

Предисловие

Еще никто не знает, какой вариант нейрокомпьютера завоюет мир, еще нет общепринятого определения нейрокомпьютера, еще неясно, в какой мере его структура должна копировать структуру мозга и ведутся споры о том, что же такое нейрон, но уже очевидно приближение новой технической революции.

Пять поколений ЭВМ следуют друг за другом. Нарождающееся шестое настолько отличается от предыдущих, что лучше говорить не о поколениях и даже не о новых видах, родах или семействах, а о новом царстве - масштаб дистанции между нейрокомпьютерами и обычными ЭВМ соответствует различиям между царствами живых организмов.

Чем отличаются машины второго царства? Ответ на этот вопрос - одна из задач книги. Но можно коротко для первого знакомства указать несколько важных отличий.

1. Большое число параллельно работающих простых элементов - нейронов (от нескольких десятков до $10^6 - 10^8$), что обеспечивает колоссальный скачок в быстродействии.
2. Место программирования занимает обучение (воспитание) - машина учится решать задачи, изменения параметры нейронов и связей между ними.

Вместе с новой техникой рождаются и новые профессии. Уже сейчас можно назвать две из них.

Нейроконструктор. Тут даже две профессиональных области: а) конструирование универсальных нейронных блоков, б) конструирование для данного класса задач нейрокомпьютера из универсальных блоков.

Воспитатель или учитель компьютеров. Место программиста у нейрокомпьютера займет человек, который будет выращивать в машине умения решать задачи и развивать у нее способности к обучению. По аналогии с педагогикой тут можно выделить и воспитание, и обучение. Опять, как и для нейроконструктора, можно указать две области деятельности: создание методик ("методист") и обучение реальным задачам, сконструированное

Горбань А.Н. Обучение нейронных сетей.

Рассматриваются обучаемые нейронные сети. Дан обзор алгоритмов обучения, основанных на принципе двойственности. С помощью описанных алгоритмов достигнуто ускорение обучения в 10000 и более раз по сравнению с методом обратного распространения ошибки.

Книга может служить для первого знакомства с нейрокомпьютерами и предназначена для научных работников, аспирантов и студентов, интересующихся этим новым направлением в информатике и вычислительной технике.

Ил. 23, библиогр. 32 назв.

CopyRight СП "Параграф", 1990



на базе универсальных методик.

Откуда берутся представители новых профессий, когда их еще не было и следовательно нет профессиональных традиций? Наивно было бы полагать, что все определяется наличием общественной потребности по схеме: есть потребность – возникнут профессионалы. Должны быть сформированы нормы новой профессиональной деятельности, прописаны основные процедуры, зафиксированы образцы. Как бы поверх этой профессиональной деятельности должна быть построена деятельность учебная, организованы обучение и вовлечение – вербовка в новую профессию.

И сразу – несколько проблем.

Первое. Кто сформирует новую деятельность? Вряд ли следует ожидать этого от разработчиков первых нейрокомпьютеров. У них другие задачи. Как тут не вспомнить, что Джон фон Нейман не был создателем профессии программиста.

Второе. Как сформировать новую деятельность, если нет еще ее главного предмета и орудия – работающего серийного нейрокомпьютера?

Третье. Откуда возьмутся кадры – люди, готовые на риск – осваивать новую профессию, пока общественная потребность не оформилась в виде рабочих мест, ставок, ... ?

Проще всего с кадрами. Есть люди, которым тесны рамки профессиональной структуры науки и техники. Это **конкингсторы** научно-технического прогресса, поставляющие кадры лидеров для новых направлений. Другая группа – люди, не нашедшие прочного места в своей научно-технической профессии, так сказать, **аутсайдеры**. Среди них немало одаренных личностей. Наконец, **молодежь**, у которой не угас еще важный первичный стимул профессиональной ориентации: интересно – значит, надо попробовать.

Когда нет предмета деятельности, а создавать и осваивать ее надо, выход один – **игра**. Многое для решения второй проблемы можно понять, взглянувши в игры детей. С чем играть? Вероятно, с программными имитаторами нейрокомпьютеров. Игра – дело серьезное, и продукты таких игр могут находить практическое применение в различных областях – от распознавания визуальных образов до медицинской

диагностики. Однако имитаторы нейрокомпьютеров на фоне Неймановских машин работают еще более-менее сносно, но обучаются медленно¹ и поэтому не всегда смогут конкурировать с обычными распознавающими программами. Это не страшно. Главное – чтобы шла Игра и создавалась новая деятельность. Полезные для практики результаты – побочный продукт Игры. Другого пути пока не видно.

И, наконец, первая проблема. Ход, позволяющий ее решить, уже описан. В Игре, игроками и разработчиками вариантов Игры могут быть заложены основы новых профессий.

Честно говоря, Игра уже началась. Некоторые серьезные исследователи высказывают это даже как упрек международному нейрокомпьютерному движению, в которое вложены уже сотни миллионов долларов. Они правы – с точки зрения своих давно оформленных профессий. Но с нейроконструированием и воспитанием компьютеров дело обстоит сложнее и без Игры не обойтись. Можно, конечно, делать серьезную мину и пускать пыль в глаза. Можно и даже нужно в рамках Игры формировать серьезные научные программы нового типа. Но надо помнить, что главное – сама Игра. Человечество осваивает новую игрушку, которая, быть может, позволит совершить прыжок к будущему.

Гарантирует ли успех? Вдруг игрушка останется всего лишь игрушкой, а развитие техники пойдет по другому пути. Что же, и это не исключено. Но задачи, обреченные на успех, не порождают быстрого развития. Кто не играет, тот не выигрывает.

Пора объявить свои цели.

1. Распространение и популяризация достижений в области конструирования нейрокомпьютеров.
2. Создание и распространение пакетов программ, имитирующих нейрокомпьютеры на ЭВМ и позволяющих решать ряд прикладных задач.
3. Проведение конкурсов по нейроконструированию и воспитанию

¹ Впрочем, использование серии усовершенствований позволило довести нейросетевые распознавающие программы для персональных компьютеров до вполне конкурентоспособного состояния: они уже и обучаются достаточно быстро.

компьютеров, а также школ, конференций, игр

Без участия фирмы "Феникс" все это было бы невозможно. Тут трудно удержаться и не ругнуть отечественную организацию науки, ругнуть за исключительную добропорядочность, ориентацию на авторитеты, а не на создание нового уже привыкшего, что новое приходит из-за рубежа. Тому много причин - от слабого общественного спроса на научные результаты, до громоздкой и тормозящей развитие науки системы аттестации. Это может (в который раз!) поставить нас в положение отстающих и агонящих.

Один пример из области нейроконструирования. Самое существенное достижение последних лет в этой области - открытие алгоритмов, позволяющих самой нейронной сети в акте двойственного функционирования вычислять направление наилучшего обучения. Это было сделано около четырех лет назад одновременно в США (Дж. Хинтон и др.) и СССР (В.А. Охонин). Одна из первых публикаций - Барцев С.И., Охонин В.А. Адаптивные сети обработки информации. Красноярск. изда. Института физики СО АН СССР, 1986. 19 с. / препринт 59Б. Алгоритм Охонина более универсален, чем алгоритмы обратного распространения ошибки, за счет явного использования идей двойственности.

И что в результате? Первые публикации Охонина прошли практически незамеченными. Подробная статья около трех лет лежала в издательстве (недавно вышла [2]). Все, в том числе и советские, исследователи ссылаются исключительно на работы американских авторов. А в США тем временем созданы уже машины, работающие по новому алгоритму.

Сколько раз советским ученым приходилось сталкиваться с таким "признанием" своих работ: результаты аналогичных и даже более слабых зарубежных исследований публикуются у нас как последние достижения и образцы для подражания. Как тут не воскликнуть вслед за генералом Ермоловым: "Государь, произведи меня в немцы!" Квасной патриотизм смешон, но не менее смешна ориентация исключительно на зарубежные авторитеты. Мировая наука едина, но новое может быть создано и у нас. Необходима отечественная школа нейроконструирования.

Несколько слов о содержании книги. В ней излагаются

методы обучения нейронных сетей, основанные на принципе двойственности. По сравнению с первыми алгоритмами Охонина и Хинтона обучение удалось ускорить в 10000 и более раз (тестовые задачи - распознавание визуальных образов и бинарная классификация). Дан набросок ассемблера нейронной сети - списки основных команд, с помощью которых можно описать большинство алгоритмов. Приложения нейрокомпьютеров рассматриваются только эпизодически - они заслуживают отдельной монографии.

Значительная часть излагаемых результатов получена при работе в следующих организациях: Вычислительном центре СО АН СССР (Красноярск), Институте биофизики СО АН СССР (Красноярск) и Научно-техническом объединении "Феникс" (Омск).

Я бы не смог написать эту книгу без сотрудничества с С.И. Барцевым, С.Е. Гилем и Е.М. Миркесом и В.А. Охониным. В частности, С.Е. Гилем и Е.М. Миркесом создан эмулятор нейрокомпьютера на IBM PC/AT, с помощью которого исследовались основные алгоритмы. Я рад случаю поблагодарить всех за помощь и сотрудничество.

Красноярск
февраль 1990 г.

А. Горбань

1. ПЕРЕЧЕНЬ ЭЛЕМЕНТОВ

Нейрокомпьютеры создаются из уже признанного набора "кубиков" - элементов. Их совокупность неоднородна - одни надстраиваются над другими. Элементы первого типа: **нейрон**, **синапс** - связь между нейронами и **сумматор**, складывающий сигналы, приходящие к данному нейрону. Иногда сумматор включают в состав нейрона. Это, конечно, не особенно важно. В наиболее распространенных моделях функционирования единственным нелинейным элементом является нейрон, сумматор осуществляет обычное линейное сложение, синапс - умножение передаваемого сигнала на число - вес синапса.

Следующий элемент - обучающий **пример**. Он задается парой наборов значений: **вход** - соответствующий **выход** и **целевой функцией**, штрафующей за отклонение выходов от предписанных значений (при данных вводах).

Различным группам нейронов, синапсов, сумматоров и примеров можно присваивать метки - "цвета". Группы элементов разных цветов могут по-разному участвовать в процессе обучения.

Следующая группа элементов - **алгоритмы обучения**. Один элемент - это шаг или цикл алгоритма. Соединяя такие шаги в цепочки, получаем методику обучения. Методику будем считать составной (незлементарной).

Итак, имеем перечень элементов: нейроны, синапсы, сумматоры, обучающие примеры, цвета нейронов, синапсов, сумматоров, примеров, алгоритмы обучения.

2. НЕЙРОН

2.1. Функционирование нейрона

Наиболее распространено представление нейрона как преобразователя один **вход** - один **выход**, действующего в дискретные моменты времени с некоторым шагом τ . На входной сигнал A такой нейрон отвечает выходным $f(A, \alpha)$, где α - характеристика нейрона или набор характеристик. Примеры:

1) пороговый нейрон, $A \geq 0, \alpha > 0$

$$f(A) = \begin{cases} 0, & \text{если } A < \alpha; \\ 1, & \text{если } A \geq \alpha; \end{cases} \quad (1)$$

2) $f(A) = \frac{A}{\alpha + |A|}, \quad (-\infty < A < \infty, \alpha > 0); \quad (2)$

3) $f(A) = \text{th} \frac{A}{\alpha} \quad (-\infty < A < \infty, \alpha > 0). \quad (3)$

В наших моделях мы обычно использовали вторую функцию.

Многопараметрические нейроны могут функционировать, например, так:

1) $f(A) = \frac{\alpha_1 A + |A|A}{\alpha_2 + \alpha_3 |A| + A^2} \quad (\alpha_2 > 0, -\infty < A, \alpha_{1,3} < \infty); \quad (4)$

2) $f(A) = \frac{\alpha_1 A + \alpha_2 A^2 + A^3}{\alpha_3 + \alpha_4 |A| + \alpha_5 A^2 + |A|^3} \quad (\alpha_3 > 0, -\infty < A, \alpha_{1,2,4,5} < \infty); \quad (5)$

и т. д. Знаки абсолютной величины расставлены так, чтобы $f(A) \rightarrow \pm 1$ при $A \rightarrow \pm \infty$, а знаменатель был всегда положителен.

Возможно ввести **нелокальность** по времени, задавая функционирование нейрона с помощью динамической системы:

$$f_{n+1} = F(f_n, A_n, \alpha),$$

где $n, n+1$ - моменты времени, α - параметр или набор параметров.

При функционировании в непрерывном времени также возможно введение нелокальности по времени - **запаздывания**. Это даже полезно, так как позволяет сглаживать отдельные

случайные выбросы. Нелокальность может осуществлять как сумматор, так и сам нейрон, например:

$$1) f(\{A(\tau) | \tau \leq t\}) = \begin{cases} 0, & \text{если } \int_0^{\infty} e^{-\alpha_1 \tau} A(t-\tau) d\tau < \alpha_2; \\ 0, & \\ 1, & \text{если } \int_0^{\infty} e^{-\alpha_1 \tau} A(t-\tau) d\tau \geq \alpha_2; \\ 0, & \end{cases} \quad (6)$$

$$A(\tau) \geq 0, \alpha_{1,2} > 0.$$

Поскольку здесь нелокальность задается линейным образом, можно положить

$$A'(t) = \int_0^{\infty} e^{-\alpha_1 \tau} A(t-\tau) d\tau$$

и считать $A'(t)$ выходом сумматора, оставляя нейрон стандартным пороговым (1). Другой пример, не сводимый в общем случае к линейным операциям, можно построить, задавая функционирование нейрона дифференциальным уравнением

$$\frac{df}{dt} = F(f(t), A(t), \alpha), \quad (7)$$

где f – как и выше, выход, A – вход, α – параметр или набор параметров, F – некоторая нелинейная функция.

Функция F в (7) должна выбираться так, чтобы обеспечивать переключения. Простейший пример:

$$F = -\alpha_1 f^3 + \alpha_2 f + A \quad (\alpha_{1,2} > 0).$$

Эта функция имеет экстремумы в точках $f = \pm f_0 = \pm \sqrt{\alpha_2 / 3\alpha_1}$. Если $A > -\alpha_1 f_0^3 + \alpha_2 f_0$, то нуль у F один и выходное значение, выдаваемое нейроном, релаксирует к этому значению f при постоянном значении на входе $A(t) \equiv A$. Аналогично, если $A < -\alpha_1 f_0^3 + \alpha_2 f_0$.

Если же $\alpha_1 f_0^3 - \alpha_2 f_0 < A < -\alpha_1 f_0^3 + \alpha_2 f_0$, то нейрон бистабилен и его релаксация при $A(t) \equiv A$, $t \rightarrow \infty$ определяется начальным состоянием.

Можно задавать релаксацию нейрона и дифференциальными уравнениями более высокого порядка, если это упростит техническую реализацию.

2.2. Обучение нейрона

Точнее было бы назвать этот раздел "нейрон в процессе

обучения", так как обучается не отдельный нейрон, а сеть в целом. В ходе обучения сети предъявляются примеры и их решение оценивается. По результатам оценки делается шаг обучения. В результате такого шага могут меняться все параметры, в том числе – характеристики нейронов α . Сами процедуры изменения α будут описаны далее в разделе, посвященном алгоритмам обучения. Здесь же – два замечания.

Во-первых, для быстрого параллельного вычисления градиента оценки потребуются элементы, вычисляющие производные выходов по параметрам: $\partial f / \partial \alpha$.

Во-вторых, для параметров есть область естественных ограничений. Например, α в формуле (2) не должно слишком отличаться от 1, естественный диапазон для нее логично принять таким: $2\alpha \geq 0.1$. Если процедура обучения на каком-либо шаге выводит из зоны естественных ограничений, то можно либо оставаться на ее границе, либо ввести в оценку функцию штрафа за выход из этой зоны.

Заметим еще, что в большинстве алгоритмов обучения сетей параметры нейронов полагаются неизменными, а все нейроны – одинаковыми. Меняются только веса синаптических связей. Это направление и его идейная база получили название "коннекционизм" – главное не элементы, а связи. Между тем, при чисто коннекционистском подходе теряются дополнительные возможности обучения и функционирования. Гетерогенная сеть, состоящая из различных нейронов с подстраиваемыми параметрами, является более гибкой.

2.3. Сложный нейрон как сеть простых

Как нетрудно видеть из предыдущих разделов, в принятой простой модели нейрона его состояние совпадает с его выходом. Здесь огромное поле возможностей для дальнейшего усложнения. Можно ввести для нейрона классическую триаду "вход, состояние, выход", задать три пространства: X_{in} – входы, S – состояния, X_{out} – выходы, функционирование в дискретном времени описывать двумя функциями:

$F_{s\alpha} : X_{in} \times S \rightarrow S$ - смена состояний,

$F_{out \alpha} : X_{in} \times S \rightarrow X_{out}$ - выход,

α , как и прежде, параметр или набор параметров.

Выход на $n+1$ -м шаге будет задаваться как функция от входа и состояния на предыдущем шаге

$$X_{out \alpha, n+1} = F_{out \alpha}(X_{in, n}, S_n),$$

аналогично будет задаваться смена состояний

$$S_{n+1} = F_{s\alpha}(X_{in, n}, S_n).$$

для сетей непрерывного времени выходы и смена состояний могут быть описаны системами дифференциальных уравнений.

На первый взгляд, сеть из таких более мощных нейронов содержит и больше возможностей. Более того, я полагаю, что сложные специализированные нейроны будут необходимым элементом будущих нейрокомпьютеров. Однако есть веские аргументы против такого усложнения всех нейронов сети.

Во-первых, есть мнение, что усложнение влечет потерю универсальности: решение одних задач может упроститься, зато других - резко усложниться.

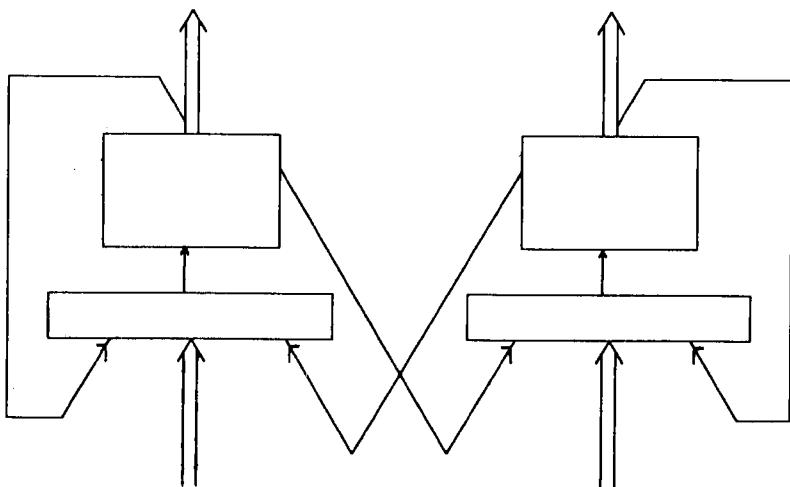


Рис. 1. Система из двух нейронов с линейными сумматорами:

□ - нейрон, ■ - сумматор, → - сигналы, * - входы и выходы.

Во-вторых, сложный нейрон, если потребуется, может вырасти в процессе обучения как малая сеть простых. Так, уже сеть из двух простых нейронов с линейными сумматорами представляет собой систему с двумерными пространствами состояний, входов и выходов (рис. 1). Параметры состояний здесь - выходы на предыдущем шаге. Возрастает и число подгноточных (обучаемых) параметров: кроме внутренних параметров отдельных нейронов сюда войдут четыре веса синапсов, на которые умножаются выходные сигналы перед подачей на сумматоры.

Резюмировать можно так: если для какого-либо важного класса задач требуются сложные специализированные нейроны, их проще вводить сразу, чем выращивать из простых. Если же такая необходимость не очевидна, то можно довериться универсальной сети из простых нейронов, она сама вырастит необходимые структуры в ходе обучения.

3. СУММАТОРЫ

По принятому нами определению, сумматор – линейный элемент, связанный с каждым нейроном и суммирующий поступающие к нему по синаптическим связям сигналы от других нейронов, а также внешние входные сигналы. Конечно, при технической реализации суммирование может быть объединено с хранением синаптических весов входящих связей и умножением на эти веса выходных сигналов других нейронов, однако при теоретическом рассмотрении функция сумматора выделена.

На первый взгляд, у сумматора не может существовать подгоночных параметров – складывает поступающие сигналы и все. Однако это не совсем так. Сумматор может отвечать за нелокальность по времени. Для функционирования в непрерывном времени он может, например, подавать на вход нейрона функцию

$$A'(t) = \int_0^{\infty} e^{-\alpha t} A(t-\tau) d\tau \quad (0 < \alpha < \infty), \quad (9)$$

где $A(t)$ – суммарный сигнал, поступающий в момент t (конечно, ∞ в пределах интегрирования надо понимать как достаточно большое конечное время).

Для функционирования в дискретном времени сумматор после каждой передачи сигнала нейрону может умножать его на некоторое α ($0 < \alpha < 1$), а потом прибавлять к произведению поступающие сигналы и так – до следующей передачи нейрону.

Накопление сигнала с затуханием памяти важно в таких сетях, где нейроны могут функционировать несинхронно. Проблема же синхронизации обостряется с увеличением числа нейронов.

Обучение сумматора состоит в оптимальном подборе параметра затухания α . Если же затухания нет, то сумматор предполагается необучаемым.

4. СИНАПС

4.1. Функционирование синапса

Синапс осуществляет связь между нейронами. Он передает выходной сигнал i -го нейрона на входной сумматор j -го и преобразует сигнал при передаче. Простейшее преобразование – умножение выходного сигнала на вес синапса – число x_{ij} (рис. 2).

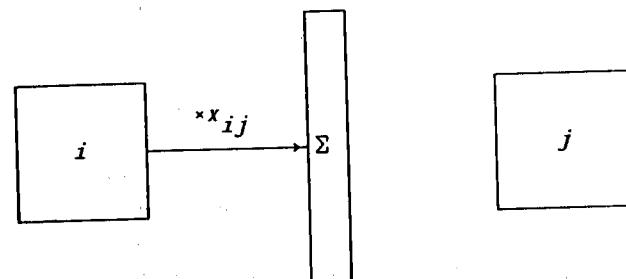


Рис. 2. Действие линейного синапса.

Возможно усложнение этой простейшей схемы по трем направлениям: увеличение размерности (для сложных нейронов), нелинейное преобразование сигнала при передаче и элементы памяти у синапса.

Увеличение размерности для линейных синапсов без памяти эквивалентно увеличению числа простейших синапсов. Действительно пусть имеется два сложных нейрона: i -й с p выходами и j -й с q выходами. Сумматор при j -м нейроне накапливает входной n -мерный вектор сигналов. Синапс осуществляет передачу выходного сигнала i -го нейрона этому сумматору и линейное преобразование сигнала при передаче. Преобразование, естественно, состоит в умножении i -го n -мерного вектора выходных сигналов на матрицу – "синаптический вес" x_{ij} . Число элементов этой матрицы для данных i , j равно $p \times n$.

То же самое можно представить и по-другому. С каждым выходом j -го нейрона связан свой сумматор – их q . От каждого

выхода i -го нейрона к каждому такому сумматору ведет простейший линейный синапс. Он передает сигнал, умножая его на свой вес. Всего таких сигналов и соответствующих весов $m \times n$.

Нелинейное преобразование сигнала делает синапс аналогичным нейрону, только без сумматора на входе. Сеть с нелинейными синапсами можно представить себе как сеть из большего количества нейронов, часть которых имеет всего две связи — одну входную и одну выходную, а сигналы по всем связям передаются без изменения.

Если допускать нелинейное преобразование проходящих по синапсам сигналов, то естественным образом приходим к новой модели сети. В ней уже нет преобразующих синапсов. Вместо них — дополнительные нейроны. Связи служат только передаче сигналов без преобразования. Вариант такой сети с простыми нейронами можно представить себе так.

Сеть состоит из нейронов, сумматоров и связей. Нейроны могут быть разными. Каждый нейрон имеет один вход и один выход. Сумматор имеет любое количество входов и один выход.

Сигнал на вход нейрона подается либо от одного (другого) нейрона, либо от какого-нибудь сумматора, либо от внешнего входа. Выходной сигнал любого нейрона может подаваться сразу на несколько разных входов как другим нейронам, так и сумматорам или выходным устройствам.

Сумматор имеет произвольное число входов и один выход, на который он передает сумму входных сигналов. Выходной сигнал сумматора может передаваться сразу нескольким нейронам.

Последовательность функционирования такой сети можно представить так: функционирование нейронов — передача сигналов от нейронов — суммирование — передача сигналов от сумматоров — функционирование нейронов.

По всей видимости, при отказе от простой схемы линейных синапсов (см. рис. 2) имеет смысл переходить к описанному представлению сети. Возможна возражение: при обычном дискретном по времени функционировании синапсы передают на входные сумматоры сигналы с предыдущего шага, а при описанной схеме передаваться будут преобразованные сигналы с позапрошлого шага.

Этого нетрудно избежать, предполагая нейроны сети разделенными на несколько групп, функционирующих последовательно друг за другом, как ранее были разделены нейроны и сумматоры. Разбиение на группы должно удовлетворять одному контролльному условию: значения сигналов на входе каждого элемента должны быть правильно сформированы в промежуток между его последовательными функционированиями.

Введение в синапсы элементов памяти также может рассматриваться как их превращение в сложный нейрон с внутренними состояниями и вновь приведет нас к описанному способу представления сети.

Вернемся к простейшему линейному синапсу (см. рис. 2). Его вес x_{ij} может принимать различные значения. Выделены два случая, самый простой — $x_{ij} = 0, \pm 1$ (связь нет, связь есть: положительная или отрицательная). Другой вариант: для x_{ij} устанавливается рабочий диапазон $-1 \leq x_{ij} \leq 1$.

4.2. Обучение синапсов и коннекционизм

В настоящее время распространенной точкой зрения на обучение нейрокомпьютеров является коннекционизм: обучение состоит в настройке значений x_{ij} — весов синапсов. Согласно коннекционизму все навыки хранятся в синаптической карте — совокупности весов x_{ij} . Свойства нейрона при этом считаются второстепенными характеристиками сети.

Настройка синаптической карты может производиться различными способами. Ограничимся пока рассмотрением двух предельных случаев.

1. Внешнее формирование (сети Хопфилда). Всплеск интереса к нейронным сетям был связан с распознавающими сетями Хопфилда. Для них синаптическая карта симметрична ($x_{ij} = x_{ji}$), а функционирование происходит в непрерывном времени. Состояние i -го нейрона (оно же — выходной сигнал) $\alpha_i(t)$ меняется во времени согласно уравнению

$$\frac{d\alpha_i}{dt} = \lambda (1 - \alpha_i^2) \alpha_i - A_i(t), \quad (10)$$

где $A_i = \sum_j x_{ij} \alpha_j$ — входной сигнал, приходящий от других нейронов.

