируются кучками на малой части поверхности сферы. Векторы весов большинства нейронов очень удалены от входных векторов и никогда не дадут лучшего соответствия, т.е. не будут участвовать в алгоритме. Оставшихся, попавших в группы входных векторов, будет недостаточно для правильного разделения x_i на классы. Очевидным решением является инициирование весов, при помещении большей их части в скопления входных векторов. Такую возможность дает метод выпуклой комбинации.

Все веса нейронной сети делаются равными $w_{ij} = 1/\sqrt{N}$, где N – размерность входного вектора. Входные векторы модифицируются по правилу $x_i = \alpha x_i + (1-\alpha)/\sqrt{N}$. Начальное значение коэффициента $\alpha \to 0$, поэтому все входные векторы сосредоточены в одной области с весами w_{ij} и имеют вид $x_i \approx 1/\sqrt{N}$. Затем, в процессе обучения α увеличивается до 1, что возвращает входным векторам их истинные значения и тренирует сеть Кохонена на них. Метод несколько замедляет обычную процедуру обучения, но дает лучшие результаты.

На основе сети Кохонена строятся более сложные структуры, называемые самоорганизующиеся сети – SOM [15]. SOM строится из нескольких слоев, построенных по принципу сети Кохонена. Чаще всего SOM выстраиваются в решетчатую структуру. Выбор ближайшего нейрона осуществляется по (2.21). Но аккредитируется, то есть обучается по (2.22), не только он, но и его соседи в окрестности R. Величина R на первых порах работы алгоритма обучения очень большая, но постепенно уменьшается до 0. Результатом работы будет не один выходной нейрон, а группа нейронов, соответствующая каждому классу. Это позволяет осуществлять более точную классификацию входных образов.

Пример использования самоорганизующейся карты признаков

Одно из важнейших свойств обученной сети Кохонена – способность к обобщению. Вектор каждого из нейронов сети заменяет группу соответствующих ему классифицируемых векторов. Это позволяет ис-

пользовать данный вид сети в области сжатия данных. Покажем это на примере компрессии изображения.

Входное изображение высотой N_x пикселей и шириной N_y пикселей разбивается на более мелкие кадры размером $n_x \times n_y$ пикселей. Далее значения компонент красного, синего и зеленого цвета каждого кадра служат обучающим набором для сети Кохонена. После самообучения на вход сети последовательно поступают все кадры исходного изображения. Веса нейрона-победителя определяют значения новых компонент основных трех цветов кадра. На рис. 2.10 представлено изображение 200 на 345 пикселей. Для компрессии использовалась сеть из 150 нейронов.

Рис. 2.10. Исходное изображение

После обучения было получено сжатое изображение (рис. 2.11). Можно заметить, что различия между ними невелики.

Рис. 2.11. Сжатое изображение

Глава 3

РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

3.1. Сети Хопфилда

У сетей прямого распространения: многослойного персептрона и сетей радиально-базисных функций, как следует из названия, обратных связей нет, то есть сигнал всегда распространяется в направлении от входов к выходам сети. В рекуррентных нейронных сетях появляются так называемые обратные связи от выходов нейронов обратно на входы. За счет обратных связей выходы всей сети при одних и тех же входах сети могут принимать произвольные состояния в различные моменты времени и не обязательно стабилизируются на одном [13].

Общая схема работы рекуррентных нейронных сетей (РНС)

Рассмотрим отдельный нейрон с обратной связью. На первом этапе работы (рис. 3.1) на вход нейрона подаются входные значения x_j и вычисляется выход нейрона z. Затем получившееся выходное значение подается на вход нейрона наряду с прочими значениями и вычисляется новое выходное значение. Этот процесс повторяется до тех пор, пока выходное значение нейрона будет мало изменяться от итерации к итерации. Если вернуться к нейронным сетям, то можно ввести следующую классификацию. Рекуррентные нейронные сети, для которых возможно получить стабилизирующиеся к определенному значению выходы, называются устойчивыми, а если выходы сети не стабильны, то неустойчивыми. В общем случае, большая часть рекуррентных нейронных сетей являются неустойчивыми. Неустойчивые сети мало пригодны для практического применения, и поэтому в рамках данной монографии не рассматриваются.

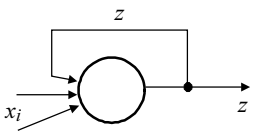


Рис. 3.1. Нейрон с обратной связью

В 1982 г. Д. Хопфилд предложил устойчивую РНС следующего вида (рис. 3.2). Сеть является биполярной, то есть оперирует только величинами {–1, 1}. В сети имеется один слой настраиваемых весов w_{ji}.

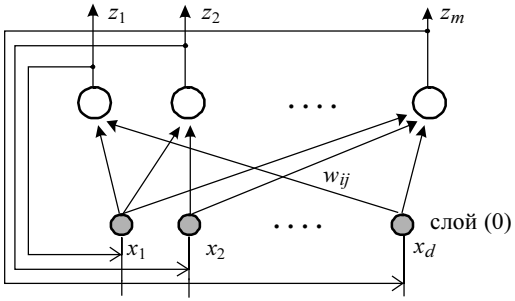


Рис. 3.2. Сеть Хопфилда

Все нейроны единственного слоя возвращают свои выходы на свой вход и входы всех остальных нейронов сети посредством распределителей (не нейронов) слоя (0). Каждый нейрон реализует следующие шаги:

1) вычисляет взвешенную сумму a своих входов:

$$a_j = \sum_{i \neq j}^{M} (w_{ji} z_i) + x_j \; ; \tag{3.1}$$

2) к сумме применяется нелинейная пороговая функция $g(a)$:

$$z_j = g(a) = \left\{ \begin{array}{c} 1, \; a_j > T_j \\ -1, \; a_j < T_j \\ \text{не меняется}, \; a_j = T_j \end{array} \right\}, \; T_j \; - \text{порог нейрона.} \tag{3.2}$$

Сеть Хопфилда функционирует следующим образом. Входные сигналы подаются один раз в начале работы, затем сеть работает только за счет возвращенных выходов. Состояние всех нейронов сети изменяется одновременно. В конце работы сеть стабилизируется в устойчивом состоянии. Так как сеть Хопфилда работает с биполярными числами, то состояние ее выходов всегда можно описать бинарным числом $z_j \in \{-1, 1\}$.

Существует простая геометрическая интерпретация. Для системы из трех выходов состояние сети описывается трехзначным биполярным числом, которое может иметь $2^3 = 8$ состояний. Пусть каждое состояние

сети – это вершина куба в пространстве. Для n выходов соответственно существует 2^n состояний и n-мерный гиперкуб. Когда подается новый входной вектор, сеть начинает переходить из вершины в вершину, пока не стабилизируется в какой-либо вершине.

Устойчивая вершина z для данного входа x задается матрицей настраиваемых весов w_{ij} и порогами T_j. Если входной вектор x частично подвергнуть искажению, то сеть Хопфилда приходит именно в состояние z, то есть можно говорить, что сеть Хопфилда обладает свойством восстановления запомненного состояния исходя из неполной информации.

Устойчивость сети Хопфилда

Сеть Хопфилда гарантированно будет устойчивой при выполнении следующих условий:

1) матрица весов W симметрична $w_{ij} = w_{ji}$;

2) имеет нули на главной диагонали $w_{ii} = 0$, $\forall i$.

Приведем упрощенное доказательство. Рекуррентную нейронную сеть можно рассматривать как динамическую систему, имеющую некоторое энергетическое состояние. Если описать энергию системы функцией E и показать, что E при любом изменении состояния всегда убывает или не изменяется, то система (нейронная сеть) будет по определению устойчивой. Используя условия 1 и 2, можно описать энергетическое состояние системы E функцией Ляпунова:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ji} z_j z_i - \sum_j x_j z_j + \sum_j T_j z_j \ . \qquad (3.3)$$

Здесь w_{ij} – веса, z_j – выходное значение нейрона j, x_j – j-й элемент входного вектора, T_j – порог нейрона j. Изменение энергии E_j, вызванное изменением состояния z_j нейрона j, есть $\Delta E_j = -\left[\sum_{i \neq j} (w_{ji} z_i) + x_j - T_j\right] \Delta z_j$.

Используя формулы (3.1) и (3.2), это выражение можно переписать:

$$\Delta E_j = -(a_j - T_j)\Delta z_j \ . \qquad (3.4)$$

Возможны три варианта:

1) $a_j > T_j$, $\forall j$. Отсюда следует, что выражение в скобках (...) > 0, также подставляя это условие в (3.2), видно, что Δz_j изменится в положительную сторону $\Delta z_j > 0$ либо не изменится. В итоге $\Delta E \leq 0$ убывает или стабилизируется;

2) $a_j < T_j$, $\forall j$. Отсюда следует, что выражение в скобках (...) < 0, также подставляя это условие в (3.2), Δz_j изменится в отрицательную сторону $\Delta z_j \le 0$, либо не изменится. В итоге $\Delta E \le 0$ убывает или стабилизируется;

3) $a_j = T_j$, $\forall j$. Следовательно, $\Delta E = 0$, то есть стабилизируется.

Таким образом, любое изменение состояний сети уменьшает энергию системы, и сеть Хопфида является устойчивой.

3.2. Ассоциативная память. ДАП

Компьютерная память является адресуемой, то есть работает по принципу: сначала предъявляется адрес, а устройство хранения возвращает информацию, содержащуюся по данному адресу. Если сформирован неверный адрес, то будут получены совершенно произвольные данные, которые будут ошибочно интерпретированы как требуемые данные. Человеческая память является ассоциативной: мозг воспринимает какую-то информацию (например имя человека) и в ответ возвращает целую гамму воспоминаний (внешность, место, эмоции и т.п.), то есть, задавая некоторую часть информации, получаем всю остальную. Очевидно, что ассоциативная память имеет очень сложную внутреннюю структуру зависимостей и связей образов. Было показано, что сети Хопфилда могут формировать упрощенную модель ассоциативной памяти [9].

Алгоритм работы ассоциативной памяти

Ш0. Первоначально все запоминаемые образы x_j, $j = 1,..., M$ кодируются биполярными векторами длины N.

Ш1. Затем веса сети Хопфилда настраиваются следующим образом:

$$w_{ij} = \sum_{d=1}^{M} x_d^i x_d^j, \text{ где вектор } x_d = (x_d^1, x_d^2, ..., x_d^N). \qquad (3.5)$$

Фактически вычисляется произведение каждого запоминаемого вектора с самим собой и суммированием матриц, полученных таким образом.

Ш2. После запоминания образов становится возможным восстановление ассоциаций. Входам придают значение образа, возможно частично искаженного, и сети предоставляется возможность колебаться до своего устойчивого состояния. Сеть в конечном итоге стабилизируется

в одном из запомненных состояний. Значения выходов при этом и есть восстановленная ассоциация. Следует заметить, что если входной вектор искажен сильно, то результат может быть неверным.

Особый интерес представляет емкость сети. Теоретически можно запоминать 2^N образов, но реально получается много меньше. Экспериментально было получено, а затем теоретически показано, что в среднем для слабо коррелирующих образов сеть из N нейронов в состоянии запомнить $\approx 0{,}15N$ образов и в идеальном случае – не больше N.

Строго говоря, ассоциативная память, реализуемая сетью Хопфилда, является автоассоциативной: образ, хранящийся в сети, может быть восстановлен по искаженному входу (этому же вектору), но не ассоциирован с произвольным другим образом.

Пример использования сети Хопфилда

З а д а ч а: Существует несколько черно-белых символов-эталонов, размер каждого из которых 6 × 6 пикселей, требуется по введенному зашумленному символу восстановить его до наиболее подходящего эталона. Примеры символов эталонов представлены на рис. 3.3.

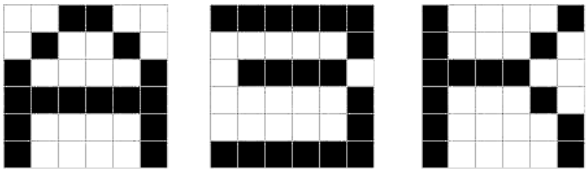


Рис. 3.3. Эталоны символов

Кодирование эталонов происходило следующим образом: каждому из пикселей эталона соответствовала координата вектора, принимающая значение 1, если пиксель имел черный цвет, и значение –1 в противном случае.

По окончании запоминания сетью Хопфилда эталонов на вход сети для проверки функционирования были предъявлены зашумленные образы, по завершении всех вычислений был получен результат, представленный на рис.3.4.

В 1980 – 83 гг. С. Гроссберг опубликовал серию работ по так называемой двунаправленной ассоциативной памяти. В этом случае происходит гетерассоциация двух векторов A и B.

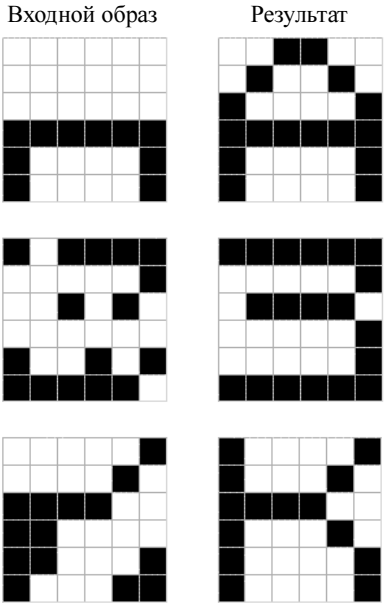


Рис. 3.4. Результаты восстановления

Двунаправленная ассоциативная память (ДАП) фактически представляет собой объединение двух слоев сети Хопфилда (рис. 3.5.). Функционирование нейронов сети ДАП совпадает со схемой работы сети Хопфилда.

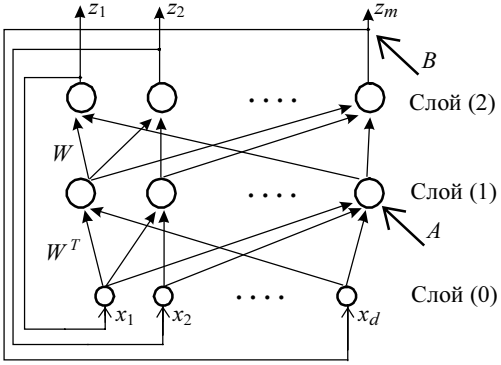


Рис. 3.5. Двунаправленная ассоциативная память

Алгоритм работы ДАП

Ш0. Для удобства будем работать в матрично-векторной форме. Ассоциируемые образы A и B кодируются двоичными векторами длины N.

Ш1. Затем веса сети ДАП задаются по формуле

$$W = \sum_i (A_i^T B_i), \text{ где вектор } A = (A_1, A_2, ..., A_d); \quad (3.6)$$

Ш2. Для восстановления ассоциации B входной вектор A подается на входы слоя (1) и обрабатывается матрицей весов W, в результате на выходе слоя должен формироваться вектор B. В матричном виде это выглядит так:

$$B = g(AW), \text{ где } g() \text{ – пороговая функция (3.2)}; \quad (3.7)$$

Ш3. Входной вектор слоя (2) B обрабатывается матрицей весов W^T, в результате на выходе слоя (2) и формируется вектор A, который затем передается на входы слоя (1):

$$A = g(BW^T), \text{ где } g() \text{ – пороговая функция (3.2)}; \quad (3.8)$$

Ш4. Шаги Ш2 и Ш3 повторяются до тех пор, пока сеть ДАП не стабилизируется, то есть ни вектор A, ни вектор B не изменятся от итерации к итерации. В результате значение вектора B будет искомой ассоциацией вектору A.

Было показано, что сеть ДАП, подобно сети Хопфилда, является устойчивой сетью, т.е. функция энергии Ляпунова либо уменьшается, либо не изменяется.

Емкость памяти ДАП, представленной сетью с N нейронами в меньшем слое, можно оценить следующей величиной:

$$L < \frac{N}{\log_2 N}, \text{ где } L \text{ – емкость память ДАП}. \quad (3.9)$$

К сожалению, емкость ДАП невысока, для $N = 1024$ нейронов сеть будет в состоянии запомнить $L < 25$ образов. По деталям практической реализации ДАП можно также обратиться к [16].

Пример использования ДАП

На рис. 3.6 представлены связанные пары векторов (первый вектор x_i – образ самолета, второй y_i – его название). Размерности x_i равны 120, а векторов y_i – 145. После определения всех параметров сети, производится тестирование – на вход подаются зашумленные образы.

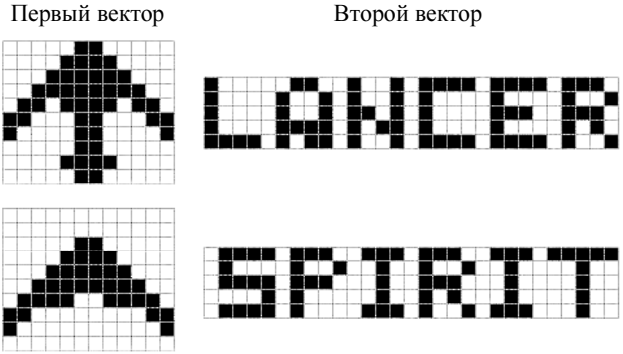


Рис. 3.6. Ассоциированные пары векторов

Результат работы сети представлен на рис. 3.7.

Рис. 3.7. Восстановление эталона и образа, ассоциированного с ним

3.5. Теория адаптивного резонанса

Человеческий мозг постоянно обрабатывает данные из окружающего мира. Именно основываясь на предыдущем опыте и воспринятой информации, мозг принимает решения. При этом постоянно возникает необходимость запоминать и усваивать новые факты. Мозг обладает замечательной способностью запоминать новое так, что ранее запомненное не искажается и не забывается. В теории нейронных сетей подобная способность сети воспринимать новые знания без потери прежних называется дилеммой стабильности-пластичности.

Традиционные нейронные сети не обладают свойством пластичности. Более того, процесс обучения и использования практически всегда разделен. Новый обучающий образ требует переобучения всей сети (рис. 3.8). Фактически, при работе «в реальном времени» сеть будет непрерывно обучаться, то есть бездействовать. Результатом исследования этой проблемы стала теория адаптивного резонанса (Adaptive Resonance Theory – ART), созданная Гроссбергом в 1986 г.

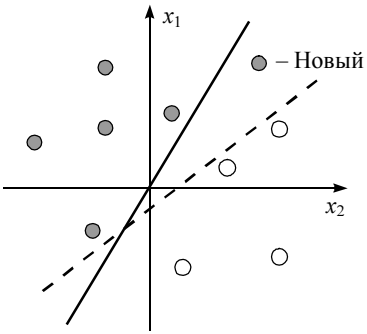


Рис. 3.8. Новый обучающий образ

В основе теории ART лежит феномен фонетического восстановления. Например, если мы слышим шум и обрывок фразы: «... чка на...», то смысл фразы ускользает. Если ее завершить чем-то вроде: «... чка на плоскости», то дополнительная информация порождает фокусировку – ожидание определенного понятия. В результате из сигнала «... чка на плоскости» имеем «точку на плоскости». Говорят, что произошел резонанс образа в памяти с поступившим сигналом. В случае, если резонанса не наступило, то считаем поступившую информацию новыми данными и запоминаем как отдельный образ. Так, фраза «... чка на память», конечно после дополнительного уточнения, может потом восстанавливаться как «фотокарточка на память».

Теория адаптивного резонанса существует в двух парадигмах: ART-1, которая работает с бинарными числами, и более сложной реализации ART-2 для работы с вещественными числами. Мы будем рассматривать простой для понимания вариант ART-1, предложенный Липпманом в 1987 г. [17].

Структура ART-сети

Рассмотрим структуру и схему работы ART-сети (рис. 3.9). Сеть ART состоит из двух основных слоев нейронов: слоя распознавания F_1 и слоя сравнения F_2. Два слоя F_1 и F_2 соединены настраиваемыми весами t_{ji} и b_{ij}. Веса b_{ij}, идущие «снизу вверх» – от выходов нейронов слоя F_1 к входам слоя F_2, называются долговременной памятью. Веса t_{ji}, идущие «сверху вниз» – от выходов слоя F_2 к входам слоя F_1, называются ожидаемыми стимулами. Значение s_i выходов слоя F_1 – это результат обра-

ботки нейронами входов x_i и сигналов «сверху вниз» от F_2 Вместе с ПУ – подсистемой управления – выходы s_i называются кратковременной памятью и, собственно, отвечают за стабильность-пластичность.

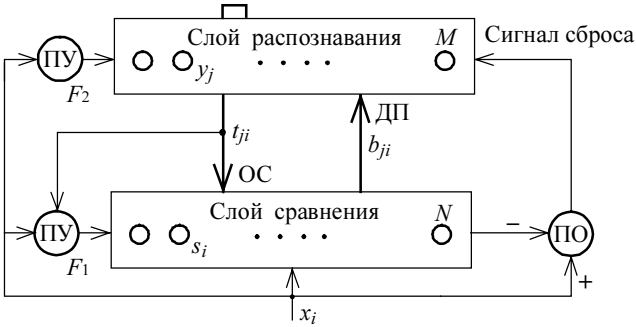


Рис. 3.9. Структура ART-сети

Выходы y_i нейронов слоя F_2 будут представлять собой результат обработки входных сигналов нейронами слоя F_2 и взаимодействия с ПО – подсистемой ориентации. Слой F_2 осуществляет классификацию входных векторов. Слой функционирует по принципу «победитель забирает всё»: только один нейрон, имеющий вектор весов b_{ij}, наиболее соответствующий по признакам входному вектору, возбуждается, а все остальные нейроны затормаживаются (п. 2.5).

Этот эффект достигается за счет специальных тормозящих связей (рис. 3.10). От нейрона с высоким выходным значением на все нейроны слоя подается полученное высокое значение, но со знаком «–», что принудительно затормаживает остальные нейроны. Дополнительно нейрон-победитель усиливает себя положительной обратной связью.

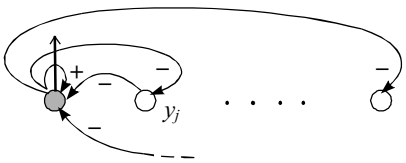


Рис. 3.10. Тормозящие связи

Схема работы ART-сети

Рассмотрим три возможных состояния ART-сети:

1) если входной вектор, поданный на слой F_1, максимально соответствует сигналам «сверху-вниз» (возникает резонанс), т.е. похож на вектор весов b_{ij} одного из нейронов слоя F_2 (нейрона-победителя), то говорят, что входной сигнал соответствует ожиданиям системы и система приходит в устойчивое состояние, определив класс входного вектора. При этом автоматически происходит дообучение сети, с устранением слабых различий запомненного и входного образов. Получается, что в процессе работы ART-сети запомненный образ постоянно уточняется;

2) в случае, если входной образ, поданный на F_1, не соответствует сигналам «сверху-вниз», то ПО подавляет нейрон-победитель, блокирует все его связи и инициирует повторную подачу входного вектора. Подобный поиск среди запомненных образов повторяется до тех пор, пока мы не добьемся эффекта резонанса или не превысим некоторый предел поиска. Если в результате поиска резонанс не был достигнут, то получается третий вариант;

3) входной образ не соответствует ожиданиям системы, и в слое F_2 достаточно нейронов, не задействованных для определения классов входных векторов. В этом случае ПУ заставит нейроны в F_2 настроиться на новый класс входных образов. Процесс настройки проходит по процедуре обучения «без учителя».

Формально алгоритм работы ART-сети выглядит следующим образом:

Алгоритм работы ART-сети

Вводится дополнительный параметр – параметр сходства p, определяющий чувствительность сети к шуму и ответственный за создание новых классов образов.

Ш0. Проводится начальная инициализация весов: $t_{ji} = 1$, $i = 1,..., N$, $j = 1,..., M$; все входные векторы приводятся в равные условия $b_{ij} = L/(L - 1 + N)$. Здесь $L \geq 1$ (обычно $L = 2$), N – число нейронов в слое сравнения F_1, а M – число нейронов в слое распознавания F_2;

Ш1. На вход слоя F_1 подается входной вектор $x = (x_1,..., x_N)$;

Ш2. Вычисляются выходы слоя F_2:

$$y_j = \sum_{i=1}^{N} b_{ij} x_i, \quad j = 1,..., M; \tag{3.10}$$

Ш3. Выбирается лучший по соответствию входному вектору нейрон y_k слоя F_2. Такой, что

$$y_k = \max_j \{y_j\}; \tag{3.11}$$

Ш4. Нейрон-победитель тестируеся на сходство:

$$\mu = \frac{\sum_{i=1}^{N} t_{ik} x_i}{\sum_{i=1}^{N} x_i}. \tag{3.12}$$

Здесь μ дает такую долю единиц во входном векторе, которая имеется и в запомненном в нейроне y_k образе. Если $\mu > p$, сходство считаем достаточным для идентификации и далее на Ш6. Иначе считается, что поступил новый входной образ, переход на Ш5.

Ш5. Временно блокируется лучший по соответствию нейрон y_k, т.е. $y_k = 0$, и переход на Ш3. Причем на данном шаге этот нейрон остается заблокированным (не учитывается);

Ш6. В y_k получен результат работы сети ART – определенный класс входного вектора. Происходит настройка нейрон-победителя. Если выполнен переход с Ш5, то добавляется незадействованный нейрон:

$$t_{ik} = t_{ik} x_i; \tag{3.13}$$

$$b_{ik} = \frac{t_{ik} x_i}{0{,}5 + \sum_{i=1}^{N} t_{ik} x_i}; \tag{3.14}$$

Ш7. Нейрон, запрещенный на Ш5, снова делается активным, и алгоритм снова начинается с Ш1.

Заметим, что выхода из алгоритма нет, т.е процесс обучения неразрывно связан с процессом работы ART-сети.

Липпман показал, что данный алгоритм хорошо работает при незначительном уровне шума. При высокой зашумленности приходится делать параметр сходства p очень малым, иначе два подобных входных образа будут отнесены к разным классам. Существуют усовершенствования алгоритма ART-1, когда p является переменной величиной, но за это приходится платить сложностью реализации и увеличением времени выполнения.