Правая часть (10) представляет собой $-\partial H / \partial \alpha_i$, где

$$H = \frac{\lambda}{4} \sum_i (1 - \alpha_i^2)^2 + \frac{1}{2} \sum_i \alpha_i x_{ij} \alpha_j. \quad (11)$$

Уравнение (10) имеет вид

$$\frac{d\alpha_i}{dt} = -\frac{\partial H}{\partial \alpha_i}, \quad (12)$$

поэтому движение стремится к одному из локальных минимумов H . Роль входного сигнала играют начальные состояния α_i .

"Программирование" сети на распознавание N эталонных образов осуществляется так. Пусть эталонам соответствует состояния сети $\alpha_i = \sigma_{1i}$, $i = 1, \dots, N$.

Положим

$$x_{ij} = -\varepsilon \sum_{l=1}^N \sigma_{1l} \sigma_{jl}. \quad (13)$$

Тогда функция H (11) примет вид

$$H = \frac{\lambda}{4} \sum_i (1 - \alpha_i^2)^2 - \frac{\varepsilon}{2} \sum_{i,j} (\alpha_i \sigma_{1j})^2. \quad (14)$$

Первое слагаемое в H дает вклад в уравнение (12), сдвигающий α_i к +1 или -1 в зависимости от начального значения - положительные α_i он тянет к +1, отрицательные - к -1. Если записать вклад второго слагаемого в векторном виде, то получим:

$$\frac{d\alpha}{dt} = \dots + \varepsilon \sum_i (\alpha, \sigma_1) \sigma_i. \quad (15)$$

Эта сумма сдвигает α пропорционально векторам σ_i , причем сдвиг больше для тех σ_i , для которых больше по величине скалярное произведение (α, σ_i) . Если предполагать, что величина $|\alpha, \sigma_i|$ имеет ярко выраженный максимум по σ_i , то получим в итоге α "похожим" на соответствующее σ_i . Искажения будут связаны с наличием других σ_i и первого слагаемого в H .

Возможен и другой подбор x_{ij} . Важен принцип - синаптическая карта формируется сразу и запоминается.

Приведем пример усовершенствования. В нем

$$H = \frac{\lambda}{4} \sum_i (1 - \alpha_i^2)^2 - \frac{\varepsilon}{2} (\alpha, P\alpha),$$

где P - ортогональный проектор α на линейное пространство, порожденное векторами σ_i .

Строится P , например, с помощью последовательной

ортогонализации. Полагаем

$$\gamma_1 = \sigma_1, \quad \delta_1 = \frac{\sigma_1}{\sqrt{(\sigma_1, \sigma_2)}};$$

$$\gamma_2 = \sigma_2 - (\sigma_2, \delta_1) \delta_1, \quad \delta_2 = \frac{\gamma_2}{\sqrt{(\gamma_2, \gamma_2)}};$$

$$\dots$$

$$\gamma_i = \sigma_i - \sum_{q, q < i} (\sigma_i, \delta_q) \delta_q, \quad \delta_i = \frac{\gamma_i}{\sqrt{(\gamma_i, \gamma_i)}};$$

$$P\alpha = \sum_i \delta_i (\alpha, \delta_i).$$

Веса синапсов в этом случае - матричные элементы: $P: P = (x_{ij})$.

2. *Обучение*. В ходе обучения происходит минимизация целевой функции, оценивающей соответствие выходных сигналов входным. По существу, при таком подходе задача обучения нейрокомпьютеров погружается в обширную и хорошо разработанную область науки, посвященную методам оптимизации.

Специфические трудности здесь будут состоять в очень большой размерности и связанных с этим ограничениях на память и быстродействие. Поэтому преимущество должно отдаваться алгоритмам оптимизации, допускающим параллельное выполнение на структурах, аналогичных нейронным сетям.

5. ВХОДЫ, ВЫХОДЫ И ФУНКЦИОНИРОВАНИЕ

5.1. Входы и выходы

Существует несколько вариантов подачи входных сигналов сети.

1. **Подача на вход нейронов.** Выделяется набор входных нейронов и на их сумматоры добавляются внешние сигналы.
2. **Подача на выход нейронов.** Внешние сигналы добавляются к выходным сигналам выделенных входных нейронов.
3. **Аддитивная подача на синапсы.** Для каждого внешнего сигнала выделяется свой входной синапс и эти внешние сигналы добавляются к сигналам, поступающим на эти синапсы.
4. **Подача сигналов как синаптических весов (мультиплексивная подача на синапсы).** Выделенному массиву синаптических весов присваиваются значения, равные входным сигналам.

В вариантах 1-3 обучаться могут все синапсы, в варианте 4 входные синапсы, очевидно, не обучаются.

Существует также несколько вариантов снятия сигналов.

1. **Снятие с выхода нейронов.** Выходные сигналы части нейронов являются и выходными сигналами нейрокомпьютера.
2. **Снятие с входа нейронов.** За выходные сигналы принимаются входные сигналы выделенных нейронов или выходные сигналы соответствующих сумматоров.
3. **Аддитивное снятие с синапсов.** На выход подаются сигналы, проходящие по выделенным синапсам, уже умноженные на соответствующий вес синапса.
4. **Снятие синаптических весов.** Выходными сигналами считаются установленные веса выделенных синапсов.

В четвертом случае процесс функционирования должен включать изменение выходных синапсов. В обычной постановке задачи это соответствует, скорее, обучению. Функционирование с изменением весов синапсов можно рассматривать как обучение с вспомогательной целевой функцией, параметры которой

являются подстраиваемыми в ходе "настоящего" обучения, устанавливающего правильное соответствие вход - выход при данном типе выходных сигналов.

В этом и следующих разделах, если специально не оговорено противное, мы ограничимся рассмотрением входов первого и второго типов и выходов первого. Некоторые другие варианты (а всего мыслимых вариантов восемь) будут обсуждаться отдельно. Ограничимся пока также нейронными сетями, синхронно функционирующими в дискретные моменты времени.

5.2. Сети периодического функционирования

Простейшие представления о функционировании нейронной сети тезки. В начальный момент состояния всех нейронов одинаковы, выходных сигналов нет. Предполагается, что характеристики нейронов это позволяют. Подаются входные сигналы, инициирующие активность сети (нулевой такт). Далее входные сигналы могут подаваться на каждом такте функционирования. На каждом такте могут сниматься выходные сигналы. После k тактов цикл функционирования заканчивается и сеть возвращается в исходное состояние, готовая к новому циклу (акту) функционирования. Между актами функционирования могут вставляться акты обучения.

В результате цикла из k тактов функционирования нейронная сеть выдает в ответ на последовательность из k наборов входных сигналов последовательность из k наборов выходных сигналов.

Функция ошибки должна улавливать взаимное соответствие входной и выходной последовательностей. Еще более упрощенный вариант - входные сигналы подаются только в самом начале, выходные - снимаются в самом конце. Несколько забегая вперед, заметим, что такое упрощение приводит к замедлению обучения - полезно оценивать ответы сети на каждом такте функционирования, постепенно ужесточая требования и также может быть полезно "напоминать" сети входной сигнал на каждом такте, постепенно уменьшая интенсивность. Это - результаты опытов по обучению сетей.

Сети периодического функционирования по характеру использования напоминают обычные ЭВМ: на вопрос следует ответ, причем воспроизводимый. Иначе обстоит дело с сетями непрерывного функционирования.

5.3. Сети непрерывного функционирования

Можно организовать функционирование нейронной сети, не приводя ее каждый раз перед подачей задания в фиксированное состояние. При этом перед получением задания сеть находится в некотором состоянии, нейроны выдают выходные сигналы. В течение k тактов функционирования подаются входные и снимаются выходные сигналы. После такого функционирования можно проводить цикл обучения, оценивая успешность функционирования, а можно переходить к следующему циклу функционирования.

Непрерывное функционирование нейронной сети (с подучиванием или без него) более соответствует имеющимся представлениям о поведении живых существ, чем периодическое. Опыт показывает, что, чередуя циклы функционирования и обучения, для таких сетей можно получить хорошие результаты адаптации.

Важная особенность сетей непрерывного функционирования: для них естественным образом можно ввести спонтанную активность нейронов. Не требуется отсутствия выходных сигналов при нулевых входных. Простейший пример — вместо характеристики нейрона (2) допускается

$$f(A) = \frac{A}{\alpha_1 + |A|} + \alpha_2, \alpha_2 > 0. \quad (16)$$

Параметры спонтанной активности (здесь α_2) для таких нейронных сетей должны подбираться в ходе обучения.

Конечно, нейроны со спонтанной активностью можно использовать в сетях периодического функционирования, однако там это менее естественно из-за необходимости стандартного начального состояния и жесткой связи входных и выходных сигналов.

6. ОЦЕНКИ И ПРИМЕРЫ

6.1. Оценки одного акта функционирования

Мы продолжаем обсуждение нейронных сетей, синхронно функционирующих в дискретные моменты времени. Добавим еще одно самоограничение — будем рассматривать только акт периодического функционирования.

Как уже оговаривалось, входные сигналы прибавляются к выходам нейронов, а выходные снимаются с выходов.

Каждый цикл функционирования может оцениваться. Пусть цикл состоит из k тактов, A_0, \dots, A_{k-1} — последовательность векторов входных сигналов, $\alpha_1, \dots, \alpha_k$ — последовательность векторов выходных сигналов. Оценка — некоторый функционал

$$H(A_0, \dots, A_{k-1}, \alpha_1, \dots, \alpha_k). \quad (17)$$

принимаем соглашение о знаке: чем меньше H , тем лучше. Цель обучения — добиться такого функционирования сети, чтобы H была как можно меньше.

Примеры. В задачах распознавания визуальных образов возможна такая организация оценки. На входные нейроны подается изображение. Число выходных нейронов k равно числу распознаваемых образов. Если изображение соответствует i -му образу, то в конце цикла функционирования (на k -м шаге) выходной сигнал на i -м выходном нейроне должен быть больше, чем на остальных выходных нейронах: $\alpha_{ki} > \alpha_{kj}$ при $j \neq i$.

Возможны различные функции оценки.

$$1. H_i = (\alpha_{ki} - 1)^2 + \sum_{j \neq i} (\alpha_{kj} + 1)^2. \quad (18)$$

Здесь участвуют сигналы с выходных нейронов на k -м такте и оценка тем лучше, чем ближе сигнал с i -го нейрона к +1, а с остальных — к -1. Индекс i — номер образа.

$$2. H_i = \sum_{q=1}^k q [(\alpha_{qi} - 1)^2 + \sum_{j \neq i} (\alpha_{qj} + 1)^2]. \quad (19)$$

Здесь учитываются и результаты предыдущих тактов функционирования с весом, равным номеру такта. Возможны модификации с заменой веса на q^α ($\alpha > 0$), $\exp(\lambda q)$ ($\lambda > 0$) и

т.п.

Опыт показывает, что оценки второго типа могут давать лучшие результаты при обучении.

3. Оценки этого типа строятся так. Обозначим E_1 множество таких n -мерных векторов $\alpha = (\alpha_1, \dots, \alpha_n)$, что $\alpha_i > \alpha_j$ при $i \neq j$. $dist(x, E)$ - евклидово расстояние от точки до множества:

$$dist(x, E) = \inf_{\alpha \in E} \|x - \alpha\|.$$

Пусть e_i - n -мерный вектор, у которого i -я координата - 1, остальные - нули. Полагаем

$$H_1\epsilon = dist(\alpha_i - \epsilon e_i, E_1). \quad (20)$$

Здесь существует параметр ϵ ($0 < \epsilon < 1$) - уровень требований к превышению i -го выходного сигнала над остальными. Нетрудно выписать явные формулы для $H_1\epsilon$.

$$4. H_1\epsilon = \sum_{q=1}^n q dist(\alpha_q - \epsilon e_q, E_1). \quad (21)$$

Эта оценка отличается от предыдущей учетом всех тактов функционирования а не только последнего и различными требованиями на каждом такте.

Опыт показывает, что успешность и скорость обучения существенно зависят от удачного выбора функции оценки (даже в простейших случаях при изменении и времени обучения может меняться в десятки раз).

В этом разделе обсуждалось оценивание одного такта функционирования в том случае, когда ответ известен заранее. Будем называть пару (вход, известный выход) обучающим примером. Подчеркнем, что "известный выход" не обязательно означает точно известные выходные сигналы, а может, например, представлять собой известные требования к выходным сигналам, как в разобранном примере - сигнал с i -го выходного нейрона больше, чем у остальных.

6.2. Оценки для нескольких обучающих примеров

Пусть принят способ оценивания одного такта функционирования и нейронная сеть может получать оценки за решение каждого обучающего примера. Однако обычно примеров много и практически никогда пример не бывает единственным. Вопрос:

как оценить функционирование сети при решении серии примеров?

Простейший способ: взятие средней арифметической оценки (или суммы оценок). Его недостаток: отдельные плохо решаемые примеры растворяются среди хорошо решенных. Можно использовать суммирование оценок в некоторой степени p :

$$H_{\Sigma}^{(p)} = \frac{1}{n} \left(\sum_{i=1}^n H_i^p \right)^{1/p}. \quad (22)$$

Здесь n - число примеров, i - номер примера, H_i - оценка сети при решении этого примера, $1 \leq p < \infty$, H_{Σ} - общая оценка.

Предельный случай - оценка наихудшего примера:

$$H_{\Sigma}^{(\infty)} = \max_{i=1}^n H_i. \quad (23)$$

Возможны и другие варианты. Например, если при $H_i < \epsilon$ пример считается удовлетворительно решенным, то можно положить

$$H_{\Sigma} = \sum_{H_i > \epsilon} H_i + s\epsilon, \quad (24)$$

где s - число тех i , для которых $H_i \leq \epsilon$.

Прежде, чем приступить к обучению нейрокомпьютера на заданном множестве обучающих примеров, надо выбрать способ оценки на всем множестве (а не только на отдельном примере), иначе будет неясно, что означает "хуже" или "лучше" работающая нейронная сеть.

7. ГРАДИЕНТЫ И ДВОЙСТВЕННОСТЬ

7.1. Обучение как оптимизация

Если выбраны множество обучающих примеров и способ вычисления суммарной оценки, задача обучения нейронной сети превращается в задачу многомерной оптимизации, вообще говоря, невыпуклой.

Необходимо оговорить сразу, что в процессе обучения и множество обучающих примеров, и способ вычисления оценки могут меняться — на начальных этапах может быть разумно ограничиться небольшим набором примеров и одним способом оценивания, расширяя множество и меняя оценки в ходе обучения. Тем не менее, такая ситуация, когда заданы множество обучающих примеров и способ оценивания, возникает на всех этапах обучения.

Решать возникающие задачи оптимизации можно различными способами. Существует немало методов, не использующих производных. Так, например, движение в случайному направлении, а также различные модификации этого метода не требуют значительной дополнительной памяти (по сравнению с задачами функционирования). Но все же методы, использующие производные, работают обычно быстрее.

Как вычислить производные функции оценки по параметрам сети? Градиент — огромный набор чисел и, если вычислять его последовательно, то это потребует очень большого времени. Естественно возникает вопрос: нельзя ли использовать саму нейронную сеть для вычисления градиента.

7.2. Двойственное функционирование сетей автоматов при вычислении градиентов сложных функций

Основную идею двойственного функционирования можно понять уже на простейшем примере. Рассмотрим вычисление производной сложной функции. Пусть заданы функции одной переменной f_1, f_2, \dots, f_n . Образуем из них сложную функцию

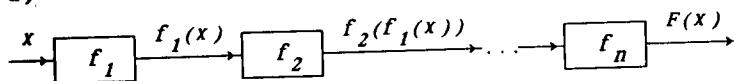
$$F(x) = f_n(f_{n-1}(\dots(f_1(x))\dots)) \quad (25)$$

можно представить вычисление $F(x)$ как результат работы п автоматов, каждый из которых имеет один вход и выдает на выходе значение $f_i(A)$, где A — входной сигнал (рис. 3.а). Чтобы построить систему автоматов, вычисляющую $F'(x)$, надо дополнить исходные автоматы такими, которые вычисляют функции $f'_i(A)$, где A — входной сигнал, и еще цепочкой из $n-1$ одинаковых автоматов, имеющих по два выхода, по одному выходу и подающих на выход произведение входов. Тогда формулу производной сложной функции

$$\frac{dF}{dx} = f'_n(f_{n-1}(\dots(x))\dots)f'_{n-1}(\dots(x))\dots x \dots \quad (26)$$

можно реализовать с помощью сети автоматов, изображенной на рис. 3.б. Сначала по этой схеме вычисления идут слева направо: на входы f_1 и f'_1 подаются значения x , после вычислений f_1 этот входной сигнал подается на входы f_2 и f'_2 и т.д. В конце цепочки оказываются вычисленными все $f_1(f_{1-1}(\dots))$ и $f'_1(f_{1-1}(\dots))$.

а)



б)

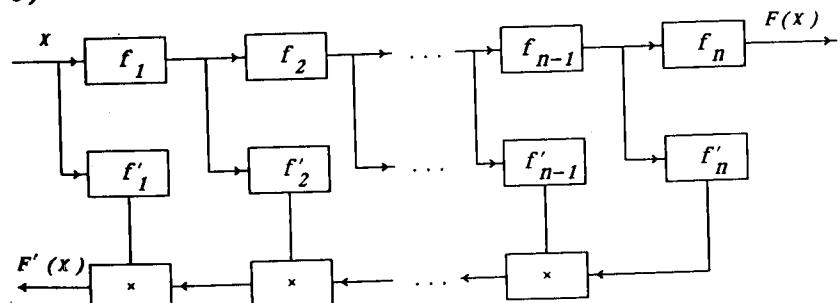


Рис. 3. Последовательность автоматов для вычисления сложной функции (а) и ее производной (б); \square изображает автомат, подающий на выход произведение входов.

Тем самым, для каждого автомата нижней строки, кроме

самого правого, оказывается сформированным по одному значению входа, а у самого правого – оба, поэтому он может сработать и начать передачу сигнала справа налево. Это обратное движение есть последовательное вычисление попарных произведений и передача их налево. В конце получаем dF/dx . дальнейшее содержание этого раздела – обобщение схем прямого и обратного функционирования (см. рис. 3) на сложные функции многих переменных.

Обратимся сначала к простым (не сложным) функциям многих переменных. Переменные, от которых зависит функция f , разобьем на две группы: параметры $\alpha = (\alpha_1, \dots, \alpha_k)$ и собственно переменные $x = (x_1, \dots, x_n)$. Смысл этого разбиения таков: если речь идет об автомате, вычисляющем f , то значения параметров хранятся при автомате (образно говоря, описывают положения "рукояток настройки"), а значения переменных подаются на входы и каждый раз – новые. Для нейронной сети это разбиение соответствует выделению параметров нейронов и весов синапсов – с одной стороны – и входных сигналов этих элементов – с другой. Для функций общего вида принципиального различия между параметрами и переменными, конечно, нет.

Будем изображать автомат, вычисляющий простую функцию, как показано на рис. 4.

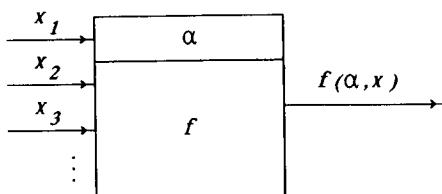


Рис. 4. Автомат, вычисляющий функцию нескольких переменных $f(\alpha, x)$.

Для вычисления сложной функции параметров $\alpha = (\alpha_1, \dots, \alpha_k)$ и переменных $x = (x_1, \dots, x_n)$ задается некоторое количество N простых функций f_i ($i = 1, \dots, N$) и правило подстановки аргументов. Среди функций f_i возможны совпадающие. Обозначим n_i – число переменных, а k_i – число параметров, от которых зависит f_i . Подчеркнем, что на входы f_i в качестве переменных могут подаваться не только x_i , но также и результаты вычисления других функций f_j .

Чтобы построить процесс вычисления для сложной функции, надо указать порядок, в котором вычисляются f_i , и способ формирования входов каждого автомата. Для единобразия обозначений входов удобно в список функций f_i включить независимые переменные $x: f_i = x_i$ ($i = 1, \dots, n$). Эти первые n "функций" не имеют параметров и вообще не вычисляются. Их объединяют с остальными f_i то, что они в равной степени могут подаваться на вход других автоматов. Далее используем такую инициализацию, изменив соответствен N .

Предполагаем, что порядок вычисления f_i согласован с выбранной нумерацией: если вычислены f_1, \dots, f_{i-1} , то определены все входы f_i .

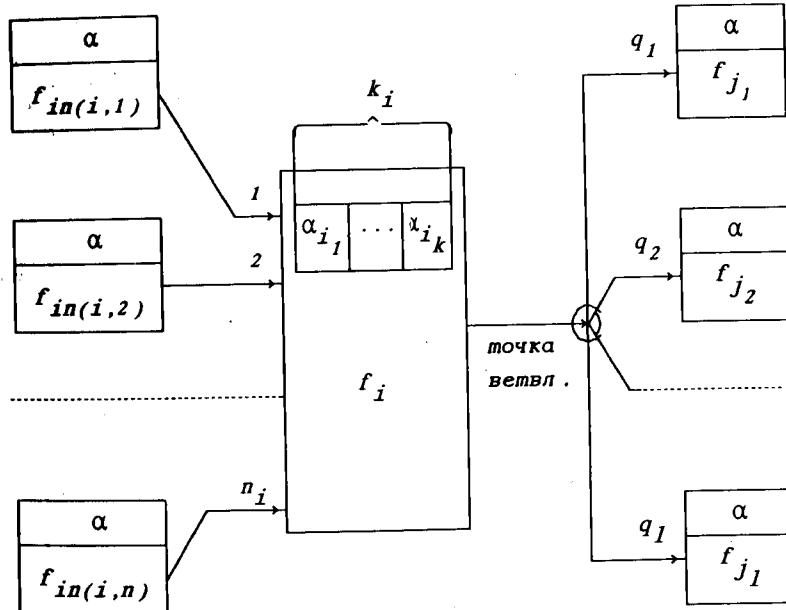


Рис. 5. Автомат, вычисляющий f_i с указанием адресов входов, выхода и точки ветвления; q_p ($p = 1, \dots, l$) – номера тех входов f_{j_1} , на которые подается выходной сигнал f_i ; при этом $((j_p, q_p) | p = 1, \dots, l) = out(i)$.

Каждому входу f_i соответствует "адрес входа" – номер j

того f_j , выход которого подается на данный вход. Адрес q -го входа f_i обозначим $in(i,q)$. Функция $in(i,q)$ определена при $n < i \leq n$ и $1 \leq q \leq n_i$. Ее значения должны удовлетворять неравенству $in(i,q) < i$. (27)

которое означает, что все входы f_i сформированы, если известны значения f_1, \dots, f_{i-1} .

Каждому f_j можно поставить в соответствие совокупность "адресов выхода" - $out(j)$. Она представляет собой совокупность таких пар (i,q) , что $in(i,q) = j$: i - это номер f_i , на q -й вход которого подается результат вычисления f_j . Множество $out(j)$ можно обозначить как $in^{-1}(j)$.

Вычисление f_i с распределением входных и выходных сигналов показано на рис. 5. Важный структурный элемент этой схемы - точка ветвления. Она возникает, если выход f_i подается на несколько различных входов, то есть число элементов в $out(i)$ больше единицы. Подчеркнем, что все сигналы, выходящие из точки ветвления, равны входящему в нее сигналу.

Работу описанной цепочки автоматов можно представить себе так: на ее вход подаются значения переменных x_i ($i=1, \dots, n$). На каждом шаге вычисляется очередная функция f_j от своих аргументов ($j=n+1, \dots, n$). На выходе при данных значениях параметров получаем сложную функцию $F(\alpha, x)$. Она представляет собой результат срабатывания последнего устройства цепочки, вычисляющего f_N (рис. 6).

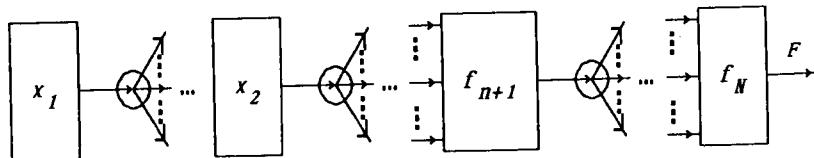


Рис. 6. Цепочка автоматов, вычисляющих сложную функцию $F(\alpha, x)$. Входы f_i формируются из выходов предыдущих элементов цепочки.

Перейдем теперь к схеме, вычисляющей ∇F - производные $\partial F / \partial x_1, \partial F / \partial x_2$. Она будет состоять из двух цепочек - прямой и обратной. Цепочка прямого функционирования кроме вычисления f_i будет включать еще вычисление частных производных этих

функций по их аргументам при тех же значениях аргументов.

Примем обозначения: $f_{i,r}$ - производная f_i по r -му аргументу (входу), f_{i,α_j} - производная f_i по α_j ($f_{i,\alpha_j} = \partial f_i / \partial \alpha_j$). При прямом функционировании значения этих производных вычисляются, запоминаются, но никуда не передаются - они будут использованы при обратном функционировании. Схематически такт прямого функционирования показан на рис. 7.

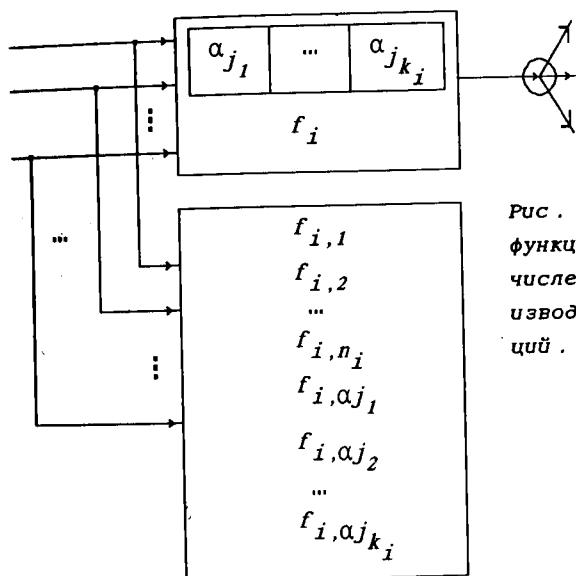


Рис. 7. Такт прямого функционирования с вычислением частных производных простых функций.

Обратное функционирование опишем сначала с помощью формул, а потом зададим схематически. Процесс обратного функционирования заключается в последовательном вычислении цепочки двойственных f_i величин μ_i , начиная с μ_N и кончая μ_1 :

$$\mu_N = 1, \mu_i = \sum_{(j,q) \in out(i)} \mu_j f_{j,q}, \quad (28)$$

здесь суммирование распространяется на все такие пары (j,q) , что величина f_j в прямом функционировании посылается на q -й вход f_i - это и означает, что $(j,q) \in out(i)$ или, что тоже самое, $i = in(j,q)$. Частные производные $f_{j,q}$ были вычислены в

ходе прямого функционирования.