Пример обучения ART-сети

На этапе 1 подается символ «С». Так как для него отсутствуют запомненные образы, то выделяется нейрон в слое F_2, веса t_{ji} которого подстраиваются соответствующим образом (рис. 3.11). Аналогично для «В» и «Е»: так как эти символы не вступают в резонанс с уже запомненной «С», то под них выделяются новые нейроны (этапы 2 и 3). При подаче символа «Е», но другого шрифта, он определяется как уже существующий в системе. Причем происходит корректировка запомненного образа (этап 4). Если подается искаженный символ «Е», для которого $\eta < p$, параметр сходства недостаточно мал, и для искаженного символа выделится новый нейрон (этап 5). Отсюда видно, что правильный выбор параметра сходства p очень важен. К сожалению, теоретического обоснования для выбора p не существует.

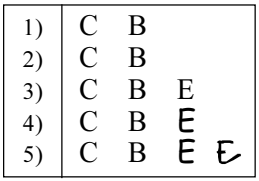


Рис. 3.11. Пример обучения ART-сети

Сеть ART является представителем рекуррентных нейронных сетей, которые являются в общем случае неустойчивыми. Для ART-сети было доказано несколько теорем, показывающих, что она будет устойчивой.

Теорема: процесс обучения ART-сети конечен и устойчив. Любая последовательность произвольных входных векторов будет производить стабильный набор весов после конечного количества обучающих серий. Повторяющиеся последовательности входных векторов не будут приводить к циклическим изменениям.

ART-сеть не лишена недостатков. Она очень чувствительна к шуму. Также имеет место связность всех запомненных образов в t_{ji} и b_{ij}: при потере одного фактически будут потеряны все остальные. Головной мозг, напротив, в состоянии что-то помнить и при частичной потере памяти. Сеть ART имеет ограниченную числом нейронов емкость памяти.

Глава 4

ЭФФЕКТИВНЫЕ НЕЙРОННЫЕ СЕТИ

Наличие разных типов нейронных сетей, алгоритмов их обучения для различных типов задач не гарантирует автоматически, что созданная нейронная сеть будет решать эффективно ту или иную проблему. Рассмотрим несколько методик, помогающих в создании и тренировке эффективных нейронных сетей.

4.1. Обработка данных

При использовании нейронных сетей на реальных данных предварительная обработка бывает полезной практически всегда. В общем виде процесс обработки состоит в этапе предобработки данных перед их подачей нейронной сети.

Предобработка предполагает приведение данных в удобный для обработки нейронной сетью вид. В некоторых случаях требуется этап последующей обработки выходов нейронной сети для восстановления первоначального вида данных (рис. 4.1). Самой простой и эффективной формой предварительной обработки является снижение размерности входных данных. Нейронная сеть с меньшим числом входов требует более простую топологию и соответственно быстрее обучается и работает. Очевидно, что уменьшение размерности ведет к потере части информации, поэтому очень важно выбрать стратегию предобработки так, чтобы оставить как можно больше важной информации и отбросить только малосущественную. Подобный подход реализуется мощной методикой снижения размерности, носящей название анализ основных компонент.

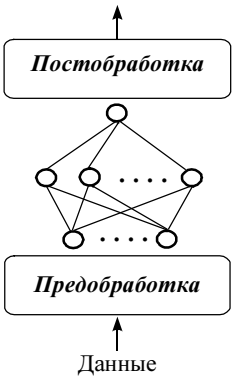


Рис. 4.1. Обработка данных

Анализ основных компонент

Рассмотрим линейное преобразование вектора $x = (x_1,..., x_d)$ из обучающей выборки $\{x^n\}$ в новый вектор $z = (z_1,..., z_M)$ в M-мерном пространстве, причем $M < d$. Вектор x может быть представлен как линейная комбинация ортонормальных векторов u_i:

$$x = \sum_{i=1}^{d} z_i u_i \ . \tag{4.1}$$

Оставим только M базисных векторов u_i и соответственно M коэффициентов z_i. Остальную часть коэффициентов мы заменим константами b_i:

$$\tilde{x} = \sum_{i=1}^{M} z_i u_i + \sum_{i=M+1}^{d} b_i u_i \ . \tag{4.2}$$

Рассмотрим весь обучающий набор $\{x^n\}$, где $n = 1,..., N$. Необходимо выбрать базисные векторы u_i и коэффициенты b_i так, чтобы приближение (4.2) давало лучший результат для всего набора данных. Ошибка при этом будет выглядеть следующим образом:

$$E = \frac{1}{2}\sum_{n=1}^{N}(x^n - \tilde{x}^n)^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{i=M+1}^{d}(z_i^n - b_i)^2 \ . \tag{4.3}$$

Можно показать, что минимум ошибки E в отношении базисных векторов u_i будет найден при минимизации величины

$$E = \frac{1}{2}\sum_{i=M+1}^{d}\lambda_i \ . \tag{4.4}$$

Здесь величины λ_i являются собственными значениями базисных векторов u_i, которые соответственно будут собственными векторами. Таким образом, минимум ошибки достигается отбрасыванием $d - M$ самых малых собственных значений и соответствующих им базисов u_i, т.е. собственных векторов.

Практически метод анализа основных компонент реализуется следующим образом. Вычисляется матрица ковариаций обучающей выборки $K = \sum_n (x^{(n)} - \overline{x})(x^{(n)} - \overline{x})^T$ – это среднее значение по всей обучающей выборке. Затем находятся собственные векторы и собственные значения матрицы K. Согласно (4.4) оставляются собственные векторы,

соответствующие наибольшим собственным значениям K. После этого проецируются векторы из $\{x^n\}$ на оставленные собственные векторы u_i.

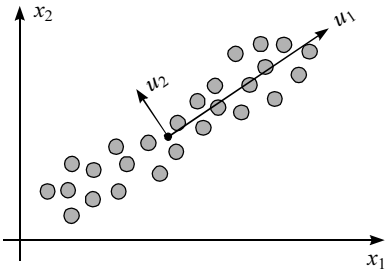


Рис. 4.2. Пример метода анализа основных компонент

В результате получен трансформированный вектор z^n меньшей размерности в пространстве размерности M, минимально отличающийся от x^n. В численном виде ошибку преобразования E можно найти по (4.4). На рис. 4.2 каждый из двумерных векторов x должен быть трансформирован в одномерный z проекцией на собственный вектор u_i.

Нормализация входных данных

Одной из самых распространенных форм предварительной обработки входов нейронной сети является нормализация данных. Очень часто бывает, что параметрами обучающего вектора могут быть семантически различные значения, например, x – возраст, вес, температура, давление пациента. В зависимости от шкалы измерения численные значения этих величин могут различаться в несколько раз. Причем большие значения ни коим образом не являются более значимыми относительно малых.

Применяя простое линейное преобразование, можно выровнять относительные значения для всех компонент входного вектора. Для этого для каждой компоненты входного вектора независимо рассчитываются среднее значение $\overline{x_i}$ и дисперсия $\overline{\sigma}_i^{\,2}$:

$$\overline{x_i} = \frac{1}{N}\sum_{i=1}^{N} x_i^n ; \tag{4.5}$$

$$\overline{\sigma}_i^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i^n - \overline{x_i})^2 . \tag{4.6}$$

Здесь $x^n = (x_1^n, ..., x_N^n)$ – вектор из обучающей выборки $\{x^n\}$. Используя (4.5) и (4.6), получим новый входной вектор $\tilde{x}^n = (\tilde{x}_1^n, ..., \tilde{x}_N^n)$, где

$$\tilde{x}_i^n = \frac{x_i^n - \overline{x}_i}{\sigma_i}. \qquad (4.7)$$

Таким образом, новое преобразованное значение $\tilde{x}^n$ будет иметь нулевое среднее значение и единичную дисперсию.

4.2. Оптимизация процесса обучения

Одной из часто используемых методик улучшения результатов обучения нейронной сети является добавление к определению функции ошибки сети так называемого регулирующего параметра:

$$\tilde{E} = E + \gamma\Omega. \qquad (4.8)$$

Здесь E – среднеквадратическое отклонение, а Ω – регулирующий параметр, γ – коэффициент, определяющий степень его влияния на ошибку $\tilde{E}$. Существует много способов задания регулирующего параметра, рассмотрим один из самых простых – метод понижения весов. В 1987 г. Г. Хинтон показал, что регулирующий параметр следующего вида существенно может улучшить работу сети [18]:

$$\Omega = \frac{1}{2}\sum w_{ij}^2. \qquad (4.9)$$

Некоторое улучшение результата работы нейронной сети можно объяснить следующим образом. Для получения аппроксимации функции с большим «стробированием» восстановленной функции требуются большие значения весов. Параметр вида (4.9) поощряет веса сети принимать небольшие значения, и аппроксимация функции будет более гладкая.

Ранняя остановка обучения

Другим эффективным методом, улучшающим процедуру обучения, является так называемая ранняя остановка обучения [19].

Метод позволяет улучшить результаты работы нейронной сети за счет устранения эффекта ее переобучения. Этот эффект заключается в том, что на реальных задачах с присутствием шума в обучающей вы-

борке нейронная сеть начинает настраиваться не только на определение необходимых закономерностей в данных, но и на шум и искажения, что сильно ухудшает результат ее работы.

Суть метода заключается в том, что вся обучающая выборка $\{x^n\}$ разбивается на три набора: $\{x^{\text{train}}\}$ – обучающий, $\{x^{\text{valid}}\}$ – проверочный и $\{x^{\text{test}}\}$ – тестовый. При обучении нейронной сети для настройки весов используется обучающий набор $\{x^{\text{train}}\}$. Ошибка обучения сети вычисляется по набору $\{x^{\text{valid}}\}$. Получается, что проверка эффективности сети производится на независимых данных. В момент обучения, когда сеть начинает настраиваться на шум, присутствующий только в выборке $\{x^{\text{train}}\}$, ошибка, вычисляемая по совершенно другим данным, начинает расти (рис. 4.3). Именно в этот момент t_0, когда сеть работает наилучшим образом и имеет смысл прекратить процесс обучения сети. Необязательный набор данных $\{x^{\text{test}}\}$ служит для окончательной проверки эффективности ее работы.

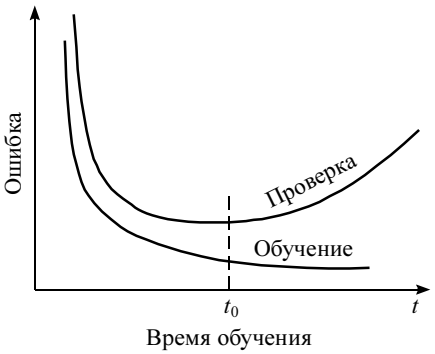


Рис. 4.3.: Изменение ошибки

Комитет сетей

Обычной практикой при решении задачи нейронными сетями является построение и обучение нескольких различных нейронных сетей, из которых затем выбирается для использования лучшая по производительности сеть. У такого подхода есть очевидные недостатки, а именно: все усилия, направленные на построение многих сетей, пропадают впустую. Второй, более серьезный недостаток – это отсутствие гарантий, что сеть, наилучшая по производительности на одних тестовых данных, будет также хороша на других.

Как решение поставленной проблемы М. Перрон и Л. Купер в 1993 г. предложили использовать комитет сетей [20]. Можно формально показать, что при использовании нескольких нейронных сетей ошибка существенно уменьшается.

Предположим, имеется набор из L произвольных нейронных сетей. Задача – построить аппроксимацию функции $y(x)$, тогда каждая сеть i должна будет реализовывать функцию $h(x)$, такую, что $y_i(x) = h(x) + \varepsilon_i(x)$. Среднеквадратическую ошибку отдельной сети можно записать как $E_i = e((y_i(x) - h_i(x))^2) = e((\varepsilon_i(x))^2)$, где $e[\cdot]$ – это среднее значение $e((\varepsilon_i(x))^2) = \int(\varepsilon_i(x))^2 p(x)dx$. Отсюда средняя ошибка по всем сетям в отдельности равна

$$E_{\text{ср}} = \frac{1}{L}\sum_{i=1}^{L} e((\varepsilon_i(x))^2). \tag{4.10}$$

Простейшей формой реализации комитета сетей будет усреднение значений выходов всех L сетей:

$$y = \frac{1}{L}\sum_{i=1}^{L} y_i(x). \tag{4.11}$$

Отсюда ошибка комитета сетей будет равна

$$E = e[(\frac{1}{L}\sum_{i=1}^{L} y_i(x) - h(x))^2] = e[(\frac{1}{L}\sum_{i=1}^{L} \varepsilon_i)^2]. \tag{4.12}$$

Если предположить, что ошибки ε_i имеют нулевое среднее $e(\varepsilon_i) = 0$ и не коррелируют $e(\varepsilon_i\varepsilon_j) = 0$, то ошибку комитета сетей можно переписать как

$$E = \frac{1}{L^2}\sum_{i=1}^{L} e(\varepsilon_i^2) = \frac{1}{L}E_{\text{ср}}. \tag{4.13}$$

Следовательно, ошибка комитета сетей в идеальном случае будет в L раз меньше ошибки отдельных нейронных сетей. Но в действительности предположения о некорррреляции ε_i и $e(\varepsilon_i) = 0$ не выполняются, поэтому получаем просто $E < E_{\text{ср}}$.

В этой главе рассматриваются известные методы и подходы, используемые для построения нейронных сетей. В следующем разделе дается краткий обзор теоретических оценок эффективности их работы. В разделе, посвященном конструктивным алгоритмам, приведен довольно полный, на наш взгляд, ряд алгоритмов построения нейронных сетей. В заключение главы детально будут рассмотрены методики эволюционного подхода к их построению.

4.3. Критерии эффективности нейронных сетей

Наличие очевидной взаимной зависимости между сложностью модели, размерами обучающей выборки и результирующей обобщающей способностью модели на независимых данных предполагает возможность определения этой зависимости тем или иным способом [21]. Иными словами, для нахождения метода построения эффективных нейронных сетей нам необходимо определиться с возможными способами оценки обобщающей способности сети $\hat{P}$.

Существующие оценки можно условно разделить на валидационные и алгебраические. Валидационные оценки основаны на дополнительном анализе данных. Алгебраический способ построения основан на выполнении некоторых предположений о распределении данных.

Валидационные оценки

Простейшим случаем валидационной оценки является разбиение всей доступной для обучения выборки данных на обучающую и проверочную выборки. Обучающая выборка используется в процедуре обучения сети, а проверочная – только для вычисления среднеквадратической ошибки модели. Тем не менее проверочная выборка является частью процедуры настройки сети, поскольку она через значение ошибки будет определять выбор сети. Данный подход эффективен в случае, если имеется большое количество примеров, достаточное для образования двух наборов.

Выбор примеров для проверочного набора всегда будет существенен. Метод кросс-валидации (КВ) предполагает усреднение эффекта выбора определенного проверочного набора путем построения и попеременного использования на всей доступной выборке данных нескольких поднаборов [19]. Весь набор разбивается на Q поднаборов S_j размера V каждый, так что $N = QV$. Обучается Q различных вариантов сети, реализующих оценки $\hat{f}_j(x, D)$ на всей выборке, исключая поочередно S_j. Таким образом, оценка обобщающей способности сети примет вид

$$\hat{P}(x)_{CV} = \frac{1}{N}\sum_{j=1}^{V}\sum_{k\in S_j}^{N}(\hat{f}_j(x_k, D) - y_k)^2 . \qquad (4.14)$$

Наиболее известен вариант КВ без одного (leave-one-out) с $V = 1$:

$$\hat{P}(x)_{\mathrm{LOO}} = \frac{1}{N}\sum_{k=1}^{N}(\hat{f}_j(x_k, D) - y_k)^2 . \qquad (4.15)$$

Использование (4.15) предполагает обучение каждой из N дополнительных нейронных сетей на $N-1$ примерах. Фактически это означает расширение пространства поиска параметров сети в пределах выбранного класса топологий сети $A_i \subset A$.

Алгебраические оценки

В статистике известна зависимость между ожидаемой ошибкой на обучающей и тестовой выборках для линейных моделей [22] при среднеквадратическом определении ошибки:

$$E_{\mathrm{test}} = E_{\mathrm{train}} + 2\sigma^2 \frac{P}{N} . \qquad (4.16)$$

Здесь P – число свободных параметров модели. В случае нейронных сетей число свободных параметров определяется архитектурой (числом слоев L, количеством нейронов сети γ и набором весов связей между нейронами сети W. Предполагается, что данные выборки D зашумлены независимым стационарным шумом с нулевым средним и дисперсией σ^2.

Как было показано Барроном в [22], (4.16) может быть применено к случаю нелинейных моделей при выполнении нескольких условий. Модель $\hat{f}_{\tilde{\theta}}(x, D)$ может рассматриваться как локально линейная в окрестности $\tilde{\theta}$. Это предположение игнорирует гессиан и форму поверхности модели в пространстве параметров высшего порядка. Модель является полной, т.е. существует такой набор параметров $\tilde{\theta}$, что $\hat{f}_{\tilde{\theta}}(x, D) = f(x)$.

На основе этого предположения строится ряд асимптотических оценок обобщающей способности сети. Одним из первых был предложен информационный критерий Акаика (AIC) [23]. Похожим критерием является предложенный Шварцем байесовский информационный критерий (BIC) [24]:

$$\hat{P}(x)_{AIC} = E_{\mathrm{train}} + 2\hat{\sigma}^2 \frac{P}{N};$$

$$\hat{P}(x)_{\mathrm{BIC}} = E_{\mathrm{train}} + 2\hat{\sigma}^2 \frac{P\ln(N)}{N} . \qquad (4.17)$$

Здесь $\hat{\sigma}^2$ – некоторая оценка дисперсии шума для данных выборки.

Акаик предложил использовать оценку $\hat{\sigma}^2 = \frac{N}{N-P} E_{\text{train}}$. Результатом будет финальная ошибка предсказания (final prediction error, FPE). Еще одним ее аналогом является обобщенная кросс-валидация (generalized cross-validation, GCV). Хотя оценки GCV и FPE будут слегка различаться для малых выборок, они являются асимптотически эквивалентными для больших *N*:

$$\hat{P}(x)_{\text{FPE}} = E_{\text{train}} \frac{N+P}{N-P};$$

$$\hat{P}(x)_{\text{GCV}} = E_{\text{train}} \frac{1}{(1-\frac{P}{N})^2}. \qquad (4.18)$$

Ключевым моментом будет являться то, что в алгебраических оценках учитывается сложность модели через число свободных параметров *P*. Очевидно, что в ряде случаев могут задействоваться не все доступные параметры модели [25], поэтому рядом авторов было предложено оценивать эффективное число параметров модели. В серии работ [26 – 28] Муди предложил критерий, названный обобщенной прогнозируемой ошибкой (generalized prediction error, GPE):

$$\hat{P}(x)_{\text{GPE}} = E_{\text{train}} + \frac{2}{N} P_{\varepsilon}, \; P_{\varepsilon} = tr(H_S H_C^{-1}). \qquad (4.19)$$

Здесь H_S и H_C – гессианы среднеквадратической ошибки и регулирующего слагаемого соответственно, а P_ε – эффективное число параметров модели. В случае широко используемого регулирующего слагаемого понижения весов число эффективных параметров P_ε примет вид

$$P_{\varepsilon} \approx \sum_{k=1}^{P} \frac{\lambda_k}{\lambda_k + \xi_k},$$

где λ_k – это собственные вектора H_S, *а* ξ – коэффициент.

В работах [21, 29] предложена аналогичная GPE оценка – сетевой информационный критерий (network information criterion, NIC).

Сравнивая два различных подхода к оценке обобщающей способности сети, можно сделать несколько общих выводов. Оценки, основанные на валидации, являются очень затратными с вычислительной точки

зрения. Асимптотический характер алгебраических оценок предполагает наличие достаточно большого набора данных, но дает более точный результат.

Использование приведенных оценок для выбора той или иной нейронной сети затруднено по причине отсутствия механизмов перебора моделей, однако для GPE в [24] были сделаны такие попытки.

Важно отметить, что Мурата в работе [19] показал, что критерии, подобные AIC, NIC и GPE, эффективнее всего применять для сравнения моделей иерархически вложенных друг в друга, т.е. когда одна нейронная сеть вложена в другую как подсеть меньшей размерности.

4.4. Конструктивный подход к построению нейронных сетей

Конструктивные методы предполагают последовательное построение (редукцию) нейронной сети путем поэтапного добавления (удаления) элементов архитектуры сети (связей, нейронов, скрытых слоев). Алгоритм конструирования функционирует по определенному правилу таким образом, что любое изменение архитектуры на каждом этапе гарантированно уменьшает значение ошибки сети. Правило подчиняется следующей схеме: локализация неверно решенных примеров и коррекция архитектуры сети только для них, не затрагивая, по возможности, корректных.

4.4.1. Бинарные алгоритмы

Первые алгоритмы автоматического конструирования нейронных сетей были связаны с построением нейронных сетей, реализующих произвольные логические функции. Эти сети были несколько упрощены: входы и выходы таких нейронных сетей могли принимать только значения $\{0, 1\}$.

Алгоритм черепичного построения

Исторически одним из самых первых подобных алгоритмов был алгоритм черепичного построения, предложенный в 1989 г. в работе [30]. Черепичный алгоритм строит нейронную сеть слой за слоем. Каждый последующий от входов слой имеет меньшее число нейронов. Построенная сеть может осуществлять некоторое бинарное отображение N_{inp}

входов в единственный выход. Обучающее множество состоит из N_P бинарных векторов, так что $N_P \le 2^{N_{\text{inp}}}$.

Идея алгоритма заключена в добавлении новых слоев нейронов таким образом, чтобы входные обучающие вектора, имеющие различные соответствующие им выходные значения, имели в нем различное внутреннее представление. Внутренним представлением входного вектора X_P на слое L будут являться все выходные значения нейронов этого слоя. Для входного слоя ($L = 0$) для каждого примера внутренним представлением будет он сам. Число классов для слоя L будет подчиняться соотношениям $N_P^L \le N_P$, $N_P^0 = N_P$. Более строгим определением классов будут совпадающие классы. Класс для слоя L будет совпадающим, если все входные вектора, попадающие в этот класс, будут иметь одинаковое желаемое выходное значение.

Работа алгоритма будет заключаться в добавлении нового слоя и проверке, что все классы для него являются совпадающими. Если это удается не для всех классов, то для корректно отображаемых классов веса замораживаются, а для некорректных строится новый слой нейронов.

Новый слой строится из одного основного нейрона – Ⓜ и нескольких дополнительных – Ⓐ (рис. 4.4). Всякий раз основной нейрон будет создаваться так, чтобы число входных обучающих векторов $N_\varepsilon{}^L$, для которых неверно выполняется отображение, уменьшалось. В случае $N_\varepsilon{}^{L^*} = 0$, L^* – последний слой, а значение основного нейрона и есть желаемое выходное значение. Дополнительные нейроны обеспечивают условие совпадающих классов: если условие не выполнено, то это означает, что существуют входные векторы из разных классов, дающие равные выходные значения. Все такие векторы объединяются под дополнительным нейроном, фактически образуя новый класс.

Веса как основного, так и дополнительных нейронов настраиваются модификацией обычного алгоритма обучения персептрона – «карманным» алгоритмом. Алгоритм обучения персептрона находит решение только в случае линейной разделимости данных, в противном случае алгоритм не сходится [4], «карманный» алгоритм всегда останавливается на варианте весов, наиболее близком к решению.

Процедура работы черепичного алгоритма имеет простую геометрическую интерпретацию (рис. 4.5). Основной нейрон (сплошная линия) наилучшим образом разделяет два класса, а дополнительные нейроны

(пунктирная линия) только создают корректирующие линейные разделители для неверно классифицированных входных векторов. Этот процесс повторяется на каждом слое, что позволяет создать нейронную сеть вида рис. 4.4.

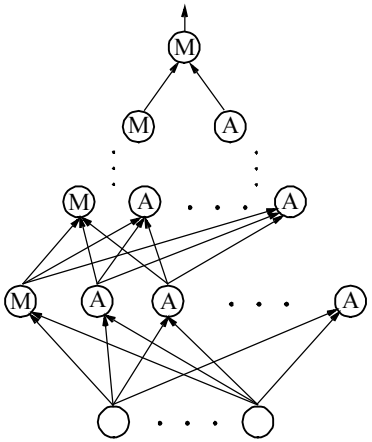


Рис. 4.4. Вид сети, построенной черепичным алгоритмом

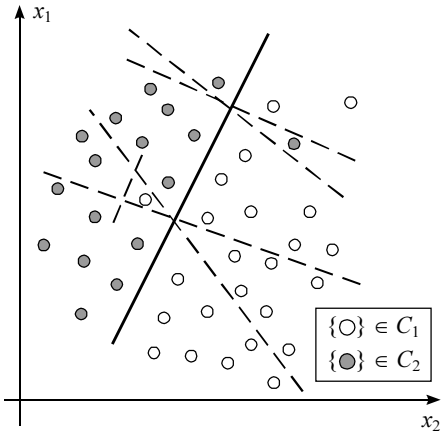


Рис. 4.5. Геометрическая интерпретация черепичного алгоритма

Сходимость черепичного алгоритма доказывается следующей т е о р е м о й: Пусть все классы слоя $(L-1)$ – совпадающие, а число ошибок основного нейрона $N_\varepsilon^{L-1} > 0$. Тогда всегда существует набор весов w, соединяющих слой $(L-1)$ с основным нейроном нового слоя L, причем так, что $N_\varepsilon^{L} < N_\varepsilon^{L-1} - 1$. Более того, такой набор весов всегда единственен. Доказательство теоремы будет являться способом построения такого набора весов и подробно приведено в работе [31].

Башенный алгоритм

Вариантом черепичного алгоритма будет являться так называемый. башенный алгоритм, предложенный в 1990 г. Галлантом. Отличием башенного алгоритма будет то, что здесь для организации совпадающих классов не используются дополнительные нейроны, а сразу строятся соединения с входными нейронами. На каждом новом слое L создается основной нейрон и соединяется со всеми входами сети. В таком случае внутренним представлением входных векторов будет являться сам

входной вектор вместе со значением основного нейрона слоя ($L - 1$). В случае N_P входных векторов всегда будет иметь место N_P совпадающих классов с одним элементом в каждом. Обучение основных нейронов происходит, как и для черепичного алгоритма, «карманным» методом. В случае нескольких выходных значений для каждого выхода строится отдельная нейронная сеть с объединением впоследствии результата.

Алгоритм быстрой надстройки

Алгоритм быстрой надстройки [32] использует несколько иную стратегию для построения нейронной сети. Сеть не наращивается постепенно от входов до момента схождения. Наоборот, новые нейроны добавляются между выходным и выходным слоями. Роль этих нейронов заключается в корректировании ошибок выходного нейрона. В общем случае нейронная сеть, построенная алгоритмом быстрой надстройки, имеет форму бинарного дерева.

Первоначально имеется сеть с P входами и одним выходным нейроном. Как и в черепичном алгоритме, сеть тренируется «карманным» алгоритмом. В результате обучения для каждого примера $X_i = \{x_1,..., x_P\}$ из обучающей выборки возможно четыре состояния выхода сети:

- верно активный: Выход сети o_P равен желаемому для X_j значению T_i и равен 1: $o_P = T_i = 1$, P_1 – соответствующий набор входных векторов;
- верно неактивный: $o_P = T_i = 0$, набор P_2;
- неверно активный: $o_P = 1$, $T_j = 0$, набор P_3;
- неверно неактивный: $o_P = 0$, $T_j = 1$, набор P_4.

Идея алгоритма заключается в добавлении двух нейронов-потомков к текущему нейрону (выходному в начале работы). Первый потомок предназначен для коррекции значений выходов, которые были неверно активны – P_3, а второй соответственно для неверно неактивных – P_4 (рис. 4.6).

Для набора P_3 ошибочные значения можно устранить, если первый потомок активен только для примеров:

$$o_1(p) = \begin{cases} 1, p \in P_3 \\ 0, p \in \{P_1, P_2, P_4\} \end{cases} \quad (4.20)$$

Здесь $o_1(p)$ выходное значение первого потомка для примера p. Для этого первый потомок соединяется с выходной связью с отрицательным значением, абсолютное значение которого больше, чем максимальный размер ошибки для набора P_3. Следует заметить, что для набора P_2

(верно неактивный) влияние первого потомка будет только повторение правильного результата. Это значит, что набор P_i можно убрать из обучающего набора (4.20) для первого потомка:

$$o_1(p) = \begin{Bmatrix} 1, p \in P_3 \\ 0, p \in \{P_1, P_4\} \end{Bmatrix}. \quad (4.21)$$

Аналогично для второго потомка который соединяется с выходной связью уже с положительным значением, имеем обучающий набор

$$o_2(p) = \begin{Bmatrix} 1, p \in P_4 \\ 0, p \in \{P_2, P_3\} \end{Bmatrix}. \quad (4.22)$$

Отображения, заданные в (4.21) и (4.22), потенциально не будут линейно разделимыми, поэтому нейроны-потомки обучаются при помощи «карманного» алгоритма. Это подразумевает, в свою очередь, создание рекурсивно корректирующих нейронов для каждого из потомков. Результатом будет нейронная сеть на рис. 4.7.

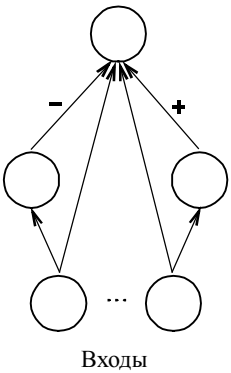


Рис. 4.6. Добавление нейронов-потомков к текущему нейрону

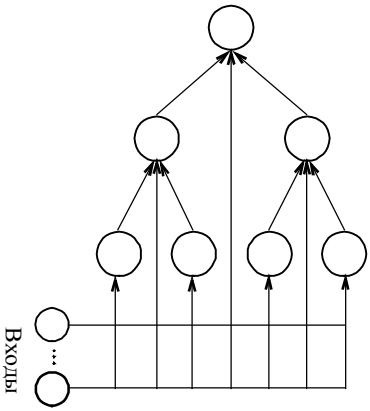


Рис. 4.7. Нейронная сеть в виде бинарного дерева, построенная алгоритмом быстрой надстройки

Сходимость алгоритма гарантирует т е о р е м а, которая гласит: потомки будут всегда производить меньше ошибок, чем родитель. Доказательство приведено в [32].

Применение принципа построения нейронной сети, использованного в бинарных алгоритмах, возможно и для вещественных данных. В работе [33] используется следующий оригинальный прием: данные выборки по определенным правилам преобразуются в бинарные значения, по ним конструируется нейронная сеть, схематично решающая проблему, которая затем окончательно обучается по исходным данным.

4.4.2. Древовидные нейронные сети

В начале 1990-х гг. независимо друг от друга двумя коллективами авторов были предложены схожие алгоритмы для генерации и обучения нейронных сетей специального вида. В 1991 г. в работе [34] были предложены древовидные нейронные сети или NT-сети, которые представляли собой древоподобную структуру с персептронами в узлах. В 1992 г. в серии работ [35 – 37] были заявлены самоформирующиеся древовидные нейронные сети (self-generating neural trees) или SGNT-сети.

SGNT-сети отличаются от сетей, предложенных в [34], тем, что имеют в узлах нейроны специального вида. Схема работы сетей обеих типов будет сильно схожа: обе сети предполагают использование обучающих данных без учителя; алгоритм настройки выполняет иерархическую кластеризацию; обе сети предполагают некоторую избыточность в структуре и методы ее устранения. Рассмотрим общую идею работы на примере SGNT-сетей.

Пусть $e_i = (a_{i1},..., a_{in})$ – пример из обучающей выборки. Нейроном n_j назовем пару вида (W_j, C_j). Здесь $W_j = (w_{j1},..., W_{in})$ – вектор настраиваемых весов нейрона, а C_j – это множество всех нейронов-сыновей. SGNT-сетью назовем дерево вида $\langle\{n_j\},\{l_k\}\rangle$, состоящее из множества нейронов $\{n_j\}$ созданного алгоритмом обучения на основе обучающей выборки и множества связей $\{l_k\}$. Между нейронами n_i и n_j существует связь только если $n_j \in C_i$. Нейрон $n_k \in \{n_j\}$ будем называть победителем множества $\{n_j\}$ для примера e_i, если выполняется условие для каждого j $d(n_k, e_i) \le d(n_j, e_i)$, где $d(n_j, e_i)$ – евклидово расстояние между нейроном n_j и примером:

$$d(n_j, e_i) = \sqrt{\frac{\sum_{k=1}^{n} p_k (a_{jk} - a_{ik})^2}{n}}. \qquad (4.23)$$

Здесь p_k будут весами k-го атрибута. Из определения вытекает следую-

щий иерархический алгоритм построения SGNT-сети. На вход подается обучающая выборка $E = \{e_i\}$, $i = 1,...,N$ и задается пороговое расстояние $\xi \geq 0$.

Алгоритм:

```
copy(Корень, e_i);
for(i=1,j=1; i<N; i++)
{
   МинимальнаяДистанция = d(e_∞, Корень);
   Победитель = СтарыйПобедитель = Корень;
   МинимальнаяДистанция = test(e_i, Корень);
   if(МинимальнаяДистанция > ξ)
   {
      if(leaf(Победитель))
      {
         copy (n_j, Победитель);
         connect (n_j, Победитель);
         j = j + i;
      }
      copy(n_j, e_i);
      connect(n_j, Победитель);
      j = j + 1;
   }
   update(Победитель, e_i);
}
```

Функции:

copy(n,e) – создает нейрон n и копирует значения элементов вектора e в веса нейрона n;

d(n,e) – определяет дистанцию между n и e согласно (4.23);

test (e, ПодВетвь) – находит победителя в текущей ветви SGNT-сети с вершиной в ПодВетвь для примера e и возвращает расстояние между победителем и e;

leaf (n) – определяет, является ли нейрон n концевой вершиной дерева;

connect (n_0, n_1) – делает нейрон n_1 потомком нейрона n_0;

update(n_i, e_{k+1}) – обновляет веса нейрона n_i в соответствии с правилом (4.24).

Веса каждого неконцевого нейрона SGNT-сети будут представлять собой среднее значение соответствующих примеров выборки E. Отсюда

последовательное правило обновления весов, где w_{jk}^{i} – k-й элемент нейрона n_j после предъявления i примеров выборки:

$$w_{jk}^{i+1} = w_{jk}^{i} + \frac{1}{i+1}(a_k^{i+1} - w_{jk}^{i}) . \qquad (4.24)$$

После того как SGNT-сеть построена, может потребоваться некоторая оптимизация ее структуры. Предлагается балансировать дерево, используя горизонтальный и вертикальный критерии. Кроме того, сеть повторно тренируется несколько раз, и «мертвые» ветви дерева, которые не продолжают расти, отсекаются как излишние (подробнее см. [35]).

Эффективность SGNT-сетей проверялась на задаче голосования в США и задаче классификации грибов [36]. При сравнимых значениях производительности время обучения-построения SGNT-сетей оказалось в несколько раз меньше, чем у сетей, созданных методом «каскадной корреляции» (п. 4.4.5). NT-сети показали свою эффективность на задаче распознавания гласных звуков. Был проведен сравнительный анализ с многослойным персептроном и сетями радиально-базисных функций [34]. Основное отличие NT-сетей от SGNT-сетей заключается в том, что SGNT-сети фактически строят только разбиение множества обучающих примеров гиперплоскостями, а NT-сети с персептронами в узлах могут организовать более сложное разбиение.

4.4.3. Алгоритмы Monoplan, NetLines и NetSphere

Алгоритмы Monoplan, NetLines и NetSphere – это алгоритмы автоматического построения нейронных сетей, объединенные общей идеей, предложенные Торрес-Морено [38]. Рассмотрим кратко их принцип работы.

Требуется по обучающей выборке $(x_1,..., x_n)(x_1,...,x_n)$ и целевым значениям принадлежности двум классам $\{t_1^0 = -1, t_2^0 = +1\}$ построить нейронную сеть. На начальном этапе имеется сеть, состоящая из входов и одного выходного нейрона. Сеть обучается с использованием любого из алгоритмов, пригодных для настройки персептрона, например, «карманным» алгоритмом. Если после обучения имеются ошибки, то добавляется и обучается нейрон $h = 1$ в единственный скрытый слой. Если ошибки классификации присутствуют, то модифицируются целевые значения такие, как $t_1^1 = +1$ для всех примеров правильно определенных нейроном и $t_1^1 = -1$ для прочих, и снова добавляется нейрон $h = 2$ в

скрытый слой. Всякий раз, когда нейрон добавлен и обучен, его веса остаются замороженными. При подобном способе построения каждый новый нейрон будет устранять хотя бы одну ошибку предыдущего нейрона, что гарантирует сходимость.

В алгоритмах NetLines и NetSphere появляются усовершенствования. Рассмотрим пример на рис. 4.8. Класс t_1 определен серым цветом, остальное – класс t_2. Следуя процедуре построения, получаем сеть с двумя нейронами в скрытом слое, задающими две границы решения w_1 и w_2. Из рис. 4.8. видно, что разные классы все еще разделены неверно. Теперь, всякий раз после добавления нейрона выходной нейрон (w_3) будет переобучаться по схеме, аналогичной описанной. Таким образом, правильное решение достигается гораздо быстрее. Алгоритм NetSphere имеет отличия только в форме пороговой функции

$$g(a) = \left| \sum_{i=1}^{N} (w_i - x_i)^2 - \theta^2 \right|, \qquad (4.25)$$

где θ – радиус окружности, w_i – веса нейрона, задающие центр окружности. Таким образом, разделяющая поверхность будет иметь более сложный вид и существенно улучшает результаты.

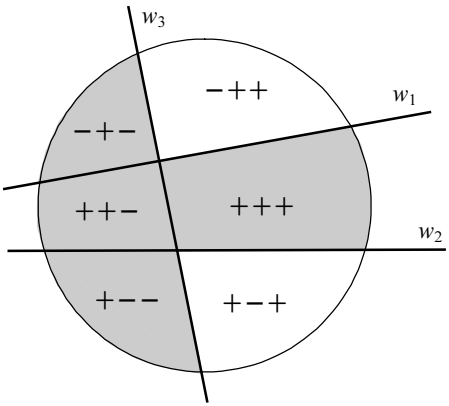


Рис. 4.8. Пример работы алгоритма NetLines

Данное семейство алгоритмов предназначено для решения классификационных задач в случае двух классов. Для расширения на произвольное число классов предложено использовать несколько нейронных

сетей с объединением результатов работы. Работа алгоритма была исследована для задач диагностики рака и заболевания диабетом [38].

4.4.4. Метод динамического добавления узлов

Еще одним интересным методом построения нейронных сетей является метод динамического наращивания узлов [39].

В оригинальном виде процесс построения начинается с исходной нейронной сети с одним нейроном в единственном скрытом слое. При обучении сети любым способом, например методом обратного распространения ошибки, вычисляется ошибка E результирующих выходов сети $y_n(x;W)$, где x – входной вектор, W – матрица весов связей. Можно считать, что сложность текущей сети не соответствует поставленной задаче, если ошибка E с числом эпох обучения уменьшается недостаточно быстро, либо не уменьшается совсем. Уменьшение скорости ниже заданного порога служит сигналом к тому, что для улучшения результатов при падении скорости ошибки в [39] предлагается добавлять по одному нейрону в скрытый слой. Формально скорость изменения ошибки можно определить следующим образом:

$$\frac{\left|E_t - E_{t-\delta}\right|}{\left|E_{t_0}\right|} < \Omega . \qquad (4.26)$$

Здесь E_{t_0} – значение ошибки в момент t_0, t_0 – момент добавления предыдущего нейрона, Ω – некоторая заданная минимальная скорость изменения ошибки, при превышении которой принимается решение об изменении сети, δ – количество эпох – циклов обучения сети перед произведением оценки скорости ошибки. Таким образом, скорость определяется изменением ошибки относительно первоначального замера значения.

Основным достоинством предлагаемого здесь подхода является то, что в нем не налагается никаких принципиальных ограничений на способ построения нейронной сети, а производится только оценка изменения производительности. Упомянутую методику можно применять и наряду с другими способами построения нейронных сетей оптимальной топологии. Метод динамического добавления узлов был опробован на нескольких простых задачах, таких как сложение двоичных чисел. Было показано, что для подобных задач дополнительные вычисления возрастают не более чем на 40% по сравнению с тренировкой нейронной сети уже оптимальной структуры.

4.4.5. Каскадная корреляция и ее модификации

Одним из самых известных и исследованных методов построения нейронных сетей является алгоритм каскадной корреляции, предложенный Фальманом в [40]. Метод содержит в себе две основные идеи: особую каскадную структуру, вариант архитектуры сетей прямого распространения, в которой новые нейроны последовательно добавляются и уже не изменяются после настройки. Вторым важным моментом является способ автоматического выбора и настройки нового нейрона. Рассмотрим алгоритм каскадной корреляции.

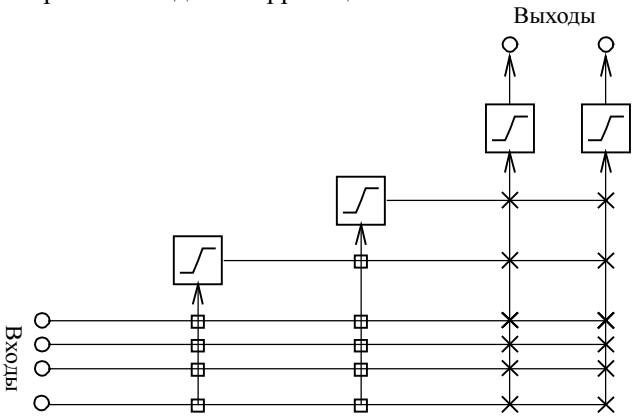


Рис. 4.9. После добавления второго нейрона:
□ – узлы заморожены, × – узлы настраиваются

Алгоритм начинает работу с обычного персептрона – сети, состоящей только из входов и выходов, с одним полносвязным слоем настраиваемых весов. Алгоритм можно разбить на две части: обучение выходных нейронов и создание нового нейрона.

На начальном этапе, в первой части, настраивается прямое соединение входов-выходов с использованием любого алгоритма обучения однослойной сети. Алгоритм использует эффективный метод обучения quick-prop [41]. После того, как за заданное число циклов обучения не происходит существенного уменьшения ошибки, принимается решение: если ошибка меньше желаемого значения, работа алгоритма завершена, в противном случае – инициируется создание нового нейрона.

Во второй части алгоритма новый нейрон-«кандидат» добавляется к существующей сети, причем его входы соединяются со всеми внешни-

ми входами сети и выходами уже добавленных прежде нейронов. Входные веса «кандидата» настраиваются особым образом так, чтобы максимизировать влияние выхода этого нейрона на ошибку сети. После настройки его выход соединяется со всеми выходными нейронами сети, входные веса замораживаются, а все выходные настраиваются алгоритмом quick-prop. На рис. 4.9. показана сеть с двумя добавленными нейронами.

Рассмотрим способ настройки весов нейрона-«кандидата» для максимизации его влияния на ошибку. Целью обучения будет максимизация функции S – суммы по всем выходным нейронам o степени корреляции между o^h, новым значением «кандидата» и E внутренней ошибкой, полученной на выходном нейроне o. Отсюда S можно определить как

$$S = \sum_o \left| \sum_p (o_p^h - \overline{o}^h)(E_{p,o} - \overline{E}_o) \right|, \tag{4.27}$$

где $E_{p,o}$ – ошибка на выходе o для входного примера p; o_p^h – выходное значение нейрона-«кандидата» для входного примера p и величины $\overline{E}_o$ и $\overline{o}^h$ – значения соответствующих величин, усредненные по всей выборке.

Для максимизации S вычисляются частные производные S в отношении каждого входного веса w_i^h нового нейрона:

$$\frac{\partial S}{\partial w_i^h} = \sum_{p,o} \sigma_o (E_{p,o} - \overline{E}_o) f_p^{/} I_{i,p} \ . \tag{4.28}$$

Здесь σ_0 – знак корреляции между выходным значением «кандидата» o_p^h и ошибкой $E_{p,o}$ выхода o; f_p' – производная активационной функции нейрона-«кандидата» для примера p; $I_{i,p}$ – это входное значение, которое получает «кандидат» от i-го, добавленного на предыдущем этапе, нейрона. Максимизация S происходит также с помощью алгоритма quick-prop. В результате его работы находятся желаемые значения входящих весов «кандидата» w_i^h. Однажды вычисленные, они остаются без изменения до конца работы всего алгоритма.

Для улучшений характеристик работы алгоритма используются несколько приемов:

- используется пул – набор из нейронов «кандидатов» с активационными функциями различного типа и случайной инициализацией весов. Выбирался тот, что показывал максимальное значение S;

- так как веса нейронов, добавленных на предыдущем этапе, не изменяются, их выходы, соответствующие различным обучающим примерам, могут быть сохранены для повторного использования.

Алгоритм каскадной корреляции был протестирован на большом количестве задач, как искусственного происхождения: N – четность, проблема «двух спиралей» [40], так и классификационных задачах на реальных данных [42, 43]. Наиболее полное исследование алгоритма для 42 различных тестовых наборов было представлено в работе [44]. Доступны также критические работы по сравнению каскадной архитектуры с методами статистического анализа [45] и анализ каскадной архитектуры нейронной сети [46].

В большинстве работ алгоритм показал результаты не хуже, а в ряде случаев лучше, чем обычный алгоритм обратного распространения. Кратко достоинства алгоритма каскадной корреляции можно сформулировать так:

1) автоматическое определение числа нейронов. Возможность использования различных активационных функций одновременно;

2) быстрое обучение сети, нет трудоемкого обратного распространения ошибки.

Алгоритм каскадной корреляции получил дальнейшее развитие. Авторами было предложено его обобщение для случая рекуррентных сетей [47] и для задач с ограниченной точностью [48].

Существует несколько интересных модификаций алгоритма каскадной корреляции. Стандартный алгоритм конструировал нейронную сеть с большим количеством слоев с единственным нейроном. Необходимость в реализации нейронных сетей на аппаратном уровне породила варианты каскадной корреляции с ограничением числа слоев [49, 50]. В этом случае нейроны добавлялись в слой до достижения некоторого заданного числа, затем формировался следующий слой и т.д. В результате получались компактные нейронные сети с характеристиками не хуже, чем для сетей с каскадированием.

Некоторый интерес представляет алгоритм, предложенный в [51]. Существенным отличием от оригинального алгоритма является отсутствие каскадирования и роста сети вверх. Новые нейроны не соединяются с входами прежде добавленных нейронов, иначе говоря, нейроны добавляются всегда только в один скрытый слой. Отсюда название алгоритма: алгоритм каскадной корреляции второго порядка, а оригинальный алгоритм считается алгоритмом высшего порядка.

Еще один вариант каскадной корреляции, предложенный в работах [52, 53] под именем BINCOR, использует стратегию добавления сразу нескольких нейронов, по числу выходов сети (рис. 4.10). Это существенно ускоряет процесс обучения, так как каждый нейрон корректирует ошибку отдельного выхода, а не всех сразу, но и в то же время приводит к заметному увеличению размеров сети.

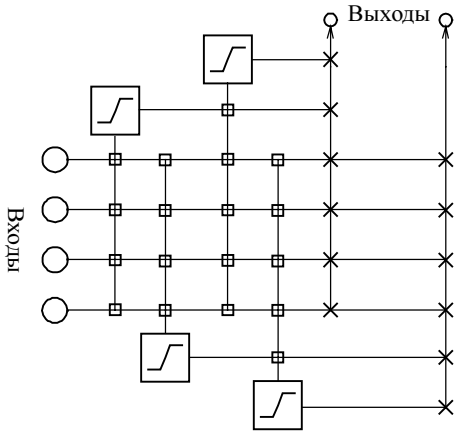


Рис. 4.10. BINCOR – вариант алгоритма каскадной корреляции

Сразу пять вариантов алгоритма каскадной корреляции разработано и исследовано Л. Пречелтом в [44]. Основной особенностью предложенных модификаций является использование в различных сочетаниях минимизации ошибки сети вместо максимизации ковариации (4.27) выбор нейрона-«кандидата» сразу по двум наборам данных: обучающему и проверочному. Лучшие результаты показал вариант, названный kogi9, в котором используются соображения, аналогичные примененным в алгоритме каскадной корреляции второго порядка: добавление нейронов в один слой. Только в kogi9 аналогично формируется сразу два скрытых слоя.

Алгоритм КасПер

Самые новейшие исследования и модификации алгоритма каскадной корреляции были проведены в серии работ Треадголдом и Гедеоном [54 – 58]. Фактически ими был предложен новый алгоритм – Кас-

Пер. Авторы предложили отбросить использование оценки корреляции по (4.27) и принцип замораживания весов, так как исследования показали, что это приводит к созданию нейронной сети завышенного размера. Сеть будет требовать больше нейронов для коррекции ошибок, допущенных предыдущими нейронами. Вместо этого в КасПер после добавления нейрона дообучается вся сеть с использованием особого алгоритма обучения.

Алгоритм КасПер начинает построение нейронной сети подобно оригинальному методу каскадной корреляции: начинает с единственного скрытого нейрона и последовательно добавляет по одному нейрону. Всякий раз после добавления нейрона используется адаптированная версия добавления алгоритма обратного распространения ошибки RPROP. RPROP модифицирован таким образом, чтобы всякий раз после добавления нового нейрона скорости обучения для весов сети устанавливались в различные значения, в зависимости от положения веса в сети. Все веса нейронной сети разбиваются на три группы: L_1, L_2 и L_3 (рис. 4.11). Набор L_1 составляют все веса, соединяющие новый нейрон со входами сети и выходами прежде добавленных нейронов. Второй набор включает в себя веса, соединяющие выход нового нейрона со всеми выходами сети. Оставшиеся веса включаются в набор L_3.

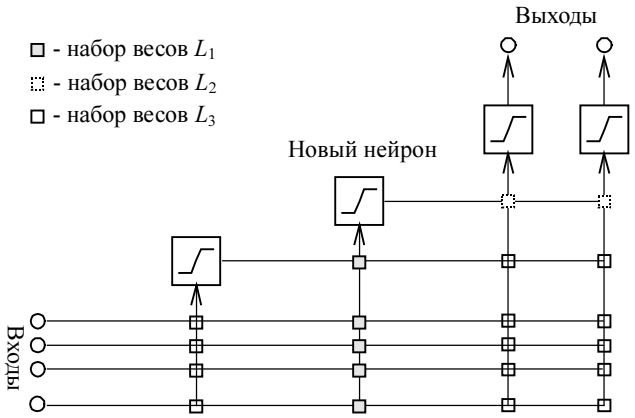


Рис. 4.11. Архитектура КасПер
в момент добавления второго нейрона

Скорости обучения для групп L_1, L_2 и L_3 принимают значения соответственно $L_1 >> L_2 > L_3$. Причина для подобного соотношения подобна причине, по которой оригинальный алгоритм каскадной корреляции использует оценку корреляции: большое значение L_1 поощряет новый нейрон к устранению оставшихся ошибок сети. А при $L_2 > L_3$ уменьшается ошибка сети без сильного влияния на веса остальных нейронов. Заметим, что таким образом КасПер реализует преимущества замораживания весов и техники корреляции оригинального метода каскадной корреляции и одновременно устраняет проблемы насыщения, обусловленные специфической оценкой корреляции и неизменным состоянием плохо настроенных нейронов.