В ходе обратного функционирования вычисляются производные $\frac{\partial F}{\partial x_i}$ - это просто величины μ_i при $i=1, \dots, n$. Для вычисления $\frac{\partial F}{\partial \alpha_j}$ требуется дополнительные операции:

$$\frac{\partial F}{\partial \alpha_p} = \sum_j \mu_j f_{j, \alpha_p} \quad (29)$$

Здесь суммирование ведется по всем тем j , для которых f_j зависит от α_p . Конечно, по существу эта формула совпадает с (28) для μ_i .

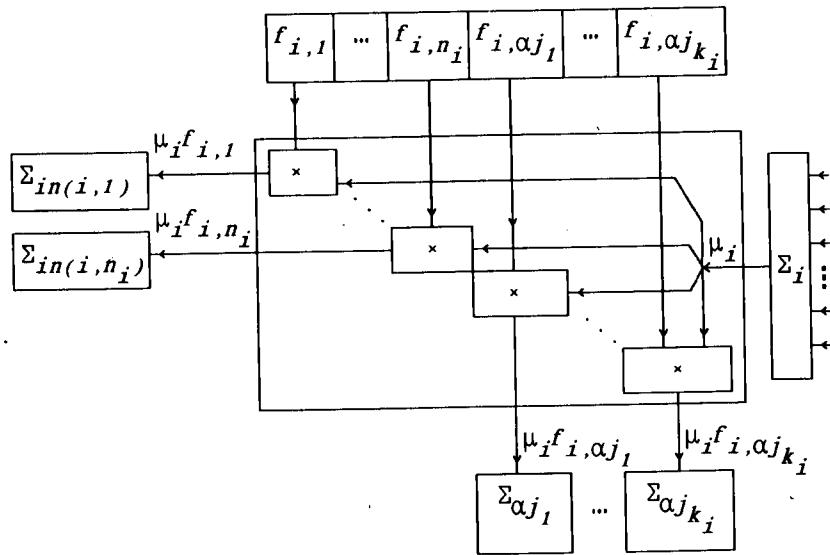


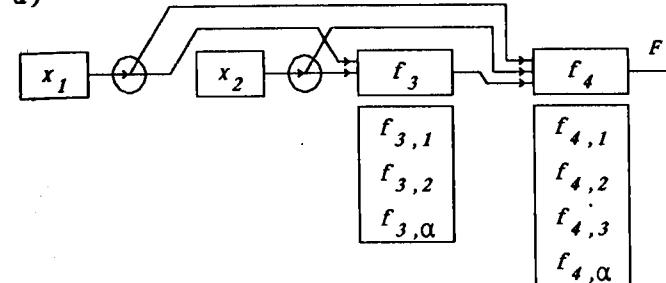
Рис. 8. Такт обратного функционирования; \square - автомат, подающий на выход произведение входов; на входы Σ_i (справа) подаются произведения $\mu_j f_{j,q}$ от Φ_j при $(j,q) \in out(i)$.

При схемной реализации формул (28, 29) умножения будут производить элементы, соответствующие f_j (будем обозначать их Φ_j), а суммирование - сумматоры, соответствующие правым частям. Для формулы (28) сумматор обозначим Σ_i , на его выходе получаем μ_i . Для формулы (29) сумматор обозначим Σ_{α_p} , на его выходе получаем $\frac{\partial F}{\partial \alpha_p}$. Такт обратного функционирования изображен на рис. 8 для $i < N$.

Случай $i=N$ отличается отсутствием сумматора Σ_N - на вход Φ_N подается $\mu_N=1$.

Элемент Φ_i представляет набор из n_i+k_i простых автоматов, каждый из которых имеет два входа, один выход и подает на выход произведение входов (см. рис. 8).

а)



б)

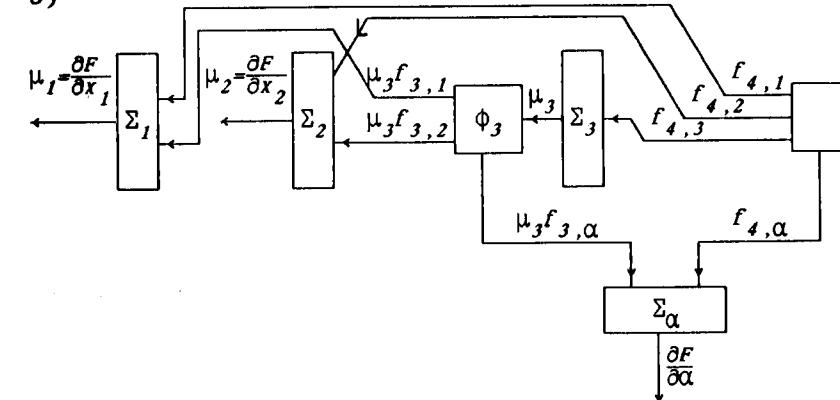


Рис. 9. Прямое (а) и обратное (б) функционирование при вычислении функции F (30) и ее производных.

В качестве входов на каждый из таких автоматов подаются: частная производная f_i , вычисленная в ходе прямого функционирования, и переменная обратного функционирования μ_i . Результат умножения подается на сумматор, определяемый аргументом, по которому взята частная производная f_i . Если это производная по j -му входу, то сигнал отправляется туда, откуда при прямом функционировании приходил вход, то есть на

$\Sigma_{i_0(1,1)}$. Если же это производная по параметру α_j , то сигнал посылается на сумматор Σ_{α_j} .

При обратном функционировании элементы Φ_i , соответствующие независимым переменным ($i=1, \dots, p$), не нужны – для них остаются только сумматоры.

Заметим, что при обратном функционировании выходы Φ_i соответствуют входам f_i , вход Φ_1 – выходу f_1 , а входной сумматор Σ_1 – точке ветвления на выходе f_1 . Входные адреса соответствуют выходным и обратно.

Несколько примеров. Пусть

$$F(\alpha, x_1, x_2) = f_4(\alpha, x_1, x_2, f_3(\alpha, x_1, x_2)). \quad (30)$$

В определении F участвуют две функции: f_3 – от одного параметра и двух переменных и f_4 – от одного параметра и трех переменных. Напомним, что номера 1, 2 зарезервированы за независимыми переменными.

Схемы прямого и обратного функционирования для вычисления F и ее производных показаны на рис. 9.

Другой пример чуть сложнее:

$$F(\alpha_1, \alpha_2, x_1, x_2) = f_5(\alpha_2, x_1, f_4(\alpha_1, x_1, f_3(\alpha_1, x_1, x_2)), f_3(\alpha_1, x_1, x_2)). \quad (31)$$

для него схемы функционирования показаны на рис. 10. Напомним, что в ходе прямого функционирования производные простых функций вычисляются при тех значениях аргументов, которые к ним поступают. Так, на схеме рис. 10

$$f_{5,1} = \frac{\partial f_5(\alpha, y_1, y_2)}{\partial y_1} \left| \begin{array}{l} \alpha = \alpha_2, \\ y_1 = f_4(\alpha_2, x_1, f_3(\alpha_1, x_1, x_2)), \\ y_2 = f_3(\alpha_1, x_1, x_2). \end{array} \right. \quad (32)$$

На рис. 10 и далее мы не указываем аргументов, передаваемых автоматам, вычисляющим $f_{i,j}$ и f_{i,α_j} в ходе прямого функционирования. Всюду предполагается, что это – те же аргументы, что и у f_i .

На схемах рис. 9, 10 видно, что часть сумматоров Σ_i при обратном функционировании служит просто звеном, передающим единственный сигнал без всякого изменения. Это происходит

тогда, когда выходной сигнал f_i передается при прямом функционировании по единственному адресу без ветвления. Тривиальное ветвление – тривиальный сумматор.

До сих пор речь шла о последовательном режиме вычисления сложной функции и соответственно последовательном обратном функционировании. В этом месте возможны усовершенствования, а именно – переход к параллельно-последовательным вычислениям. Если иметь в виду нашу основную задачу – обучение нейрокомпьютеров, то такой переход необходим.

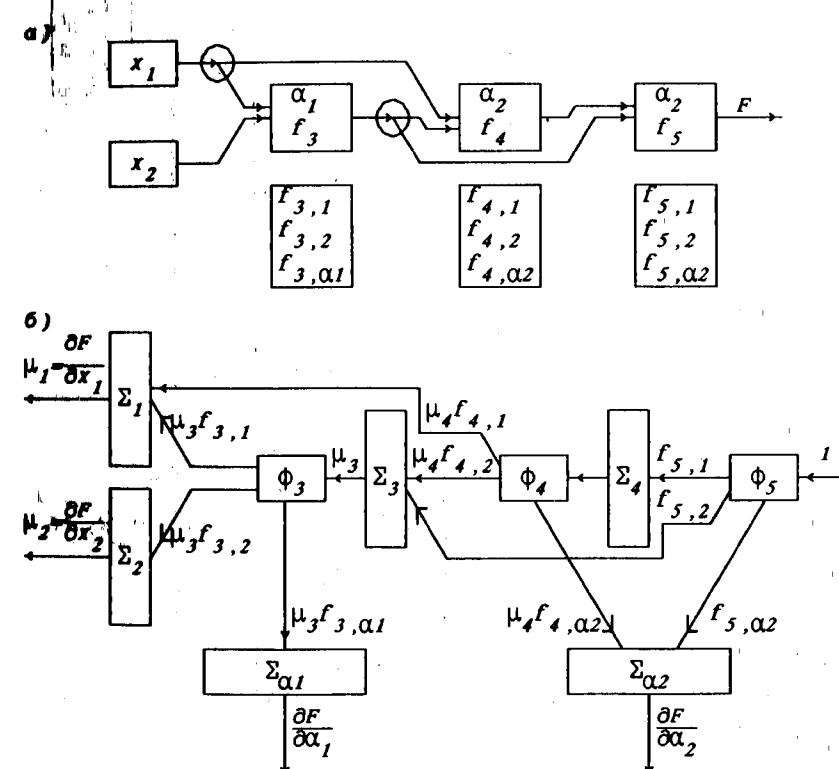


Рис. 10. Прямое и обратное функционирование при вычислении функции F (31) и ее производных.

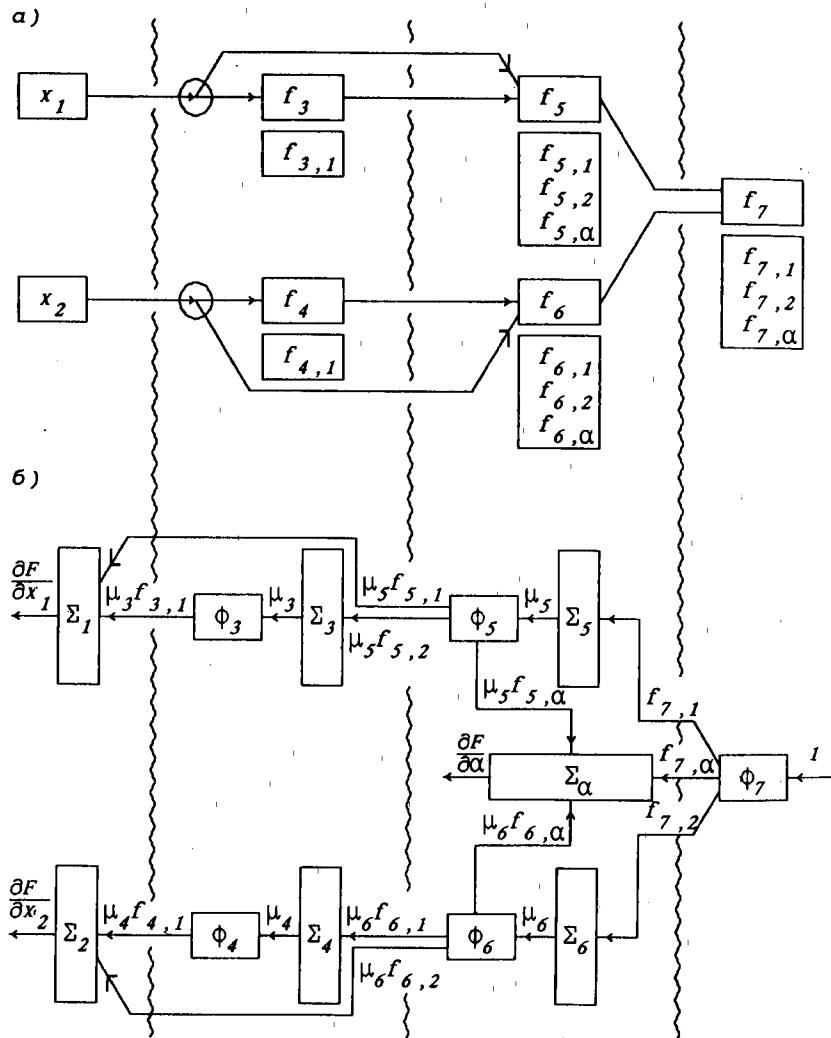


Рис. 11. Послойное прямое (а) и обратное (б) функционирование при вычислении функции F (34) и ее производных. { - линия раздела слоев.

мы будем рассматривать цепочку функций f_1, \dots, f_n участвующую в вычислении сложной функции F , и ее разбиение на подмножества. Подмножество цепочки будем задавать набором индексов J . Пусть задано разбиение цепочки на несколько (непересекающихся) множеств с наборами индексов J_1, \dots, J_p и для любых $i \in J_r$, $1 \leq j \leq n_i$ выполнено

$$in_i(j) \in J_q \text{ при некотором } q < r. \quad (33)$$

Последнее означает, что для вычисления f_i ($i \in J_r$) требуются значения только тех функций, номера которых лежат в J_q ($q=1, \dots, r-1$). Если такое разбиение возможно, то функции, номера которых лежат в одном J_r , можно вычислять независимо и следовательно одновременно. Схема вычисления F приобретает послойный вид - в одном слое лежат те f_i , номера которых принадлежат одному J_r . Вычисления в одном слое производятся параллельно, между слоями есть последовательные связи. Аналогично, послойную структуру приобретает процесс обратного функционирования.

Ограничимся одним простым примером. Пусть

$$F(\alpha, x_1, x_2) = f_7(\alpha, f_6(\alpha, f_4(x_2), x_2), f_5(\alpha, x_1, f_3(x_1))). \quad (34)$$

На первом слое стоят подача переменных x_1, x_2 , на втором - вычисление f_3, f_4 , на третьем - f_5, f_6 , на четвертом - f_7 . Аналогично, только в обратном порядке идет обратное функционирование (рис. 11).

7.3. Двойственное функционирование сетей автоматов. Обобщения

Цель этого раздела - перейти от цепочек автоматов, описанных в предыдущем разделе, к более общим сетям. Первое обобщение - введение сумматоров перед каждым входом всех элементов. При обратном функционировании эти сумматоры передадут в точки ветвления.

Сразу будем рассматривать послойно организованные сети. Каждый элемент вычисляет некоторую функцию входных сигналов и характеризуется парой номеров i, j - i -номер слоя, j - номер элемента в слое. Обозначим n_{ij} число входов i, j -го элемента, f_{ij} - саму функцию. Для каждой тройки i, j, q (i - номер слоя, j - номер элемента в слое, q - номер входа) определено

множество адресов входа $in(i, j, q)$. Множество $in(i, j, q)$ – множество пар (p, r) (номер слоя, номер элемента в слое), причем $p < i$ – сигналы подаются с предыдущих слоев. На q -й вход i, j -го элемента подается сумма выходных значений f_{pr} p, r -х элементов

$$A_{1j} = \sum_{(p, r) \in in(i, j, q)} f_{pr}. \quad (35)$$

Суммирование производится по всем (p, r) из $in(i, j, q)$. На нулевом слое A не вычисляется, f_{0j} – значение j -го независимого переменного. На выходе i, j -го элемента получаем число

$$f_{ij} = f_{ij}(A_{1j1}, \dots, A_{1jn_{ij}}). \quad (36)$$

Функционирование i, j -го элемента можно представить схемой – см. рис. 12.

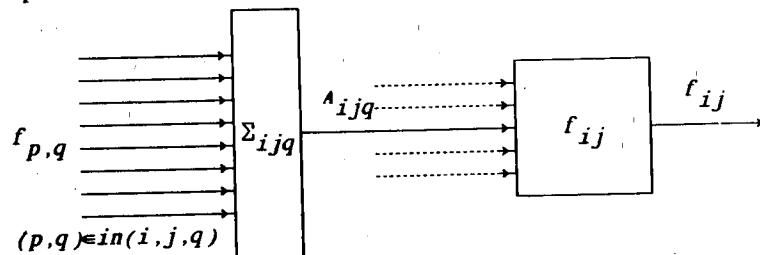


Рис. 12. Функционирование элемента с сумматорами перед входами.

На последнем N -м слое стоит один элемент f_n . Позднее при рассмотрении нейронных сетей он превратится в вычислитель оценки.

Кроме того, каждый элемент f_{ij} зависит от некоторого набора параметров. Эти параметры α_k могут быть общими для разных элементов, а могут быть и различными.

Пусть на вход сети поданы переменные x_i ($i = 1, \dots, n$) и определены параметры α . Тогда после срабатывания всех слоев сети получаем на выходе f_n . Это значение есть некоторая функция $F(x, \alpha)$. Ее частные производные можно найти, соответствующим образом организовав прямое и обратное функционирование. Для этого в ходе прямого функционирования кроме f_{ij} надо вычислять еще набор производных

$$f_{1j,q} = \frac{\partial f_{1j}}{\partial A_{1j,q}}; \\ f_{1j,\alpha_p} = \frac{\partial f_{1j}}{\partial \alpha_p}.$$

При обратном функционировании сумматоры заменяются точками ветвления, точки ветвления – сумматорами. Вычисляются переменные обратного функционирования μ_{ij} , двойственные f_{ij} :

$$\mu_n = 1, \mu_{ij} = \sum_{r, s, l: (i, j) \in in(r, s, l)} \mu_{rs} f_{rs, l}. \quad (38)$$

Суммирование распространяется здесь на все тройки (r, s, l) (номер слоя, номер элемента в слое, номер входа), для которых при прямом функционировании f_{ij} подается на входной сумматор i -го входа f_{rs} .

Множество $in(i, j, q)$ обладает свойством:

$$p < i \text{ для любых } (p, r) \in in(i, j, q). \quad (39)$$

Поэтому для всех слагаемых в (38) $r > i$, вычисления суммы можно проводить внутри каждого слоя параллельно, а в последовательности слоев – от большего номера к меньшему. Каждое произведение, входящее в сумму (38), можно вычислять в соответствующем r, s -ом элементе обратного функционирования. Схема такого элемента представлена на рис. 13. Отличие от элемента, показанного на рис. 8, – наличие разветвлений на выходах. Эти разветвления двойственны сумматорам при входах прямого функционирования. Обозначение сумматоров обратного функционирования отличается от сумматоров прямого штрихом (Σ' вместо Σ).

При вычислении μ_{ij} , спустившись до нулевого слоя, получаем

$$\mu_{0j} = \frac{\partial F}{\partial x_j}. \quad (40)$$

Произведение $\partial F / \partial x_j$ накапливается в специальных сумматорах так же, как и в предыдущем разделе.

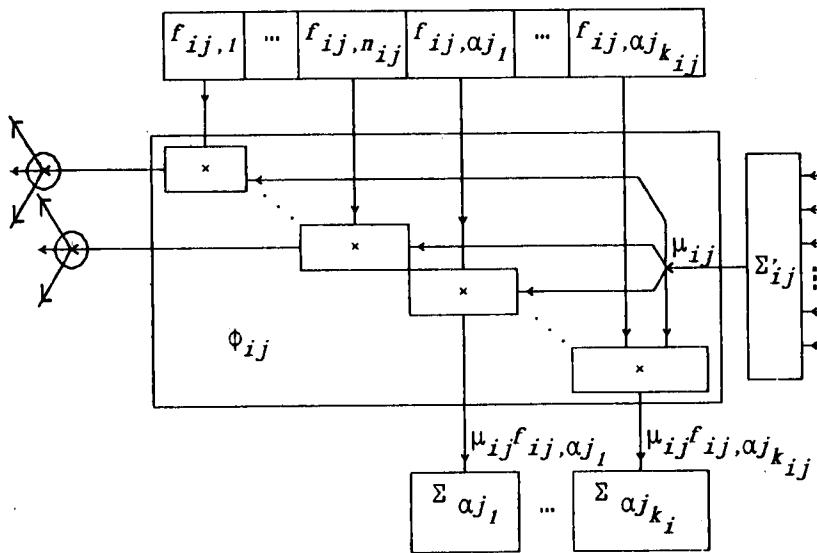


Рис. 13. Схема элемента обратного функционирования Φ_{ij} при наличии сумматоров при входах прямого функционирования. На сумматор Σ'_{ij} подаются $\mu_{rs} f_{rs,1}$ от таких Φ_{rs} , что $(i,j) \in \text{in}(r,s,1)$. Получаемые на выходе (слева) сигналы $\mu_{ij} f_{ij,\alpha j_k}$ передаются через точки ветвления ко всем сумматорам Σ_{pq} , для которых $(p,q) \in \text{in}(i,j,k)$.

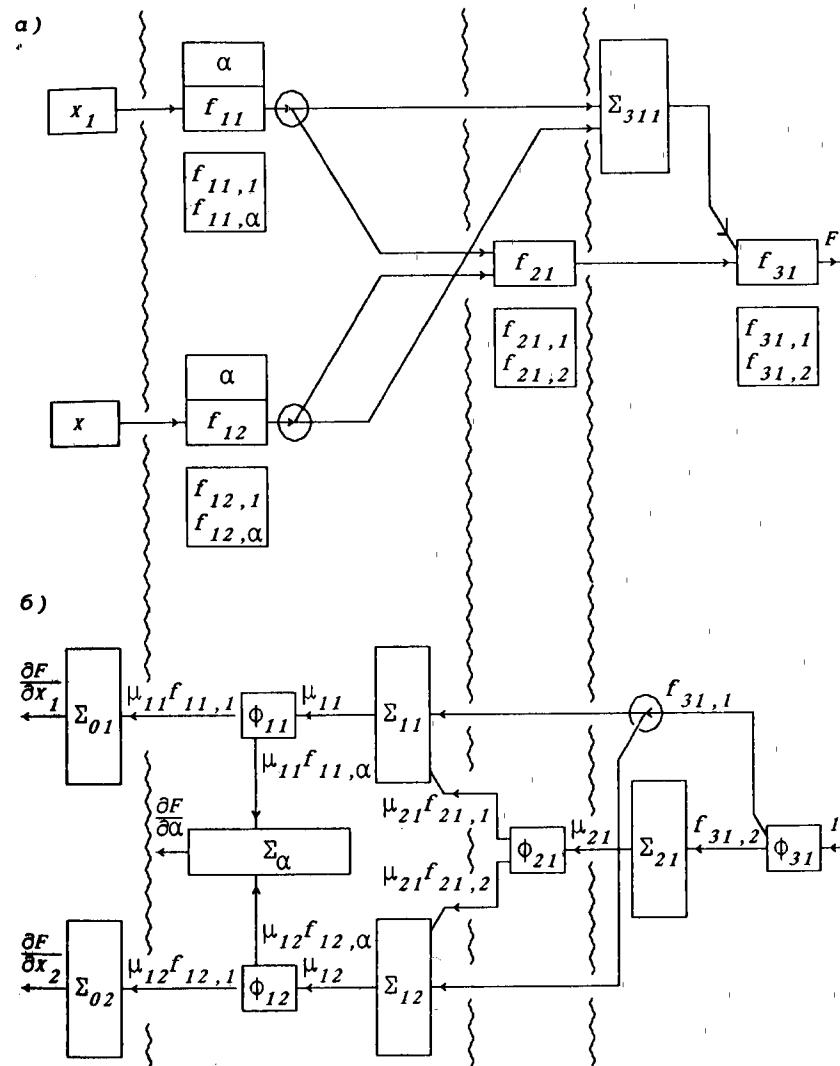


Рис. 14. Схема прямого (а) и обратного (б) функционирования при наличии сумматора на входе f_{31} . } - линия раздела слоев.

Приведем простой пример:

$$F(\alpha, x_1, x_2) = f_{31}(f_{11}(\alpha, x_1) + f_{12}(\alpha, x_2), f_{21}(f_{11}(\alpha, x_1), f_{12}(\alpha, x_2))) \quad (41)$$

Прямое и обратное функционирование для этой F показаны на

рис. 14. Сумматор здесь стоит на первом входе f_{31} , соответственно при обратном функционировании первый выходной сигнал ϕ_{31} разветвляется (см. рис. 14).

Обратимся теперь к проблеме обучения сети автоматов, синхронно функционирующих в дискретном времени. Для обучения желательно вычислять градиент ошибки - частные производные этой функции по параметрам. Одновременно могут быть вычислены частные производные этой функции по входным сигналам. Об использовании таких производных будет рассказано в следующем разделе. Опишем сначала функционирование сети, а потом построим двойственный процесс.

Каждый автомат имеет несколько входов (n), несколько выходов (p) и конечный набор (s) параметров состояния. Он вычисляет $s+p$ функций от $n+s$ переменных. Аргументы этих функций - входные сигналы (их n) и текущие параметры состояния (их s). Значение функций - выходные сигналы (их p) и параметры состояния на следующем шаге (их s). Каждый такой автомат можно представить как систему из $s+p$ более простых автоматов (рис. 15). Эти простые автоматы вычисляют по одной функции от $n+s$ переменных. Смена состояний достигается за счет того, что часть значений этих функций на следующем шаге становится аргументами - так соединены автоматы (см. рис. 15).

Таким образом, без потери общности можно рассматривать сеть автоматов как набор устройств, каждое из которых вычисляет функцию нескольких переменных $f_i(\alpha, A_1, \dots, A_n)$ ($i=1, \dots, N$).

Каждая из этих функций может зависеть от параметров $\alpha = (\alpha_1, \dots, \alpha_k)$. Конечно, в конкретных приложениях f_i зависит только от некоторых из α_j и это может привести к существенным упрощениям, но пока ограничимся общим случаем.

На каждом шаге функционирования всем функциям f_i подаются входные сигналы A_{ij} ($j=1, \dots, n$). Значение A_{ij} называем j -м входным сигналом f_i . Эти входные сигналы снимаются с входных сумматоров Σ_{ij} . На входные сумматоры могут поступать сигналы двух видов - значения некоторых f_k на предыдущих шагах функционирования и внешние сигналы.

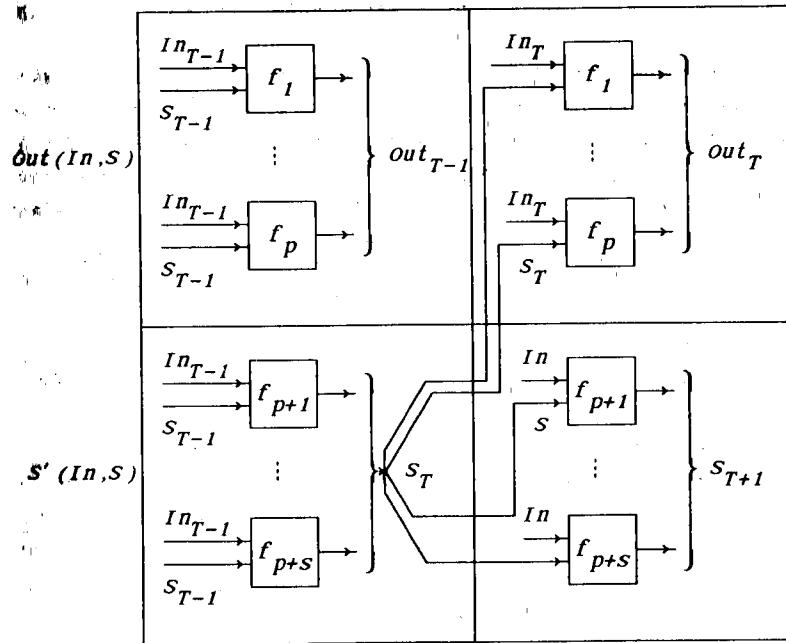
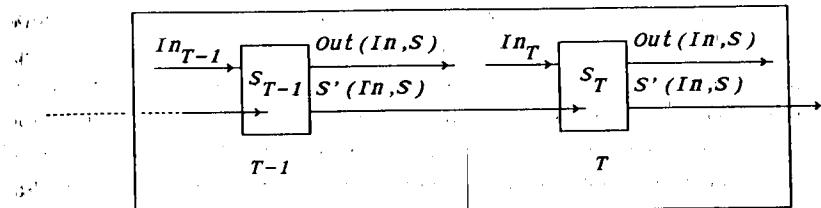


Рис. 15. Функционирование автоматов в последовательные моменты времени: а - в общем виде, б - в разбиении на более простые автоматы. Обозначения:

$In = (x_1, \dots, x_n)$, $S = (x_{n+1}, \dots, x_{n+s})$;

$Out = f_i(in, S)$ ($i=1, \dots, p$); $S' = f_i(in, S)$ ($i=p+1, \dots, p+s$);

T - дискретное время.

Обозначим T текущее дискретное время. Оно может принимать целые значения. Величины, относящиеся к времени T ,

будем отмечать индексом T .

В каждый момент времени системе подается входной сигнал $x_t = (x_{1t}, \dots, x_{nt})$. Входным сумматорам Σ_{ij} соответствует набор адресов входа $in(i, j)$. Он состоит из двух множеств $in_1(i, j)$ – номера тех x_p , которые подаются на Σ_{ij} , $in_s(i, j)$ – номера тех f_i , значения которых на предыдущем шаге $f_{i, t-1}$ подаются на Σ_{ij} . В соответствии с введенными обозначениями значение на T -м шаге A_{ijT} равно

$$A_{ijT} = \sum_{p \in in_1(i, j)} x_p + \sum_{q \in in_s(i, j)} f_{q, t-1}. \quad (42)$$

Индексы I и S у in соответствуют внешнему входу (In) и состоянию (S).

Возможно обобщение, состоящее в том, что на входные сумматоры в момент T подаются сигналы не только из набора текущих внешних входных x_t и предыдущих выходных f_{t-1} , но и из $x_{t-1}, \dots, f_{t-2}, \dots$ Формально оно строится так. Каждому входному сумматору Σ_{ij} ставится в соответствие множество адресов входа $in(i, j)$, состоящее из множеств $in_{10}(i, j)$, $in_{11}(i, j), \dots, in_{s1}(i, j)$, $in_{s2}(i, j), \dots$. Величина j -го входа f_i на шаге T формируется так:

$$A_{ijT} = \sum_{r=0}^{R_I} \sum_{p \in in_{1r}(i, j)} x_{p, T-r} + \sum_{r=0}^{R_S} \sum_{q \in in_{s_r}(i, j)} f_{q, T-r}. \quad (43)$$

Верхние пределы суммирования по r – времена запаздывания.

Еще одно обобщение. Сама система автоматов может иметь послойную структуру – состоять из поочередно срабатывающих подсистем. Вспомниме функционирование нейронной сети: нейроны \rightarrow синапсы \rightarrow сумматоры \rightarrow нейроны \dots . Такт функционирования такой системы разбивается на несколько более мелких единиц – тактов подсистем (слоев). Сигнал на Σ_{ij} может подаваться не только с предыдущего такта системы, но и с предыдущих слоев того же такта. Формализуем и эту ситуацию. У f и A появится еще один индекс – номер слоя.

Итак, системы с дополнительной послойной структурой (последовательно-параллельные системы). Они также составлены из устройств, вычисляющих функции нескольких переменных. Эти устройства объединены в слои. Каждая функция задается двумя номерами: f_{ij} , i – номер слоя, j – номер функции в слое.

Каждому входу соответствует сумматор Σ_{ijk} (k – номер входа). Переменная, выдаваемая сумматором Σ_{ijk} на k -й вход f_{ij} , обозначается A_{ijk} . Каждому сумматору Σ_{ijk} соответствует набор адресов $in(i, j, k)$ составленный из множеств $in_{10}(i, j, k)$, $in_{11}(i, j, k), \dots, in_{s0}(i, j, k)$, $in_{s1}(i, j, k), \dots$. Элементы $in_{1r}(i, j, k)$ – номера внешних входных сигналов с шагом $T-r$, подаваемых на вход Σ_{ijk} в момент T . Элементы $in_{sr}(i, j, k)$ – пары индексов p, q , для которых выходной сигнал $f_{pq, T-r}$ (момент $-T-r$) подается на вход Σ_{ijk} в момент T . Основное отличие от предыдущего случая – наличие $in_{s0}(i, j, k)$. Элементы этого множества должны удовлетворять неравенству:

$$\text{если } (p, r) \in in_{s0}(i, j, k) \text{ то } p < i. \quad (44)$$

Оно означает, что сигналы, берущиеся в Σ_{ijk} на том же такте системы, должны приходить с предшествующих слоев.

Аналог (43) теперь принимает вид

$$A_{ijT} = \sum_{r=0}^{R_I} \sum_{p \in in_{1r}(i, j, k)} x_{p, T-r} + \sum_{r=0}^{R_S} \sum_{q \in in_{s_r}(i, j, k)} f_{pq, T-r}. \quad (45)$$

Будем далее в этом разделе рассматривать сети и с запаздыванием, и с послойной организацией. При необходимости будем возвращаться к менее громоздким частным случаям (42, 43).

Рассмотрим общую сеть автоматов в режиме периодического функционирования. В начальный момент ($T=1$) она должна обратиться за входными сигналами к предыдущим тактам функционирования ($T=0, -1, \dots$), а они отсутствуют. Поэтому работа сети в режиме периодического функционирования предполагает фазу инициации – нужно задать числа, которые будут в соответствии с (45) посыпаться на входы Σ_{ijk} . Это – числа x_{p1}, f_{pqm} ($l=0, -1, \dots, 1-R_I; m=0, -1, \dots, 1-R_S$). Даже в отсутствие запаздывания нужно задать f_{pqm} ($m=0$).

Существуют два способа инициации.

1. **Стандартная инициация.** Неизвестные сигналы с отсутствующих предыдущих тактов определяются некоторым стандартным образом. Чаще всего они полагаются нулевыми.
2. **Обучаемые параметры инициации.** Сигналы, соответствующие недостающим предыдущим тактам, рассматриваются как дополнительные подстраиваемые параметры. Они участвуют в процессе обучения наравне с другими параметрами.

Итак, пусть сеть автоматов инициирована - заданы недостающие величины с предыдущих шагов. Всюду далее они называются параметрами инициации. Потом подаются входные сигналы x_1 , после чего последовательно срабатывают все слои сети, потом подаются входные сигналы x_2 и т.д. до последнего срабатывания при $T=\theta$. На этом цикл функционирования ("акт" в отличие от такта) кончается. Теперь нужно оценить функционирование сети и вычислить производные оценки по всем подстроенным параметрам.

Оценка цикла функционирования есть некоторый функционал от входных и выходных сигналов. В качестве выходных могут использоваться любые $f_{1,1T}$ ($T=1, \dots, \theta$). Здесь мы рассмотрим все $f_{1,1T}$ ($T=1, \dots, \theta$) как выходные сигналы. Возможно упрощение из-за того, что не все $f_{1,1T}$ используются как выходные сигналы. Оно приведет к уменьшению слагаемых в дальнейших формулах.

Оценка - функция от всей совокупности переменных x_{1T} , $f_{1,1T}$: $H=H(x_{1T}, \{f_{1,1T}\})$.

Функционирование сети с вычислением оценки можно представить себе как вычисление сложной функции, описанное в предыдущем разделе. Вычисление происходит послойно, входные независимые переменные - это параметры инициации и входные сигналы x_{1T} .

Автомат, вычисляющий $f_{1,1}$, в разные моменты времени T можно представить как разные экземпляры одного автомата, существующие одновременно. При этом временное функционирование сети представляется как последовательное срабатывание нескольких экземпляров сети. Такое одновременное представление удобно. Номером слоя, к которому принадлежит $f_{1,1T}$, становится пара (i, T) со следующим определение порядка:

$$(i_1, T_1) > (i_2, T_2) \text{ если } T_1 > T_2 \text{ или } (T_1 = T_2 \text{ и } i_1 > i_2). \quad (46)$$

Знак $\dots > \dots$ означает "... следует за ..." в функционировании".

Кроме функций $f_{1,1T}$, в новом слое, следующем за всеми этими функциями, появляется автомат f_n , вычисляющий функционал оценки. На его входы подаются x_{1T} , $f_{1,1T}$ ($T=1, \dots, \theta$), на выходе получаем оценку H .

Представив период функционирования сети таким образом, можно воспользоваться с соответствующими переобозначениями

формулами (38) для описания обратного функционирования и вычисления производных H по параметрам α , параметрам инициации и входным сигналам x_{1T} . Каждая из производных будет накапливаться в соответствующем сумматоре.

Итак, цикл функционирования и его оценивание для сети автоматов представлен как вычисление сложной функции послойно организованной сетью автоматов. При этом временные такты располагаются друг за другом пространственно, аналогично последовательным слоям. Обратное функционирование проходит в обратном направлении - от f_n к выходам, а результате в специальных сумматорах накапливаются значения градиента оценки.

7.4. Смысл двойственных переменных

Переменные обратного функционирования μ (38) появляются как вспомогательные при вычислении производных сложной функции. Переменные такого типа появляются не случайно. Они постоянно возникают в задачах оптимизации и являются множителями лагранжа.

Мы вводим μ , исходя из правил дифференцирования сложной функции. Возможен другой путь, связанный с переходом от функции лагранжа к функции Гамильтона. Изложим его и параллельно получим ряд дальнейших обобщений.

Для всех сетей автоматов, встретившихся нам в предыдущих разделах, можно выделить три группы переменных: внешние входные сигналы x , переменные функционирования - значения f и переменные обучения α (многоточиями заменяются различные наборы индексов). Объединим их в две группы - вычисляемые величины y - значения f и задаваемые - β . Упростим индексацию, перенумеровав f и β натуральными числами: $f_1, \dots, f_n; \beta_1, \dots, \beta_n$.

Пусть функционирование системы задается набором из N уравнений

$$\Phi_i (y_1, \dots, y_n, \beta_1, \dots, \beta_n) = 0. \quad (47)$$

Простейший пример такого уравнения

$$\Phi_i = f_i (\beta_{i_1}, \dots, \beta_{i_n}) - y_i, \quad i = \text{in}(i, j) < i. \quad (48)$$

Предполагается, что система уравнений (47) задает способ вычисления u_1 .

Пусть имеется целевая функция (лагранжиан) $H(u_1, \dots, u_n, \beta_1, \dots, \beta_n)$. Эта функция зависит от β_i и явно, и неявно — через переменные функционирования u . Если представить, что уравнения (47) разрешены относительно всех u_i ($u_1 = u_1(\beta)$), то H можно представить как функцию от β :

$$H = H_1(\beta) = H(u_1(\beta), \dots, u_n(\beta), \beta). \quad (49)$$

где β — вектор с компонентами β_i .

Для задачи обучения требуется найти производные $\partial H_1(\beta) / \partial \beta_i$. Непосредственно и явно это сделать трудно. Поступим по-другому. Введем новые переменные μ_1, \dots, μ_n — множители лагранжа — и производящую функцию

$$W(u, \beta, \mu) = H(u, \beta) + \sum_i \mu_i \psi_i(u, \beta). \quad (50)$$

В функции W u , β и μ — независимые переменные. Уравнения (47) можно записать как

$$\frac{\partial W}{\partial \mu_i} = 0. \quad (51)$$

Заметим, что для тех u , β , которые удовлетворяют уравнениям (47), при любых μ

$$W(u, \beta, \mu) \equiv H(u, \beta). \quad (52)$$

Это означает, что для истинных значений переменных функционирования u при данных β функция $W(u, \beta, \mu)$ совпадает с оценкой.

Попытаемся подобрать такую зависимость $\mu_i(\beta)$, чтобы для D_1 , используя (52), получить наиболее простые выражения. На многообразии решений (51)

$$\begin{aligned} D_1 &= \frac{\partial H}{\partial \beta_1} + \sum_j \frac{\partial H}{\partial u_j} \frac{\partial u_1}{\partial \beta_1} = \\ &= \frac{\partial W}{\partial \beta_1} + \sum_j \frac{\partial W}{\partial u_j} \frac{\partial u_1}{\partial \beta_1} + \sum_j \frac{\partial W}{\partial \mu_j} \frac{\partial \mu_1}{\partial \beta_1} = \\ &= \sum_j \left[\frac{\partial H}{\partial u_j} + \sum_k \mu_k \frac{\partial \psi_k}{\partial u_j} \right] \frac{\partial u_1}{\partial \beta_1} + \sum_j \mu_j \frac{\partial \psi_j}{\partial \beta_1} + \frac{\partial H}{\partial \beta_1}. \end{aligned} \quad (53)$$

Если выбрать такие μ_k , что слагаемые в первой сумме выражения (53) обратятся в нуль, то формула для D_1 резко упростится. Положим поэтому

$$\frac{\partial H}{\partial u_j} + \sum_k \mu_k \frac{\partial \psi_k}{\partial u_j} = 0. \quad (54)$$

Это — система уравнений для определения μ_k ($k=1, \dots, n$). Если μ_k определены согласно (54), то

$$D_1 = \frac{\partial H}{\partial \beta_1} + \sum_j \mu_j \frac{\partial \psi_j}{\partial \beta_1}; \quad (55)$$

это — в точности те выражения (с точностью до переобозначения индексов), которыми мы пользовались при поиске производных сложных функций, вычисляемых сетью автоматов. В наших вычислениях мы пользовались явным описанием функционирования, что соответствует выбору ψ_1 в виде (48). Метод множителей лагранжа допускает и более неявные описания сетей.

7.5. Обратное функционирование нейронных сетей

Предыдущие два раздела были посвящены обобщениям. Формулы для быстрого вычисления производных сложной функции были распространены на широкий класс автоматов, функционирующих в периодическом режиме "начальное состояние — задание — ответ — оценка ответа".

Мы научились вычислять производные функции оценки по входным сигналам и подстраиваемым параметрам. Вычисление производится в параллельно-последовательном режиме обратного функционирования. Для того, чтобы его осуществить, необходимо нагружить прямое функционирование дополнительным вычислением частных производных простых (не сложных) функций по их входным сигналам и тем параметрам, от которых эти функции зависят.

В данном разделе мы займемся частными случаями, в которых формулы приобретают наиболее простой вид, и применением формул к конкретным конструкциям нейронных сетей.

Первый вопрос: для каких функций нагрузка на прямое функционирование, необходимое для обратного, минимальна? Вот несколько вариантов ответов.

1. Линейные функции входов:

$$f_1(\alpha, a_1, \dots, a_{n_1}) = \sum_j a_j(\alpha) A_j. \quad (56)$$

Здесь производные по значениям входов не зависят от них:

$$f_{1,j} = a_j(\alpha). \quad (57)$$

Поэтому, например, в различных тактах функционирования т величины $f_{1,1T}$ одинаковы и определяются формулами (57). Ситуация еще более упрощается, когда каждый коэффициент a_j в (56) – линейная функция одного параметра:

$$\begin{aligned} a_j &= b_j + c_j \alpha_{kj}, \\ f_{1,j} &= b_j + c_j \alpha_{kj}, \\ f_{1,\alpha_k} &= c_j A_j. \end{aligned} \quad (58)$$

Здесь производная по параметру пропорциональна значению A_j с постоянным коэффициентом.

Самый простой случай – суммирование входных сигналов. Для такого элемента, как уже неоднократно говорилось, прямое функционирование происходит без нагрузки, а при обратном он заменяется на двойственный элемент – точку ветвления.

2. Элементы с одним входом. Для них дополнительная нагрузка на прямое функционирование уменьшается за счет того, что вычислять нужно только одну величину типа $f_{1,j}$ (но, быть может, много величин типа f_{1,α_j}).

3. Каждая функция зависит от своих параметров α . В этом случае упрощается обратное функционирование – нет необходимости заводить специальные сумматоры для накопления производной оценки по α_i . Эти производные могут храниться в специальных элементах памяти, связанных с автоматами, вычисляющими f_i . Такое хранение тем более удобно, что при обучении сети нужно изменять α , например, в направлении антиградиента. Если каждая функция f_i зависит только от своих α , то производные оценки по α не нужны для обучения устройств, вычисляющих другие f_j . Конечно, сохраняются сумматоры, соответствующие параметрам инициации и входным сигналам.

4. Линейные функции параметров α ($A = (A_1, \dots, A_n)$):

$$f_1(\alpha_{j_1}, \dots, \alpha_{j_k}, A) = \sum_p a_p(A) \alpha_{j_p} + b(A),$$

$$f_{1,\alpha_p} = a_p(A). \quad (59)$$

Если величины $a_p(A)$ вычисляются при вычислении функции $f_{1,j}$, то нахождение f_{1,α_j} практически не требует дополнительной работы, а только увеличения памяти и запоминания этих промежуточных результатов $a_p(A)$.

Приведем теперь расчетные формулы для обратного

функционирования и вычисления градиента функции оценки для различных вариантов нейронных сетей. Начнем с довольно общего варианта, потом постепенно перейдем к более частным случаям. Будем задавать сеть по элементам. Каждый элемент зависит от своих параметров – осуществлен упрощающий вариант предыдущего списка.

1. Нейрон. В данном варианте сети предполагаем, что все нейроны имеют по одному входу и одному выходу. Зависимость выхода от входа для всех нейронов определяется с помощью одной общей функции от входного сигнала A и набора параметров α :

$$f_i = f(\alpha_i, A), \quad (60)$$

отличия – в значениях параметров. Для каждого нейрона они свои. В формуле (60) i – номер нейрона, α_i – вектор параметров i -го нейрона. Функция f предполагается дифференцируемой.

2. Синапсы. Каждый нейрон соединен с каждым (в том числе и с собой) синапсами. Синапсы нумеруются парами индексов i, j : от i -го нейрона к j -му. Они преобразуют выходные сигналы нейронов с такта функционирования T и передают их на входные сумматоры в такт функционирования $T+1$. Закон преобразования определяется параметрами синапсов и некоторым набором функций, общим для всех синапсов. Если выходной сигнал i -го нейрона в момент (такт) T есть f_{1iT} , то на входной сумматор j -го нейрона в такт $T+1$ i, j -й синапс передает сигнал

$$s_{1jT} = \alpha_{1ji} \Phi_1(f_{1iT}) + \alpha_{1j2} \Phi_2(f_{1iT}) + \dots + \alpha_{1jk} \Phi_k(f_{1iT}). \quad (61)$$

Здесь α_{1jr} ($r=1, \dots, k$) – параметры синапса, преобразователи $\Phi_r(f)$ – функции, общие для всех синапсов. Принят упрощенный вариант 4 – линейная зависимость от параметров.

3. Сумматоры. Отличие от обычных сумматоров состоит в существовании "синапсов памяти", преобразующих выходной сигнал сумматора в такт T и посылающих результат на вход того же сумматора в такт $T+1$. За счет такого ветвления сигнала на выходе (одна ветвь – на вход нейрона, другая – к синапсу памяти) при обратном функционировании появится двойственный сумматор, складывающий два сигнала.

4. Синапсы памяти. Они преобразуют выходные сигналы в

такт T сумматоров, стоящих перед входом нейронов, и подают их на вход этих сумматоров в такт $T+1$. Закон преобразования аналогичен (61). Синапсам памяти, так же, как и входным сумматорам, присваивается номер того нейрона, перед входом которого они расположены. На вход i -го синапса памяти поступает тот же сигнал A_{1T} , что и на вход i -го нейрона. На выходе получается сигнал $\Psi_{1T} = \alpha_{m11} \Phi_1(A_{1T}) + \dots + \alpha_{mq} \Phi_q(A_{1T})$, где i — номер синапса памяти, Ψ — знак того, что параметр относится к синапсу памяти, α_{mij} — параметры синапсов памяти, $\Phi_j(A)$ — функции, общие для всех таких синапсов.

5. *Параметры инициации.* При непрерывном функционировании сеть передает на следующий такт через синапсы связи выходные сигналы нейронов и синапсов памяти. Поэтому можно отождествить параметры инициации с набором сигналов, передаваемых по сети в момент $T=1$ "как бы" от выходов нейронов и синапсов памяти в такт $T=0$. Эти сигналы подаются соответственно на синапсы и входные сумматоры нейронов. Сигналы, подаваемые на синапсы i -го нейрона, обозначим f_{10} , на входные сумматоры i -го нейрона — Ψ_{10} .

Прямое и обратное функционирование сети в периодическом режиме опишем, объединив различные способы организации входов — подача на входные сумматоры нейронов, прибавление к выходным сигналам и аддитивная подача на синапсы. Описание проведем в три этапа: прямое функционирование, прямое функционирование, нагруженное вычислением дополнительных производных простых функций, обратное функционирование с вычислением градиента функции оценки.

1. *Прямое функционирование.* Перед началом работы формируют параметры инициации f_{10} , Ψ_{10} . Входные сигналы нейронов в первый такт равны

$$A_{11} = \Psi_{10} + x_{11(in)}, \quad (62)$$

где $x_{11(in)}$ — внешние входные сигналы (первого типа — на входы) на первом такте.

После того, как сработают нейроны, на их выходы от них поступают сигналы

$$f(\alpha_1, A_{11}). \quad (63)$$

К этим сигналам добавляются параметры инициации и входные

сигналы второго типа

$$f(\alpha_1, A_{11}) + f_{10} + x_{11(out)}. \quad (64)$$

Кроме того, на вход каждого синапса подаются дополнительные входные сигналы $x_{1j1(syn)}$. Итак, на вход i, j -го синапса подается сигнал

$$\sigma_{1j1} = f(\alpha_1, A_{11}) + f_{10} + x_{11(out)} + x_{1j1(syn)}. \quad (65)$$

На выходах синапсов формируются сигналы

$$\Psi_{1j1} = \sum_k \alpha_{1jk} \Phi_k(\sigma_{1j1}). \quad (66)$$

На входы синапсов памяти подаются сигналы A_{10} (62), на выходах получаем

$$\Psi_{11} = \sum_k \alpha_{mik} \Phi_k(A_{11}). \quad (67)$$

Первый такт прямого функционирования описан. Переходим к такту с номером $T>0$. С предыдущих тактов сеть получает сигналы: Ψ_{1jT-1} , Ψ_{1T-1} . Кроме того, она получает три набора входных сигналов $x_{1T(in)}$, $x_{1T(out)}$, $x_{1T(syn)}$. На входной сумматор i -го нейрона в такт T приходят сигналы:

$$\Psi_{1jT-1} \quad (j=1, 2, \dots), \quad \Psi_{1T-1}, \quad x_{1T(in)}. \quad (68)$$

Вход i -го нейрона A_{1T} — их сумма:

$$A_{1T} = \Psi_{1T-1} + x_{1T(in)} + \sum_j \Psi_{1jT-1}. \quad (69)$$

Эта же сумма подается на вход i -го синапса памяти. Выход i -го нейрона в момент T

$$f_{1T} = f(\alpha_1, A_{1T}). \quad (70)$$

К нему прибавляется входной сигнал $x_{1T(out)}$. Сумма подается i, j -му синапсу, на входе которого добавляется еще входной сигнал $x_{1jT(syn)}$. Поэтому на входе i, j -го синапса в такт T формируется сигнал

$$\sigma_{1jT} = f_{1T} + x_{1T(out)} + x_{1jT(syn)}. \quad (71)$$

Напомним, что при появлении в прямом функционировании сумм в обратном появляются точки ветвления.

На $T+1$ -й такт сеть передает сигналы синапсов:

$$\Psi_{1jT} = \sum_k \alpha_{1jk} \Phi_k(\sigma_{1jT}), \quad (72)$$

$$\Psi_{1T} = \sum_k \alpha_{mik} \Phi_k(A_{1T}). \quad (73)$$

Внешний выход — выходные сигналы нейронов f_{1T} . Таким

образом, прямое функционирование сети описано. Оно происходит от $T=1$ до $T=\theta$.

2. **Нагруженное прямое функционирование.** Для поиска градиента функции оценки при прямом функционировании надо вычислять набор дополнительных величин. Для каждого функционального элемента это - производные функций по входным сигналам при заданных параметрах и по параметрам при данных значениях на входе.

Для нейронов это

$$f_{1T,A}(\alpha_1, A_{1T}) = \frac{\partial f(\alpha_1, A)}{\partial A} \Big|_{A=A_{1T}} ; \quad (74)$$

и совокупность производных по α_1 :

$$f_{1T,\alpha k}(\alpha_1, A_{1T}) = \frac{\partial f(\alpha_1, A_{1T})}{\partial \alpha_{1k}} \Big|_{\alpha=\alpha_1} . \quad (75)$$

Напомним, что α_1 - наборы параметров $\alpha_{11}, \alpha_{12}, \dots$; все f_1 - одинаковые функции набора параметров и входных сигналов, а нейроны отличаются друг от друга значениями параметров.