Алгоритм КасПер также использует методику снижения весов для улучшения характеристик построенной сети. Градиент ошибки для КасПера выглядит следующим образом:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial w_{ij}} - k \cdot \text{sign}(w_{ij}) w_{ij}^2 \cdot 2^{-0{,}01 N_{\text{Epoch}}}, \qquad (4.29)$$

где N_{Epoch} – число эпох обучения после добавления последнего нейрона, $\text{sign}(w_{ij})$ – знак операндов w_{ij}. Решение о добавлении нового нейрона принимается, если скорость уменьшения ошибки становится меньше 1% за время $15+\alpha N$. Здесь N – число уже установленных нейронов сети с поправочным коэффициентом.

4.4.6. Методы редукции

Другим основным направлением, которое можно выделить среди конструктивных алгоритмов, являются методы редукции нейронной сети. Все подходы к редукции объединены следующей общей схемой работы. Исходной является некоторая, уже обученная, нейронная сеть. В предположении, что размеры этой сети можно уменьшить, отсекаются определенные каким-либо образом избыточные нейроны или существующие связи между ними. Как правило, в качестве критерия выбирается степень воздействия нейрона (связи) на величину ошибки, а убираются соответственно слабо влияющие нейроны (веса). После упрощения нейронной сети повторяется процесс дообучения и повторного усечения до тех пор, пока величина ошибки не достигнет желаемой величины или топология нейронной сети не достигнет необходимой простоты. Очевидными недостатками подхода усечения являются неопреде-

ленность в принципах построения исходной нейронной сети и большой объем вычислений. Обычно метод усечения применяется для экономии ресурсов при необходимости аппаратной реализации уже найденной нейронной сети.

Редукция нейронов

Исторически, первой техникой упрощения сети была техника усечения, предложенная в 1989 г. Мозером и Смоленским [59]. Ими была введена следующая оценка значимости j-го нейрона:

$$S_j = E(\text{без нейрона } j) - E(\text{с нейроном } j). \qquad (4.30)$$

Вычисление (4.30) будет трудоемким, так как потребуется полный проход по всей обучающей выборке для каждого нейрона сети. Вместо точного значения s_j предложено вычисление оценки $\hat{s}_j$. Для этого пороговая функция $g(a)$ каждого нейрона дополняется коэффициентом α_j:

$$z_j = g(\alpha_j \sum_i w_{ji} z_i)\,;\ g(0) = 0\,;\ g(a) = \text{th}(a)\,. \qquad (4.31)$$

Таким образом, для $\alpha_j = 0$ нейрон будет отсутствовать, для $\alpha_j = 1$ присутствовать. Отсюда (4.30) можно переписать:

$$s_j = E(\alpha_j = 1) - E(\alpha_j = 0)\,. \qquad (4.32)$$

Для оценки s_j авторами было предложено использовать производную ошибки по α_j, которую можно автоматически вычислять с помощью модифицированного алгоритма обратного распространения ошибки:

$$\hat{s}_j = -\frac{\partial E}{\partial \alpha_j}\Big|\alpha_j = 1\,. \qquad (4.33)$$

Контрастирование сетей

Более исследованным подходом к упрощению нейронной сети является техника усечения весов, основанная на оценке значимости весов связей. В отечественной литературе техника усечения весов носит название процедура контрастирования и впервые была предложена в работе [60].

Пусть имеется нейронная сеть, правильно решающая все примеры обучающего набора. При работе метода обратного распространения ошибки сеть вычисляет градиент функции оценки (ошибки) $H(w)$ по весам связей $w_j - \partial E/\partial w_j$. Пусть w^0 – текущий набор весов сети, а ошибка

сети для текущего примера равна E^0. Тогда в линейном приближении можно записать функцию оценки в точке w как:

$$E(w) = E^0 + \sum_i \frac{\partial E}{\partial w_i}(w_p - w_p^0). \quad (4.34)$$

Используя приближение (4.34), можно оценить изменение оценки при замене w_p^0 на w_p^* как

$$\aleph(p,q) = \left|\frac{\partial E}{\partial w_i}\right| \cdot \left|(w_p^0 - w_p^*)\right|. \quad (4.35)$$

Здесь q – индекс примера обучающего набора, для которого были вычислены оценка и градиент. Величину $\aleph(p,q)$ будем называть показателем чувствительности к замене w_p^0 на w_p^* для примера q. Далее необходимо вычислить показатель чувствительности, не зависящий от номера примера. Обычно используется равномерная норма максимума модуля:

$$\aleph(p) = \max_q(\aleph(p,q)).$$

Таким образом, процедура контрастирования заключается в следующем. Каким-либо образом обучается нейронная сеть с требуемым малым значением ошибки $\hat{E}$. На каждом шаге работы определяем минимальный показатель чувствительности – $\aleph(p^*)$, заменяем соответствующие веса $w_{p^*}^0$ на $w_{p^*}^*$ и исключаем его из сети. Вся процедура контрастирования повторяется до тех пор, пока значение ошибки $\hat{E}$ не начинает увеличиваться.

Из способа вычисления (4.34) и (4.35) видно, что необходимо знать вид оценочной функции и процедуры обучения для правильной работы процедуры контрастирования. Однако существует вариант контрастирования, который позволяет обойтись без этих сведений.

Пусть дана только обученная нейронная сеть и обучающее множество, причем вид функции оценки и процедура обучения неизвестны. В этом случае контрастирование весов производится понейронно. В каждом нейроне стоит адаптивный сумматор, который суммирует входные сигналы нейрона, умноженные на соответствующие веса связей. Если обозначить входные сигналы нейрона при решении q-го примера как x_p^q, получим формулу для показателя чувствительности весов:

$$\aleph(p,q) = \left|x_p^q(w_p - w_p^*)\right|. \quad (4.36)$$

Получаем показатель чувствительности

$$\aleph(p) = \max_q (x_p^q) \left| (w_p - w_p^*) \right| .$$

В работе [58] в качестве основных целей контрастирования указаны: упрощение технической реализации нейронной сети и выявление знаний, полученных сетью в ходе обучения. Результаты экспериментов по контрастированию нейронных сетей опубликованы в [61].

Optimal Brain Damage и Optimal Brain Surgeon

Для направления упрощения сети методом усечения весов широко известными являются методы «оптимального повреждения мозга» (Optimal Brain Damage, OBD) [62] и его более позднее развитие – «лучший нейрохирург» (Optimal Brain Surgeon, OBS) [63]. Рассмотрим эти методы подробнее.

Пусть изменение ошибки δE в результате некоторых малых изменений значений весов w_i на $w_i + \delta w_i$

$$\delta E = \sum_i \frac{\partial E}{\partial w_i} \delta w_i + \frac{1}{2} \sum_i \sum_j H_{ij} \delta w_i \delta w_j + O(\delta w^3) . \qquad (4.37)$$

Здесь $H_{ij} = \dfrac{\partial^2 E}{\partial w_i \partial w_j}$ – элементы матрицы вторых производных, гессиан.

В предположении, что процесс обучения завершен и ошибки нет в (4.37) можно отбросить первое слагаемое. В работе [62] было предложено рассматривать приближенное значение гессиана по элементам его главной диагонали, отсюда следует, что

$$\delta E = \frac{1}{2} \sum_i H_{ii} \delta w_i^2 . \qquad (4.38)$$

Тогда, если исходный вес w_i сделать равным нулю, то увеличение ошибки можно оценить по (4.38) с $\delta w_i = w_i$. Отсюда легко можно определить значимость веса w_i как $(H_{ii} w_i^2)/2$.

В результате алгоритм OBD можно сформулировать в следующих этапах. Для уже имеющейся обученной нейронной сети с малым значением ошибки $\hat{E}$ вычисляются вторые производные H_{ii} для каждого веса

w_i; по (4.38) определяется значимость весов и отбрасывается часть с малыми значениями до тех пор, пока не будет достигнут некоторый более общий критерий (размер сети, величина ошибки).

Однако предположение, что гессиан можно аппроксимировать по его диагонали, не всегда дает хорошее решение. Более совершенная процедура усечения весов – OBS избегает подобных предположений [61].

Предположим, установлен вес $w_i = 0$. Оставшиеся веса сети корректируются так, чтобы минимизировать увеличение ошибки сети – δw. Из (4.37), предполагая, что сеть была предварительно обучена, и отбрасывая малую величину $\theta(\delta w^3)$, получаем в векторной форме

$$\delta E = \frac{1}{2}\delta w^T H \delta w \,. \tag{4.39}$$

Изменения значений весов должны удовлетворять условию

$$e_1^T \delta w + w_1 = 0 \,. \tag{4.40}$$

Здесь e_1 – единичный вектор в пространстве весов, параллельный оси w_i. Изменение весов δw, минимизирующее δE, можно найти, используя множитель Лагранжа:

$$\delta w = -\frac{w_i}{[H^{-1}]_{ii}} H^{-1} e_1 \,. \tag{4.41}$$

Соответствующее увеличение ошибки

$$\delta E_i = \frac{1}{2}\frac{w_i^2}{[H^{-1}]_{ii}} \,. \tag{4.42}$$

Заметим, что в случае, если гессиан является диагональным, выражения (4.41) фактически сводятся к предыдущему методу.

Практический алгоритм OBS можно представить следующим образом. Имеется уже обученная нейронная сеть с малым значением функции ошибки. Затем вычисляется обратная матрица гессиана H^{-1}, по (4.42) рассчитываются δE_i для каждого веса i и выбирается i, дающий минимальное увеличение ошибки. При помощи (4.41) обновляются значения всех весов сети. Вся процедура повторяется, пока не будет достигнут некоторый более общий критерий (размер сети, величина ошибки).

На рис. 4.12. наглядно иллюстрируется сопоставление методов OBD и OBS для случая двух весов. Локальный минимум, найденный градиентным спуском, находится в точке w^*. На рисунке также отражено простейшее соображение, когда усечение весов производится по эмпирическому правилу: меньшее значение соответствует менее значимому весу.

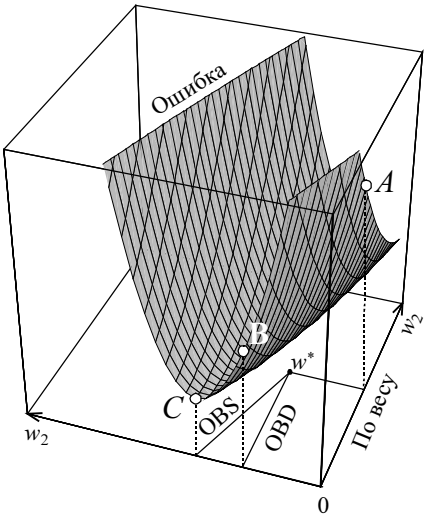


Рис. 4.12. Сравнение методов Optimal Brain Damage и Optimal Brain Surgeon для случая двух весов

Такой способ отбросит меньший вес w_2 и переместит вектор весов в точку *A*. Метод OBD, следуя правилу (4.38), отбросит вес w_1 и переместит вектор весов в *B*. Метод OBS кроме того, что отбросит вектор w_1, в соответствии с (4.41) скорректирует вес w_2. Таким образом, величина ошибки будет подчиняться неравенству

$$E(\text{по весу}) \geq E(OBD) \geq E(OBS).$$

Исследования показали, что действительно метод OBS гарантирует лучший результат, чем более ранний OBD [63, 64].

4.5. Эволюционный способ создания нейронных сетей

4.5.1. Генетические алгоритмы

Современным способом построения нейронных сетей является использование эволюционного подхода к поиску по пространству возможных архитектур. Исторически сложились три методологии эволюционного поиска [65]: эволюционное программирование Фогела, 1966 г., эволюционные стратегии Реченберга, 1973 г. и генетические алгоритмы (ГА) Холланда, 1975 г. Идеи этих методов похожи, но они различаются в выборе представления индивидуумов, механизмах селекции, форме генетических операторов и способах оценки приспособленности.

Наиболее универсальным подходом являются ГА Холланда [66 – 68]. Идея ГА основана на принципе эволюции биологических существ: из всей совокупности особей данного вида выживают только наиболее приспособленные к сложившимся условиям окружающей среды индивидуумы. Полезные свойства закрепляются и накапливаются на генетическом уровне в последующих поколениях путем обмена генами. Возвращаясь к нейронным сетям, отметим, что в данном случае набор всех возможных нейронных сетей будет выступать в качестве популяции, архитектура сети (число слоев, нейронов, связей) будет генетическим кодом, критерием приспособленности будет наилучшая точность результатов и малый размер сети.

Каждый индивидуум популяции будет представлять собой возможное решение поставленной проблемы. Параметры решения кодируются в строку символов – генетическую строку. В оригинальном методе, предложенном Холландом, использовалась двоичная строка фиксированной длины. Если проблема имеет несколько параметров, генетическая строка содержит набор подстрок, или генов, соответствующих каждому параметру. Современные варианты ГА применяют генетическую строку переменной длины [69] различной структуры: векторы, графы, упорядоченные списки, Лисп-выражения [65]. Используя биологическую терминологию, можно сказать, что информация, содержащаяся в строке, будет генотипом индивидуума популяции. Результат же – восстановленное решение задачи, можно назвать фенотипом. Очевидно, что способ кодирования задачи в генетическую строку будет существенно определять эффективность работы ГА. Кодирование решения задачи в генетическую строку производится пользователем ГА. Сами ГА в процессе функционирования не владеют информацией о значении за-

кодированной строки. Основной процесс их работы заключается в манипулировании наборами генетических строк – популяциями в репродукционном процессе с помощью генетических операций.

В репродукционном процессе новые решения (индивидуумы) создаются при помощи выбора, рекомбинаций и изменений существующих решений, основываясь на оценочной функции, часто называемой функцией приспособленности (фитнесс-функцией). Функция приспособленности оценивает приспособленность каждого индивидуума по отношению к «окружающей среде». Так как каждый индивидуум являет собой возможное решение задачи, то «окружающая среда» есть поставленная задача. Если поставлена задача найти оптимальную нейронную сеть при помощи ГА, то функцией приспособленности может быть величина, обратная ошибке сети или равная нулю в случае, если сеть не может решить проблему. Следует отметить, что ГА использует только значение функции приспособленности, но не смысловую нагрузку. Это позволяет строить произвольно сложную оценочную функцию путем комбинирования нескольких факторов.

Механизмы селекции

Для выделения индивидуума из популяции и последующей обработки генетическими операциями используются механизмы селекции [70]. По аналогии с биологическими механизмами селекции индивидуумы с большим значением функции приспособленности (более приспособленные) вероятнее всего выбираются для «размножения». Обычно выделяют два механизма выбора. Первый вариант считается классическим и носит название выбор по колесу рулетки, предложенное в [71]. Выбор по колесу рулетки предполагает выбор генетических строк с вероятностью, пропорциональной значениям их функции приспособленности относительно данной популяции.

Данный подход прекрасно работает на начальных этапах ГА, но ближе к концу значения функции приспособленности отличаются друг от друга незначительно, что препятствует выбору действительно лучших индивидуумов. Другим недостатком приведенного способа выбора будет то, что в случае доминирования некоторого большого значения функции приспособленности над всей популяцией результатом может стать скатывание ГА к неоптимальному решению.

В альтернативном подходе: выбор по рангу в популяции [72] шанс быть выбранным для индивидуума зависит от ранга или относительного

положения в популяции. Вся популяция сортируется по значению функции принадлежности. Ранг R индивидуума определяется по формуле

$$R = \frac{N - i + 1}{M};\ M = \sum_{p=1}^{N} p, \tag{4.47}$$

где i – позиция индивидуума в популяции, а N – их количество. Для популяции из $N = 10$ индивидуумов, согласно (4.47), ранг первого индивидуума будет равен $R_1 = \frac{10}{55} \approx 0{,}18$, а последнего $R_{10} = \frac{1}{55} \approx 0{,}01$. Такой подход позволяет решить все поставленные выше проблемы и, кроме того, абстрагироваться от абсолютного значения функции приспособленности.

Операция скрещивания

Операция скрещивания, или кроссинговер, моделирует воспроизводство потомка от двух родителей. Скрещивание создает новых членов популяции путем комбинирования различных частей двух генетических строк родителей, как показано на рис. 4.13. Число и позиция точек пересечения выбирается, как правило, случайным образом. Использование операции скрещивания позволяет закреплять и усиливать в следующих поколениях выработанные «полезные гены» – параметры решения.

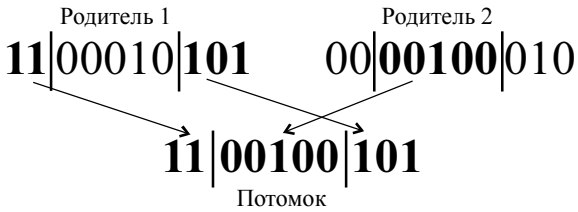


Рис. 4.13. Операция для двух точек сечения

Операция мутации

Операция мутации генерирует новых индивидуумов, случайным образом изменяя произвольное количество битов (символов, генов) генетической строки (рис. 4.14). Задача этой операции – производить случайные скачки по пространству возможных решений и выводить ГА из локальных решений, полученных операцией кроссинговер. Для того

чтобы ГА не сводились к алгоритмам простого случайного поиска, вероятность применения оператора мутации делается малой.

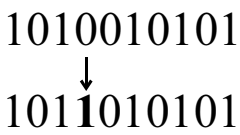


Рис. 4.14. Операция мутации. Изменяется четвертый бит

Таким образом, принципиальная схема работы ГА будет выглядеть следующим образом (рис. 4.15).

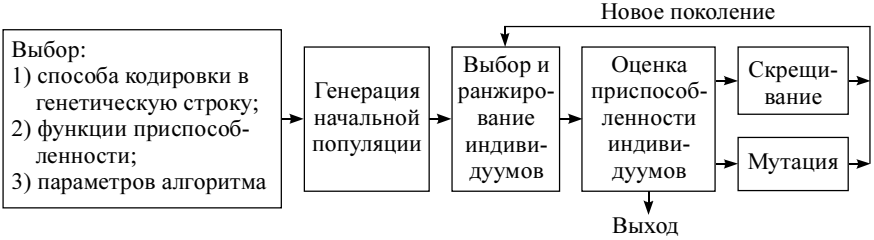


Рис. 4.15. Схема работы генетических алгоритмов

Попробуем теоретически обосновать, почему такой простой принцип работы и несложные генетические операции дают столь мощный поисковый механизм. Рассмотрим простой пример. Пусть задача заключается в том, чтобы при помощи ГА найти максимум функции $f(x) = x^2$ для $x \in [0, 255]$. В качестве генетической строки возьмем строку из 8 элементов, являющую двоичное представление x.

Если имеются только следующие строки (рис. 4.16) и неизвестен вид функции приспособленности F, лучшим способом для получения новых строк будет поиск совпадений между строками с максимальным приспособлением.

Ген. строка	Приспособленность
0 0 0 0 0 1 1 0 1	169
1 1 0 0 0 1 1 0	39204
0 0 1 1 1 0 0 1	3249
1 1 1 1 0 0 0 1	57600

Рис. 4.16. Значения функции приспособленности для $f(x) = x^2$

Для рис. 4.16 можно сделать вывод, что 1 в старшем разряде влияет на высокое значение приспособления. Идея использования малых частей строк с высоким приспособлением для производства новых строк формально описывается концепцией генетических схем и строительных блоков.

Назовем шаблоном структуру, описывающую подстроки с совпадениями [66]. Шаблон [1 * 0001 * 0] (* – любой символ) соответствует любой из строк: {10000100,10000110, 11000110}. Оценим количество всех возможных шаблонов *S,* заключенных в популяции из *n* строк длины *l*. Если каждая генетическая строка строится алфавитом из *k* символов ($k = 2$ для двоичных строк), тогда $S = (k + 1)^l$, так как кроме *k* символов может быть и знак «*». Для примера на рис. 4.16 $S = (2 + 1)^8 = 6561$, тогда как число всех возможных строк $2^8 = 256$. Для популяции строк размера *n* число вовлеченных шаблонов будет лежать в диапазоне от 2^l до $n \cdot 2^l$. Таким образом, даже малая популяция может содержать огромное количество информации о важных совпадениях строк.

Назовем длиной шаблона δ расстояние между первым и последним элементом не*. Операция скрещивания имеет тенденцию к разбиванию шаблонов большой длины, когда точки разбиения выбираются равномерно случайно, например шаблон 1*****10 имеет больший шанс быть разбитым, чем * * * * *10* (6/7, 86% против 1/7, 14%).

Нижнюю границу для вероятности выживания $p_в$ при скрещивании оценим как

$$p_в \geq 1 - p_c \, \delta / (l - 1) . \tag{4.48}$$

Здесь p_c – это вероятность применения скрещивания.

Оценим эффект применения операции выбора. Когда имеется $m(t)$ строк, заданных определенным шаблоном на момент времени *t* в популяции, можно ожидать:

$$m(t+1) = m(t) n \cdot f_в \Big/ \sum_{i=1}^{n} f_i . \tag{4.49}$$

Здесь *n* – размер популяции, $f_в$ – среднее значение приспособленности для строк, представленных данным шаблоном. Если переписать формулу (4.49), используя среднее значение приспособленности для популяции $f_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} f_i$, получим

$$m(t+1) = m(t) \cdot f_в \Big/ f_{\text{avg}} . \tag{4.50}$$

Шаблон со значением приспособленности выше среднего имеет больше шансов быть перенесенным в следующем поколении и получает при этом увеличение числа попадающих в него строк.

Общий эффект от операций скрещивания и выбора может быть оценен как комбинация (4.48) и (4.50):

$$m(t+1) \geq m(t)\frac{f_{\text{в}}}{f_{\text{avg}}}\left[1 - p_{\text{c}}\frac{\delta}{l-1}m(t)\right]. \qquad (4.51)$$

Каждый шаблон увеличивает свое присутствие в зависимости от двух факторов: отношения приспособленности шаблона к средней приспособленности популяции и длины шаблона. Таким образом, определяется теорема шаблонов [70]: шаблоны с большим значением приспособленности и короткой длиной продвигаются экспоненциально через поколения. Такие шаблоны называются строительными шаблонами. Можно сказать, что механизмы ГА ведут к комбинированию лучших частичных решений в лучшее общее решение.

Экспоненциальное продвижение шаблонов малой длины и высокой приспособленности производится параллельно в пределах одной популяции из n генетических строк. В работе [71] было оценено более точно количество шаблонов, обработанных в каждом поколении: их количество оценено около n^3. Так как за одно поколение производится только n вычислений функции приспособленности, эта особенность ГА была названа неявным параллелизмом.

4.5.2. Эволюционные алгоритмы для нейронных сетей

Идея комбинирования нейронных сетей и генетических алгоритмов представляется довольно естественной хотя бы вследствие того, что обе методики были позаимствованы из биологических систем [73]. Можно сказать, что таким объединением моделируется естественная ситуация, когда индивидуумы, выжившие в результате эволюционного процесса и приспособившиеся к создавшимся условиям окружающей среды (генетические алгоритмы), затем непосредственно обучаются решать более конкретные задачи (нейронные сети). Можно выделить три основных подхода к объединению ГА и нейронных сетей [74]:

1) использование ГА как инструмента поиска параметров алгоритма обучения нейронной сети;

2) для непосредственного обучения сети (поиска по пространству весов);

3) для нахождения архитектуры нейронной сети.

В нашем обзоре будет рассматривается только последний вариант.

Общую схему поиска архитектуры нейронных сетей с помощью ГА можно представить следующим образом. Информация об архитектуре нейронной сети специальным образом кодируется в генетический код. Затем генерируются поколения, к которым применяются стандартные генетические операции. Оценка каждого индивидуума производится следующим образом: нейронная сеть восстанавливается из генетической строки и производится ее тестирование. Значение функции приспособленности, как правило, будет зависеть от значения ошибки сети на проверочной выборке. Работа ГА завершается по достижению желаемого значения ошибки [75].

Для вычисления функции принадлежности применяется два подхода: с использованием процедуры обучения сети и без обучения. Эффективность использования процедуры обучения нейронной сети для оценки ее приспособленности основано на «эффекте Балдвина». В своей работе «Новый фактор в эволюции» в 1896 г. Балдвин предположил, что в эволюционном процессе через поколения в генетическом коде передается не сами полученные навыки, а только гены, несущие способность к обучению, что естественным образом повышает адаптационные возможности индивидуума. В работе [76] дается вывод о том, что при использовании «эффекта Балдвина» ГА дают значительно лучшие результаты для нейронных сетей, правда, при этом время работы несколько увеличивается.

Описанная выше процедура поиска нейронных сетей при помощи ГА вполне очевидна и проста. Основной трудностью здесь будет удачный выбор способа кодирования информации об архитектуре сети в генетический код. Эта проблема породила массу различных подходов и вариаций. Рассмотрим основные трудности, возникающие при кодировании/декодировании нейронных сетей, и дадим краткий обзор наиболее интересных подходов.

4.5.3. Подходы к кодированию нейронных сетей

Можно выделить два подхода к кодированию нейронных сетей: прямое и порождающее кодирование [77]. Прямое кодирование подразумевает хранение полной информации о структуре сети в генетическом коде. В противоположность этому порождающее кодирование предполагает хранение правил порождения нейронной сети. Каждое из правил может порождать часть сети.

Основным достоинством прямого кодирования будет порождение малого количества «нежизнеспособных» нейронных сетей, т.е. сетей с архитектурой, делающей невозможным их функционирование. Главной причиной появления порождающего кодирования была невозможность использования прямого кодирования для нейронных сетей произвольно большой размерности. Объем генетического кода для больших сетей возрастал пропорционально, что существенно отражалось на производительности. Многие алгоритмы вынуждены были искусственно ограничивать размер сети и соответственно уменьшать пространство поиска. Для порождающего кодирования размеры сети не будут существенны, но применение генетических операций к правилам влечет за собой порождение некоторого количества «нежизнеспособных» архитектур, которые следовало распознавать и отбрасывать. Как было отмечено в [78], для порождающего кодирования время поиска несколько выше, чем для прямого.

4.5.4. Прямое кодирование

Прямая кодировка по Миллеру

Одной из самых простых является техника кодирования, предложенная Миллером в [79]. Все соединения сети из N нейронов кодируются матрицей связности нейронов размерностью $N\cdot(N+1)$. Все строки матрицы затем конкатенируются в генетическую строку (рис. 4.17).