Для синапсов дополнительные вычисления состоят в поиске производных по значениям на входе и по параметрам. Производные по входным значениям:

$$s_{1jT,0} = \frac{ds_{1jT}}{d\sigma_{1jT}} = \sum_k \alpha_{1jk} \phi'_k(\sigma_{1jT}) , \quad (76)$$

$$m_{1T,A} = \frac{dm_{1T}}{df_{1T}} = \sum_k \alpha_{1ik} \phi'_k(A_{1T}) . \quad (77)$$

Производные же по параметрам для синапса выглядят очень просто:

$$s_{1jT,\alpha k} = \phi_k(\sigma_{1jT}) ; \quad (78)$$

$$m_{1jT,\alpha k} = \phi_k(A_{1T}) . \quad (79)$$

Производные по параметрам инициации совпадают с соответствующими производными по входным параметрам при $T=1$, так как эти параметры подаются на входы элементов просто прибавлением к входному сигналу (62), (64), (65).

3. **Оценка цикла работы и обратное функционирование.** После того, как пройден цикл работы $T=1, 2, \dots, \theta$, получается набор выходных сигналов f_{1T} и можно вычислять функцию оценки

$$H(\{f_{1T}\}, \{x_{1T(1n)}\}, \{x_{1T(out)}\}, \{x_{1jT(syn)}\}) , \quad (80)$$

где {...} обозначается совокупность величин при всевозможных значениях индексов.

Часто функция оценки для всего цикла может быть получена суммированием отдельных потактовых оценок. Это слегка облегчает вычисление, но не является обязательным.

Как уже отмечалось, H задается как явная функция выходных сигналов и через них - как неявная функция параметров и входных сигналов. Последнее - в добавление к явной зависимости от входов (80). Задача обратного функционирования - поиск производных H как функции от параметров и входных сигналов. Результаты нагруженного прямого функционирования предполагаются при этом известными.

При обратном функционировании производные будут накапливаться в специальных сумматорах. Каждый входной сигнал подается одному своему функциональному элементу и каждый параметр также связан только с одним элементом, поэтому и сумматоры удобно расположить при элементах. По крайней мере, это можно сделать в логике рассмотрения, а вопрос об аппаратной реализации оставить открытым.

Сумматоры, соответствующие параметрам i -го нейрона $\alpha_{11}, \alpha_{12}, \dots$ естественно расположить при этом нейроне и обозначить $\Sigma_{\alpha 1k}$; сумматоры, соответствующие параметрам синапсов связи α_{1jk} , - при соответствующих синапсах и обозначить их $\Sigma_{\alpha 1jk}$, для параметров синапсов памяти обозначим сумматоры $\Sigma_{\alpha 1k}$. Сумматоры для производных по входным сигналам $\Sigma_{x1T(1n)}$ и $\Sigma_{x1T(out)}$ располагаются при i -м нейроне, а $\Sigma_{x1jT(syn)}$ - при i, j -м синапсе.

Определим число слагаемых, которые придут в ходе обратного функционирования на эти сумматоры. На каждый сумматор $\Sigma_{\alpha 1k}$ в каждый тakt обратного функционирования подается один сигнал. Всего накапливается их θ - по числу тактов. Так же обстоит дело и для других сумматоров, накапливающих производные по параметрам синапсов.

Сумматоры $\Sigma_{x1T(1n)}$ и $\Sigma_{x1jT(syn)}$ могут называться сумматорами лишь условно - на них подается только по одному сигналу и только в тakt T .

Сумматор $\Sigma_{x_{1T(\text{out})}}$ получает сигналы в такт T со всех элементов, двойственных синапсам, ведущим от i -го нейрона к другим. Число суммируемых сигналов равно числу синапсов. Следует, однако, заметить, что это суммирование и так производится в ходе обратного функционирования. Поясним: в прямом функционировании после выхода i -го нейрона стоит точка ветвления — сигнал посыпается другим нейронам; перед этой точкой ветвления стоит сумматор, прибавляющий к f_{1T} входной сигнал $x_{1T(\text{out})}$; при обратном функционировании точка ветвления заменяется двойственным сумматором, а сумматор складывающий f_{1T} и $x_{1T(\text{out})}$ — двойственной точкой ветвления, от которой один экземпляр сигнала идет к элементу Φ_{1T} , двойственному нейрону (в такт T), а другой — к $\Sigma_{x_{1T(\text{out})}}$. В результате $\Sigma_{x_{1T(\text{out})}}$ тоже ничего не складывает при обратном функционировании, а служит только ячейкой памяти.

После окончания цикла обратного функционирования к содержимому сумматоров должны добавляться еще частные производные H (80) по явному вхождению соответствующих входных сигналов.

Перейдем к описанию обратного функционирования. Сеть обратного функционирования будет состоять из элементов, двойственных элементам прямого функционирования в каждый такт времени T : Φ_{1T} — двойственен i -му нейрону в такт T , S_{1jT} — i, j -му синапсу связи в такт T , M_{1T} — i -му синапсу памяти в такт T . Сумматоры заменяются двойственными точками ветвления, точки ветвления — сумматорами.

Входной сигнал элемента обратного функционирования называется *двойственной переменной* к выходному сигналу элемента прямого функционирования и обозначается так:

$$\begin{aligned}\mu_{1T} & \text{ — для } \Phi_{1T}; \\ \mu_{s1jT} & \text{ — для } S_{1jT}; \\ \mu_{m1T} & \text{ — для } M_{1T}.\end{aligned}$$

Первый такт обратного функционирования ($T=0$). На элементы Φ_{10} подаются величины

$$\mu_{10} = \frac{\partial H}{\partial f_{10}}, \quad (81)$$

где H — явная функция входов и выходов (80). На выходе

получаем сигнал

$$\mu_{10} f_{10,1}. \quad (82)$$

На элементы S_{1j0} и M_{1j0} подаются нулевые входные сигналы:

$$\mu_{s1j0} = \mu_{m1j0} = 0. \quad (83)$$

На выходе этих элементов также нули, то есть эти элементы фактически отсутствуют.

Вход i -го нейрона A_{10} есть сумма сигналов прямого функционирования (69):

$$A_{10} = m_{10} + x_{1(1n)} + \sum_j S_{j10}. \quad (84)$$

В соответствии с этим сигнал Φ_{10} рассыпается по элементам, двойственным источникам сигналов из суммы (84): он посыпается M_{10} , $\Sigma_{x_{10}(1n)}$ и S_{j10} . Как всегда, сумматору двойственна точка ветвления. Тем самым,

$$\begin{aligned}\mu_{s10} &= \mu_{10} f_{10,1}; \\ \mu_{m10} &= \mu_{10} f_{10,1}.\end{aligned} \quad (85)$$

Кроме того, после первого такта обратного функционирования содержимое $\Sigma_{x_{10}(1n)}$ равно величине (82).

В сумматоры Σ_{aik} после первого ($T=0$) такта обратного функционирования подаются сигналы

$$\mu_{10} f_{10,ak}, \quad (86)$$

где второй множитель вычисляется по формуле (75) в нагруженном прямом функционировании. На каждом последующем такте в эти сумматоры будут добавляться еще слагаемые.

Второй ($T=0-1$) и последующие такты обратного функционирования начинаются так: на входы S_{1jT} и M_{1T} подаются сигналы μ_{s1jT} , μ_{m1T} с предыдущего такта.

На выходах этих элементов получаем:

$$\begin{aligned}\mu_{s1jT} S_{1jT,0} & \text{ — на выходе } S_{1jT}, \\ \mu_{m1T} M_{1T,0} & \text{ — на выходе } M_{1T}.\end{aligned} \quad (87)$$

На в сумматоры, накапливающие значения производных H , после этого посыпаются сигналы:

$$\begin{aligned}
 \mu_{s_{ijT}} s_{ijT, \alpha_k} &= \Sigma_{\alpha_{ijk}}, \\
 \mu_{M_{iT}} m_{iT, \alpha_k} &= \Sigma_{\alpha_{ik}}, \\
 \mu_{s_{ijT}} s_{ijT, \sigma} &= \Sigma_{x_{ijT}(\text{syn})}, \\
 \sum_j \mu_{s_{ijT}} s_{ijT, \sigma} &= \Sigma_{x_{iT}(\text{out})}.
 \end{aligned} \tag{88}$$

Заметим, что последние два выражения – это просто выходные сигналы s_{ijT} и их сумма по второму индексу j соответственно.

На вход Φ_{iT} подается сумма по j всех выходов s_{ijT} и $\frac{\partial H}{\partial f_{iT}}$.

$$\mu_{iT} = \frac{\partial H}{\partial f_{iT}} + \sum_j \mu_{s_{ijT}} s_{ijT, \sigma}. \tag{89}$$

Последнее выражение (88) входит как целое в формулу (89). Конечно, ни при аппаратной, ни при программной реализации не следует вычислять эту сумму дважды.

На выходе Φ_{iT} получаем

$$\mu_{iT} f_{iT, A}, \tag{90}$$

этот же сигнал посыпается в сумматоры $\Sigma_{x_{iT}(\text{in})}$; в сумматоры же $\Sigma_{\alpha_{ik}}$ посыпается сигнал

$$\mu_{iT} f_{iT, \alpha_k}. \tag{91}$$

После выхода Φ_{iT} расположены сумматоры, складывающие выходной сигнал Φ_{iT} и поступающий с предыдущего шага обратного функционирования сигнал от M_{iT} . Эта сумма

$$\mu_{iT} f_{iT, A} + \mu_{M_{iT}} m_{iT, A}. \tag{92}$$

через точку ветвления, двойственную входному сумматору i -го нейрона, передается элементам M_{iT-1} и s_{ijT-1} . Итак,

$$\mu_{s_{ijT-1}} = \mu_{M_{iT-1}} = \mu_{iT} f_{iT, A} + \mu_{M_{iT}} m_{iT, A}. \tag{93}$$

движение происходит от $T=0$ до $T=1$. Единственное отличие последнего такта обратного функционирования ($T=1$) от предыдущих состоит в наличии специальных сумматоров, точнее – ячеек памяти для производных H по параметрам инициации. Содержимое ячеек совпадает с сигналом поступающим на этом шаге в $\Sigma_{x_{iT}(\text{in})}$, – для производной по α_{10} , в $\Sigma_{x_{iT}(\text{out})}$ – для производной по f_{10} .

дадим теперь сводку формул и схему (рис. 16) процессов прямого и обратного функционирования.

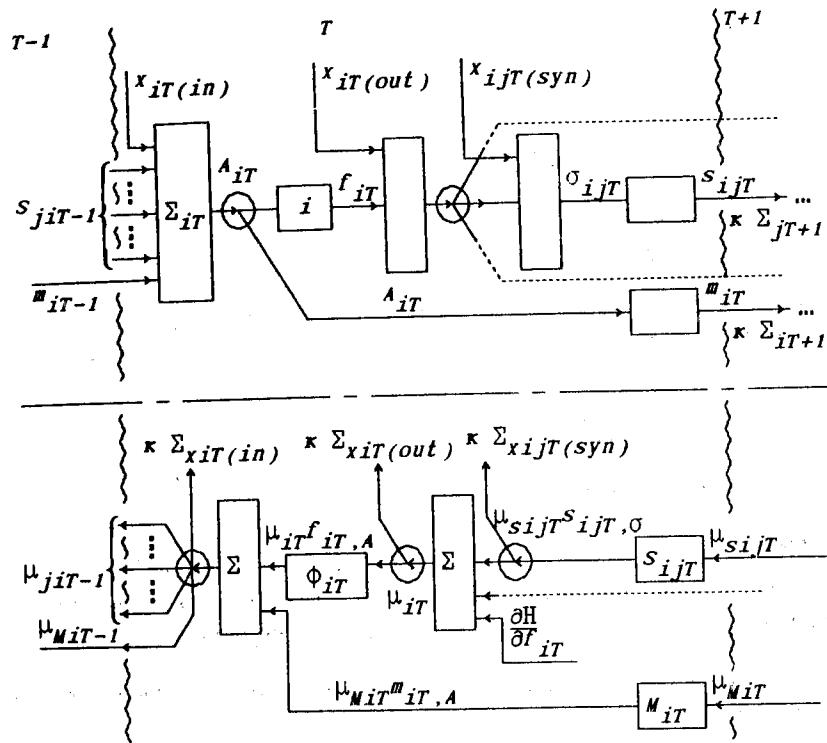


Рис. 16. Акты прямого и обратного функционирования звена нейронной сети (показаны частично).

Нагруженное прямое функционирование:

а) входы нейронов

$$A_{11} = m_{10} + x_{11(i_{in})},$$

$$A_{iT} = m_{iT-1} + x_{iT(i_{in})} + \sum s_{j_{iT-1}} \quad (T=2, \dots, \theta);$$

б) выходы нейронов

$$f_{iT} = f(\alpha_{iT}, A_{iT}),$$

$$f_{iT, A} = \frac{\partial f(\alpha, A)}{\partial A} \Big|_{\alpha=\alpha_{iT}, A=A_{iT}},$$

$$f_{iT, \alpha_k} = \frac{\partial f(\alpha, A)}{\partial \alpha_{iT}} \Big|_{\alpha=\alpha_{iT}, A=A_{iT}},$$

в) входы синапсов связи

$$\sigma_{ij1} = f_{11} + f_{10} + x_{11(out)} + x_{ij1(syn)},$$

$$\sigma_{ijT} = f_{1T} + f_{1T} + x_{1T(out)} + x_{ijT(syn)} \quad (T=2, \dots, \theta);$$

г) выходы синапсов связи

$$s_{ijT} = \sum_k \alpha_{ijk} \phi_k(\sigma_{ijT}),$$

$$s_{ijT, \sigma} = \sum_k \alpha_{ijk} \phi'_k(\sigma_{ijT}),$$

$$s_{ijT, \alpha_k} = \phi_k(\sigma_{ijT});$$

а) входы синапсов памяти в такт T совпадают с A_{1T}

с) выход синапсов памяти

$$m_{1T} = \sum_k \alpha_{1ik} \phi_k(A_{1T}),$$

$$m_{1T, \sigma} = \sum_k \alpha_{1ik} \phi'_k(A_{1T}),$$

$$m_{1T, \alpha_k} = \phi_k(A_{1T}).$$

Функция оценки: $H = H(\{f_{1T}\}, \{x_{1T(in)}\}, \{x_{1T(out)}\}, \{x_{ijT(syn)}\})$.

Удобно иметь отдельное обозначение для H , как функции от параметров и внешних входных сигналов: $H = H_0(\{\alpha\}, \{x\})$.

Обратное функционирование; вычисление двойственных переменных:

а) для нейронов

$$\mu_{1\theta} = \frac{\partial H}{\partial f_{1\theta}},$$

$$\mu_{1T} = \frac{\partial H}{\partial f_{1T}} + \sum_j \mu_{s1jT} s_{ijT, \sigma} \quad (T=\theta-1, \theta-2, \dots);$$

б) для синапсов связи

$$\mu_{s1j\theta} = 0,$$

$$\mu_{s1jT} = \mu_{jT+1} f_{jT+1, \alpha} + \mu_{jT+1} m_{jT+1, \alpha} \quad (T=\theta-1, \theta-2, \dots);$$

в) для синапсов памяти

$$\mu_{1iT} = \mu_{1iT+1} f_{1iT+1, \alpha} + \mu_{1iT+1} m_{1iT+1, \alpha} \quad (T=\theta-1, \theta-2, \dots).$$

Обратное функционирование; вычисление производных функции оценки:

$$\frac{\partial H_0}{\partial \alpha_{1k}} = \sum_{t=\theta} \mu_{1t} f_{1t, \alpha_k},$$

$$\frac{\partial H_0}{\partial \alpha_{1ik}} = \sum_{t=\theta-1} \mu_{1it} f_{1it, \alpha_k},$$

$$\frac{\partial H_0}{\partial \alpha_{1jk}} = \sum_{t=\theta-1} \mu_{s1jt} f_{s1jt, \alpha_k},$$

$$\frac{\partial H_0}{\partial \alpha_{1T(in)}} = \mu_{1T} f_{1T, \alpha} + \frac{\partial H}{\partial \alpha_{1T(in)}},$$

$$\frac{\partial H_0}{\partial \alpha_{1T(out)}} = \mu_{1T} + \frac{\partial H}{\partial x_{1T(out)}},$$

$$\frac{\partial H_0}{\partial \alpha_{ijT(syn)}} = \mu_{s1jT} s_{ijT, \sigma} + \frac{\partial H}{\partial x_{ijT(syn)}},$$

$$\frac{\partial H_0}{\partial \alpha_{10}} = \mu_{11} f_{11, \alpha},$$

$$\frac{\partial H_0}{\partial \alpha_{10}} = \mu_{11}.$$

Перепишем эти формулы для одного важного частного случая. Пусть каждый нейрон характеризуется двумя параметрами – характеристикой пологости α_{11} и спонтанной активностью α_{12} :

$$f(A) = \frac{A}{\alpha_{11} + |A|} + \alpha_{12}, \quad (94)$$

синапсы связи и памяти – простейшие линейные:

$$s_{ij}(\sigma) = \alpha_{1j} \sigma,$$

$$m_{1i}(\alpha) = \alpha_{1i} \alpha.$$

Пусть также есть только одна группа входных сигналов $x_{1T} = x_{1T(in)}$ и только одна группа параметров инициации – α_{10} . Тогда **нагруженное прямое функционирование** принимает вид:

а) входы нейронов

$$A_1 = m_{10} + x_{11},$$

$$A_{1T} = \alpha_{1i} A_{1T-1} + \sum_j \alpha_{1j} f_{jT-1};$$

б) выходы нейронов

$$f_{1T} = \frac{A_{1T}}{\alpha_{11} + |A_{1T}|} + \alpha_{12},$$

$$f_{1T,A} = \frac{\alpha_{11}}{(\alpha_{11} + |A_{1T}|)^2},$$

$$f_{1T,\alpha_1} = \frac{A_{1T}}{(\alpha_{11} + |A_{1T}|)^2},$$

$$f_{1T,\alpha_2} = 1;$$

в) входы и выходы синапсов связи

$$\sigma_{1jT} = f_{1T},$$

$$s_{1jT} = \alpha_{1j} f_{1T},$$

$$s_{1jT,\sigma} = \alpha_{1j},$$

$$s_{1jT,\alpha} = f_{1T};$$

г) выходы синапсов памяти

$$m_{1T} = \alpha_{11} A_{1T},$$

$$m_{1T,A} = \alpha_{11}.$$

Обратное функционирование; вычисление двойственных переменных:

а) для нейронов

$$\mu_{1\theta} = \frac{\partial H}{\partial f_{1\theta}},$$

$$\mu_{1T} = \frac{\partial H}{\partial f_{1T}} + \sum_j \alpha_{1j} \mu_{s1jT} \quad (T=\theta-1, \theta-2, \dots);$$

б) для синапсов связи

$$\mu_{s1j\theta} = 0,$$

$$\mu_{s1jT} = \mu_{jT+1} f_{jT+1,A} + \mu_{mjt+1} \alpha_{jm} \quad (T=\theta-1, \theta-2, \dots);$$

в) для синапсов памяти

$$\mu_{m1\theta} = 0,$$

$$\mu_{m1T} = \mu_{1T+1} f_{1T+1,A} + \mu_{m1T+1} \alpha_{11} \quad (T=\theta-1, \theta-2, \dots).$$

Обратное функционирование; вычисление производных функций оценки - по сравнению с общим случаем сразу можно упростить три формулы:

$$\frac{\partial H_0}{\partial \alpha_{12}} = \sum_{T=\theta}^1 \mu_{1T},$$

$$\frac{\partial H_0}{\partial \alpha_{11}} = \sum_{T=\theta-1}^1 \mu_{m1T} A_{1T},$$

$$\frac{\partial H_0}{\partial \alpha_{1j}} = \sum_{T=\theta-1}^1 \mu_{s1jT} f_{1T}.$$

Еще проще выглядят формулы в чисто коннекционистской модели. В этой модели обучение состоит в подстройке параметров синапсов, а нейроны неизменны. Нейроны, конечно, могут отличаться друг от друга, но эти отличия не меняются в ходе обучения и производные по параметрам нейронов не вычисляются.

7.6. Back-back процедура

При организации обучения полезно использовать вторые производные функции ошибки. Вычислять всю матрицу вторых производных и хранить ее в памяти может быть слишком сложно. Существует промежуточный вариант - вычислять градиенты от некоторых функций градиента H , например, от

$$H_2 = \sum \frac{\partial H}{\partial \alpha}, \quad (95)$$

где суммирование ведется по всем параметрам α .

Поскольку градиент H вычисляется в параллельно-последовательном режиме сетью автоматов прямого и обратного функционирования, то можно организовать обратное функционирование этой сети для вычисления нужных производных. Это "двойное обратное" функционирование названо нами "back-back" процедурой. Чтобы не загромождать изложение сложными, но по существу - тривиальными формулами, ограничимся при описании этой процедуры квадратом градиента (95) для чисто коннекционистских моделей, линейных синапсов связи и при отсутствии синапсов памяти.

Допустим различия между нейронами. Пусть i -й нейрон вычисляет функцию входного сигнала $f_i(A)$. Ограничимся только одним типом входных сигналов $x_{iT} = x_{iT(i)}$. Формулы предыдущего раздела для прямого и первого обратного функционирования в рамках принятых предположений можно упростить еще сильнее.

Сеть вычисляет величины

$$A_{11} = x_{11}, \quad A_{1T} = x_{1T} + \sum_j f_{jT-1} \alpha_{j1} \quad (T=2, 3, \dots, \theta),$$

$$f_{1T} = f_1(A_{1T}),$$

$$f_{1T,A} = f'(A_{1T}),$$

$$\mu_{1\theta} = \frac{\partial H}{\partial f_{1\theta}},$$

$$\mu_{1T} = \frac{\partial H}{\partial f_{1T}} + \sum_j \alpha_{1j} \mu_{jT+1} f_{jT+1,A} \quad (T=\theta-1, \theta-2, \dots),$$

$$H_{1j} = \frac{\partial H}{\partial \alpha_{1j}} = \sum_{T=\theta-1}^1 \mu_{jT+1} f_{1T} f_{jT+1,A}.$$

В обратно-обратном функционировании требуется вычислить производные функции

$$H_2 = \sum_{i,j} H_{ij}^2 \quad (96)$$

по параметрам α_{1j} .

Исходно H_2 задана как функция H_{ij} – величин, получаемых на выходе специальных сумматоров системы обратного функционирования. При обратно-обратном функционировании сумматоры заменяются точками ветвления, выходы – выходами и на эти новые входы подаются частные производные H_2 (96) по H_{ij} . то есть $2H_{ij}$.

далее разворачивается уже неоднократно описанный в предыдущих разделах обратный процесс. Прежде, чем приступить к его описанию, дадим специально приспособленное для этого схематическое представление прямого и первого обратного функционирования, а также введем некоторые дополнительные обозначения:

$$h_{1jT} = f_{1T} \mu_{jT+1} f_{jT+1,A} \quad (T=1, 2, \dots, \theta-1); \quad H_{1j} = \sum_{T=\theta-1}^1 h_{1jT}. \quad (97)$$

При составлении схемы (рис. 17) следует учесть, что теперь величина $f_{1T,A}$ полноправно участвует в прямом и первом обратном процессах, а не только как дополнительная нагрузка для обеспечения обратного функционирования. Роль дополнительной нагрузки будут играть вторые производные

$$f_{1T,AA} = \frac{d^2 f_1(A)}{dA^2} \Big|_{A=A_{1T}} = f_1''(A_{1T}). \quad (98)$$

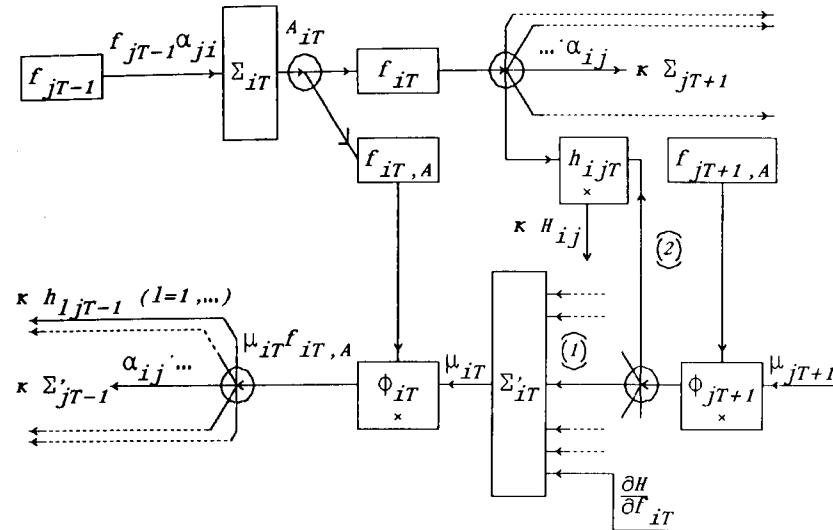


Рис. 17. Схема звена прямого и обратного функционирования для построения обратно-обратного. По связи (1) передается сигнал $\alpha_{ij} \mu_{jT+1} f_{jT+1,A}$; по связи (2) – сигнал $\mu_{jT+1} f_{jT+1,A}$.

В процессе первого обратного функционирования участвует много элементов, призывающих умножение двух входных сигналов – это Φ_{iT} и h_{1jT} . Двойственные к ним элементы обратно-обратного функционирования имеют по одному входу и по два выхода.

Если исходный элемент Y перемножал сигналы A , B , то выходы двойственного элемента Y' передают сигналы к элементам, двойственным источникам A и B соответственно. Обозначим здесь эти источники через A' и B' . При этом на A' подается входной сигнал Y' , умноженный на B ($d(AB)/dA=B$), а на B' – входной сигнал Y' , умноженный на A ($d(AB)/dB=A$).

Основные элементы прямого и первого обратного процессов – это f_{iT} , $f_{iT,A}$ и Φ_{iT} . Обозначим двойственные им элементы обратно-обратного функционирования через Φ_{iT} , $\Phi_{iT,A}$ и Φ_{iT} соответственно. Входные сигналы для этих элементов обозначим λ_{iT} , $\delta_{iT,A}$, ν_{iT} – основные переменные обратно-обратного

функционирования. Проще всего определяются $\delta_{iT,A}$, так как сигнал от $f_{iT,A}$ поступает только одному элементу Φ_{iT} (при первом обратном функционировании):

$$\delta_{iT,A} = v_{iT} \mu_{iT} \quad (99)$$

Обратно-обратное функционирование удобно представить в виде двух цепей: верхней, двойственной прямому процессу, и нижней, двойственной обратному. Обе они используют результаты прямого и первого обратного процессов. Верхняя цепь представлена на рис. 18. От нижней цепи ей поступают сигналы $\delta_{iT,A}$ (99).