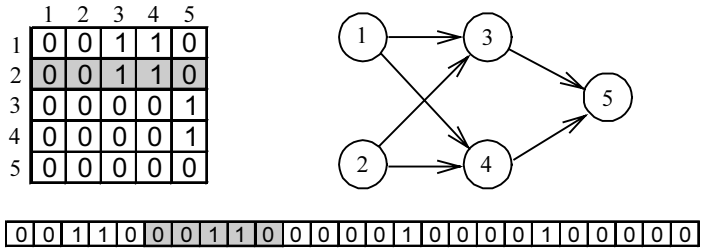


Рис. 4.17. Прямая кодировка по Миллеру

Все восстановленные сети, не попадающие под определение нейронных сетей прямого распространения, отбрасываются. Кодировка была исследована для задачи XOR на популяции из 50 индивидуумов

[79]. Аналогичный принцип кодирования, с некоторыми модификациями, применяется и в работах других авторов [80 – 83].

GENITOR

Одним из классических подходов к прямому кодированию является алгоритм GENITOR, предложенный в [72]. В данном способе кодирования в генетический код заносится информация как об архитектуре, так и о весах связей. Индексный бит обозначает присутствие соединения, а 8-битовое число представляет целое значение соответствующего веса в интервале [–127, +127] (рис. 4.18). GENITOR требует приблизительное (максимальное) определение архитектуры нейронной сети до начала работы. Вследствие того, что операция скрещивания может иметь точки сечения в произвольном месте битовой строки, потомки могут иметь значения некоторых весов, существенно отличающиеся от родительского. В дальнейшем этот метод получил развитие для вещественных чисел. В работе [84] исследуется влияние размера генетической строки, построенной подобным образом, на конечный результат.

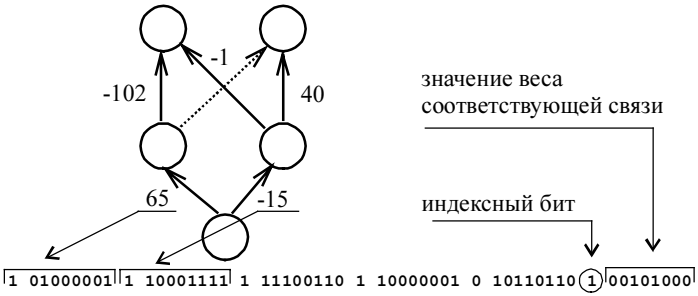


Рис. 4.18. Кодирование алгоритмом GENITOR

BP-Generator

Описанные выше методы, основанные на кодировании соединений, имеют один существенный недостаток: необходимо указывать размеры максимальной сети. Этого неудобства лишены методы, базирующиеся на стратегии кодирования нейронов. Одним из первых подобный способ был предложен в работах [85 – 87] под именем BP-Generator .

Строка генетического кода представляла собой набор пронумерованных нейронов. Каждому нейрону предшествует список связанных с

ним нейронов (рис. 4.19). Существует вариант, когда строка кода содержит также информацию о весах связей.

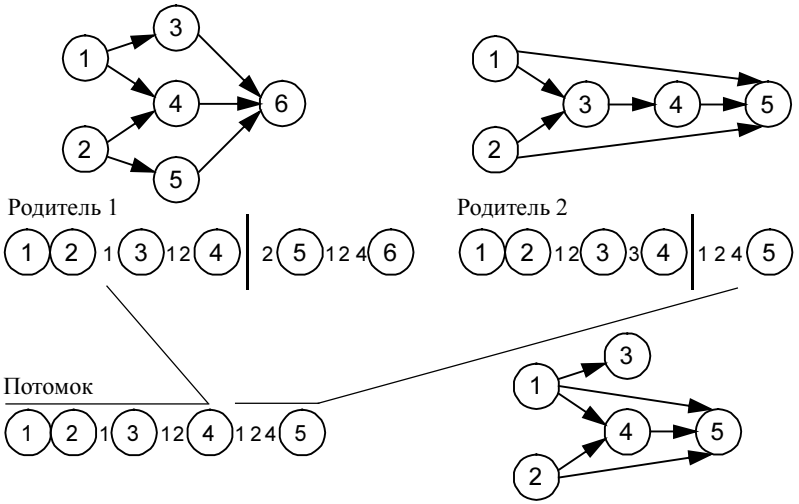


Рис. 4.19. Применение операции скрещивания для кодировки BP-Generator

Алгоритм предлагает следующий оператор мутации: случайное удаление и внесение слабых соединений (малое значение веса) и просто добавление новых нейронов. Схема работы оператора скрещивания представлена на (рис. 4.19). Точка сечения может быть только между нейронами и не разбивает списка связанных нейронов. Если в генетическом коде после применения генетических операций появляются нейроны без входных и выходных соединений, они исключаются из кода. В работе [88] применяется аналогичный способ кодирования с использованием распараллеленного эволюционного процесса.

Древовидное представление

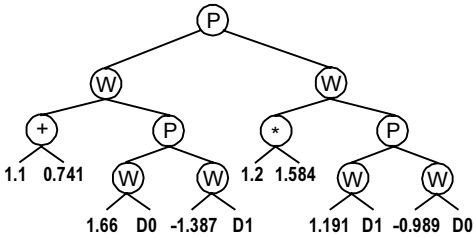
В способе кодирования, предложенном Козой в работе [89], была применена кодировка нейронов и весов связей в виде дерева, записанного в виде выражения на Лиспе. Такое представление также позволяет не заботиться об определении предварительных размеров сети. На рис. 4.20. приведен пример такой записи. Вершины дерева – (P) представляют собой нейроны, а вершины (W) – веса связей. Для восстановления

нейронной сети дерево разбирается снизу вверх от выходов сети. Каждый выход требует свое собственное дерево, поэтому фактически выходы будут независимы друг от друга. Каждый нейрон (P) может иметь произвольное количество поддеревьев, начинающихся с (W). Вес (W) имеет два поддерева: одно для значения веса, второе для обозначения нейрона источника соединения. Два входных нейрона обозначены как D0 и D1. Значения весов может быть записано как вещественным числом, так и выражением. Операция скрещивания заключается в обмене поддеревьями двух родителей, мутация произвольно изменяет поддеревья и значения весов. В работе [90] аналогичная схема кодирования использована в модуле построения нейронных сетей в известной программе NeuroForecaster.

LISP S-выражение

```
(P (W (+ 1.1 0.741) (P (W 1.66 D0)   (W -1.387)))
   (W (* 1.2 1.584) (P (W 1.191 D1) (W -0.989 D0))))
```

Представление в виде дерева



Нейронная сеть

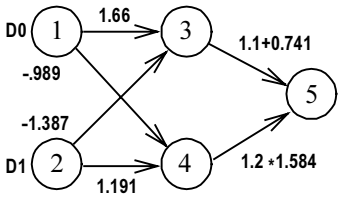


Рис. 4.20. Древовидное представление архитектуры

Послойное представление

Схема прямого кодирования, основанная на представлении нейронной сети в генетическом коде по слоям, была предложена в [91]. Основной структурой будет список блоков с информацией о слоях и блок параметров сети (рис. 4.21). Значения весов не кодируются. В качестве параметров сети сохраняются параметры алгоритма обучения. Для каждого слоя в блоке описания сохраняется информация о размере сети, выходных и входных соединениях слоя с другими слоями. Выходные соединения предполагают, что каждый нейрон слоя имеет связь со следующим смежным слоем. Входные соединения приходят с предшест-

вующего слоя сети, заданного в поле соединение. Поле радиус определяет распределение соединений по нейронам присоединенного слоя, поле плотность – число нейронов в этой области. Все три поля: соединение, радиус и плотность задаются относительно числа нейронов соответствующего слоя.

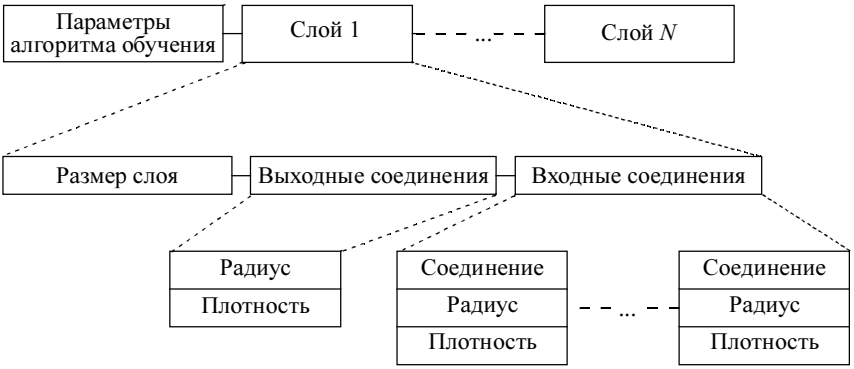


Рис. 4.21. Послойное представление

Оператор мутации предполагает случайное изменение параметров сети или любого из параметров произвольно выбранного слоя. Точки разбиения для операции скрещивания выбираются четко между слоями сети, обмен генетической информацией происходит послойно.

Эта система кодирования применялась для задач выделения углов на изображении, распознавания символов алфавита и аппроксимации [91].

4.5.5. Порождающее кодирование

Для порождающего кодирования характерным является сохранение в генетическом коде не параметров нейронной сети, а правил, по которым генерируются эти параметры. Мотивацией к применению правил является способ формирования биологических систем: для головного мозга количество составляющих нейронов будет на порядки превышать число нуклеотидов в геноме. Эффективность биологического кода основана также на модульной организации и определенной степени фрактальности. Большая часть систем с порождающим кодированием для задания правил построена на принципах, взятых из формальных грамматик.

Грамматика графовой генерации

Одним из первых порождающих способов кодирования нейронных сетей была предложенная Китано в [92] грамматика графовой генерации – разновидность контекстно-свободной грамматики. Продукционные правила грамматики имеют вид:

$$S \rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix}. \qquad (4.52)$$

Левая часть правила – символ грамматики, правая часть представляет собой 2 × 2 матрицу символов из алфавита {A,B,... ,Z,a,b,.. .p, 0,1}. Символы с A по Z являются нетерминальными символами, из которых и составляется генетическая строка. В строке обязательно должен присутствовать стартовый символ S, с которого будет производиться грамматический разбор. Вершины алфавита {a, b,..., p} являются псевдотерминальными и определяют 16 фиксированных правил грамматики, генерирующих все возможные 2 × 2-матрицы из нулей и единиц (рис. 4.22).

Рис. 4.22. Грамматика графовой генерации, согласно Китано

В конце разбора терминальные вершины {0,1} составляют финальную матрицу присутствия/отсутствия связи между соответствующими нейронами, подобно тому, как было приведено в [79] (п. 4.5.4). Топология нейронной сети приводится к виду прямого распространения путем от-

брасывания всех обратных связей. На рис. 4.22 приведен пример разбора и порождения сети. В процессе восстановления нейронной сети возможно появление «мертвых» символов – без правил вывода. Такие символы интерпретируются как нули по правилу

$$0 \to \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{4.53}$$

Если в генетической строке имеются два правила с одинаковыми левыми частями, используется только первое встреченное. Грамматический разбор прекращается после некоторого предопределенного числа шагов. Все нетерминальные символы, присутствующие в матрице, заменяются нулями.

Китано использовал операцию выбора «по колесу рулетки», элитизм (сохранение в следующем поколении части лучших представителей популяции), скрещивание с одной и несколькими точками сечения. Мутации производились путем случайного изменения символов в генетической строке на прочие, с вероятностью от 2 до 30%.

Китано произвел сравнение своего метода с прямым кодированием, предложенным Миллером (п. 4.5.4) для задач 4 × 4 и 8 × 8 кодирования. Результаты предполагают несколько более быструю сходимость и лучшее масштабирование для грамматики графовой генерации. Было исследовано влияние длины генетического кода на результаты для 5, 10, 20 и 40 правил порождения в коде. Было отмечено, что большее число правил дает лучшие результаты в эволюционном процессе.

Аксоновые деревья

В модели, предложенной Нолфи [93 – 94], нейроны кодируются положением точки на плоскости. Отображение информации о нейроне из генетической строки в фактический нейрон производится один к одному. Соединения нейронов в слои производится по специальным правилам, параметры которых сохранены в генетическом коде.

Процесс построения связей между нейронами отображен на рис. 4.23. Нейроны, попавшие в крайнюю левую часть плоскости, рассматриваются как нейроны первого скрытого слоя, соответственно нейроны, попавшие в правую часть, становятся выходными нейронами. Соединения между нейронами определяются путем построения специальных аксоновых деревьев. Аксоновые деревья растут по правилу порождения:

$$F \to F[-F][+F]. \tag{4.54}$$

Как видно, деревья представляют собой фрактальные структуры, основанные на правилах L-системы (п. 4.5.5). Число итераций роста каждого дерева фиксируется (обычно 5) (рис. 4.23, *б*). Длина сегмента дерева и угол ветвления определяются из генетической информации (рис. 4.23, *а*). Соединение считается установленным всякий раз, когда ветвь дерева достигает другого нейрона (рис. 4.23, *в*).

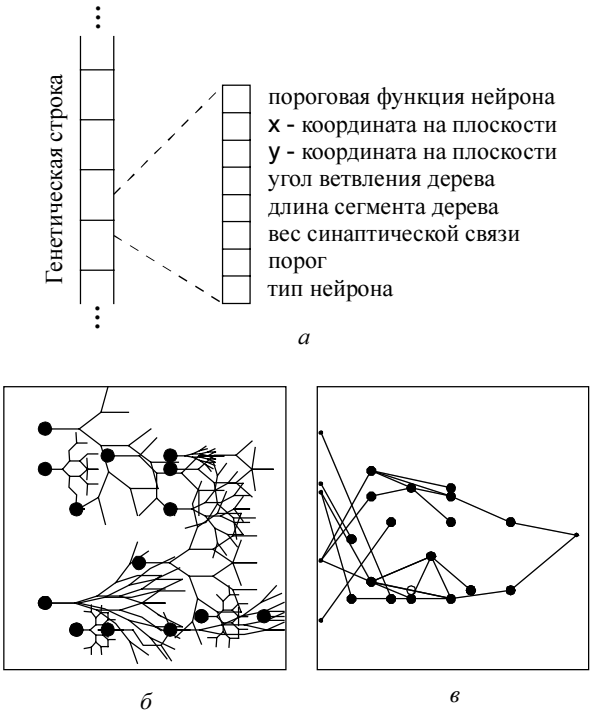


Рис. 4.23. Построение сети, согласно Нольфи

Индексное значение i входного или выходного нейрона сети u_i определяется по значению типа нейрона. Для входов индекс i_1 = тип нейрона mod N_1. Для выходов $i_0 = N - N_0 +$ тип нейрона mod N_0, где N_1 – фактическое число входов, N_0 – фактическое число выходов, N – общее число нейронов.

Значения весов связей сохраняется в генетической строке, поэтому никакого дополнительного алгоритма обучения не было использовано.

Нолфи использовал нейронные сети для исследования и симуляции искусственной жизни, заключающейся в поиске пищи и воды псевдосуществами [94, 95]. Также были исследованы возможности метода для построения автономных роботов [96].

Предложенный метод был в дальнейшем расширен и модифицирован в работе Канджелози [97]. Вместо прямого кодирования каждого нейрона точкой на плоскости в генетический код были добавлены правила деления и миграции для некоторых «псевдоклеток», обеспечивая тем самым создание достаточного числа нейронов. Похожие правила для генерации нейронов сети использованы Фуллмером в [98].

Правила роста нейронов сильно напоминают процесс построения в известной игре «Жизнь». Порождение начинается с «праклетки», которая делится на дочерние клетки. После некоторого числа итераций клетки «созревают» в нейроны. Во всем, что касается построения связей между нейронами, метод остался без изменений.

L-системы

Этот подход был разработан с использованием методологии так называемых *L*-систем, предложенных Линденмайером в 1976 г. как способ моделирования морфогенеза растений. *L*-система представляет собой формальную грамматику, продукционные правила которой используются для генерации строк символов, описывающих состояние некоторой биологической системы. Например, если имеются начальная строка «*F*» и правило:

$$F \to F - F + + F - F \,, \tag{4.55}$$

то после применения (4.55) на втором шаге строится строка

$$F - F + + F - F - F - F + + F - F + + F - F + + F - F - F - F + + F - F$$

Полученная строка может иметь следующую интерпретацию: «*F*» – передвинуть рисующую след «черепашку» вперед, «–» – повернуть под углом налево, «+» – повернуть направо (рис. 4.24).

Если ввести в алфавит два новых символа «[» и «]», можно работать с более натуралистичной моделью растения (рис. 4.25). Первый символ «[» сохраняет позицию и направление черепашки в стеке, второй «]» – восстанавливает верхнее состояние из стека. Перепишем правила:

$$X \to F[-[[X] + X] + F[+FX] - X] \,, \tag{4.56}$$

$$F \to FF \,.$$

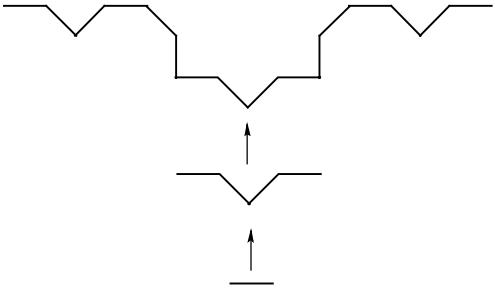


Рис. 4.24. Выполнение двух итераций по (4.55)

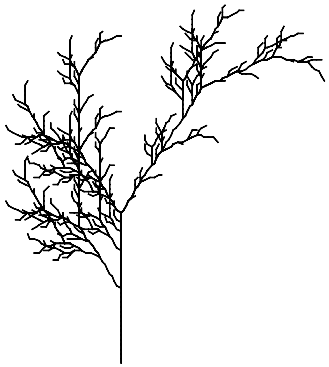


Рис. 4.25. Грамматика (4.56) с углом 33° для пяти итераций

Боер и Куипер предложили использовать для кодирования архитектуры нейронной сети грамматику, являющуюся вариантом *L*-системы [99, 100]. В генетической строке, описывающей нейронную сеть, сохраняются продукционные правила вида

$$L < P > R \to S \,. \tag{4.57}$$

Правила вида (4.57) принадлежат к классу контекстно-зависимых *L*-систем. Символ языка *P* из предыдущей строки преобразуется в строку символов *S* в новой строке, если он имел символ *L* своим левым соседом, а символ *R* своим правым. В один момент времени к текущему предложению последовательно применяются все возможные продукционные правила.

В алгоритме широко используется принцип модульности, взятый из биологических систем. Модуль – это выделенная группа не связанных между собой нейронов сети, каждый из которых соединен с одним и тем же определенным набором узлов.

Все четыре части правила (4.57) представляют собой либо нейроны, либо группы нейронов (модули), закодированные строкой определенного вида. Отдельные нейроны модуля обозначаются буквами алфавита {A,..., Z}. Модуль нейронов выделяют скобками: [C,DE] и обрабатывается в продукционных правилах как единственный символ.

Каждый нейрон или модуль, по умолчанию, считается соединенным со смежными нейронами. Не соединенные друг с другом нейроны или модули разделяются запятой: C, D. Для обозначения связи несмежных

модулей используется число N – скачок через N нейронов (модулей) вперед. Строка A1[C, DE]B означает, что нейрон A имеет соединения с нейроном B, но не имеет связи с модулем [C, DE].

Подобной нотации оказалось достаточно, чтобы задавать правила порождения нейронных сетей прямого распространения.

1) $A \rightarrow BBB$;

2) $B > B \rightarrow [C,D]$;

3) $B \rightarrow C$;

4) $C < D \rightarrow C$;

5) $D > D \rightarrow C1$.

В результате применения продукционных правил 1) –5) к аксиоме *A,* мы поэтапно получаем генетические строки. Итоговая строка [C, C1][C, C]C восстанавливается в сеть, изображенную на рис. 4.26.

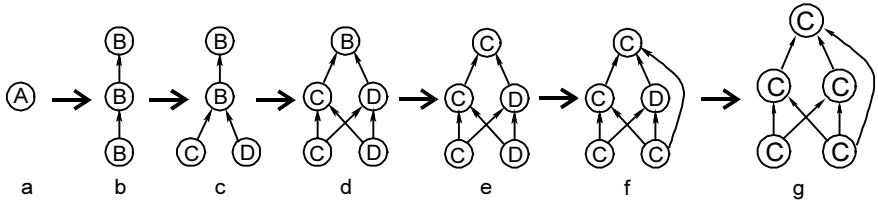


Рис. 4.26. Восстановление нейронной сети с использованием правил 1 – 5. Результирующая строка [C, C1][C, C]C дает сеть (g)

В генетической строке продукционные правила вида (4.57) кодируются как трехбитовые двоичные числа. Эти числа переводятся в символы в алфавите {A,..., G, 1,..., 5, [,],*}, где * – служебный символ разделитель правил, с использованием специальной трансляционной таблицы. Таким образом, генетическая строка представляет собой бинарную строку, к которой применимы классические операции мутации и скрещивания. Для восстановления правил триплеты считываются из строки в три фазы и в обоих направлениях (по аналогии с процессом считывания ДНК). Подобный способ способствует получению большого числа строк, которые не могут быть интерпретированы как правила, поэтому используется методика выявления и вычленения ошибок.

Метод был применен для задачи «исключающего или» и к упрощенной задаче распознавания символов. Исследования данного подхода к кодированию нейронной сети показали, что метод не лишен недостатков [101]. Изначально провозглашенное свойство модульности претер-

пело значительные упрощения при кодировании: модуль может быть связан только с одним или максимум с двумя другими модулями. Так как конструирование сети идет на уровне модулей, каждый из которых допускает произвольно сложную структуру, и нет механизма поощрения развития более простых особей, полученная в результате сеть не исключает перегруженность топологии.

Клеточное кодирование

Один из самых развитых методов кодирования нейронных сетей был предложен Ф. Груо в серии работ [102 – 108] под названием клеточное кодирование. Клеточное кодирование использует грамматическое дерево для воспроизведения процесса развития «клеток» в нейронную сеть. Генетическая строка состоит из набора инструкций, применяющихся к первоначальной «клетке», которая соединена со всеми входами и выходами сети. В процессе выполнения инструкций восстанавливается нейронная сеть большего размера.

Клетка начинает считывать инструкции с вершины дерева операций вида рис. 4.27, *а*. Узлы дерева представляют собой отдельные инструкции. Клетки – результат применения инструкций делятся и модифицируются дальше, путем перемещения считывающей головки вниз по ветвям дерева. Основными инструкциями можно считать:

1) последовательное деление (SEQ), дает в результате две клетки *A* и *B,* где *A* получает все входы исходной клетки и выход, соединенный с *B,* а клетка *B* соответственно соединена с выходами исходной клетки;

2) параллельное деление (PAR), для которого обе дочерние клетки наследуют входы и выходы исходной клетки;

3) завершение чтения (END), эта инструкция останавливает рост клетки, к которой была применена;

4) рекурсивный вызов (REC), производится рекурсивное повторение всех примененных перед этим операций к данной клетке. Рекурсивные вызовы генерируют модульную структуру сети и позволяют создавать крупные нейронные сети при небольшом количестве инструкций.

Кроме перечисленных имеется более десятка других операций как для манипулирования числом и расположением клеток, так и для модификации контекста клетки (весов связей, порога и т.п.). Обработка операции и соответственно рост сети производится до момента, когда все клетки считают инструкцию «конец». На рис. 4.27, *б* приведен пример этапов построения нейронной сети.

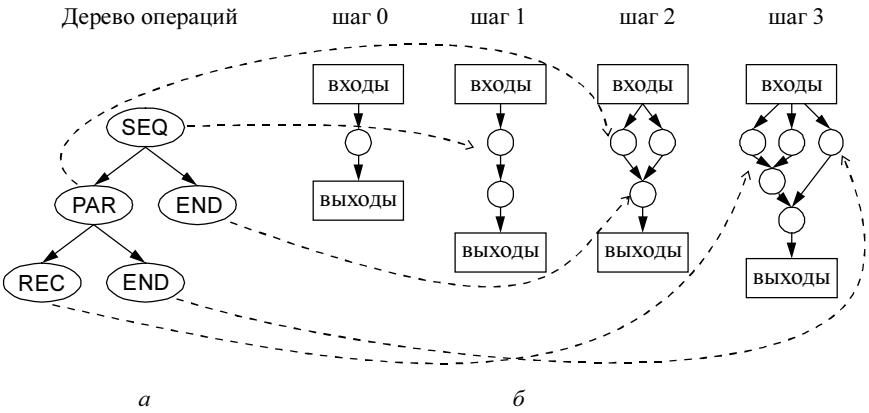


Рис. 4.27. Процесс «клеточного» построения сети

Способы задания генетических операций, применяемых в методе клеточного кодирования, схожи с теми, которые использовал Коза в парадигме генетического программирования (п. 4.5.4). Мутация реализована путем изменения вершин дерева (команд) на команды с аналогичной арностью. Операция скрещивания выполняется путем обмена поддеревьями между родителями.

Груо был сформулирован набор свойств генетического кода, способного эффективно кодировать нейронные сети. Основные свойства следующие:

1. Полнота – способ кодирования считается полным по отношению к классу архитектур, если любая возможная архитектура может быть им представлена;

2. Компактность – способ кодирования А считается более компактным, чем В, если для любой архитектуры сети длина кода А меньше длины кода В;

3. Замкнутость – в отношении класса архитектуры нейронных сетей существует тогда, когда любая возможная строка кода восстанавливается в архитектуру данного класса;

4. Модульность – способ кодирования является модульным, если он допускает повторное ссылочное использования одного участка кода в нескольких местах генетической строки [106].