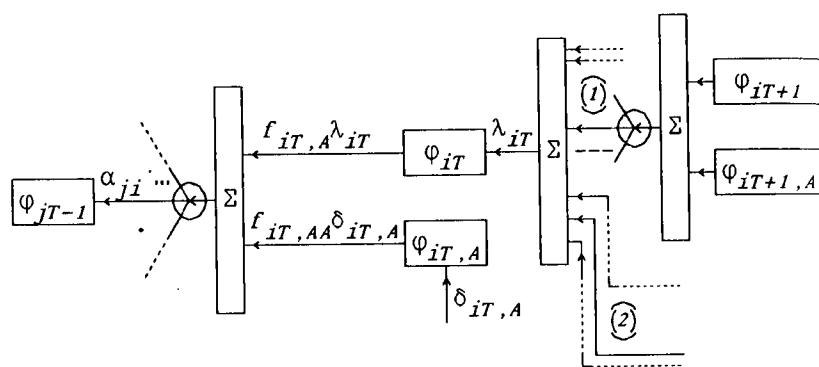


Рис. 18. Звено верхней цепи обратно-обратного функционирования. По связи (1) передается сигнал $\alpha_{ij}(f_{jT+1,A} \lambda_{jT+1} + f_{jT+1,AA} \delta_{jT+1,A})$, по связи (2) – сигнал $\frac{\partial H_2}{\partial H_{ij}} \mu_{jT+1} f_{jT+1,A}$.

Формулы функционирования верхней цепи (от $T=0$ до $T=1$):

$$\lambda_{i0} = 0,$$

$$\begin{aligned} \lambda_{iT} = & \sum_j \alpha_{ij} (f_{jT+1,A} \lambda_{jT+1} + f_{jT+1,AA} \delta_{jT+1,A}) + \\ & + \sum_j \frac{\partial H_2}{\partial H_{ij}} f_{jT+1,A} \mu_{jT+1} \quad (T=0-1,0-2, \dots, 1). \end{aligned}$$

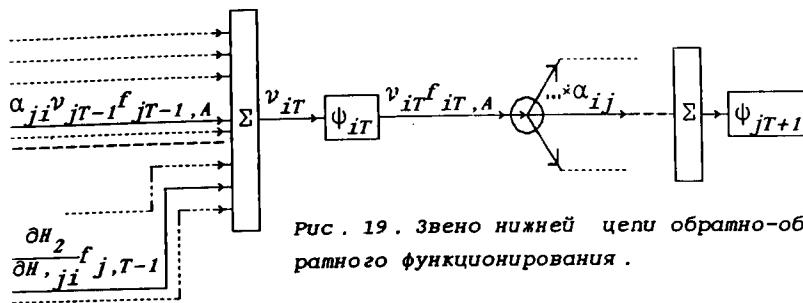


Рис. 19. Звено нижней цепи обратно-обратного функционирования.

Нижняя цепь представлена на рис. 19. Ее функционирование описывается формулами (от $T=0$ до $T=\Theta$):

$$v_{i1} = 0, \quad v_{iT} = \sum_j \alpha_{ij} v_{jT-1} f_{jT-1,A} + \sum_j \frac{\partial H_2}{\partial H_{ij}} f_{jT-1,A} \quad (T=2,3, \dots, \Theta).$$

обратно-обратное функционирование проводится так: сначала от $T=1$ до $T=\Theta$ работает нижняя цепь, потом вычисляется δ (99) и от $T=\Theta$ до $T=1$ работает верхняя цепь. Производные H_2 как неявной функции α_{ij} по этим переменным можно накапливать в специальных сумматорах. Они определяются так:

$$\begin{aligned} \frac{\partial H_2}{\partial \alpha_{ij}} = & \sum_{T=1}^{\Theta-1} v_{iT} \mu_{jT+1} f_{jT+1,A} f_{jT+1,A} + \\ & + \sum_{T=1}^{\Theta-1} f_{iT} (f_{jT+1,A} \lambda_{jT+1} + f_{jT+1,AA} \delta_{jT+1,A}). \end{aligned}$$

Первая сумма соответствует нижней цепи, вторая – верхней.

Итак, показано, как в параллельно-последовательном режиме вычислять производные функционалов от градиента функции оценки. Использование этих производных в обучении будет продемонстрировано в разделе, посвященном алгоритмам обучения.

7.7. Несколько заключительных замечаний по двойственности

В математическом фольклоре бытует такой тезис: сложная задача оптимизации может быть решена только при эффективном использовании двойственности. Мы определили обучение нейронных сетей как задачу оптимизации и потратили немало

места на описание конструкций, свойственных нейронной сети. Что же позволяет сделать использование двойственности? Попытаемся в нескольких пунктах ответить на этот вопрос.

1. Двойственные сети позволяют быстро в последовательно-параллельном режиме вычислять градиент функции оценки как неявной функции параметров и входов.
2. Если требуется, двойственная сеть безо всяких изменений своей структуры и параметров может вычислять производные по параметрам сети и входам для любой другой функции выходных сигналов.
3. Сеть обратно-обратного функционирования может эффективно вычислять производные любых функционалов от указанных в пп. 1, 2 градиентов.

И это все? Да, пока все. Отметим еще только, что для чисто коннекционистских моделей и линейных синапсов цепи обратного и обратно-обратного функционирования используют ту же структуру связей между элементами и те же веса α_{ij} , что и при прямом функционировании. Это существенно упрощает реализацию таких сетей. Именно использование при обратном функционировании той же системы связей, что и при прямом, имеют в виду, когда говорят, что сеть сама вычисляет направление наискорейшего обучения.

8. АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ

8.1. Зачем нужен анализ чувствительности

Рассматривается нейронная сеть, решающая предъявленный ей пример. Из общих соображений вроде бы ясно, что не все параметры сети одинаково важны для решения этого примера. К изменениям одних параметров решение будет более чувствительно других – менее. Разделение параметров на две или более групп по степени чувствительности может использоваться при обучении.

Вот один из способов использования: параметры, к изменениям которых решение данного примера или группы примеров мало чувствительно, при обучении этим примерам не менять вовсе. Вероятная польза состоит в том, что к изменениям этих параметров может оказаться чувствительным решение других примеров, которым сеть учились прежде и, возможно, будет учиться в дальнейшем. Не стоит зря портить то, что для других примеров может быть полезно.

Другой способ, исходящий из тех же оснований. Пусть задано несколько примеров и для каждого из них выделены параметры, к изменениям которых их решение наиболее чувствительно. Можно учить по очереди всем примерам, но при обучении очередному примеру не менять параметры из групп повышенной чувствительности для других примеров.

Может случиться так, что при этом вообще не будут меняться параметры из групп повышенной чувствительности. Это произойдет, если каждый такой параметр принадлежит группам повышенной чувствительности сразу для нескольких примеров.

Даже такое странное обучение, когда наиболее сильно влияющие параметры совсем не меняются, нельзя заранее определить как плохое или хорошее. Для этого нужно сравнить ситуацию с обучением всем примерам с изменением всех параметров – без ограничений.

Если такое обучение по разным примерам сдвигает параметр

из группы повышенной чувствительности в разные стороны, то может оказаться более успешной та стратегия обучения, когда этот параметр не сдвигается вовсе. Не то возникает ситуация лебедя, Рака и Щуки из известной басни. Если же сдвиг для разных примеров происходит в одну сторону, то двигаться безусловно надо.

Можно и по-другому использовать разбиение параметров по группам чувствительности. Некоторые варианты будут разобраны в разделах, посвященных алгоритмам обучения. А пока зафиксируем: анализ чувствительности состоит в разбиении параметров сети на группы в зависимости от того, насколько сильно изменение этих параметров влияет на решение. Чувствительность может сравниваться по ряду показателей. Перейдем к их обсуждению.

8.2. Определение чувствительности по производным функции оценки

Наиболее очевидный подход к анализу чувствительности состоит в упорядочивании параметров α_i по величине $|\partial H / \partial \alpha_i|$ – чем больше абсолютная величина производной, тем выше чувствительность.

Вторые производные также могут служить для формализации идеи чувствительности: даже если мало $|\partial H / \partial \alpha_i|$, но сравнительно велика вторая производная $|\partial^2 H / \partial \alpha_i \partial \alpha_j|$ при некотором j , то изменение α_i может сильно изменить значение градиента H и тем самым сильно повлиять на обучение. Частный случай учета вторых производных – выбор в качестве параметров, характеризующих чувствительность, величин $|\partial(\nabla H)^2 / \partial \alpha_i|$. Их можно найти с помощью back-back процедуры.

Главный вывод этого краткого раздела: существует много показателей чувствительности. Каждый показатель как-то позволяет сортировать параметры. Вопрос: как использовать сразу несколько показателей? Этот вопрос обостряется тем, что многие показатели могут вычисляться на каждом шаге обучения, но вряд ли стоит на каждом шаге менять разделение по группам чувствительности.

Закрепим за показателями чувствительности греческую букву α . Наиболее перспективны для использования три набора показателей:

- 1) $\alpha_i = |\partial H / \partial \alpha_i|$;
- 2) $\alpha_i = \max_j |\partial^2 H / \partial \alpha_i \partial \alpha_j|$;

$$3) \alpha_i = \left| \frac{\partial}{\partial \alpha_i} \left(\sum_j \left(\frac{\partial H}{\partial \alpha_j} \right)^2 \right) \right|. \quad (100)$$

Заметим, что без первого из них в этом наборе не обойтись – производная может быть велика и при нулевых вторых производных. Поэтому второй и третий наборы без первого использовать не могут. Они дают важную, но все же дополнительную информацию.

8.3. Анализ чувствительности по прямому функционированию

Показатели чувствительности, приведенные в предыдущем разделе, используют производные H . В некоторых случаях это невозможно или нежелательно. Например, если нет гладкой функции оценки, а функционирование оценивается бинарно: "хорошо – плохо". Конечно, в этом случае можно ввести вспомогательную функцию выходных сигналов, вычислить ее производные и по ним определять чувствительность. Такой функцией может быть, например, сумма квадратов выходных сигналов или взвешенная сумма квадратов с весами, соответствующими относительной важности сигналов.

Можно попытаться, однако, обойтись без обратного функционирования и вычисления градиентов. Основанием служат следующие соображения. Пусть при прямом функционировании некоторый элемент сети Φ получает на такте T вектор входных сигналов A_T и на выходе выдает функцию $\Phi(\alpha, A_T)$. При обратном функционировании двойственный Φ в такт T элемент получает на входе двойственную переменную $\mu_{\Phi T}$. В сумматор, накапливающий $\partial H / \partial \alpha$, этот двойственный элемент отправляет величину

$$\mu_{\Phi T} \frac{\partial \Phi(\alpha, A_T)}{\partial \alpha}. \quad (101)$$

Чтобы найти первый сомножитель в (101), требуется обратное

функционирование. Второй может быть найден и в ходе прямого, точнее, нагруженного прямого. Вот этот второй сомножитель и можно выбрать для оценки чувствительности, полагая

$$x_{\alpha} = \max_{\Phi, T} \left| \frac{\partial \Phi(\alpha, A_T)}{\partial \alpha} \right|, \quad (102)$$

где максимум берется по всем тактам функционирования T и по всем элементам Φ , зависящим от параметра α .

для нейронной сети простейшего типа и линейных синапсов связи и памяти имеем

$$x_{\alpha_{ij}} = \max_T |f_{it}|, \quad x_{\alpha_{it}} = \max_T |f_{it}|. \quad (103)$$

Иными словами, для синаптических весов чувствительность характеризуется максимальным сигналом, проходившим по синапсу за вход-выход в виде цикла функционирования.

Непосредственное использование максимальных по T величин (102), (103) может привести к ряду парадоксов. Дело в том, что на разных тактах обратного функционирования величины μ могут существенно различаться. Поэтому естественнее, опуская множитель μ в (103), сохранять для каждого параметра столько показателей чувствительности, сколько тактов включает функционирование. Получим для каждого T набор

$$x_{\alpha_T} = \max_{\Phi} \left| \frac{\partial \Phi(\alpha, A_T)}{\partial \alpha} \right|. \quad (104)$$

Выбор наиболее значимых параметров по показателям чувствительности следует производить отдельно для каждого T , а потом объединять полученные множества.

8.4. Системы, элементы которых линейны по параметрам

В предыдущем разделе описаны показатели чувствительности, которые вычисляются в ходе только прямого функционирования. Для линейного синапса такой показатель есть абсолютная величина проходившего сигнала. Это достаточно очевидно — если сигнала нет, то и вес синапса роли не играет.

для более сложных элементов с нелинейной зависимостью от параметров вычисление показателей (100-104) требует все же нагруженного, а не просто прямого функционирования. Нагруз-

ка состоит в вычислении производных, входящих в (101). Этого недостатка лишены системы, элементы которых линейны по параметрам. Такие системы достаточно интересны и обладают рядом преимуществ при обучении. Остановимся на них немного подробнее.

Пусть характеристики вход-выход для нейронов имеют вид

$$f_i(\alpha, A) = \sum_{i,j} \alpha_{ij} \Phi_j(A). \quad (105)$$

Тогда производные f_i по параметрам — просто величины $\Phi_j(A)$. Они и могут служить показателями чувствительности.

Если, например, мы хотим регулировать в ходе обучения характеристики нейрона, то можно задать соответствие

$$f(\alpha, A) = \alpha \frac{A}{a_{1 \min} + |A|} + (1-\alpha) \frac{A}{a_{1 \max} + |A|}, \quad 0 \leq \alpha \leq 1. \quad (106)$$

Величины $a_{1 \min}$ и $a_{1 \max}$ характеризуют предельно допустимые варианты крутизны.

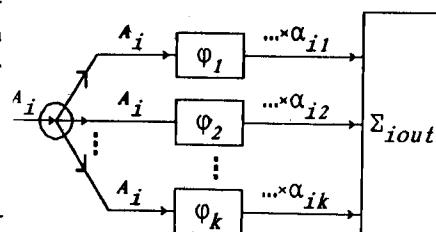


Рис. 20. Коннекционистское представление нейрона, линейного по параметрам.

Характеристики вида (105) можно графически представить так: существует не один i -й нейрон с подстраиваемыми параметрами α_{ij} , а несколько. Каждый из них имеет характеристику $\Phi_j(A)$, зависящую от номера i . Эти нейроны подают выходные сигналы через свои выходные синапсы на i -й выходной сумматор (рис. 20). Тем самым, обучаемый нейрон представляется посредством малой нейронной сети с нерегулируемыми нейронами, о обучаемыми синапсами. Получаем чисто коннекционистское представление обучаемого нейрона.

8.5. Формирование списка параметров, к изменениям которых система наиболее чувствительна

При выделении совокупности параметров, к изменениям которых система наиболее чувствительна, учитываются различные характеристики.

1. **Интервал показателей чувствительности α .** Среди текущих значений показателей чувствительности выделяются максимальное α_{\max} , минимальное α_{\min} и находится среднее α_{av} . Определяется интервал значений, соответствующий повышенной чувствительности. Например, это могут быть значения между средним и максимальным, включая, естественно, максимальное. Другой пример - выбирается отрезок повышенной чувствительности $[\alpha_{av} + \epsilon, (\alpha_{\max} - \alpha_{av}) \cdot \alpha_{\max}]$, а ϵ ($0 \leq \epsilon < 1$) подбирается из других соображений в ходе обучения.
2. **Число параметров, включаемых в список.** Задается количество параметров, включаемых в список. Оно может определяться из различных соображений, связанных с ограничениями по времени, по памяти и т.п. Естественный способ задания этого числа - KN , где N - количество нейронов, K - натуральное число, не зависящее от N ($2, 3, 4, \dots$).
3. **Ограничения, определяемые структурой сети.** Они могут состоять, например, в том, что для каждого нейрона в список должно попасть хотя бы два параметра - один для какого-нибудь входящего, один для выходящего синапса. Возможны и другие варианты.
4. **Динамика показателей чувствительности.** Процесс обучения развертывается во времени и показатели могут вычисляться в некоторой последовательности шагов. Информацию о ходе их изменения можно и нужно учитывать при составлении списка. В качестве примера приведем принцип k -максимального термометра. Для каждого параметра сети определяется максимум показателя чувствительности за k последних шагов, а потом по этому массиву максимальных показателей определяется список наиболее значимых для обучения параметров.

Кроме наиболее значимых параметров можно выделить

наименее значимые. Их можно потом устремлять к нулю. Это упростит структуру сети.

Приведем пример алгоритма для выделения наиболее и наименее значимых параметров. Рассматривается чисто коннекционистская нейронная сеть периодического функционирования с линейными синапсами. Цикл функционирования включает Θ тактов сети. Идет обучение нескольким примерам. Их оценки H_1, \dots, H_n . Минимизируется суммарная оценка H . Для каждого шага обучения в двойственном функционировании получаем градиенты всех H_i и их сложением - градиент H . На каждом такте T обратного функционирования вычисляются величины (101) для синаптических весов α_{ij} :

$$\alpha_{ijT}^{1q} = f_1(\alpha_{ijT-1}^{1q}) f_1'(\alpha_{ijT}^{1q}) \mu_{ijT}^{1q} \quad (T=2, \dots, \Theta), \quad (107)$$

где верхний индекс 1 - номер примера, q - номер шага обучения.

Суммируя их по примерам, получаем

$$\alpha_{ijT}^{q} = \left| \sum_{i=1}^n \alpha_{ijT}^{1q} \right| \quad (108)$$

Этот модуль суммы и будет играть роль показателя чувствительности. Далее, согласно принципу k -максимального термометра, положим

$$\alpha_{ijT}^{*q} = \max_{p=q, q-1, \dots, q-k+1} \{\alpha_{ijT}^{p}\} \quad (109)$$

Чтобы для каждого нейрона в каждый момент в число значимых вошли веса хотя бы одного входного и одного выходного синапса, найдем

$$\max_{j} \{\alpha_{ijT}^{*q}\}; \arg \max_{j} = j(i, q, T);$$

$$\max_{i} \{\alpha_{ijT}^{*q}\}; \arg \max_{i} = i(j, q, T).$$

В список значимых включаются не более $2N(\Theta-1)$ параметров

$$\alpha_{ij(i, q, T)}, \alpha_{ij(j, q, T)}, i, j = 1, \dots, N, T = 2, \dots$$

Исключаем их и соответствующие показатели α из списков. Среди оставшихся показателей α_{ijT}^{*q} для каждого T найдем M наибольших. Включим соответствующие α_{ij} в список значимых параметров, если кроме того

$$\alpha_{ijT}^{*q} > \frac{1}{N^2} \sum_{i, j} \alpha_{ijT}^{*q}, \quad (110)$$

то есть показатель чувствительности больше среднего. В итоге получаем не более $(2N+M)(\theta-1)$ параметров α_{ij} .

Список наименее значимых параметров строится так. Для всех T находим

$$\mathcal{X}_{Tqav}^{*q} = \frac{1}{N^2} \sum_{i,j} \mathcal{X}_{ijT}^{*q},$$

$$\mathcal{X}_{Tqmin}^{*q} = \min_{i,j} \mathcal{X}_{ijT}^{*q}.$$

Задается некоторое ϵ ($0 < \epsilon \leq 1$). Параметр α_{ij} вносится в список наименее значимых, если

$$\mathcal{X}_{ijT}^{*q} < \mathcal{X}_{Tqmin}^{*q} + \epsilon (\mathcal{X}_{Tqav}^{*q} - \mathcal{X}_{Tqmin}^{*q})$$

при всех $T=2, \dots, \theta$ и α_{ij} не входит при этом в список наиболее значимых параметров.

Для определения списков достаточно k шагов обучения, после чего предлагается устремлять наименее значимые параметры к нулю, а в ходе обучения некоторое время менять только наиболее значимые параметры. То, что значимых параметров сравнительно мало (по сравнению с N^2), позволяет использовать более эффективные методы оптимизации.

Подчеркнем, что это – только один из примеров. Выбор наиболее значимых параметров предполагает сортировку большого массива. Это трудоемкая операция и желательно перейти от нее к менее трудоемким и параллельно выполняемым. Например, можно ограничиться выбором наиболее значимых параметров среди отдельных локальных наборов. Так можно упорядочивать по чувствительности веса синапсов, связанных с каждым отдельным нейроном и еще, к тому же, раздельно входных и выходных. Отдельно можно сравнивать по чувствительности параметры разных нейронов. Эти палиативные варианты допускают высокую параллельность исполнения, так как в них сортировке подлежит много ($N+1$) сравнительно небольших ($\approx N$) массивов.

9. КОНТРАСТИРОВАНИЕ СЕТИ

9.1. Контрастирование для интерпретации навыков сети

Программируемая ЭВМ понята – последовательность ее действий имеет ясную логическую структуру. Действия нейронной сети обычно непонятны. Конечно, их можно описать как пересылку, суммирование и преобразование большого числа сигналов, но понятийная интерпретация крайне затруднена. Каждый, кто знакомился с синаптической картой обученной сети, знает: интерпретировать ее практически невозможно, нейронная сеть делает то, что она делает, навык нельзя прочитать по значениям параметров, его можно выявить на тестовых испытаниях.

Неприменимость, непонятность – хорошо это или плохо? Конечно, плохо: если можно, то лучше иметь возможность понимания. Другой вопрос – а если понять все же нельзя, тогда как? Что же, если по-другому нельзя получить быстродействующие самообущающиеся устройства, тогда ничего не поделешь. Но вот всегда ли нельзя? Иными словами, для любой ли задачи, решаемой нейрокомпьютером, невозможна понятийная интерпретация его действий. Это вряд ли. Поэтому возникает цель: по возможности обеспечивать интерпретируемость результатов обучения. Этой цели можно достичь с помощью контрастирования сети.

Сначала одно замечание. Чем меньше в сети элементов, тем понятнее ее работа. В частности, чем меньше синапсов связи, тем проще понять действия сети.

Контрастирование сети в целях интерпретируемости ее навыков состоит в том, что при обучении стараются обратить в ноль максимально возможное число параметров сети. При этом, конечно, навык решения задач должен сохраняться.

9.2. Контрастирование для технической реализации нейрокомпьютера

Если наша цель – построить на основании предварительно обученных нейронных сетей специализированную вычислительную

машину для решения данного класса задач, то во-первых, мы должны учесть, что чем меньше элементов и связей – тем проще техническая реализация. Во-вторых, если при обучении лучше иметь дело с многоразрядными числами, то при реализации – чем меньше разрядов, тем лучше. В порядке возрастания сложности варианты значений для синаптических весов например можно расположить так: нет синапса ($\alpha=0$), 1-битовые веса ($\alpha=\pm 1$), 2-битовые веса ($\alpha=\pm 0.5, \pm 1$) и т.д. Желательно ограничиться 1-битовыми весами ($\alpha=\pm k/128, k/1, 2, \dots, 128$). Возможны, конечно, и другие варианты.

Контрастирование для технической реализации состоит в том, что при обучении должен производиться подбор параметров из данного множества значений, причем каждое значение имеет свою "цену" (и не только в виде математически описываемого штрафа за то или иное ненулевое значение, а саму настоящую денежную).

9.3. Как проводить контрастирование

Мы вынуждены несколько забежать вперед и коснуться процесса обучения. В начальный момент параметры нейронной сети имеют некоторые значения. Если сеть порождается впервые и ничему не обучалась, то эти параметры, вообще говоря, случайны. Исходя из всевозможных предварительных соображений, некоторым группам параметров могут быть назначены различные интервалы начальных значений, но случайность, тем не менее, сохраняется.

Уже на начальном этапе, сделав несколько шагов обучения, можно проанализировать чувствительность, определить группу наименее значимых параметров и приравнять их к нулю. Если это сильно изменит функционирование сети, то можно часть значений приравнять не нулю, а ближайшим выделенным ("дешевым") значениям.

Если в ходе обучения решается и задача контрастирования, то в каждый момент параметры разделены на две группы: те, значения которых жестко зафиксированы и равны выделенным ("замороженные" параметры) и те, значения которых меняются в

ходе обучения.

для меняющихся параметров α в целевую функцию вводится штраф –

$$\varepsilon \sum \Phi_r(\alpha - \alpha_r), \quad (111)$$

где $\varepsilon > 0$ – возрастающая в ходе обучения интенсивность контрастирования, α_r – выделенные значения, Φ_r – потенциалы "притяжения" к выделенным значениям.

Можно выбирать Φ_r в виде

$$\Phi_r(x) = \frac{a_r}{b_r + x^2}, \quad a_r, b_r > 0, \quad (112)$$

где a_r/b_r – "глубина" потенциальной ямы, b_r – ее "ширина".

Глубина и ширина должны подбираться из стоимости данного значения.

Величина ε постепенно нарастает в ходе обучения. Кроме того, для успешного контрастирования необходимо производить обучение с периодически повторяемыми случайными сдвигами обучаемых параметров и постепенно увеличивающимся уровнем этих сдвигов. Опыт показывает, что при этом достигается устойчивость к сдвигам, достигающим 0.1 максимальной величины параметров и даже более. Такая устойчивость позволяет без существенного нарушения навыков сети производить ее контрастирование.

Контрастирование предполагает процедуры замораживания и размораживания параметров. Замораживание может производиться, например, если значение параметра отличается от выделенного не более, чем на достаточно малое $\delta > 0$. Тогда параметру присваивается это выделенное значение и он перестает меняться в ходе обучения. Размораживание производится, если обучение сильно замедлилось, а навык еще не выработан. Тогда на основе анализа чувствительности наиболее значимые параметры сети размораживаются, то есть включаются в обучение.

Возможно, что после обучения и контрастирования останется несколько наиболее "упорных" элементов, не поддающихся контрастированию без разрушения навыков сети. Что делать с ними? Либо смириться и оставить все как есть, либо разморозить побольше параметров, повысить уровень случайных возмущений и учить с контрастированием дальше.

9.4. Сглаживание характеристик - средство для улучшения интерполяционных и экстраполяционных навыков

Правильное решение сетью обучающих примеров еще, строго говоря, не означает, что она будет хорошо работать при других значениях входных сигналов. Важно еще, насколько полученные навыки допускают интерполяцию и экстраполяцию. При этом большую роль играет крутизна характеристик тех элементов, из которых составлена сеть.

Поясним роль крутизны на простейшем примере. Пусть необходимо построить устройство, вычисляющее положительную функцию одного^{*} переменного $f(x)$, и в нашем распоряжении - набор устройств, вычисляющих функции $\varphi(x-a)$ с "колоколообразным" графиком (рис. 21а). Требуется по заданным значениям $f(x_i)$ ($i=1, \dots, n$) построить линейную комбинацию

$$\sum_{j=1}^n \alpha_j \varphi(x-a_j), \quad n \geq n, \quad \alpha_j > 0, \quad (113)$$

приближающую данную функцию.