Клеточное кодирование дало толчок для создания большого количества модификаций и вариаций на данную тему. В работах Фридриха и

Морага [109, 110] было предложено использовать вещественные веса и расширенный набор команд. Публикации других авторов Кодйабахиана и Мейера [111, 112] предлагают геометрически ориентированное клеточное кодирование, в котором добавлена парадигма выращивания связей между нейронами. Люк и Спектор представили предварительные результаты для значительно модифицированного клеточного кодирования [113] – кодирования по краям. Основным изменением является то, что для представления нейронной сети используется «лес» сетей, где отдельные сети имеют возможность ссылаться и вызывать рекурсивно прочие сети. В варианте Талко [114] метод расширен правилами вывода активационных функций нейронов и параметров алгоритма обучения.

В серии работ Хуссейна [115 – 123] был предложен способ кодирования, основанный на грамматике, извлеченной из клеточного кодирования, под именем атрибутная грамматика (NGAGE). В своем очень подробном и обстоятельном обзоре [124] Хуссейн рассматривает сильные и слабые характеристики клеточного кодирования. Основным недостатком он находит его неоднозначность. В частности, идентичные поддеревья разных деревьев не всегда будут восстанавливаться в одинаковые части нейронной сети, благодаря использованию некоторых команд.

Клеточное кодирование было применено для решения ряда задач робототехники [125, 126], а также для задач классификации [127]. Довольно подробное исследование и сравнение результатов с конструктивными методами было проведено Фредриксоном [128] для целого ряда практических задач.

Проделанный обзор методов автоматического построения нейронных сетей позволяет сделать следующие общие выводы. Конструктивные алгоритмы позволяют быстро создавать сети, но жестко заданные правила построения не позволяют совершать отклонения в пространстве архитектур от некоторой окрестности. Это позволяет говорить о конструктивном подходе, как о подходе, реализующем локальный поиск, и сильно подверженный попаданию в локальные минимумы.

Особое место занимают алгоритмы редукции: они позволяют получить неплохие результаты, но предполагают полноту имеющейся модели до начала работы. Иными словами, до начала работы алгоритма уже предполагается наличие некоторой нейронной сети, успешно решающей проблему, для которой делается предположение о возможном ее упрощении. Это условие сильно ограничивает круг возможного применения методов усечения.

Эволюционные алгоритмы несомненно реализуют охват всего пространства архитектур и соответственно реализуют глобальный поиск, с большей вероятностью попадания в глобальный минимум. Но подобный способ организации поиска занимает очень много времени.

4.6. Исследование подходов к построению нейронных сетей

Все способы автоматического построения нейронных сетей можно условно разделить на два направления – конструктивные алгоритмы и методы глобального поиска [23]. И в том и в другом случаях универсальную процедуру для получения нейронной сети с желаемыми свойствами в общем виде можно определить следующим образом. В пространстве всех возможных нейронных сетей по специальной процедуре производится перебор вариантов нейронных сетей, которые последовательно оцениваются. Попытаемся примерно оценить размерность пространства архитектур нейронных сетей.

4.6.1. Размерность пространства поиска

Все множество подклассов A, определяемых выбранной топологией сети, можно оценить по рекурсивной формуле, предложенной в [86].

Допустим, некоторым образом стало известно число скрытых слоев L и число нейронов γ, необходимых для решения некоторой проблемы. Возможное распределений нейронов по слоям можно определить рекурсивной формулой $T(L,\gamma)$. Для простейшего случая имеем

$$T(L,\gamma)=1\text{, если } L=1 \text{ или } \gamma=1. \tag{4.58}$$

Если число нейронов $\gamma < L$, то полагаем $T(L,\gamma) = 0$. В случае $\gamma > L$ мы получим

$$T(L,\gamma)=T(L-1,\gamma-1)+T(L,\gamma-1). \tag{4.59}$$

Формула (4.59) рекурсивно учитывает число возможных разбиений, опираясь на более простую архитектуру с $\gamma - 1$ нейронами. Первое слагаемое в (4.59) учитывает число возможных разбиений для случая, когда новый дополнительный слой содержит единственный дополнительный нейрон. Второй член предполагает, что дополнительный нейрон был добавлен к последнему скрытому слою более простой архитектуры, которая имеет L слоев.

Кроме множества вариантов архитектур в пределах каждого класса $A_i \subset A$ нейронные сети будут характеризоваться еще вектором взвешенных связей между нейронами сети $W = \{w^i, i = 1,..., p\}$. Попытаемся оценить размерность p вектора весов сети W для фиксированного значения числа нейронов сети γ.

В таком случае все возможные топологии будут варьироваться в пределах двух крайних классов архитектур: от $A^{\gamma}_{L=1} \subset A$ – сети только с одним скрытым слоем ($L = 1$) до максимально вытянутой сети $A^{\gamma}_{L=\gamma} \subset A$ по одному нейрону на скрытом слое ($L = \gamma$).

Будем предполагать полную связность всех нейронов сети, то есть каждый слой нейронов, включая входы сети, соединен со всеми последующими от него слоями, включая все выходы сети. Все нейронные сети с меньшим числом весов W могут быть получены из полносвязной сети путем установки соответствующих весов в нулевое значение. Таким образом, для первого варианта максимальное количество весов оценивается соотношением

$$p_{L=1} = \gamma(k + o) + k \cdot o\,. \tag{4.60}$$

Здесь предполагается, что сеть имеет k входов и o выходов.

В случае максимально вытянутой сети размерность увеличивается за счет взаимного соединения слоев:

$$p_{L=\gamma} = \frac{\gamma^2 + \gamma}{2} + \gamma(k + o) + k \cdot o\,. \tag{4.61}$$

Очевидно, что в реальных задачах граничные варианты архитектуры будут встречаться довольно редко. Вместо них будет использован некоторый промежуточный вариант и реальное значение p будет лежать между $p_{L=0}$ и $p_{L=\gamma}$.

Необходимо также подчеркнуть, что даже для известной архитектуры с известным количеством слоев и числом нейронов, с фиксированным числом весов сети, собственно для получения нейронной сети, корректно решающей поставленную задачу, существует необходимость в определении конкретных значений весов сети или, иными словами, в обучении сети.

Таким образом, получено представление о значительной размерности пространства архитектур нейронных сетей, по которому предстоит осуществлять поиск. Исходя из этого, можно сделать вывод о том, что прямой перебор архитектур реализовывать нерационально. Отсюда сле-

дует, что единственным разумным способом построения нейронной сети является использование некоторой эвристики перебора моделей. Именно такой способ автоматического построения нейронных сетей реализуют конструктивный и эволюционный подходы.

4.6.2. Анализ известных направлений

Конструктивные методы, исторически более ранние, реализуют поиск архитектуры путем итеративного изменения сложности выбранной модели. Решение о способе модификации сети принимается в соответствии с некоторым заданным алгоритмом построения. Алгоритм построения задается таким образом, чтобы добавление новых нейронов (связей) на каждом этапе гарантированно уменьшало (в случае усечения не увеличивало) значение ошибки работы сети.

Общий принцип работы подобных алгоритмов носит характер локализации неверно решаемых нейронной сетью примеров обучающей выборки *D,* фиксирование уже построенной сети для правильно решаемых примеров и добавление архитектуры нейронами (связями), корректирующими работу сети для неверно решаемых примеров. Наглядным примером могут являться бинарные алгоритмы (п. 4.4.1), где этот принцип выражен в явном виде. В одном из самых исследованных методов построения нейронных сетей – каскадной корреляции – также можно проследить эту схему: новый нейрон-«кандидат» обучается изолированно от уже построенной сети, причем так, чтобы максимально влиять на остаточное значение ошибки всей сети. Построение завершается, когда ошибка уменьшается до желаемого значения.

Приведенный выше принцип построения модели предполагает некоторую избыточность в структуре готовой нейронной сети. Так, в работах [51, 56] отмечено, что техника «заморозки» весов каскадной корреляции приводит к построению нейронной сети несколько большего размера, чем необходимо. Избыточность неизбежно возникает при конструктивном подходе, при необходимости обеспечения относительной изоляции уже построенной сети и соответственно той части обучающей выборки, для которой верно находится решение, от воздействия нового нейрона (веса). В ряде современных конструктивных алгоритмов: Kogi9, КасПер (п. 4.4.5), делается попытка уйти от этой проблемы.

Большая часть конструктивных алгоритмов жестко фиксирует способ наращивания архитектуры, например добавлением нейронов в единственный скрытый слой сети, или же, наоборот, присоединением

всякий раз нового скрытого слоя с единственным нейроном (п. 4.4.5). Очевидно, что тот или иной фиксированный способ будет обеспечивать уменьшение ошибки в равной степени не для всех типов задач.

Так, в работе Хванга и др. [45] было показано, что каскадная корреляция на некоторых задачах классификации и регрессии будет создавать сети с неоптимальной обобщающей способностью. Хванг объясняет этот результат использованием в алгоритме механизмов каскадирования и корреляции, что влечет за собой слишком зубчатые значения выходов сети.

Конструктивные алгоритмы строят нейронные сети путем итеративного изменения сложности модели. Подобная организация поиска затрагивает, фактически, только смежные классы архитектур A_i в пространстве архитектур A. Иными словами, в таком случае можно только рассчитывать на попадание в ближайший минимум, как правило, локальный. В итоге результаты будут значительно зависеть от выбора начальных параметров алгоритма.

Основным преимуществом конструктивного подхода можно считать небольшое время работы. Это следует напрямую из алгоритма построения, при котором всякий раз имеем дело с нейронной сетью, уже, хотя бы частично, решающей задачу. Каждое дальнейшее усложнение архитектуры только улучшает результат. Еще одним положительным результатом подобного подхода к построению сетей будет являться иерархическая организация архитектур созданных нейронных сетей. В ряде случаев при использовании некоторых эффективных критериев оценок обобщающей способности нейронной сети, для корректного сравнения различных построенных нейронных сетей свойство вложенности будет важным (п. 4.3).

Особое место среди конструктивных алгоритмов занимают алгоритмы редукции сети: они позволяют получить неплохие результаты, но предполагают полноту имеющейся модели до начала работы. Иными словами, до начала работы алгоритма уже предполагается наличие некоторой нейронной сети, успешно решающей проблему, для которой делается предположение о возможном ее упрощении. Очевидно, что не для всех случаев предварительно может существует такая сеть. Это условие сильно ограничивает круг возможного применения методов редукции.

Существенным шагом вперед на пути разработки алгоритмов, строящих нейронные сети, является использования такого механизма

глобального поиска, как эволюционные алгоритмы. Методы глобального поиска, как следует из названия, основаны на поиске в пределах всех возможных моделей в пространстве архитектур. Среди всех топологий нейронных сетей целенаправленно ищется единственная сеть, которая будет формировать наилучшую оценку $\hat{f}(x)$.

Применительно к нейронным сетям генетические алгоритмы дают достаточно эффективный механизм выбора архитектуры. Эволюционный подход обеспечивает глобальный охват пространства поиска и, очевидно, в процесс поиска будут вовлекаться как простые, так и сложные архитектуры. Немаловажным отличием эволюционного подхода от конструктивного можно считать обеспечение охвата всех классов архитектур A, а не только смежных. Поиск по всему пространству дает большие шансы для нахождения глобального минимума. Кроме этого, способ организации работы генетических алгоритмов дает возможность находить решение при произвольно сложном виде оценочной функции приспособленности.

Слабым местом при организации построения нейронных сетей методами эволюционного поиска будет являться весьма значительное время получения результата. Это обусловлено необходимостью на каждом этапе эволюции производить сначала построение, а затем и перебор большого числа индивидуумов, при этом вычисляя значения функции приспособленности для каждого. Именно за счет поддержки популяции индивидуумов – возможных решений поставленной задачи – реализуется больший охват пространства архитектур.

В некоторых реализациях делаются попытки снижения времени работы ГА, например у Китано (п. 4.5.5), в генетической строке ограничивается размер представимой нейронной сети, тем самым искусственно сужается пространство поиска.

Другим популярным способом является распараллеливание эволюционного процесса на мультипроцессорных архитектурах [128]. Генетические алгоритмы очень эффективно могут быть распараллелены: всю популяцию можно произвольным образом разбить на субпопуляции, и вычисление функции приспособленности может вестись независимо во всех субпопуляциях. Вычисление лучшего среди всей популяции индивидуума делается в момент синхронизации данных о победителях в каждой субпопуляции. Этот способ является очень эффективным, но требует соответствующей технической базы.

Проведенные исследования обоих направлений по построению нейронных сетей позволяют говорить о том, что однозначного лидера не существует. Оба приведенных выше подхода обладают и достоинствами и недостатками. Конструктивные алгоритмы за разумное время получают нейронную сеть, слегка отличающуюся от оптимальной. С другой стороны, эволюционный подход к построению обеспечивает, как правило, лучший результат, но при этом затрачивается значительное количество времени.

Было бы интересным сделать попытку объединения лучших сторон обоих подходов: использовать принцип поэтапного построения нейронной сети, присущий конструктивному направлению, одновременно с мощным поисковым механизмом, реализуемым генетическими алгоритмами. В результате может получиться алгоритм с неплохими характеристиками.

4.7. Метод мониторинга динамики изменения ошибки

В данном разделе рассматривается модификация метода динамического добавления нейрона путем усовершенствования мониторинга скорости изменения ошибки [129, 130]. Метод динамического наращивания узлов среди всех конструктивных алгоритмов можно выделить как наиболее интересный в плане общего определения схемы построения (п. 4.4.4). Основным моментом, в котором была произведена модификация, является способ определения скорости изменения ошибки.

Оригинальный метод динамического наращивания предлагает довольно простую схему построения нейронной сети [39]. Процесс поиска архитектуры начинается с нейронной сети с одним нейроном в единственном скрытом слое. В [39] предполагается, что сложность сети не соответствует поставленной задаче, если ошибка сети с числом эпох обучения уменьшается недостаточно быстро. Значение скорости уменьшения ошибки вычисляется по формуле (4.26). При падении скорости уменьшения ошибки ниже заданного порога в скрытый слой добавляется нейрон.

Как видно из (4.26), при вычислении значения скорости ошибки принято рассматривать абсолютное изменение ошибки по отношению к некоторому первоначальному значению ошибки и не учитывать возможность ее кратковременного увеличения вследствие, например, эффекта переобучения.

В варианте, предложенном авторами данной монографии, несколько изменился способ оценки скорости ошибки: принимается во внимание динамика изменения ошибки на всем интервале обучения δ:

$$\frac{\sum_{i=1}^{\delta}\left\{E_{t-\delta_{i+1}} - E_{t-\delta_i}\right\}^2}{\delta} < \Omega\,,\ (t \geq t_0), \delta_i \in [1,\delta]\,. \qquad (4.62)$$

Здесь $E_{t-\delta_{i+1}}$ и $E_{t-\delta_i}$ представляют собой значения ошибок на интервале обучения δ, Ω – некоторая заданная минимально возможная скорость изменения ошибки, при превышении которой принимается решение об изменении сети, δ – количество эпох-циклов обучения сети перед произведением оценки скорости ошибки.

Заметим, что при оценке скорости по (4.62) учитывается возможность наличия периодов временного роста ошибки на интервале δ, причем за счет эффекта усреднения отсекаются локальные провалы производительности на интервале обучения δ. В результате подобного подхода к определению скорости ошибки общая производительность метода динамического наращивания узлов улучшается. За счет оценки общей динамики скорости на значительном интервале обучения δ отпадет необходимость усложнения топологии нейронной сети, вызванная временным, не системным, замедлением скорости уменьшения ошибки. Более того, интервал δ можно теперь сделать большим по длительности.

Помимо изменения способа определения скорости ошибки была предложена иная схема построения нейронной сети. Алгоритм работы метода мониторинга динамики изменения ошибки выглядит следующим образом. Исходной является нейронная сеть прямого распространения с одним скрытым слоем. Обучение производится методом обратного распространения ошибки с инерцией и адаптивной скоростью спуска. После δ эпох обучения производится оценка скорости ошибки по формуле (4.62). При скорости падения ошибки, меньшей некоторого допустимого порога Ω, производится модификация нейронной сети: добавляется несколько нейронов в текущий скрытый слой – единственный на начальном этапе. При достижении некоторого первоначально заданного из эмпирических соображений числа γ_L нейронов в слое создается новый скрытый слой и нейрон добавляется уже в него. При добавлении нового слоя число нейронов в предыдущем активном слое уменьшается в T раз.

Дополнительно было исследовано два варианта работы алгоритма:

- найденные на предыдущем шаге значения весов сети и параметров нейронов «замораживались» и использовались после модификации нейронной сети. Добавившиеся веса связей и пороги нейронов инициировались случайными значениями. Данный вариант применялся только для сети с одним скрытым слоем;

- при добавлении нового скрытого слоя, когда число нейронов на предыдущем слое уменьшалось, веса всех нейронов сбрасывались, т.е. сеть полностью переобучалась.

В качестве тестовой проблемы были взяты данные реальной задачи из области предсказания землетрясений [131]. В работе исследуется зависимость между интенсивностью импульсов электромагнитных полей тектонических разломов и возможностью землетрясения. Непосредственно перед тем, как начинаются заключительные процессы подготовки землетрясения, происходит прекращение движения блоков относительно соседних с ним участков. Следствием этого становится уменьшение интенсивности регистрируемого сигнала, что должно говорить о подготовке к тектоническому возмущению. Результаты практического применения предложенного алгоритма к данному кругу проблем опубликованы в [132].

В данном разделе ограничимся констатацией, которая позволяет сделать следующие выводы:

1) метод динамического добавления нейрона, используемый без применения к более информированной стратегии построения нейронной сети, можно считать эффективным только для малых задач, т.е. требующих построения небольших нейронных сетей. Для задач большей размерности для получения результата, близкого к оптимуму, требуется специальный подбор параметров алгоритма (порог скорости, δ и т.д.);

2) достоинством предлагаемого подхода можно считать то, что он не накладывает никаких ограничений на топологию нейронной сети и способы ее построения, поэтому его можно свободно комбинировать с другими алгоритмами оптимизации нейронных сетей;

3) в рамках дальнейшего усовершенствования приведенного метода мониторинга динамики изменения ошибки возможно использование адаптивно меняющегося значения величины порога Ω в процессе работы алгоритма;

4) несмотря на то, что алгоритм мониторинга динамики изменения ошибки оказался достаточно эффективным, исследования в области со-

четания эволюционного и конструктивного подходов следует вести несколько в ином направлении, чему и посвящены последующие разделы;

5) идея мониторинга динамики изменения ошибки оказалась достаточно продуктивной и использовалась нами в качестве составного элемента развиваемого ниже более общего метода.

4.8. Эволюционное накопление признаков

4.8.1. Предлагаемая организация поиска архитектуры

Рассматривая алгоритмы, реализующие конструктивный и эволюционный подходы, приходим к следующим заключениям. Конструктивный подход позволяет строить нейронные сети за очень малое время, но обеспечивает только локальный поиск по пространству архитектур, заведомо обрекая на попадание в ближайший локальный минимум. Схема построения большинства конструктивных алгоритмов жестко фиксирована по причине необходимости уменьшения ошибки на каждой итерации. Тем самым рост архитектуры в пространстве реализуется только в одном направлении. Эволюционный подход обеспечивает глобальный поиск в пространстве архитектур и охват всех классов архитектур, а не только смежных. Поиск по всему пространству дает больше шансов для нахождения глобального минимума. Основным недостатком эволюционного поиска можно считать очень длительное время нахождения решения.

Практические исследования нейронных сетей показывают, что для многих задач, в условиях зашумленных наборов данных ограниченной размерности, совершенно различные нейронные сети обеспечивают практически одинаковые результаты [24]. Таким образом, при поиске архитектуры нейронной сети не обязательно охватывать все пространство целиком. Для нахождения приемлемого результата достаточно будет реализовать стратегию перемещения по пространству небольшими скачками с подробным поиском в пределах некоторой области.

Предлагается следующая стратегия организации поиска архитектуры. Основополагающим остается принцип, согласно которому сложность нейронной сети в процессе поиска итеративно увеличивается. Изменения, которые необходимо внести в первоначальную модель, определяются при помощи генетических алгоритмов. Для этого нами разработан специальный способ кодировки архитектуры нейронной сети в генетическую строку, учитывающий наличие постоянной части сети.

Формально эта процедура будет выглядеть следующим образом. Имеется некоторая базовая нейронная сеть с архитектурой A. На каждом этапе алгоритма будем получать новую нейронную сеть $\tilde{A}$ путем добавления к сети некоторого количества нейронов ($\gamma_\Delta > 1$). Способ добавления допускает образование как новых слоев L_Δ, так и дополнительных связей p_Δ. Фрагмент Δ будем называть наращивающим блоком нейронной сети. Способ добавления достраивающего блока Δ будем определять при помощи эволюционной процедуры

$$\tilde{A} = A + \Delta . \qquad (4.63)$$

С другой стороны, на данный способ организации поиска можно взглянуть, как на разновидность конструктивного алгоритма. Здесь фактически реализован аналогичный итеративный принцип построения сети, за единственным исключением: вместо жесткого способа построения сети, например каскадированием или по слоям, способ надстройки на каждом этапе будет выбираться произвольно, в зависимости от потребности задачи. Необходимость будет оцениваться специальной эволюционной процедурой, реализующей целенаправленный перебор и оценку вариантов. Более того, условие возможного добавления сразу нескольких нейронов допускает реализацию одновременно и каскадного и послойного наращивания.

Следует заметить, что многие эффективные алгоритмы конструктивного направления для того, чтобы сгладить ограничения жесткого способа построения, в неявном виде реализуют переборную стратегию в неявном виде. Так, во всех модификациях алгоритма каскадной корреляции для выбора нейронов-«кандидатов» используется так называемый пул из нескольких нейронов. Нейроны всего пула поочередно оцениваются, и из всего пула выбирается тот, что будет влиять на ошибку сети больше всех.

Как было показано выше, эволюционная процедура, предлагаемая в данном подходе, имеет специальный способ кодировки архитектуры нейронной сети в генетическую строку, учитывающий наличие постоянной части сети. Фактически это будет обозначать разделение генетической строки на постоянную и варьируемую части. В терминах теоремы шаблонов положительный эффект от подобной организации можно обосновать следующим образом.

Как известно, по мере сходимости эволюционной процедурыалгоритм формирует генетическую строку, отдельные участки которой изменяются все реже и реже, по мере образования строительных шабло-

нов. Такой эффект свидетельствует о близости конечного решения. Постоянная часть генетической строки будет обусловлена конкретной архитектурой нейронной сети, полученной на предыдущей итерации. Из способа построения предполагается, что сформированная подобным образом постоянная часть даст лучшие частичные решения для решаемой нейронной сетью задачи. Можно считать, что постоянная часть нашей генетической строки будет фактически являться строительным шаблоном, зафиксированным принудительно. Таким образом, время сходимости генетической процедуры значительно ускоряется. Необходимо добавить, что при этом также сокращается длина генетической строки, что дополнительно увеличивает скорость работы.

Организация постоянной и переменной частей переложена на разработанный нами метод кодирования нейронной сети в генетическую строку, подробно изложенный в следующем разделе. Заметим также, что постоянная часть не будет содержать в себе полное описание архитектуры базовой сети, а только тех ее элементов, которые существенны для переменной части.

Выделяя постоянную часть в генетическом коде, тем самым реализуем следующую, более естественную концепцию эволюционного процесса, когда основная часть генетической информации через поколения сохраняется неизменной. Классические генетические операции, такие, как скрещивание и мутация, могут создавать потомков, радикально отличающихся от своих родителей по всему набору признаков. Очевидно, что подобные революционные изменения в генотипе не характерны для живой природы. Если на микроуровне быстрая смена генотипа микроорганизмов еще возможна, то на макроуровне некий базовый набор признаков не изменяется или меняется медленнее. Вряд ли можно предположить, что появление наземного животного мира было обусловлено формированием легких у водоплавающих в пределах всего одного-двух поколений. Таким образом, выделение постоянной и переменной частей в генетическом коде можно считать более естественным способом для воссоздания эволюционного процесса получения решения.

4.8.2. Реализация кодирования путями

В немалой степени эффективность генетического поиска будет зависеть от правильности выбора способа кодирования информации об архитектуре нейронной сети в генетическую строку. Более того, организация построения нейронной сети по предложенному в предыдущем

пункте способу налагает дополнительные требования на кодирующую схему. Генетическая информация в строке должна содержать сведения о способе добавления достраивающего блока в основной сети.

Исследования показывают, что порождающее кодирование для ряда задач оказывается значительно менее эффективным [133, 134] и требующим больше времени для достижения результата [78]. Поэтому для предложенной схемы поиска эффективнее было бы использование прямой схемы кодирования.

В данной работе было использовано развитие варианта прямого кодирования, основанного на «путях» передачи сигнала в нейронной сети, предложенного в [135]. Данный вариант прямого кодирования предполагает сохранение в генетическом коде не информации о количестве слоев, нейронов или связях в нейронной сети, а условные «пути» передачи сигнала в сети от входов к выходам.

В виде, предложенном в [135], путь в сети будет определяться списком нейронов, начиная только с любого из входных нейронов и заканчиваясь только одним из выходных нейронов. Никаких ограничений на представление и порядок нейронов внутри такого «пути» не налагается. Оригинальный метод имеет некоторые ограничения: он предполагает ограничение размеров сети сверху и может порождать нейронные сети произвольного вида, не ограничиваясь классом нейронных сетей прямого распространения.

Метод кодировки «путями» авторы подвергли существенным изменениям применительно к поставленной задаче. Все требования к модифицированному построению нейронной сети путями можно изложить следующим формальным образом.

1) До начала формирования генетической строки имеем некоторую базовую нейронную сеть прямого распространения, состоящую из нейронов $\Gamma = \{h_1,..., h_\gamma\}$ и имеющую k входов и o выходов. Зададим некоторое количество строящих нейронов $U = \{h_{\gamma+1},..., h_{\gamma+u}\}$. В формировании «путей» будут участвовать все нейроны $\{\Gamma,U\}$, то есть новые нейроны будут равноправно участвовать при построении «путей» наряду с нейронами базовой сети.