Чем уже колоколообразные графики φ , тем легче построить сумму, совпадающую с f в выбранных точках. Однако при этом ухудшаются интерполяционные возможности (рис. 21б). Обратно, при увеличении ширины построение приближенного выражения (113) усложняется, ошибка на заданном обучающем множестве увеличивается, но интерполяционные возможности полученного приближения повышаются. Начиная с некоторого предела ширины, характерного для функции f , ошибка при дальнейшем расширении начинает превышать разумные ограничения на точность. Поэтому существует оптимальная ширина - ошибка еще не слишком велика, а интерполяционные свойства уже достаточно хороши.

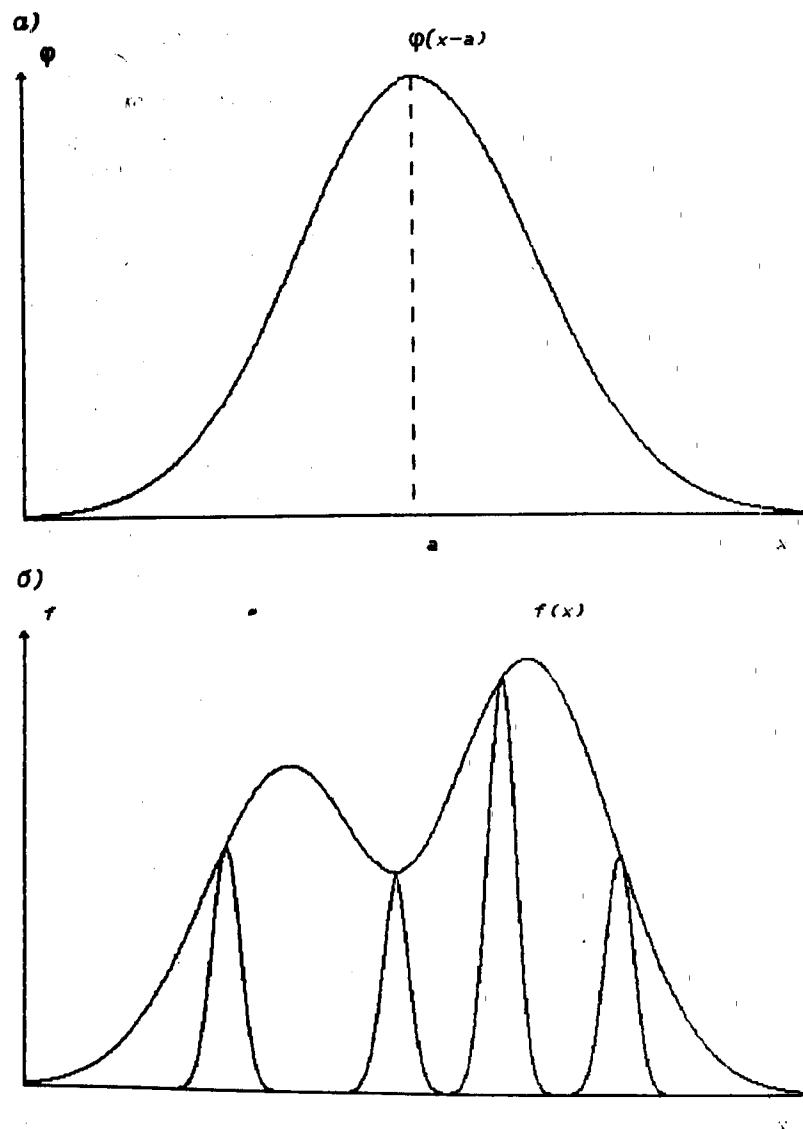


Рис. 21. Колоколообразная функция φ (а) и приближение функции f в заданных точках суммой узких пиков (б).

Приведенные соображения нетрудно формализовать, введя соответствующие критерии. Простейший способ: задать максимальную допустимую ошибку на обучающем множестве и увеличивать ширину Φ , пока эта точность достичима. Другой способ - ввести штраф за малую ширину и штраф за ошибку на обучающем множестве и минимизировать их сумму.

Аналогично обстоит дело и с нейрокомпьютером - чем более пологие характеристики имеют его элементы, тем больше у него способности к экстраполяции навыков, но, увы, тем хуже точность на обучающем множестве.

Так, например, для нейронов с характеристикой вход-выход вида

$$f(A) = \frac{A}{h+|A|} \quad (h > 0),$$

чем больше h - тем выше способности к экстраполяции и интерполяции, но хуже точность решения обучающих примеров. Компромисс достигается введением в оценку штрафа за крутизну, например,

$$\epsilon \sum (-\ln h_i), \quad (114)$$

где i - номер нейрона, $\epsilon > 0$ - параметр, подбираемый в ходе обучения. Из-за функции штрафа (114) в производные $\partial h_i / \partial h_j$ войдут слагаемые $-1/h_i$, которые будут отталкивать обучаемые характеристики нейронов от нуля.

Полагаем, что сглаживающие процедуры необходимы для систем с обучаемыми нелинейными элементами. По способу выполнения они близки к отдельным частям процедуры контрастирования - налагаются штрафы за нежелательные значения параметров.

10. АЛГОРИТМЫ ОБУЧЕНИЯ

10.1. ОБУЧЕНИЕ НЕЙРОННЫХ СЕТЕЙ И АЛГОРИТМЫ ОПТИМИЗАЦИИ

Предлагается рассматривать обучение нейронных сетей как задачу оптимизации. Это означает, что весь мощный арсенал методов оптимизации может быть испытан для обучения. Так и видится: нейрокомпьютеры наискорейшего спуска, нейрокомпьютеры Ньютона, Флетчера и т.п. - по названию метода нелинейной оптимизации.

Существует, однако, ряд специфических ограничений. Они связаны с огромной размерностью задачи обучения. Число параметров может достигать 10^8 - и даже более. Уже в простейших программных имитаторах на персональных компьютерах подбирается $10^3 - 10^4$ параметров.

Из-за высокой размерности возникает два требования к алгоритму:

1. **Ограничение по памяти.** Пусть n - число параметров. Если алгоритм требует затрат памяти порядка n^2 , то он вряд ли применим для обучения больших нейрокомпьютеров. Вообще говоря, желательно иметь алгоритмы, которые требуют затрат памяти порядка Kn , $K = \text{const}$.
2. **Возможность параллельного выполнения** наиболее трудоемких этапов алгоритма и желательно - нейронной сетью. Если какой-либо особо привлекательный алгоритм требует память порядка n^2 , то его все же можно использовать, если с помощью анализа чувствительности и, возможно, контрастирования сократить число обучаемых параметров до разумных пределов.

Еще два обстоятельства связаны с компьютерной спецификой.

1. Обученный нейрокомпьютер должен с приемлемой точностью решать все тестовые задачи (или, быть может, почти все с очень малой частью исключений). Поэтому задача обучения становится по существу многокритериальной задачей

оптимизации: надо найти точку общего минимума большого числа функций. обучение нейрокомпьютера исходит из гипотезы о существовании такой точки. Основания гипотезы - очень большое число переменных и сходство между функциями. Само понятие "сходство" здесь трудно формализовать, но опыт показывает что предположение о существовании общего минимума или, точнее, точек, где значения всех оценок мало отличается от минимальных, часто оправдывается.

2. Обученный нейрокомпьютер должен иметь возможность приобретать новые навыки без утраты старых. Возможно более слабое требование: новые навыки могут сопровождаться потерей точности в старых, но эта потеря не должна быть особенно существенной, а качественные изменения должны быть исключены. Это означает, что в достаточно большой окрестности найденной точки общего минимума оценки оценки произведенных изменений.

Итак, имеем четыре специфических ограничения, выдающих пример из серии придется несколько раз - и при выборе обучения нейрокомпьютера из общих задач оптимизации: направления изменения параметров, и при оценке производимых астрономическое число параметров, необходимость высокого шага. Быстрая память дорога и объем ее ограничен. Наконец, параллелизма при обучении, многокритериальность решаемых задач, необходимость найти достаточно широкую область, в которой значения всех минимизируемых функций близки к наименьшему количеству наименее простых примеров. Поэтому минимальным.

Еще одно - мы пока ни слова не сказали о возможной бинарной оценке: хорошо-плохо, угадал-не угадал. Возможность улучшать работу сети при наличии только бинарных оценок тоже существует. об этом - далее.

10.2. Способ предъявления примеров и принцип постраничного обучения

Простейший способ предъявления примеров состоит в том,

что они предъявляются по одному, с помощью алгоритма оптимизации находится изменение параметров сети, улучшающее ее работу с этим примером, далее это изменение осуществляется и сети с новыми значениями параметров предъявляется следующий пример.

Опыт показывает, что при таком поодиночном предъявлении обучение происходит медленно и, увы, неустойчиво. Оценки за решаемые примеры то падают, то вновь растут, возможны длительные колебания, некоторые алгоритмы не сходятся вообще. Характерный пример графика оценок для удачного обучения приведен на рис. 22.

В чем дело? Ситуация типичная для многокритериальных задач. Направление обучения для них лучше формировать, используя сразу несколько критериев (примеров), в идеале - все. Тем более, нельзя ограничиваться одним примером при исключении.

Итак, из априорных соображений лучше использовать сразу значения этих функций незначительно отличаются от минимальных. Мало того, что должна быть найдена точка все примеры. увы, тут есть немало ограничений. Во-первых, общего минимума, так она еще должна лежать в достаточно всех примеров заранее может не быть - исходный "задачник" широкой измененности, где значения всех минимизируемых может расширяться. Во-вторых, при обработке серии примеров функций близки к минимуму. для решения этой задачи нужны жалательно, чтобы обмен с памятью был достаточно быстрым и не специальные средства. Они описаны далее.

Итак, из априорных соображений лучше использовать сразу значения этих функций незначительно отличаются от минимальных. Мало того, что должна быть найдена точка все примеры. увы, тут есть немало ограничений. Во-первых, общего минимума, так она еще должна лежать в достаточно всех примеров заранее может не быть - исходный "задачник" широкой измененности, где значения всех минимизируемых может расширяться. Во-вторых, при обработке серии примеров функций близки к минимуму. для решения этой задачи нужны жалательно, чтобы обмен с памятью был достаточно быстрым и не специальные средства. Они описаны далее.

Страница - серия примеров, предъявляемая сети для обучения. Выбор направления и шага для изменения параметров сети осуществляется с помощью анализа сразу всех примеров страницы. При работе с данной страницей каждый шаг изменения параметров служит для минимизации некоторой функции от оценок, которые сеть получает при решении примеров данной страницы. Это может быть сумма оценок (средняя оценка), сумма с весами, средняя квадратичная оценка и т. п. - см. раздел 6.2.

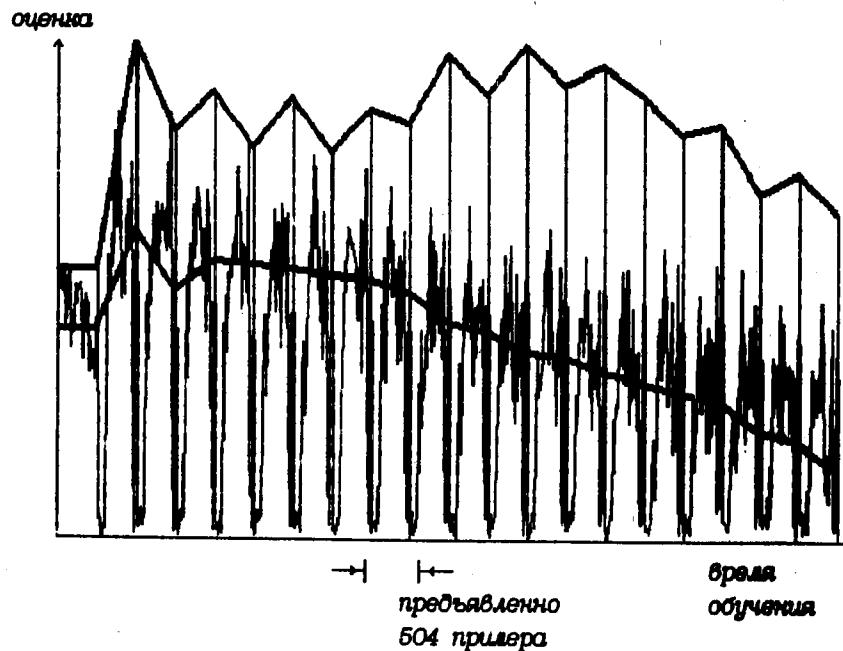


Рис. 22. Характерный график оценок при обучении предъявляемых по очереди примеров. Неоднократно последовательно шло обучение 504-м примерам. Звенья вершин ломаной соединяют точки максимальных оценок (из 504 примеров). Нижняя ломаная показывает динамику средней оценки 504 примеров.

Веса оценкам могут назначаться, исходя из субъективного представления о важности данного примера, текущей оценки, скорости, с которой сеть учится решать пример (медленно учится – можно увеличить вес), и т. п.

Преимущества постраничного обучения можно пояснить так. Если примеры предъявляются сети по одному, то она на каждом этапе нарабатывает давать на данный вход нужный выход. Но можно ли отличить это умение от просто выдачи данного выхода независимо от входов? Меняется предъявленный пример – и происходит привыкание к новым требованиям к выходным сигналам.

Если долго так учить, то что-то может и остаться кроме умения отвечать так, как это требовалось в последний раз. Это "что-то" и есть нужный навык, но идти к нему столь косвенным путем слишком долго и не всегда надежно. Как показывает опыт, постраничное обучение в задачах распознавания визуальных образов и классификации дает выигрыш не менее, чем в 10-100 раз при прочих равных. Важно, чтобы каждая страница была достаточно разнообразной и в ней присутствовали представители разных образов.

Еще одно замечание: если примеры предъявляются по одному, то шаг изменения параметров для улучшения решения одного примера должен быть достаточно малым. Иначе навык работы с другими примерами необратимо разрушится. Поэтому в этом случае нет смысла применять быстросходящиеся методы, дающие значительное улучшение.

В ходе обучения объем страницы и разнообразие примеров на ней можно увеличить: совсем необученная сеть слишком медленно учится на больших страницах, а после "начального образования" появляются возможности для быстрого освоения все больших страниц.

10.3. Две идеологии в построении обучения сети: популяционная и генетическая

Существует как минимум два принципиально различных подхода к построению алгоритмов, обучающих нейронную сеть. В

первом из них продуктом, выдаваемым для использования, является отлаженный и по возможности наилучший алгоритм обучения. Усилиями различных исследователей создается популяция алгоритмов. Пользователь выбирает из них тот, что пригоден для решаемых задач. В популяции алгоритмов идет отбор, некоторые вымирают, другие - захватывают жизненное пространство. Такой путь назовем **популяционным**.

Если создатель библиотеки алгоритмов следует популяционному пути, то он должен собрать различные алгоритмы и предоставить пользователю возможность выбора.

Однако вряд ли существует алгоритм, одинаково эффективный для всех задач, а многообразие возможных алгоритмов слишком велико, чтобы у одного пользователя была возможность апробировать их все. Поэтому дополнительно к популяционному подходу возникает другой. Его можно назвать **генетическим**. В нем основным элементом является "ген" - отдельная процедура, а пользователю даются широкие возможности собирать алгоритмы из процедур, а не только выбирать готовые алгоритмы.

При таком подходе популяция возможных алгоритмов резко возрастает, возрастают и возможности подбора алгоритма под задачу. Недостаток у этого подхода тоже есть - повышенные требования к пользователю, который должен понимать устройство отдельных процедур и логику сборки алгоритма.

В нашей исследовательской группе принят, скорее, генетический подход. Опыт показывает, что отбор алгоритмов и при нем идет весьма интенсивно, а некоторые процедуры вовсе выпадают из пользования. Интересно, что различные пользователи даже на задачах одного типа обнаруживают разные склонности и по конструкции алгоритма нередко можно угадать автора.

Если создатель библиотеки алгоритмов следует генетическому пути, то он должен собрать различные процедуры и построить программу-редактор, учитывающую логику конструирования алгоритмов.

Конечно, вряд ли найдутся сторонники чисто генетического или чисто популяционного подхода. Истина, как это часто

бывает, где-то между крайностями. Однако нам кажется, что в настоящий момент и ближайшие несколько лет генетический подход более перспективен - он дает большее потенциальное разнообразие для эволюции алгоритмов и, кроме того, имеет большее образовательное значение.

10.4. Операции, выполняемые сетью

При описании алгоритмов обучения полезно выделить операции, выполняемые самой сетью. Комбинируя их, управляющая программа осуществляет обучение.

Из операций, выполняемых сетью, можно выделить те, которые выполняются с одним примером. Над ними надстраиваются операции со страницей примеров.

Итак, операции первого этажа, выполняемые с одним примером.

1. **Прямое функционирование**: получить выходные сигналы данного примера, функционировать θ тактов. На каждом такте могут дополнительно получаться входные сигналы. На выходе - массив выходных сигналов. Они могут сниматься с различных тактов.
2. **Прямое функционирование с оценкой**. Надстраивается над прямым функционированием. Дополнительно вводится функция оценки, вычисляемая для каждого примера по выходным сигналам. На выходе - оценка того, как сеть решает пример и само решение - выходные сигналы.
3. **Запоминание карты**. Для многих алгоритмов нужно не только изменять параметры, но и помнить, какими они были до нескольких последних изменений. Поэтому нужна дополнительная память на один или несколько массивов значений параметров и операция: запоминание карты параметров. Число карт, которое можно запомнить - важная характеристика нейрокомпьютера.
4. **Нагруженное прямое функционирование**. Надстраивается над прямым функционированием. Производятся дополнительные вычисления, а также запоминание информации. Подробно это было описано в разделах, посвященных двойственности.

ограничимся напоминанием. Если в обычном прямом функционировании работает набор элементов, вычисляющих функции выхода от своих входов A_i и вектора параметров L_i :
 $f_i(A_{it}, L_i)$, (t - номер такта),

то при нагруженном нужно запомнить все значения f_i , вычислить все производные f_i по входным сигналам и параметрам, запомнить значения этих производных. Кроме того, в нагруженное прямое функционирование может входить вычисление и запоминание некоторых дополнительных показателей чувствительности. Это дополнение сейчас обсуждаться не будет. Для линейных f_i дополнение состоит в запоминании входных сигналов. Организация памяти для того, чтобы, с одной стороны, избежать дублирования запоминаемых величин, а с другой - обеспечить их быструю пересылку для исполнения нужных операций - важнейшая задача, но она выходит за рамки нашей темы.

5. **Обратное функционирование.** Выполняется после нагруженного прямого функционирования. Вычисляются и запоминаются производные функции оценки по параметрам. Дополнительно могут вычисляться и запоминаться производные оценки по входным сигналам.

6. **Дважды нагруженное прямое функционирование**, в ходе которого дополнительно вычисляются все величины, необходимые для обратно-обратного функционирования - см. раздел 7.6.

7. **Обратно-обратное функционирование.** Выполняется после последовательно проведенных дважды нагруженного прямого функционирования и обратного функционирования. В обратно-обратном функционировании вычисляется градиент произвольного подаваемого сети функционала от градиента функции оценки. Обычно - градиент скалярного квадрата градиента.

Операции 1, 2, 3, являются обязательными для обучения сети любыми методами, использующими оценку, операции 4, 5, - для обучения градиентными методами, а результаты 6, 7, используются для дополнительного ускорения градиентных методов.

Вычисление оценки по известным выходным сигналам лучше - быстрее производить с участием сети, но в принципе можно и без него. Это же относится к вычислению производных оценки по входным сигналам, то есть тех величин, которые подаются на вход сети при обратном функционировании.

Описаны основные операции, выполняемые сетью при работе с одним примером. Из них составляются операции, выполняемые со страницей.

8. **Оценка страницы.** Для всех примеров страницы проводится прямое функционирование (операция 1), вычисляются оценки для каждого примера. Исходя из них, вычисляется оценка страницы, например, сумма оценок по примерам, взвешенная сумма, либо другая функция, назначаемая пользователем извне.
9. **Накопление градиента.** Для каждого примера страницы проводятся нагруженное прямое функционирование и обратное функционирование, последовательно накапливается градиент функции, оценивающей работу сети со страницей:

$$\nabla H_{pq}(H_1, \dots, H_p) = \sum_i \frac{\partial H_{pq}}{\partial H_i} \nabla H_i;$$

вычисляются производные по параметрам и, если необходимо, по входным сигналам.

10. **Накопление градиента взвешенной суммы квадратов градиентов** отдельных примеров. Аналогично предыдущей операции накапливаются производные

$$\nabla \left(\sum_i (\nabla H_i)^2 \right).$$

Это происходит в результате проведения на каждом примере дважды нагруженного прямого функционирования, обратного и обратно-обратного.

11. **Формирование локальных массивов показателей чувствительности** по примерам страницы. Составляются массивы показателей чувствительности для различных частей сети, например, для весов входных синапсов каждого нейрона. Для конкретной реализации должен быть выбран тип показателей чувствительности и способ составления массивов (см. соответствующие разделы).

12. **Сортировка локальных массивов показателей**

чувствительности и выбор для каждого из них К наиболее или наименее значимых параметров сети. Число К задается сетью извне.

13. **Замораживание** заданного числа наименее значимых параметров - их выделение и присвоение им ближайших выделенных для контрастирования фиксированных значений. Значимость определяется по локальному анализу чувствительности, число замораживаемых параметров - одно для каждого локального массива.

14. **Размораживание** заданного числа наиболее значимых параметров - удаление их из списка замороженных параметров. Значимость опять же определяется по локальному анализу чувствительности, число размораживаемых параметров задается одно для всех локальных массивов.

Операция 8 необходима для любого алгоритма обучения, 9 - для обучения градиентными методами и их обобщениями, 10, 11, 12, для алгоритмов, ускоряющих работу градиентных методов, 13, 14 - для контрастирования.

Заметим, что штрафы за нежелательные значения параметров легко учесть как в процедурах оценивания, так и при вычислении всех производных. Отдельно обсуждать это не будем.

Наконец, есть группа операций, выполняемых сетью без примеров - операции по изменению карты параметров. Обращение управляющей системы к каждому отдельному параметру системы невозможно, или, по крайней мере, непрактично из-за больших затрат времени. Поэтому целесообразно выделять во множестве параметров отдельные подмножества - "раскрашивать" сеть. Внешнее управляющее устройство может передавать сети команды, требующие выполнить однотипные операции со всеми параметрами данного цвета.

Кроме текущего массива параметров сети есть несколько параллельных ему массивов: предыдущие параметры (до последнего изменения, до начала данного цикла обучения, до начала некоторых предыдущих циклов обучения и т.п.); градиент функции оценки H_{pg} - массив частных производных H_{pg} по параметрам, градиенты на некоторых предыдущих циклах обучения, градиент суммы квадратов градиентов по отдельным

примерам показатели чувствительности и т.п. Все эти массивы раскрашиваются так же, как и массив соответствующих параметров.

15. **Элементарное изменение** в раскрашенной сети состоит в следующем: каждый параметр данного цвета заменяется на некоторую функцию от этого параметра и соответствующих ему элементов других массивов. Например:

$$\alpha_i := \alpha_i - \varepsilon \frac{\partial H}{\partial \alpha_i}$$

градиентный шаг, величина шага ε назначается извне и зависит от цвета; или

$$\alpha_i := \alpha_i - \varepsilon (\alpha_i - \alpha_{i0})$$

партак шаг, α_{i0} - значение α_i на одном из прошлых шагов обучения, ε назначается извне и зависит от цвета; или

$$\alpha_i := \alpha_i - \varepsilon \alpha_i \frac{\partial H}{\partial \alpha_i}$$

и т.д.

Вычисление изменений α_i использует только величины, лежащие в том же срезе параллельных массивов. Поэтому здесь можно достичь очень высокой степени параллельности вычислений. Какие функции допускаются при вычислении изменений? Возможны варианты. Перечислим основные из них в порядке усложнения.

А. линейные функции с задаваемыми извне коэффициентами.
Б. линейные функции с задаваемыми извне коэффициентами и проективным учетом ограничений. Дело в том, что для многих параметров сети из различных соображений возможны ограничения вида.

$$\alpha_{i \max} \geq \alpha_i \geq \alpha_{i \min}$$

Тогда проективный учет ограничений состоит в том, что после вычисления необходимого изменения его результат сравнивается с $\alpha_{i \max}$ и $\alpha_{i \min}$. Если он больше $\alpha_{i \max}$, то α_i приравнивается $\alpha_{i \max}$. Если же меньше $\alpha_{i \min}$, то α_i приравнивается $\alpha_{i \min}$. Например, пусть

$$w = \alpha_i - \varepsilon \frac{\partial H}{\partial \alpha_i}$$

$$\alpha_i := \begin{cases} w, & \text{если } \alpha_{i \max} \geq w \geq \alpha_{i \min}; \\ \alpha_{i \max}, & \text{если } w > \alpha_{i \max}; \\ \alpha_{i \min}, & \text{если } w < \alpha_{i \min}. \end{cases}$$

В. Многочлены с определяемыми извне коэффициентами.

Г. То же, что и В, но с проективным учетом ограничений.

Д. Рациональные функции – отношения многочленов с определяемыми извне коэффициентами.

Е. То же, что и Д, но с проективным учетом ограничений.

Кроме добавления определенной функции к параметрам α_i , есть еще одна простая, но очень важная операция.

16. Случайное изменение параметров. К каждому α_i данного цвета добавляется случайная величина, равномерно распределенная в некотором отрезке. Границы задаются извне, ограничения учитываются проективно. Для введения и изменения ограничений должны существовать соответствующие операции.

17. Задать для всех параметров данного цвета ограничения

$$\alpha_{\max} \geq \alpha_i \geq \alpha_{\min}.$$

Величины α_{\max} , α_{\min} одинаковы для всех параметров данного цвета и задаются извне.

Необходимы также операции по раскраске сети. Потенциальное разнообразие здесь очень велико. Ограничимся пока перечислением четырех основных способов раскраски.

А. Априорный (или структурный). Цвета назначаются до обучения, исходя из структуры сети. Например, один цвет – параметры нейронов, другой – синапсов связи, третий – синапсов памяти. Другой пример – для многослойной сети синапсы каждого слоя окрашиваются в свой цвет.

Б. Точечный. Раскраска производится по условиям, проверяемым в одном срезе параллельных массивов: используются α_i , $\partial n_{pg} / \partial \alpha_i$, χ_i и т.п., то есть величины, относящиеся к α_i .

В. Локальный. Раскраска производится по анализу локальных массивов, например, по массивам весов входных синапсов, принадлежащих одному нейрону, и параллельным им локальным массивам – производных n_{pg} , предыдущих значений параметров, показателей чувствительности и т.п. Пример локальной раскраски – выделение наиболее значимых параметров по

локальному анализу чувствительности (операция 12).