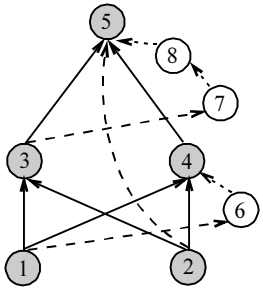
2) Архитектура нейронной сети задается набором множеств – «путей» продвижения сигнала $\{P_i\}$. Путь продвижения сигнала в нейронной сети задается упорядоченным списком нейронов из Γ и U, тем самым моделируя расположение нейронов последовательно по всем слоям, начиная от слоя расположения начального и заканчивая слоем расположения конеч-

ного. Каждый путь *P* может начинаться только с нейрона из набора допустимых стартовых нейронов *S* и может заканчиваться одним нейроном из набора допустимых выходных нейронов *E.* Повторения нейронов в пределах одного «пути» не допускается. Никакого другого ограничения на порядок нейронов внутри пути не накладывается.

3) Возможно несколько способов формирования наборов *S* и *E*. При практической реализации данного способа кодирования авторы остановились на следующем. Набор допустимых стартовых нейронов *S* формируется из всех нейронов базовой нейронной сети *Г*, за исключением выходов сети и последнего скрытого слоя. Набор допустимых выходных нейронов *E* соответственно из всех нейронов базовой сети *Г* за исключением входов сети и первого скрытого слоя. Естественно, что в оба набора не попадают строящие нейроны *U*.

4) Непосредственно в генетической строке нейроны представляются своими индексами. Каждый из путей отделяется друг от друга специальным служебным символом '#'.

Предложенную схему может проиллюстрировать рис. 4.28.



Строящие нейроны $U = \{ (6) (7) (8) \}$

Набор допустимы стартовых нейронов $S = \{ 1\ 2\ 3\ 4 \}$

Набор допустимых выходных нейронов $E = \{ 3\ 4\ 5 \}$

Возможные пути:

{ 1 6 4 5 }
{ 3 7 8 5 }
{ 2 5 }

Генетическая строка:

(варьируемая часть)

#3 7 8 5#	#1 6 4 5#	#2 5#

(Постоянная часть, в эволюции не участвует)

#1 3 5#	#1 4 5#	#2 3 5#	#2 4 5#

Рис. 4.28. Предложенный вариант кодировки путями

Генетические операции

В оригинальном методе к построенному пути применяются следующие генетические операции: оператор скрещивания с двумя точками разбиения и четыре варианта оператора мутации [135]. В модификации метода «путей» для работы с генетическими строками использовались

следующие генетические операции: скрещивание с одной точкой разбиения и два варианта операции мутации. Операция скрещивания, или пересечения, служит для закрепления полезных для решения поставленной задачи свойств генетической строки. Скрещивание выбирает случайным образом точку между путями в генетической строке двух индивидуумов. Результатом будут являться два потомка, составленные из путей, принадлежащих обоим родителям (рис. 4.29).

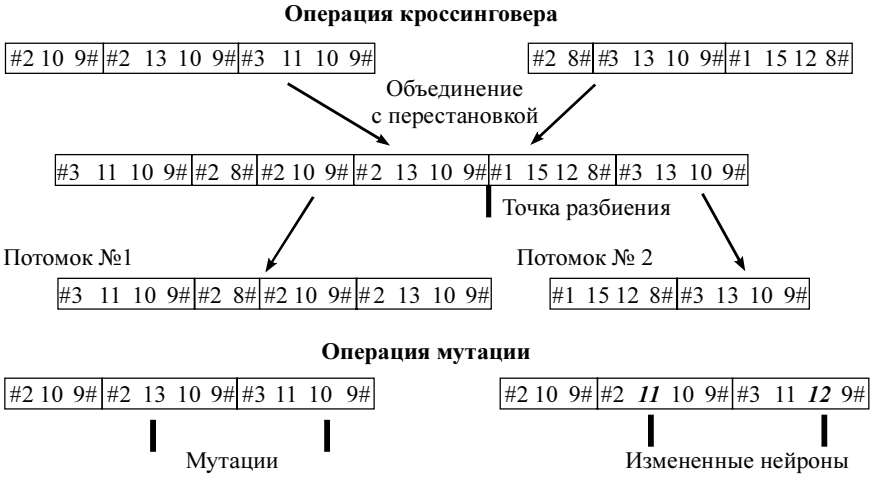


Рис. 4.29. Генетические операции

Как было уже сказано, используется два способа внесения мутаций в генетический код. Первый вариант оператора мутации производит случайным образом замещение нейронов в произвольном числе «путей», составляющих генетическую строку индивидуума. Второй вариант оператора мутации предусматривает создание случайным образом генетической строки полностью. Внесение случайных изменений – мутаций – позволяет производить скачки в пространстве архитектур и выбираться из локальных экстремумов, куда нас может привести поиск оптимальной архитектуры. Необходимо отметить, что применение генетических операций может привести к появлению некорректных архитектур для нейронной сети прямого распространения. Распознавание и исправление генетического кода подобных индивидуумов производится на этапе восстановления нейронной сети из генетического кода.

Новизна и эффективность приведенного подхода заключается в способе организации эволюционного перебора. В отличие от общепринятых подходов к эволюционному построению нейронных сетей на каждом этапе генетический код, определяющий Δ, всегда будет иметь малый размер и соответственно меньший размер пространства поиска. Общепринятые подходы имеют дело с индивидуумами, пытающимися решить сразу всю задачу.

В короткой генетической строке гораздо быстрее образуются строительные блоки, что ускоряет сходимость метода.

4.8.3. Функция приспособленности

В генетических алгоритмах под «приспособленностью» отдельного индивидуума популяции к среде рассматривается его способность к эффективному решению некоторой заданной целевой функции – в нашем случае построения эффективной нейронной сети. Существует множество вариантов задания функции приспособленности для построения нейронных сетей эволюционными методами. Далее предлагается оригинальный вариант определения функции приспособленности. По форме задание функции приспособленности, предложенное авторами, попадает под общий принцип, известный как принцип описания минимальной длины [136]. В исследованиях использовалась функция приспособленности F следующего вида:

$$F = \frac{1}{E_{\text{train}}} + \frac{1}{E_{\text{valid}}} + C + S\,. \tag{4.64}$$

Здесь E_{train} – среднеквадратическая ошибка сети, восстановленной из генетической строки, заданная на обучающей выборке, E_{valid} – среднеквадратическая ошибка сети на проверочной выборке, C – оценка сложности топологии нейронной сети, S – скорость уменьшения ошибки нейронной сети на проверочных данных.

Величина C служит для определения сложности топологии сети и оценивается по следующей эмпирической формуле:

$$C = \begin{Bmatrix} F_1\left(\frac{1}{N} + c_1\frac{1}{\sigma}\right), & \sigma > 0 \\ F_1\left(\frac{1}{N} + c_1\right), & \sigma \le 0 \end{Bmatrix}, \quad \text{где } F_1 = \frac{1}{E_{\text{train}}}. \tag{4.65}$$

Здесь через N обозначено общее число нейронов в сети,

$$\sigma = \left[\frac{1}{N} \sum_{i=1}^{N} (n_i - \overline{n}_i) \right]^{\frac{1}{2}}$$

представляет собой стандартное отклонение и характеризует степень разброса количества нейронов в скрытых слоях сети от некоторого среднего числа нейронов $\overline{n}_i$ по слоям, n_i – число нейронов в скрытом слое *i.* Согласно (4.65), большую величину *C* будут иметь нейронные сети, имеющие малое число нейронов, расположенных максимально равномерно по скрытым слоям. Именно последнее слагаемое в (4.65) поощряет нейронные сети без резких отличий числа нейронов на скрытых слоях.

Скорость уменьшения ошибки сети на проверочных данных *S* определяется в (4.66) и характеризует архитектуру нейронной сети:

$$\frac{\sum_{i=1}^{\delta} \left\{ E_{t-\delta_{i+1}} - E_{t-\delta_i} \right\}^2}{\delta} < \Omega \quad (t \geq t_0), \quad \delta_i \in [1, \delta]. \tag{4.66}$$

Здесь $E_{t-\delta_{i+1}}$ и $E_{t-\delta_i}$ представляют собой значения ошибок на интервале обучения δ, Ω – некоторая заданная минимально возможная скорость изменения ошибки, при превышении которой принимается решение об изменении сети, δ – количество эпох-циклов обучения сети перед произведением оценки скорости ошибки.

Можно считать, что сложность сети не соответствует поставленной задаче, если ошибка с числом эпох обучения уменьшается недостаточно быстро либо не уменьшается совсем. Способ оценки скорости взят из предложенного метода мониторинга динамики изменения ошибки (п. 4.7). При данной оценке скорости ошибки принимается во внимание динамика изменения ошибки на некотором интервале обучения δ. Здесь $E_{t-\delta_i}$ представляют собой значения ошибки на интервале обучения δ в момент времени *i.* Заметим, что при оценке скорости по (4.66) учитывается возможность наличия периодов временного роста ошибки на интервале δ, причем за счет эффекта усреднения отсекаются локальные провалы производительности, вызванные временным замедлением скорости уменьшения ошибки.

4.9. Алгоритм эволюционного наращивания нейронной сети

Соображения по поводу организации поиска нейронной сети, предлагаемого способа кодирования и реализации функции приспособленности реализовываются в виде предлагаемого нового алгоритма для построения нейронных сетей, названного алгоритмом эволюционного наращивания нейронной сети [137 – 142]. Опишем формально схему работы алгоритма.

Основная идея алгоритма эволюционного наращивания нейронной сети заключается в том, чтобы строить нейронную сеть в процессе обучения. Сеть строится на основе исходной сети, состоящей в общем виде только из входов и выходов. Сеть достраивается новыми нейронами и связями как между существующими, так и новыми нейронами. В качестве новых добавочных блоков нейронов используются подсети нейронов, полученные эволюционным способом. Подсеть выбирается эволюционным способом таким образом, чтобы уменьшить общую ошибку нейронной сети на предложенной обучающей выборке. Алгоритм можно условно разбить на два этапа. Основной этап работы алгоритма будет заключаться в реализации механизма поиска наращивающих блоков. Вторым этапом алгоритма будет надстройка нейронной сети и проверка эффективности ее работы. Рассмотрим алгоритм эволюционного наращивания нейронной сети по этапам.

Поиск наращивающих блоков

Предлагаемый в данном алгоритме способ осуществления эволюционного поиска представляет собой основу данного метода построения нейронной сети и выделяет его среди прочих методик поиска топологии нейронных сетей с помощью генетических алгоритмов. Его основной отличительной чертой будет являться то, что в каждый момент в эволюционном переборе будет участвовать лишь часть нейронной сети, в то время как в большинстве методик в генетический поиск вовлекается вся сеть.

На этапе инициализации эволюционного поиска предполагается, что уже имеется некоторая нейронная сеть, которую необходимо улучшить. В начале работы алгоритма берется сеть, состоящая только из входов и выходов – без скрытого слоя нейронов. Задается некоторое максимально возможное количество нейронов, которые разрешено добавить к сети

в результате эволюционного поиска. На основе нейронов этой сети и полученной информации о дополнительных нейронах случайным образом генерируется некоторая начальная популяция. Затем выполняется этап генетического поиска: для каждого индивидуума из текущей популяции производится вычисление оценочной функции приспособленности. Данная функция в нашем случае будет определять эффективность и точность работы нейронной сети, восстановленной из генетической строки, при решении поставленной задачи. Все индивидуумы ранжируются в соответствии со своим значением функции приспособленности.

Для получения нового поколения, генетически более приспособленного к среде, к выбранным индивидуумам применяются генетические операции мутации и скрещивания. После этого опять производится вычисление функции приспособленности и выбор новых кандидатов в родители. Весь процесс эволюционного порождения останавливается, когда не происходит существенного увеличения величины лучшего значения функции приспособленности для популяции. На этом эволюционный этап работы алгоритма заканчивается. Среди лучших представителей популяции выбирается тот, который даст способ наращивания нейронной сети. Пример генетического кода индивидуума победителя, давшего наибольшее улучшение работы нейронной сети, представлен на рис. 4.30.

#2	10	9#	#2	13	10	9#	#3	11	10	9#	#2	8#	#3	13	10	9#	#1	15	12	8 #

Рис. 4.30. Генетическая строка

При извлечении информации из данной генетической строки получаются два наращивающих блока (рис. 4.31, *а*). После нахождения строительных блоков для нейронной сети эволюционный этап считается завершенным.

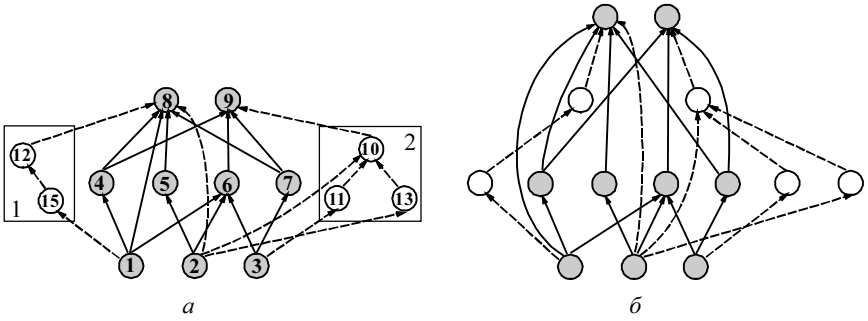
Надстройка сети

Надстройка нейронной сети является очевидным следующим шагом, логично завершающим этап эволюционного поиска. Строительные блоки, полученные на предыдущем шаге, добавляются к сети. На рис. 4.31, *а* показана фаза добавления полученных надстроечных блоков 1 и 2 к текущей нейронной сети. Светлым цветом выделены новые добавленные нейроны, пунктирной линией отмечены новые связи между нейронами.

Хотелось бы отметить, что в принципе добавление новых нейронов при наращивании сети необязательно. Нейронная сеть может развиваться только за счет добавления новых синаптических связей между существующими нейронами.

Генетическая строка:

#2 10 9#	# 2 13 10 9#	# 3 11 10 9#	#2 8#	#3 13 10 9#	#1 15 12 8 #



а *б*

Рис. 4.31. Схема алгоритма эволюционного накопления признаков

На рис. 4.31, *б* показана получившаяся в результате модификации новая нейронная сеть. Полученная нейронная сеть проверяется на обучающей и на проверочной выборках. В случае, если значение ошибки работы сети устраивает, работа всего алгоритма считается завершенной, а текущая сеть считается искомой нейронной сетью оптимальной архитектуры для данной задачи. В противном случае, полученная сеть рассматривается, как база для наращивания и управление передается обратно на эволюционный этап.

Обсуждение алгоритма

Предложенный алгоритм сочетает в себе лучшие качества конструктивного и эволюционного подходов. От конструктивного направления взят принцип, согласно которому сложность нейронной сети в процессе поиска итеративно увеличивается. Изменения, которые необходимо внести в первоначальную модель, определяются при помощи генетических алгоритмов. Для этого разработан специальный способ кодировки архитектуры нейронной сети в генетическую строку, учитывающий наличие постоянной части сети.

В результате представленный алгоритм, за счет применения генетических алгоритмов, позволяет реализовать более широкий охват пространства архитектур по сравнению с конструктивным направлением. Результатом будет улучшенное качество решения. Итеративный принцип увеличения сложности позволяет значительно уменьшить временные затраты сравнительно с большей частью методов построения нейронных сетей эволюционного направления.

Нелинейные модели имеют овражистое пространство решений с большим числом локальных минимумов. Представленный алгоритм позволяет реализовать некоторую непрерывность в пространстве моделей, создавая последовательность вложенных моделей, когда одна нейронная сеть вложена в другую как подсеть меньшей размерности. В ряде работ [21, 24] отмечена необходимость построения иерархически вложенных друг в друга моделей для эффективного использования критериев оценки обобщающей способности модели.

Как способ дальнейшего развития предложенного алгоритма эволюционного наращивания нейронной сети можно использовать распараллеливание эволюционного этапа алгоритма на системах многопроцессорной обработки информации. Сделать это позволяет наличие нескольких индивидуумов в популяции и независимая реализация вычисления фитнесс-функции каждого индивидуума. В дополнение к этому, можно применить к построенной нашим алгоритмом нейронной сети любой из эффективных методов усечения.

ЗАКЛЮЧЕНИЕ

За свою менее чем полувековую историю существования теория нейронных сетей показала свою эффективность в решении многих реальных задач. Нейронные сети успешно применяются для анализа и прогнозирования на финансовом рынке, построения систем медицинской диагностики. Очень широко нейронные сети в настоящее время применяются в робототехнике и в системах управления. Как яркий пример можно привести появление целой плеяды электронных домашних игрушек, которые ведут себя, как настоящие живые существа, могут обучаться и даже приобретать индивидуальные черты характера! Нейронные сети используются везде, где невозможно построить четкий алгоритм решения задачи: при выделении отдельных элементов изображения, распознавании текста, предсказании погоды, сочинении музыки и других областях, где раньше применение машины было немыслимо.

Вычислительный принцип, который нейронные сети позаимствовали у своего биологического прототипа головного мозга, распределенная обработка информации независимыми простейшими вычислителями несомненно является чрезвычайно эффективным. В романе фантаста С. Лема «Непобедимый» описана ситуация, когда мощный космический корабль оказался бессилен перед механистической формой жизни – примитивными электронными «мушками», которые, объединяясь в «облака» в огромных количествах, увеличивали свой интеллектуальный потенциал и могли противостоять любой опасности. Причем повреждение любой части такого «облака» влекло за собой только наращивание числа «мушек» и соответственно общего интеллекта до количества, адекватного угрозе. Тот же принцип объединения простых вычислителей лежит в основе теории нейронных сетей, что делает нейронные сети способными создавать мощные вычислительные комплексы с высокой степенью параллелизма и надежности.

В заключение можно сказать, что история исследований в области нейронных сетей еще довольно коротка и они еще не успели проявить себя в полную силу. Возможно, по мере того, как мы будем больше узнавать об устройстве головного мозга, на основе нейротехнологий будут развиваться новые, еще более необычные способы познания мира.

ЛИТЕРАТУРА

1. *Calvin W.H. and Ojemann G.A.* Conversations with Neil's Brain: The Neural Nature of Thought And Language. – Addison-Wesley, 1994. (http://faculty.washington.edu/wcalvin/).
2. *Busis N.A.* Neurology and Neurosurgery. Neuroscience Internet Guides. (http://www.neuroguide.com/neurogui/).
3. *Розенблатт Ф.* Принципы нейродинамики. Перцептрон и теория механизмов мозга. – М.: Мир, 1965. – 480 с.
4. *Минский М., Пайперт С.* Персептроны. – М.: Мир, 1971.
5. *Rumelhart D.E., Hinton G.E., and Williams R.J.* Learning internal representations by error propogation // Parallel distributed processing: Exploratians in the Microstructures of Cognitron / D.E. Rumelhart, J.I. McCelland, (Eds.). – Cambridge: MIT Press, 1986. – V. 1.
6. *Bishop C.M.* Neural Network for Pattern Recognition. – Oxford: Oxford University Press, 1997. – 482 p.
7. *Дуда Р., Харт П.* Распознавание образов и анализ сцен. – М.: Мир, 1976. – 512 с.
8. *Muller B., Reinhardt J., Strickland M.T.* Neural Networks. – Springer-Verlag, 1995. – 242 p.
9. *Горбань А.Н., Дунин-Барковский В.Л., Кирдин А.Н.* Нейроинформатика. – Новосибирск: Наука, 1998. – 296 с. (http://www.neuropower.de/rus/).
10. *Gorban A.N. and Wunsch D.C.* The General Approximation Theorem // Proceedings of Intern. Joint Conf. on Neural Networks'98. – 1998.
11. *Иванов В.В., Пурэвдорж Б., Пузырин И.В.* Методы второго порядка для обучения многослойного перцептрона // Математическое моделирование. – 1998. – Т. 10. – № 3. – С. 117 – 124.
12. *Hertz J., Krogh A., Palmer R.* Wstep do teorii obliczen neuronowych. Wyd. II. – Warszawa: WNT, 1995.
13. *Уоссермен Ф.* Нейрокомпъютерная техника: теория и практика. – 1992. – 184 с. (http://www.neuropower.de/rus/).
14. *Короткий С.* Нейронные сети: обучение без учителя. – 1998. (http://www.orc.ru/stasson/n3.zip).
15. *Терехов С.А.* Лекции по теории и приложениям искусственных нейронных сетей. – Снежинск: ВНИИТФ, 1994. (http://www.vniitf.ru/ nimfa/).
16. *Короткий С.* Нейронные сети Хопфилда и Хэмминга. – 1998. (http://www.orc.ru/stasson/n4.zip).
17. *Camargo F.A.* Learning Algorithms in Neural Networks // Tech. Rep. CUCS-062-90, Columbia University, NY, 10027, December 1990. (ftp://archive.cis. ohio-state.edu/pub/neuroprose/Camargo.learning.ps).

18. *Sjoberg J. and Ljung L.* Overtraining, Regularization, and Searching for Minimum in Neural Networks. // Tech.Rep., Department of Electrical Engineering, Linkoping University, 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/sjoberg.overtraining.ps. gz).
19. *Amari S., Murata N. and Muller K.R.* Asymptotic Statistical Theory of Overtraining and Cross-Validation // Tech.Rep. METR-95-06, Univ. of Tokyo, 1995. (ftp:// archive.cis.ohio-state. edu/pub/neur opr ose/amari. overtraining .ps. gz).
20. *Monasson R. and Zecchina R.* Learning and Generalization Theories of Large Committee-Machines // Tech. Rep. I-10129, Politecnico di Torino, 1995. (ftp://archive.cis. ohio-state.edu/pub/neuroprose/zecchina.committee.ps.gz).
21. *Murata N., Yoshizawa S., and Amari S.* Network Information Criterion – Determining the number of hidden units for an Artificial Neural Network model // IEEE Transactions on Neural Networks. – November 1994. – V. 5. – No. 6. – P. 865 – 872. (http://www. islab.brain.riken.go.jp/ mura/paper/mura94nic.ps.gz).
22. *Barron A.* Predicted squared error: a criterion for automatic model selection. – NY, 1984.
23. *Ripley B.D.* Pattern Recognition and Neural Networks. – Cambridge: Cambridge University Press, 1997. – 403 p.
24. *Moody J.E. and Utans J.* Architecture selection strategies for neural networks: application to corporate bond rating prediction. – Wiley, 1995. – P. 277 – 300. (ftp://cse. ogi.edu/pub/tech-reports/1994/94-036.ps.gz).
25. *Lawrence S., C. Lee Giles, and Ah Chung Tsoi.* What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backprop-agation // Tech. Rep. UMIACS-TR-96-22 and CS-TR-3617, Univ. of Maryland, 1996.
26. *Utans J. and Moody J.* Selecting Neural Network Architectures Via the Prediction Risk: Application to Corporate Bond Rating Prediction // Tech. Rep., Yale Univ., 1991. (ftp://archive.cis.ohio-state. edu/pub/neur opr ose/utans. bondrating.ps. gz).
27. *Moody J.E.* Note on generalization, regularization and architecture selection in nonlinear learning systems // IEEE Computer Society Press, 1991. – P. 1 – 10. (ftp:// neural.cse.ogi.edu/pub/neural/papers/ moody91 .generalize.ps.Z).
28. *Moody J.E.* The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. – 1992. – P. 847 – 854. (ftp://neural.cse. ogi.edu/pub/neural/papers/moody9 1.peffective.ps.Z).
29. *Murata N., Yoshizawa S., and Amari S.* Learning Curves, Model Selection and Complexity of Neural Networks // M.Kaufmann, San Mateo, CA, 1993. – V. 5. – P. 607 – 614, (http://www.islab.brain.rik-en.go.jp/mura/paper/mura93nips92. ps.gz).
30. *Mezard M., and Nadal J.P.* Learning in feedforward layered networks: The Tiling algorithm // Journal of Physics. – 1989. – V. A22. – P. 2191 – 2203,
31. *Bodenhausen U.* Automatic Structuring of Neural Networks for Spatio-Temporal Real-World Applications // PhD thesis, der Fakultat fur Informatik der Universitat Karlsruhe, 1994. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/bodenhausen.thesis. ps.gz).
32. *Frean M.* The Upstart Algorithm: A Method for Constructing and Training Feed-Forward Neural Networks // Tech. Rep. 89/469, Edinburgh Univ., 1989. (ftp:// archive.cis.ohio-state.edu/pub/neuroprose/ frean.upstart.ps.gz).

33. *Thimm G. and Fiesler E.* Two Neural Network Construction Methods // Neural Processing Letters. – 1997. – V. 6. – P. 25 – 31, (http://www.wkap.nl/issuetoc. htm/).
34. *Sankar A. and Mammone R.J.* Speaker Independent Vowel Recognition Using Neural Tree Networks. // Proceedings of Intern. Joint Conf. on Neural Networks, Seattle, 1991. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/sankar.ijcnn9 11.ps.gz).
35. *Wen W.X., Jennings A.? and Liu H.* Learning a Neural Tree. // Proceedings of Intern. Joint Conf. on Neural Networks'92, Beijing, China, 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/wen.sgnt-learn.ps.gz).
36. *Wen W.X., Jennings A., Liu H.? and Pang V.* Some Performance Comparisons for Self-Generating Neural Tree. // Proceedings of Intern. Joint Conf. on Neural Networks'92, Beijing, China, 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/wen.sgnt-learn.ps.gz).
37. *Wen W.X., Pang V., and Jennings A.* A Comparative Study Between SGNT and SONN. IIAГ'92, Hobart, Australia, Nov 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/wen.sgnt-learn .ps. gz).
38. *Torres-Moreno J.M.* Apprantissage et generalisation par des reseaux de neurones: etude de noveaux algorithmes constructifs // Ph.D.thesis, de l'Institut National Polytechique de Grenoble, 1997. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/torres.thesis.ps.Z).
39. *Ash T.* Dynamic Node Creation in Back-Propagation Networks // Connection Science. – 1989. – V. 1.
40. *Fahlman S.E. and Lebiere C.* The cascade-correlation learning architecture // In Advances in Neural Information Processing II. – 1990. – P. 524 – 532, (ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.cascor-tr. ps.gz).
41. *Fahlman S.E.* An Empirical Study of Learning Speed in Back-Propagation Networks // Tech. Rep. CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, 1988. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.quickprop-tr.ps.gz).
42. *Yang J. and Honavar V.* Experiments with the Cascade-Correlation Algorithm // Tech. Rep. 91-16, Department of Computer Science, Iowa State University, 1991.(ftp://archive.cis.ohio-state.edu/pub/neuroprose/zecchina.committee.ps.gz).
43. *Squires C.S., Jr. Jude, W. Shavlik.* Experimental Analysis of Aspects of the Cascade-Correlation Learning Architecture // Neural Networks. – 1991. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/squires.cascor.ps.gz).
44. *Prechelt L.* Investigation of the CasCor Family of Learning Algorithms // Neural Networks. – May 1997. – V. 10. – No. 5. – P. 885 – 896. (ftp://ftp.ira.uka.de/pub/neuron/neurnetw97.ps.gz).
45. *Jenq-Neng Hwang, Shih-Shien You, Shyh-Rong Lay, and I-Chang Jou.* What's Wrong with A Cascaded Correlation Learning Network: A Projection Pursuit Learning Perspective // IEEE Trans. Neural Networks. – 1996. – V. 7. – No. 2. – P. 278 – 289. (ftp://archive.cis.ohio-state. edu/pub/neur opr ose/hwang. cclppl. ps. gz).
46. *Shultz T.R. and Elman J.L.* Analyzing Cross Connected Networks // Advances in Neural Information Processing Systems. – 1990. – V. 6. – P. 1117 – 1124. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/shultz.cross.ps.gz).

47. *Fahlman S.E.* The Recurrent Cascade-Correlation Architecture // Tech. Rep. CMU-CS-91-100, School of Computer Science, Carnegie Mellon University, 1991. (ftp://archive.cis.ohio-state. edu/pub/neuroprose/fahlman.rcc.ps.gz).
48. *Hoehfeld M. and Fahlman S.E.* Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm.II Tech.Rep. CMU-CS-91-130, School of Computer Science,Carnegie Mellon University, 1991. (ftp://archive.cis.ohio- state.edu/pub/ neuroprose/ hoehfeld.precision.ps. gz).
49. *Klagges H. and Soegtrop M.* Limited Fan-in Random Wired Cascade – Correlation, 1991. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/klagges.rndwired-cascor. ps.gz).
50. *Phatak D.S. and Koren I.* Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture // Tech. Rep. TR-92-CSE-27, ECE Dept., UMASS, Amherst, 1994. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/phatak. layered-cascor.ps.gz).
51. *Sjogaard S.* A Conceptual Approach to Generalization in Dynamic Neural Networks // PhD thesis, Computer Science Department, Aarhus University, 1991. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/sjogaard.concept.ps.gz).
52. *Simon N., Kerckhoffs E., and Corporaal H.* Variations on the Cascade-Correlation Learning Architecture for Fast Convergence // Neural Network World. – 1992. – V. 2. – P. 497 – 510.
53. *Simon N.* Constructive Supervised Learning Algorithms for Artificial Neural Networks // PhD thesis, Delft University of Technology, Faculty of Electrical Engineering, 1993. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/simon.thesis. ps.Z).
54. *Treadgold N.K. and Gedeon T.D.* Exploring Constructive Cascade Networks // IEEE Transactions on Neural Networks, 1999. (http://www.cse.unsw.edu.au/nickt/doc/ acasper.ps).
55. *Treadgold N.K and Gedeon T.D.* Exploring Architecture Variations in Constructive Cascade Networks // Proc. Int. Joint Conf. on Neural Networks, Anchorage, 1998. – P. 343 – 348. (http://www.cse.unsw. edu.au/nickt/doc/tower.ps).
56. *Treadgold N.K. and Gedeon T.D.* A Cascade Network Algorithm Employing Progressive RPROP. // Int. Work Conf. on Artificial and Natural Neural Networks, Lanzarote, 1997. – P. 733 – 742. (http://www.cse.unsw.edu.au/nickt/doc/casper.ps).
57. *Treadgold N.K. and Gedeon T.D.* Extending and Bench marking the CasPer Algorithm // Australian Conference on Artificial Intelligence, Perth, 1997. – P. 398 – 406. (http://www.cse.unsw.edu.au/nickt/doc/casperpclass.ps).
58. *Treadgold N.K. and Gedeon T.D.* Extending CasPer: A Regression Survey. // Int. Conf. on Neural Information Processing, Dunedin, 1997. – P. 310 – 313. (http:// www.cse.unsw.edu.au/nickt/doc/casperpreg.ps).
59. *Mozer M.C., Smolensky P.* Skeletonization: a technique for trimming the fat from a network via relevance assessment // Advances in Neural Information Processing Systems. – 1989. – V. 1. – P. 107 – 115.
60. *Горбань А.Н.* Обучение нейронных сетей. – М.: СП «ParaGraph», СССР – США, 1990. – 160 с.
61. *Еремин Д.И.* Контрастирование. II Нейропрограммы / Под ред. А.Н. Горбаня. – Красноярск: Изд-во КГТУ, 1994. – С. 88 – 108.

62. *Le Cun Y., Denker J.S., and Solla S.A.* Optimal Brain Damage // Advances in Neural Information Processing Systems II (Denver1989). – 1990. – P. 598 – 605.
63. *Hassibi B. and Stork D.G.* Second Order Derivatives for Network Pruning: Optimal Brain Surgeon // Neural Information Processing Systems. – 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/stork.obs.ps.gz).
64. *Pedersen M.W., Hansen L.K., and Larsen J.* Pruning with Generalization Based Weight Saliences: OBD, OBS., 1994. (ftp://archive.cis.ohio-state.edu/pub/neuroprose /pedersen.pruning.ps.gz).
65. *Spears W.M., A. De Jong, Back T., et al.* An Overview of Evolutionary Computation // Proceedings of the 1993 European Conference on Machine Learning, 1993. (http://www.aic.nrl.navy.mil/spears/papers/ecml93.ps).
66. *Holland J.H.* Adaptation in Natural and Artificial Systems. Univ. of Michigan Press., Second Ed. 1992, MA: The MIT Press edition, 1975.
67. *Дмитрович А.И.* Интеллектуальные информационные системы. Тетрасистемс. – Минск, 1997. – 367 с.
68. *Hussain T.S.* An Introduction to Evolutionary Computation // Tech. Rep., CITO Researcher Retreat, Ontario, 1998. (http ://www. cs. queensu. ca/home/hussain/).
69. *Fjalldal J.B.* Evolving Neural Network Controllers Using Genetic Algorithms with Variable Length Genotypes // Tech. Rep., School of Cognitive and Computing Sciences at the Univ. Of Sussex, March 1999. (http://www.cogs.susx.ac.uk/users/johannf/report/report.html).
70. *Koza J.R.* Survey Of Genetic Algorithms And Genetic Programming. II Tech. Rep., Computer Science Department Margaret Jacks Hall Stanford University, 1995. (http://smi-web.stanford.edu/people/koza/).
71. *Goldberg D.E.* Genetic algorithms in search, optimization and machine learning. – Addison-Wesley, Reading, 1989.
72. *Whitley D.* The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best // Proceedings of the 3rd International Conference on Genetic Algorithms and their applications (ICGA). – 1989. – P. 116 – 121,
73. *Hussain T.S.* Methods of Combining Neural Networks and Genetic Algorithms. // Tech. Rep., ITRC/TRIO Researcher Retreat, Ontario, May 1997. (http://www.cs.queensu.ca/home/hussain/).
74. *Xin Yao.* A Review of Evolutionary Artificial Neural Networks // International Journal of Intelligent Systems, 1991. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/yao.eann.ps.gz).
75. *Branke J.* Evolutionary Algorithms for Neural Network Design and Training // 1st Nordic Workshop on Genetic Algorithms and its Applications, 1995. (ftp://ftp.aifb.uni-karlsruhe.de/pub/jbr/Vaasa.ps.gz).
76. *Boers E.J.W., Borst M.V., and Sprinkhuizen-Kuyper I.G.* Evolving Artificial Neural Networks Using the «Baldwin Effect» // Tech. Rep. TR95-14, Computer Science Department, Leiden Univ., 1995. (ftp://ftp.wi.leidenuniv.nl/pub/CS/ MScTheses/tr95-14.ps).
77. *Koehn Ph.* Combining Genetic Algorithms and Neural Networks: The Encoding Problem // PhD thesis, The University of Tennessee, Knoxville, 1994. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/koehn.encoding.ps.gz).

78. *Gruau F., Whitley D., and Pyeatt L.* A comparison between cellular encoding and direct encoding for genetic neural networks // Proceedings of the First Genetic Programming Conference. – 1996. – P. 81 – 89.
79. *Miller G., Todd P. and Hegde S.* Designing neural networks using genetic algorithms // Proceedings of the 3rd International Conference on Genetic Algorithms and their applications (ICGA). – 1989. – P. 379 – 384.
80. *Marti L.* Genetically Generated Neural Networks I: Representational Effects // Tech. Rep. CAS/CNS-TR-92-014, Boston University, Center for Adaptive Systems, 1992.
81. *Marti L.* Genetically Generated Neural Networks II: Searching for an Optimal Representation // Tech. Rep. CAS/CNS-TR-92-015, Boston University, Center for Adaptive Systems, 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/marti.ga2. ps.gz).
82. *Belew R.K., McInerney J., and Schraudolph N.N.* Evolving Networks: Using the Genetic Algorithm with Connectionist Learning // Tech. Rep. CS90-174, Computer Science Dept. Univ. California at San Diego, 1990. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/belew.evol-net.ps.gz).
83. *Dasgupta D.* Evolving N euro-Controllers for a Dynamic System Using Structured Genetic Algorithms // Applied Intelligence. – 1998. – V. 8. – P. 113 – 121. (http://www.wkap.nl/issuetoc.htm/).
84. *Korning P.G.* Training Neural Networks by means of Genetic Algorithms Working on Very Long Chromosomes // PhD thesis, Aarhus University Ny Munkegade, 1997. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/korning.nnga. ps.gz).
85. *Schiffmann W., Joost M., and Werner R.* Performance Evaluation of Evolutionarily Created Neural Network Topologies // Proc. of Parallel Problem Solving from Nature, Lect. Notes in Computer Science, 1991, Schwefel, H.P. and Maenner, R., Springer. – P. 274 – 283, 1991. (http://www.uni-koblenz.de/evol/mertenpublications.html).
86. *Schiffmann W., Joost M., and Werner R.* Synthesis and Performance Analysis of Multilayer Neural Network Architectures // Tech. Rep. 16/1992, University of Koblenz, Institute fur Physics, 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose /schiff. gann .ps. gz).
87. *Schiffmann W., Joost M., and Werner R.* Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons // Proc. of Artificial Neural Networks and Genetic Algorithms, Innsbruck 1993, Albrecht et al. – Springer Verlag, 1993. – P. 675 – 682. (http://www.uni-koblenz.de/evol/mertenpublications.html).
88. *Figueira Pujol J.C. and Poli R.* Evolving the Topology and the Weights of Neural Networks Using a Dual Representation // Applied Intelligence. – 1998. – V. 8. – P. 73 – 84. (http://www.wkap.nl/issuetoc.htm/).
89. *Koza J.R. and Rice J.P.* Genetic Generation of Both the Weight and Architecture for a Neural Network // Proceedings of the International Joint Conference on Neural Networks. – 1991. – V. II. – P. 397 – 404.
90. *Wong F.* Genetically Optimized Neural Networks // NIBS Technical Report, TR-940216, 1994.(ftp://archive.cis.ohio-state.edu/pub/neuroprose/wong.nnga.ps.gz).
91. *Mandischer M.* Representation and Evolution of Neural Networks // Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms. – 1993. – P. 643 – 649.

92. *Kitano H.* Designing neural network using genetic algorithm with graph generation system // Complex Systems. – 1990. – V. 4. – P. 461 – 476.
93. *Nolfi S. and Parisi D.* Growing Neural Network // Artificial life. – June 1992. – V. III. – P. 16. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/nolfi.growing. ps.gz).
94. *Nolfi S. and Parisi D.* Self-selection of input stimuli for improving performance. // Tech. Rep., Institute of Psychology, CNR, Italy, 1992. (ftp://archive.cis.ohio-state. edu/pub/neuroprose/nolfi.self-sel.ps.gz).
95. *Nolfi S., Miglino O., and Parisi D.* Phenotypic Plasticity in Evolving Neural Networks // Tech. Rep. PCIA-94-05, Dpt. of Cognitive Processes and Artificial Intelligence, Italy, May 1994. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/nolfi.plastic.ps.gz).
96. *Nolfi S., Floreano D., Miglino O., and Mondada F.* How to evolve autonomous robots: different approaches in evolutionary robotics // Tech. Rep. PCIA-94-03, Dpt. of Cognitive Processes and Artificial Intelligence, Italy, May 1994. (ftp://archive.cis. ohio-state.edu/pub/ neuroprose/nolfi.erobot.ps.gz).
97. *Cangelosi A., Parisi D., and Nolfi S.* Cell division and migration in a genotype for neural networks // Network: Computation in Neural Systems. – 1993. – V. 5. – P. 497 – 515.
98. *Fullmer B. and Miikkulainen R.* Using Marker-Based Genetic Encoding Of Neural Networks To Evolve Finite-State Behaviour // Proceedings of the First European Conference on Artificial Life (ECAL-91), Paris, 1991.
99. *Boers E. and Kuiper H.* Biological Metaphors and the Design of Modular Artificial Neural Networks // PhD thesis, Departments of Computer Science and Experimental and Theoretical Psychology at Leiden University, the Netherlands, 1992. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/boers.biological-metaphors. ps.gz).
100. *Boers E., Kuiper H., Happel B., and Sprinkhuizen-Kuyper I.* Designing Modular Artificial Neural Networks // Tech. Rep. 93-24, Department of Computer Science Leiden University, The Netherlands, 1993. (ftp://ftp.wi.leidenuniv.nl:21/pub/CS/MScTheses/ tr93-24.ps.gz).
101. *Тютерев В.В., Шевелев О.Г.* Использование L-грамматик для определения нейронной сети оптимальной топологии методом генетических алгоритмов // IIIV межвузовская конференция студентов, аспирантов и молодых ученых «Наука и образование»: Тез. докл. – Томск: ТГПУ, 2000.
102. *Gruau F.* Cellular Encoding of Genetic Neural Networks // Tech. Rep. 92-21, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, 1992. (http://www.cwi.nl/gruau/gruau/RR92-21.ps.Z).
103. *Gruau F. and Whitley D.* Adding Learning to the Cellular Developmental Process: A Comparative Study // Tech. Rep. RR93-04, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, 1993. (http://www.cwi.nl/gruau/ gruau/RR93-04.ps.Z).
104. *Gruau F.* Automatic definition of sub-neural networks // Tech. Rep. RR94-28, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, 1994. (http://www.cwi.nl/gruau/gruau/RR94-28.ps.Z).
105. *Gruau F.* Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm // PhD thesis, Ecole Normale Superieure de Lyon, 1994. (ftp://lip.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD94-01-E.ps.Z).

106. *Hussain T.S.* Modularity Within Neural Networks // PhD thesis, Department of Computing and Information Sciences, 1995. (http://www.cs.queensu.ca/home/ hussain/).
107. *Gruau F.* Automatic Definition of Modular Neural Networks // Adaptive Behavior. – 1995. – V. 3. – P. 151 – 183. (http://www.cwi.nl/gruau/gruau/AB. ps.Z).
108. *Gruau F.* Genetic Programming of Neural Networks: Theory and Practice // Intelligen Hybrid Systems. – 1995. – P. 245 – 271. (http://www.cwi.nl/gruau/gruau/ aigp94.ps.Z).
109. *Friedrich C.M. and Moraga C.* Using Genetic Engineering to Find Modular Structures and Activation Functions for Architectures of Artificial Neural Networks // Computational Intelligence, Theory and Applications. – 1997. – P. 150 – 161. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/friedrich.nnarch-activation.ps.gz).
110. *Friedrich C.M. and Moraga C.* An Evolutionary Method to Find Good Building-Blocks for Architectures of Artificial Neural Networks // Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '96); Granada, Spain (1996). – 1996. – P. 951 – 956.
111. *Kodjabachian J. and Meyer J.A.* Evolution and deveopment of control architectures in animats // Robotics and Autonomous Systems. – 1995. – V. 16. – P. 161 – 182. (www.biologie.ens.fr/perso/meyer/ publications.html).
112. *Kodjabachian J. and Meyer J.A.* Evolution and development of modular control architectures for 1-D locomotion in six-legged animats // (неопубликована). – 1997. (www.biologie.ens.fr/perso/meyer/publications. html).
113. *Luke S. and Spector L.* Evolving graphs and networks with edge encoding: Preliminary report // Late-Breaking Papers of the Genetic Programming'96 Conference, 1997. (www.cs.umd.edu/seanl/papers/graph-paper.ps).
114. *Talko B.* A Rule-Based Approach for Constructing Neural Networks Using Genetic Programming // PhD thesis, Univ. of Melbourne, Australia, 1999. (http://www.cs.mu. oz.au/research/vislab/).
115. *Browse R.A. Hussain T.S., and Smillie M.B.* Using Attribute Grammars for the Genetic Selection of Backpropagation Networks for Character Recognition // Proceedings of Applications of Artificial Neural Networks in Image processing IV (January 25 – 28, San Jose, CA), 1999. (http://www.cs.queensu.ca/home/ hussain/).
116. *Hussain T.S. and Browse R.A.* Genetic Encoding of Neural Network Processing and Control // Accepted for publication in Graduate Student Workshop of GECCO99: Genetic and Evolutionary Computation Conference (July 13 – 17, Orlando, FL), 1999. (http://www.cs.queensu.ca/home/hussain/).
117. *Hussain T.S. and Browse R.A.* Genetic Operators with Dynamic Biases That Operate on Attribute Grammar Representations of Neural Networks // Birds-of-a-Feather Workshop on Advanced Grammar Techniques within Genetic Programming and Evolutionary Computation, held at GECCO99, Orlando, FL, 1999. (http://www.cs. queensu.ca/home/hussain/).
118. *Hussain T.S. and Browse R.A.* Network Generating Attribute Grammar Encoding // Proceedings of the 1998 IEEE International Joint Conference on Neural Networks (May 4 – 9, Anchorage, Alaska). – 1998. – V. 1. – P. 431 – 436. (http://www.cs. queensu.ca/home/hussain/).

119. *Hussain T.S. and Browse R.A.* Genetic Encoding of Neural Networks Using Attribute Grammars // Tech.Rep., CITO Researcher Retreat, Ontario, 1998. (http://www.cs.queensu.ca/home/hussain/).
120. *Hussain T.S. and Browse R.A.* Using Attribute Grammars for Genetic Encoding of Neural Networks and Syntactic Constraint of Genetic Programming // Twelfth Canadian Conference on Artificial Intelligence: Workshop on Evolutionary Computation, V. June 17, Vancouver, BC, 1998. (http://www.cs.queensu.ca/home/hussain/).
121. *Hussain T.S. and Browse R.A.* Basic Properties of Attribute Grammar Encoding. // Late Breaking Papers at the Genetic Programming 1998 Conference (July 22 – 25, Madison, Wisconsin). – 1998. – P. 256. (http://www.cs.queensu. ca/home/hussain/).
122. *Hussain T.S. and Browse R.A.* Including Control Architecture in Attribute Grammar Specifications of Feedforward Neural Networks // Proceedings of the 1998 Joint Conference on Information Sciences: Second International Workshop on Frontiers in Evolutionary Algorithms (October 23 – 28, Research Triangle Park, North Carolina). – 1998. – V. 2. – P. 432 – 436. (http://www.cs.queensu.ca/home/hussain/).
123. *Hussain T.S.* Network Generating Attribute Grammar Encoding // Tech. Rep., Queen's University, Ontario, Canada, March 1998. (http://www.cs.queensu.ca/home/hussain/).
124. *Hussain T.S.* Cellular Encoding: Review and Critique // Tech. Rep., Queen's University, July 19, 1997. (http://www.cs.queensu.ca/home/hussain/).
125. *Gruau F.* Cellular Encoding for interactive Robotics // Tech. Rep. 425, Sussex University, School of Cognitive and Computing Science, 1996. (ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp425.ps.Z).
126. *Gruau F. and Quatramaran.* Cellular encoding for interactive evolutionary robotics // ECAL97, 1997. (http://www.cwi.nl/gruau/gruau/e.ps.gz).
127. *Dellaert F. and Vandewalle J.* Automatic Design of Cellular Neural Networks by Means of Genetic Algorithms: Finding a Feature Detector // Tech.Rep., Dept. of Computer Engineering and Science Case Western Reserve University, Cleveland, Dept. of Electrical Engineering Katholieke Universiteit, Belgium, 1996.
128. *Soegtrop M. and Klagges H.* A Massively Parallel Neurocomputer // Tech. Rep., IBM Research Division Physics Group Munich, 1997. (ftp://archive.cis.ohio-state.edu/pub/neuroprose/klagges.massively-parallel.ps.gz).
129. *Тютерев В.В.* Методы оптимизации нейронных сетей со сложной топологической структурой // Труды регион, науч.-практ. конф. «Сибирская школа молодого ученого». – Томск: Изд-во ТГПУ, 1999. – Т. 4. – С. 28 – 30.
130. *Тютерев В.В.* Подход к моделированию эффективных по размеру нейронных сетей // Материалы XXXVII Международной научной студенческой конференции «Студент и Научно-технический прогресс»: Математика. – Новосибирск: Изд-во Новосиб. ун-та, 1999. – С. 141 – 142.
131. *Малышков Ю.П. и др.* Предсказание землетрясений методом измерения литосферных импульсов // Вулканология и сейсмология. – 1998. – № 1. – С. 92.
132. *Тютерев В.В.* Определение эффективного размера нейронной сети в процессе обучения методом динамического наращивания узлов // Сборник трудов VI

Всероссийского семинара «Нейрокомпьютеры и их применение». – М., 2000. – С. 549 – 551.

133. *Тютерев В.В.* Построение нейронных сетей эффективного размера методом генетических алгоритмов // IV Сибирский конгресс по прикладной и индустриальной математике (ИНПРИМ-2000): Тез. докл. – Новосибирск: Изд-во Ин-та математики, 2000. – Т. 2. – С. 126 – 127.
134. *Siddiqi A.A., Lucas S.M.* A comparison of matrix rewriting versus direct encoding for evolving neural networks // Proceedings of Intern. Joint Conf. on Neural Networks'98, Anchorage, Alaska, 1998.
135. *Jacob W., Rehder M.* Evolution of neural net architectures by a hierarchical grammar-based genetic system // Proc. of the International Joint Conference on Neural Networks and Genetic Algorithms. – 1993. – P. 72 – 79.
136. *Hinton E., Drew van Camp.* Keeping Neural Networks Simple by Minimizing the Description Length of the Weights, 1993. (http://www.cs.utoronto.ca/drew/colt93.ps).
137. *Тютерев В.В.* Применение генетических алгоритмов для определения оптимальной топологии нейронных сетей // Нейроинформатика и ее приложения: Материалы VIII Всероссийского семинара / Под общ. ред. А.Н.Горбаня. – Красноярск: ИПЦ КГТУ, 2000. – С. 171.
138. *Иваненко Б.П., Парфенов А.Н., Тютерев В.В.* Исследование генетически построенных нейронных сетей на примере моделирования системы взаимодействующих нефтяных скважин // Моделирование неравновесных систем – 2000: Материалы III Всероссийского семинара / Под общ. ред. А.Н.Горбаня. – Красноярск: ИПЦ КГТУ, 2000. – С. 99.
139. *Тютерев В.В.* Применение генетических алгоритмов для автоматического построения нейронных сетей // Материалы междунар. науч.-практ. конф. «Компьютерные технологии в науке, производстве, социальных и экономических процессах». – Новочеркасск: Юж.-Рос. гос. техн. ун-т., НАБЛА, 2000. – Т. 2. – С. 33 – 34.
140. *Тютерев В.В.* Алгоритм эволюционного наращивания нейронной сети // Сборник трудов III Всероссийской научн.-технич. конференции «Нейроинформатика-2001». – М.: МИФИ, 2001. – Т. 1. – С. 213 – 218.
141. *Тютерев В.В., Новосельцев В.Б.* Автоматическое построение нейронных сетей методом эволюционного наращивания. – Томск: Том. ун-т, 2001. – 22 с. Деп в ВИНИТИ 11.09.2001, 1944-÷2001, 2001.
142. *Тютерев В.В., Новосельцев В.Б.* Исследования алгоритма автоматического построения нейронной сети // Исследования по анализу и алгебре. – Томск: Изд-во Том. ун-та, 2001. – Т. 3. – С. 269 – 281.

ОГЛАВЛЕНИЕ

ДЛЯ ЗАМЕТОК

Сергей Владимирович Аксенов
Виталий Борисович Новосельцев

ОРГАНИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ НЕЙРОННЫХ СЕТЕЙ (методы и технологии)

Редактор *Т.С. Портнова*
Верстка *Д.В. Фортес*

К-ОКП ОК-005-93, код продукции 953380

Изд. лиц. ИД № 04000 от 12.02.2001. Подписано к печати 12.04.2006.
Формат 60 × 84 $^{1}/_{16}$. Бумага офсетная. Печать офсетная. Гарнитура «Таймс».
Усл. п. л. 7,44. Уч.-изд. л. 8,33. Тираж 500 экз. Заказ № 8.

ООО «Издательство научно-технической литературы»
634050, Томск, пр. Ленина, 34а, тел. (382-2) 53-33-35

Отпечатано в типографии ЗАО «М-Принт», г. Томск, ул. Пролетарская, 38/1