Глобальная раскраска использует сразу все параметры и является дорогостоящей по времени операцией. В больших сетях ее использование вряд ли полезно. Завершая наш список, укажем две операции с цветами.

18. Назначение нового цвета. Сеть получает тип назначения – точечный, локальный или глобальный и условие, по которому формируется новый цвет. В конкретизации здесь нуждаются допускаемые условия.

19. Слияние цветов. Параметры нескольких цветов окрашиваются в один цвет, прежние цвета отменяются.

Итак, мы обсудили, что может и должна делать сеть, выделили основные операции. Приведенное описание не относится к какому-либо конкретному нейрокомпьютеру и потому, естественно, имеет эскизный характер. Однако оно позволяет двигаться дальше в обсуждении алгоритмов обучения.

10.5. Учет ограничений при обучении

В предыдущем разделе уже упоминалось, что для параметров сети возможны ограничения простейшего вида

$$\alpha_{i \max} \geq \alpha_i \geq \alpha_{i \min}.$$

Они вводятся из разных соображений: чтобы избежать слишком крутых или, наоборот, слишком пологих характеристик нейронов, чтобы предотвратить появление слишком больших коэффициентов усиления сигнала на синапсах и т.п.

Учитывать ограничения можно, например, методом штрафных функций либо методом проекций.

Использование метода штрафных функций означает, что в оценку добавляются штрафы за выход параметров из области ограничений. Это не требует никаких дополнительных изменений сети – меняется только функция оценки, а в градиенты и вводятся производные штрафных функций.

Проективный метод предполагает, что если сети предлагается изменение параметров

$$\alpha_i := w_i$$

и w_i для некоторых i выходит за ограничения, то следует положить

$$\alpha_i = \begin{cases} w_i & \text{если } \alpha_{i\max} \geq w_i \geq \alpha_{i\min}; \\ \alpha_{i\max} & \text{если } w_i > \alpha_{i\max}; \\ \alpha_{i\min} & \text{если } w_i < \alpha_{i\min}, \end{cases}$$

После этого обучение (то есть минимизация H_{pg}) продолжается.

В нашем опыте проективный метод не приводил к каким-либо затруднениям. Обращение со штрафными функциями было менее успешным.

Далее мы будем полагать, что ограничения учтены одним из указанных методов. Будем говорить об обучении как о безусловной минимизации, подразумевая либо наличие штрафных функций, либо описанную проекционную поправку, вносимую во все изменения параметров.

10.6. Одномерная оптимизация

Все пошаговые методы оптимизации состоят из двух важнейших частей: выбор направления и выбор шага в данном направлении. Методы одномерной оптимизации дают эффективный способ для выбора шага.

Заметим, что для обучения по отдельным примерам, то есть без постраничной организации методы одномерной оптимизации неприменимы – они слишком эффективны, поэтому давая заметное улучшение на одном обучающем примере, они часто разрушают навык решения предыдущих. При постраничном обучении такого обычно не происходит.

Из огромного количества алгоритмов одномерной оптимизации опишем один. Он хорошо показал себя в обучении нейронных сетей.

Алгоритм использует процедуру вычисления минимизируемой функции (функции ошибки на странице $H=H_{pg}$). Перед началом работы он получает начальную точку – вектор α^0 , направление движения – вектор s , начальный шаг d_0 и информацию о том, является ли s безусловным направлением спуска. Последнее

означает априорную информацию о том, существует ли такое $\epsilon > 0$, что при $0 < d < \epsilon$

$$H(\alpha^0) > H(\alpha^0 + d s).$$

Если известно, что существует, то s – безусловное направление спуска и при минимизации разумно ограничиваться случаем $d > 0$. Если неизвестно, то может оказаться полезным и выбор $d < 0$.

Безусловным направлением спуска является антиградиент n .

Кроме того, алгоритм может получать вычисленные ранее значения

$$H(\alpha^0), H(\alpha^0 + d_0 s).$$

Если какое-либо из них или оба неизвестны, то эти значения вычисляются и сравниваются.

Если $H(\alpha^0) \geq H(\alpha^0 + d_0 s)$, то выполняется процедура Increase (возрастание шага). В противном случае приходим к процедуре Decrease (убывание шага).

Обратите внимание на знак неравенства в условии: \geq , а не $>$. Это существенно при программной реализации – шаг может оказаться слишком малым, а улучшение – незаметным. Для проверки шаг надо увеличивать, а не уменьшать.

Increase. Процедура получает α^0 , s , d_0 и значения минимизируемой функции $H(\alpha^0)$, $H(\alpha^0 + d_0 s)$. При этом $H(\alpha^0) \geq H(\alpha^0 + d_0 s)$. Используется процедура вычисления минимизируемой функции.

В процедуре Increase используются константа возрастания r , три переменных шага $d_1 < d_2 < d_3$ и три – для соответствующих значений функции H_1 , H_2 , H_3 . Получив входные данные, полагаем

$$d_1 := 0, d_2 := d_0, d_3 := rd_0, H_1 := H(\alpha^0),$$

$$H_2 := H(\alpha^0 + d_2 s)$$

Вычисляем $H_3 = H(\alpha^0 + d_3 s)$. Если $H_3 > H_2$, то переходим к параболическому поиску – процедура Parabola. В противном случае проверяем, образует ли тройка аргументов $d_{1,2,3}$ и значений $H_{1,2,3}$ выпуклую комбинацию:

$$H_2 < \frac{d_3 - d_2}{d_3 - d_1} H_1 + \frac{d_2 - d_1}{d_3 - d_1} H_3. \quad (115)$$

Если это верно, то следующий шаг делается в вершину параболы, график которой проходит через точки (d_i, H_i) ($i=1, 2, 3$); $H_i = ad_i^2 + bd_i^2 + c$ ($i=1, 2, 3$). Вычисляем

$$W = -\frac{b}{2a} = \frac{d_3 + d_1}{2} - \frac{H_3 - H_1}{\frac{H_3 - H_2}{d_3 - d_2} - \frac{H_2 - H_1}{d_2 - d_1}}. \quad (116)$$

Если $W > d_3$, то полагаем $d_1 := d_2$, $d_2 := d_3$, $d_3 := W$, $H_1 := H_2$, $H_2 := H_3$, вычисляем $H_3 = H(\alpha^0 + d_3 s)$ и возвращаемся к проверке условия $H_3 > H_2$.

Если $W < d_3$ (заметим, что при указанных условиях всегда $W > d_2$), то полагаем $d_1 := d_2$, $d_2 := W$, $H_1 := H_2$, вычисляем $H_2 = H(\alpha^0 + d_2 s)$ и возвращаемся к началу.

Если же условие выпуклости (115) не выполняется, то полагаем $d_1 := d_2$, $d_2 := d_3$, $d_3 := rd_3$ и тоже возвращаемся к началу.

Дополнительные проверки. Бесконечный шаг: $d_3 > A$, где A – заранее заданное большое число. Процедура одномерной оптимизации для безусловного направления спуска заканчивается сообщением "бесконечный шаг", для небезусловного возможен перезапуск с $d_0 := -d_0$ (идем в противоположную сторону). Остановка: если после вычисления вершины параболы W , присвоения новых значений d_1 , H_1 и вычисления недостающего H_1 выполнены условия

$$\frac{|d_3 - d_2|}{d_3} < \varepsilon, |H_3 - H_2| < \delta,$$

где ε и δ – заданные малые числа, то оптимизация заканчивается, в качестве результата процедуры выдается наименьшее из H_2, H_3 и соответствующая этому наименьшему значению величина шага.

Decrease. Эта процедура существенно различна для безусловных и небезусловных направлений спуска. Для безусловных направлений спуска шаг дробится (умножается на константу r , $0 < r < 1$) до тех пор, пока $H(\alpha^0 + ds)$ не станет

меньше, чем $H(\alpha^0)$, либо шаг не станет нулевым (меньше заранее заданного малого $\gamma > 0$). Если шаг стал нулевым, одномерная оптимизация заканчивается и выдается сообщение "нулевой шаг". Если после очередного уменьшения шага стало $H(\alpha^0) > H(\alpha^0 + pd\alpha)$ ($< H(\alpha^0 + ds)$), то переходим к параболическому поиску – процедуре *Parabola*.

Для небезусловных направлений оптимизации сразу делается шаг в противоположном направлении и вычисляется $H(\alpha^0 - d_0 s)$. Если

$$H(\alpha^0) > H(\alpha^0 + d_0 s),$$

то переходим к процедуре *Increase*, полагая $d_0 := -d_0$.

В противном случае переходим к параболическому поиску – процедуре *Parabola*.

Parabola. Получает α^0 , s три различных значения шага $d_1 < d_2 < d_3$ и соответствующие значения функции H_1 , H_2 , H_3 , среднее из которых меньше одного из крайних и не превосходит другого:

$$H_1 \geq H_2 < H_3 \text{ или } H_1 > H_2 \leq H_3. \quad (117)$$

На каждом шаге снова строится тройка значений d , для которых значения H удовлетворяют условию (117). Используется процедура вычисления оценки.

Вычисляется W (116) и $H_w = (\alpha^0 + ws)$. Из двух величин H_2 , H_w выбирается меньшая. Если это H_w , то выясняется $W > d_2$ или нет. Если $W > d_2$, то полагаем $d_1 := d_2$, $d_2 := W$. Если нет, то полагаем $d_3 := d_2$, $d_2 := W$. Если же $H_2 < H_w$, также сравниваем W и d_2 . Если $W > d_2$, то полагаем $d_3 := W$. Если нет, то полагаем $d_1 := W$. Соответственные значения присваиваются и H_1 .

После этого возвращаемся к началу и вычисляем новое W .

Выход из параболического поиска может совершиться по нескольким основаниям.

- Если в последовательно вычисляемых вершинах значения H отличаются меньше, чем на заранее заданную малую величину.
- Если $d_3 - d_2$ или $d_2 - d_1$ становятся меньше некоторой заранее заданной малой величины.
- Если парабола строилась N раз, где N – заранее заданное натуральное число.

Условие выхода конструируется из 1, 2, 3 либо как

дизъюнкция - 1 или 2 или 3, либо как (1&2) или 3.

На выходе из процедуры получаем наименьшее из построенных значений H и соответствующую величину шага.

Итак, в результате одномерной оптимизации получаем величину шага и значение H , либо сообщение о том, что улучшить оценку в данном направлении не удается.

Кроме того, можно получить полезную дополнительную информацию - оценку вторых производных H в данном направлении в точке выбранного шага. Результативный выход из оптимизации возможен только после построения параболы. Пусть она строится по трем значениям шага $d_1 < d_2 < d_3$ и соответствующим трем значениям оценки H_1, H_2, H_3 . Тогда в качестве приближения можно принять вторую производную этой квадратичной зависимости:

$$\frac{d^2}{dx^2} H(\alpha^0 + x s) \approx 2 \left[\frac{H_3 - H_2}{d_3 - d_2} - \frac{H_2 - H_1}{d_2 - d_1} \right] : (d_3 - d_1). \quad (118)$$

Ее можно использовать в процедурах многомерной оптимизации.

10.7. Двумерная оптимизация

Использование для оптимизации сразу двух направлений нередко оказывается намного эффективнее, чем одного. Как выбирать направления, мы обсудим позднее, а пока опишем приемлемый алгоритм двумерной оптимизации.

Алгоритм получает на входе начальный вектор α^0 , направления s_1, s_2 и начальные шаги в этих направлениях, а также указания на то, будет ли s_1 направлением безусловного спуска из точки α^0 , а s_2 - из точки, получаемой одномерной оптимизации из α^0 в направлении s_1 . Используется процедура вычисления оценки H ($H = H_{pg}$).

Примечания. 1. От перестановки s_1 и s_2 меняются начальные условия для работы алгоритма и, следовательно, результат. 2. Вектор s_2 может быть получен не сразу а после одномерной оптимизации в направлении s_1 , например, он может быть антиградиентом H , вычисленным в точке, которая найдена в результате первой одномерной оптимизации.

Алгоритм строится так. Сначала - одномерная оптимизация из начальной точки α^0 в направлении s_1 . Обозначим найденный шаг d_1 , соответствующее значение H - H_1 , а оценку второй производной (118) a_1 . Потом - второе применение одномерной оптимизации, начальная точка $\alpha^0 + d_1 s_1$, направление s_2 , результирующий шаг обозначим d_2 , соответствующее значение функции $H(\alpha^0 + d_1 s_1 + d_2 s_2) - H_2$, оценку второй производной - a_2 . Наконец, третье применение одномерной оптимизации. Начальная точка α^0 , направление $s_3 = d_1 s_1 + d_2 s_2$, начальный шаг 1 (и сразу Increase), на выходе - значение шага x , значение функции $H_3 = H(\alpha^0 + x(d_1 s_1 + d_2 s_2))$, оценка второй производной (118) a_3 .

В результате трех применений одномерной оптимизации получены три точки $\alpha^0 + d_1 s_1, \alpha^0 + d_1 s_1 + d_2 s_2, \alpha^0 + x(d_1 s_1 + d_2 s_2)$ три соответствующих значения функции $H_1 > H_2 > H_3$ и три оценки вторых производных (118):

$$\begin{aligned} a_1 &\approx \frac{d^2}{dy^2} (\alpha^0 + y s_1); \\ a_2 &\approx \frac{d^2}{dy^2} H(\alpha^0 + d_1 s_1 + y s_2); \\ a_3 &\approx \frac{d^2}{dy^2} H(\alpha^0 + y(d_1 s_1 + d_2 s_2)). \end{aligned}$$

Из a_1 можно получить оценку матрицы вторых производных функции h двух переменных y, z :

$$h(y, z) = H(\alpha^0 + y s_1 + z s_2);$$

$$\frac{\partial^2 h}{\partial y^2} \approx a_1; \quad \frac{\partial^2 h}{\partial y^2} \approx a_2; \quad \frac{\partial^2 h}{\partial y \partial z} \approx \frac{a_3 - d_1^2 a_1 - d_2^2 a_2}{2 d_1 d_2} (= a_{12}). \quad (119)$$

Далее проверяем положительную определенность матрицы

$$\begin{bmatrix} a_1 & a_{12} \\ a_{12} & a_2 \end{bmatrix} \quad (120)$$

Если она не является положительно определенной, то выходим из цикла двумерной оптимизации. На выходе - предлагаемое изменение вектора параметров $x(d_1 s_1 + d_2 s_2)$ и соответствующее значение функции H_3 .

Если же матрица (120) положительно определена, то переходим к дальнейшему улучшению по модифицированному

методу Ньютона с постоянной аппроксимацией матрицы вторых производных (120). Для очередного шага метода необходимы три точки (y_1, z_1) , (y_2, z_2) , (y_3, z_3) , не лежащие на одной прямой, и значения $h(y, z)$ (119) в этих точках. Находим такую линейную функцию $L(y, z) = py + qz + c$, что

$$h(y_i, z_i) = L(y_i, z_i) + Q(y_i, z_i), \quad i=1, 2, 3, \quad (121)$$

где Q – квадратичная форма

$$Q(y, z) = \frac{1}{2}a_1 y^2 + a_{12}yz + \frac{1}{2}a_2 z^2.$$

После этого находим минимум полинома $L(y, z) + Q(y, z)$. Точку минимума обозначим y_0, z_0 . Полагаем $(y_1, z_1) = (y_2, z_2)$; $(y_2, z_2) = (y_3, z_3)$; $(y_3, z_3) = (y_0, z_0)$. Вычисляем $h(y_3, z_3)$. Из линейных уравнений (121) снова ищем коэффициенты L – числа p, q, c . Вновь находим минимум $L + Q$ и т.д.

Критерии остановки:

1. Векторы (y_1, z_1) , (y_2, z_2) , (y_3, z_3) лежат на одной прямой:

$$|(y_3 - y_1)(z_3 - z_2) - (y_3 - y_2)(z_3 - z_1)| < \epsilon,$$

ϵ – заранее заданное малое число.

2. Новый вектор (y_3, z_3) мало отличается от предыдущего:

$$|y_3 - y_2| + |z_3 - z_2| < \delta,$$

δ – заранее заданное малое число.

3. Новое значение h мало отличается от предыдущего:

$$|h(y_3, z_3) - h(y_2, z_2)| < \gamma,$$

γ – заранее заданное малое число.

4. Число шагов превысило N , N – заранее заданное натуральное число.

Условие остановки – 1 или (2&3) или 4.

Замечание. Поскольку при знаконеопределенности формы Q происходит окончание работы, а не дальнейший поиск, быть может, естественнее назвать описанный алгоритм – алгоритм двумерного улучшения (а не оптимизации). Возможен вариант: для знаконеопределенных форм Q ищется собственный вектор v матрицы (120), соответствующий неположительному собственному числу и в направлениях $\pm v$ с начальной точкой $y = x d_1$, $z = x d_2$, проводится одномерная оптимизация. После этого – выход с наименьшим из найденных значений h и соответствующими

значениями y, z .

На выходе из описанной процедуры двумерной оптимизации (улучшения) получаем вектор изменений $y d_1 s_1 + z d_2 s_2$ и значение H или сообщение о том, что применить алгоритм не удается.

10.8. Наискорейший спуск, случайный поиск и партан методы

Мы уже имеем в своем распоряжении процедуры одномерной оптимизации и улучшающего поиска в двумерном пространстве. Вопрос: в каких направлениях проводить этот поиск.

Пусть задано начальное значение параметров α^0 и вычислена функция оценки $H = H(\alpha^0)$. Процедура одномерной оптимизации даст приближенное положение минимума $h(x) = H(\alpha^0 + xs)$ (вообще говоря, локального). Процедура двумерного поиска уменьшает значение $h(x, y) = H(\alpha^0 + xs_1 + ys_2)$. Теперь нужно определить направления s или s_1 и s_2 и начальные шаги.

Наиболее очевидный выбор s для одномерной оптимизации – направление анградиента H : $s = -\nabla H$. Выберем на каждом шаге это направление, потом – проведем одномерную оптимизацию, потом – снова вычислим градиент H и т.д. Это – метод наискорейшего спуска – примитивнейший из градиентных методов. Изредка работает хорошо.

Другой способ – случайный выбор направления s для одномерной оптимизации. Получаемый метод требует большого числа шагов, но зато предъявляет минимальные требования к сети – ему необходимо только прямое функционирование с вычислением оценки. Кроме того, встречаются ситуации, в которых метод наискорейшего спуска не работает, а поиск в случайном направлении медленно, но верно вытаскивает в нужную область параметров.

Для того, чтобы исправить недостатки наискорейшего спуска, существует огромное число методов. В этом разделе обсудим одно семейство таких методов – итерационный и модифицированный партан-методы.

Итерационный партан-метод (k -партан) строится так. В

начальной точке α^0 вычисляется градиент H и делается шаг наискорейшего спуска — для этого используется одномерная оптимизация. Далее — снова наискорейший спуск и так k раз. Начальные параметры α^0 при этом хранятся в памяти, промежуточные — нет. После k шагов наискорейшего спуска получаем α^k и проводим одномерную оптимизацию из α^0 в направлении $s = \alpha^k - \alpha^0$ (от α^0 к α^k) с начальным шагом $d=1$ (сразу Increase). После этого цикл повторяется.

Модифицированный партан-метод также требует запоминания дополнительной карты параметров. Он строится так. Из α^0 делается два шага наискорейшего спуска. Получаем α^1 , α^2 . Далее — одномерная оптимизация из α^0 в направлении $\alpha^k - \alpha^0$. Получаем α^3 . После этого α^0 уже не используется. Далее — наискорейший спуск из α^3 . Получаем α^4 . Потом одномерная оптимизация из α^2 в направлении $\alpha^4 - \alpha^2$, получаем α^5 и т.д. Четные α^{2k} получаем наискорейшим спуском из α^{2k-1} . Нечетные α^{2k+1} — одномерной оптимизацией из α^{2k-2} в направлении $s = \alpha^{2k} - \alpha^{2k-2}$ (начальный шаг $d=1$, сразу Increase).

По нашим наблюдениям, модифицированный партан-метод в задачах обучения работает лучше, чем k-партан.

Из наискорейшего спуска, одномерной оптимизации в случайному направлении и партан-методов можно построить небольшой конструктор алгоритмов. Обозначим s - шаг наискорейшего спуска, R - шаг оптимизации в случайному направлении. Запись будем читать слева направо. Например, запись

$$\alpha^0 \mathbf{SRRSS} \quad (122)$$

означает, что из начальной точки α^0 делается шаг наискорейшего спуска, получаем α^1 , потом из α^1 шаг оптимизации в случайном направлении, получаем α^2 , из α^2 - шаг оптимизации в случайном направлении, получаем α^3 , из α^3 - наискорейший спуск в α^4 , из α^4 - наискорейший спуск в α^5 .

Специфические шаги итерационного и модифицированного партан-методов будем обозначать квадратными скобками. Знак [означает, что параметры, полученные в процедурах, описанных до этого знака, запоминаются и будут служить начальной точкой для партан-шага. Знак] означает, что из вектора параметров

α^k , который был получен в процедурах, описанных до открытия скобки, делается шаг в направлении $\alpha^1 - \alpha^k$ ($1 < k$). Вектор α^1 получен в результате процедур, описанных до закрытия скобки. Величина шага подбирается методом одномерной оптимизации, его начальное значение равно 1 (сразу Increase). Например, если в предыдущей записи (122) поставить скобки

$$\alpha^0 \mathbf{S} \mathbf{R} [\mathbf{R} \mathbf{S}] \mathbf{S}, \quad (123)$$

это будет означать, что результат последовательного применения к α^0 сначала шага наискорейшего спуска, потом - одномерной оптимизации в случайном направлении (т.е. α^2) запоминается, потом - оптимизация в случайном направлении - получаем α^3 , из α^3 - наискорейший спуск в α^4 . Потом скобка закрывается, значит предпринимается шаг из точки, полученной перед открытием скобки в направлении точки, полученной перед закрытием: из α^2 проводится одномерная оптимизация в направлении $s = \alpha^4 - \alpha^2$. Получаем новое α^5 - не то, что прежде. И, наконец, наискорейший спуск из α^5 в α^6 . Результат - α^6 .

Схема итерационного партан-метода (k -партан)

$$\alpha^0 [SS \dots S] \dots [SS \dots S] \dots \dots$$

Схема модифицированного партан-метода:

$\alpha^0 [ss []s []s []s [] \dots$

Математическое обеспечение нейрокомпьютеров должно предполагать конструктор алгоритмов. Уже три операции: **R**, **S**, **U** дают возможность конструирования.

Укажем еще несколько привлекательных схем

$$\alpha^0 [RR \dots R] [RR \dots R] \dots$$

— **к раз** **к**

$$\alpha^* [[\dots [\underset{k}{s} [s] [s] [s] \dots [\underset{k}{s}]]]]$$

Заметим, что при использовании модифицированного партан-метода можно накапливать информацию о вторых производных и после каждого партан-шага производить несколько шагов двумерной оптимизации модифицированным методом Ньютона. Это недорогое по затратам времени

усовершенствование иногда бывает весьма эффективным. Его всегда можно использовать после того, как трижды выполнена процедура одномерной оптимизации по трем направлениям, лежащим в одной плоскости, если эти направления не являются параллельными.

10.9. Использование back-back процедуры для ускорения обучения

Направление наискорейшего спуска – антиградиент функции оценки H – традиционно считается неудачным для оптимизации. Во всех учебниках по оптимизации приводятся примеры уже довольно простых функций, с которыми метод наискорейшего спуска не может справиться.

В тех случаях, когда матрица вторых производных оценки D_2 положительно определена, наилучшим считается ньютоновское направление

$$-D_2^{-1}\nabla H. \quad (124)$$

Квадратичные формы минимизируются с использованием (124) за один шаг. Однако использовать эту формулу трудно по многим причинам.

1. Время. Поиск всех вторых производных и обращение D_2 требуют слишком много вычислительных затрат.
2. Память. Для решения задач большой размерности N приходится хранить N^2 элементов D_2^{-1} – слишком много.
3. А что, если D_2 не является положительно определенной?

для того, чтобы справится с этими трудностями, придумана масса ухищрений. По существу, вся литература по методам гладкой оптимизации посвящена этому.

Идея первая. Можно не дифференцировать H и не обращать матрицу D_2 , а использовать итерационную процедуру, позволяющую за много шагов накопить "что-то вроде D_2^{-1} ".

Идея вторая. Можно хранить в памяти не матрицу, а только результаты ее действия на некоторые векторы.

Идея использовать back-back процедуру основана на следующих соображениях. Пусть $\gamma > 0$ – некоторое число. Направление

$$s = -(1 + \gamma D_2)^{-1} \nabla H \quad (125)$$

является весьма перспективным. При больших γ оно приближается к ньютоновскому (124), при малых – к направлению наискорейшего спуска. Если D_2 положительно определена, то направление (125) при всех $\gamma > 0$ есть безусловное направление спуска:

$$(\nabla H, (1 + \gamma D_2)^{-1} \nabla H) > 0. \quad (126)$$

Разложим (125) по γ , оставив только линейное слагаемое. Получим

$$s' = -(1 - \gamma D_2) \nabla H. \quad (127)$$

При достаточно малых γ это – безусловное направление спуска (если $\gamma < 1/\lambda_{\max}$, λ_{\max} – наибольшее собственное число D_2).

Заметим, что

$$s' = -\nabla H + \gamma D_2 \nabla H = -\nabla H + \frac{\gamma}{2} \nabla (\nabla H)^2. \quad (128)$$

для каждого отдельного примера вектор $\nabla (\nabla H)^2$ вычисляется нейронной сетью в back-back процедуре, а потом результаты суммируются для всех примеров страницы.

Остается вопрос о выборе γ . Конечно, нетрудно установить, что в задачах минимизации функции H в формуле (127) полезно выбирать γ так:

$$\gamma = \frac{1}{\lambda_{\min} + \lambda_{\max}}, \quad (129)$$

где λ_{\min} и λ_{\max} – наименьшее и наибольшее собственные числа D_2 .

Но, во-первых, они неизвестны, а во-вторых, и матрицу D_2 лучше не вычислять.

Поступим следующим образом. Обозначим

$$B_{pg} = \sum_i \nabla (\nabla H_i)^2$$

сумму результатов back-back процедуры по примерам страницы. Отнормируем векторы ∇H_{pg} и B_{pg} – найдем их евклидовы нормы и разделим на них. Нормировка может быть выполнена в параллельном режиме и займет не больше времени, чем решение сетью одного примера.

Представим искомое направление спуска в виде

$$s(\gamma) = \frac{\nabla H_{p_g}}{\|\nabla H_{p_g}\|} \rightarrow \gamma \frac{B_{p_g}}{\|B_{p_g}\|}, \gamma > 0. \quad (130)$$

Как выбирать γ ? Если γ лежит между нулем и единицей, то s – безусловное направление спуска. Можно задаться фиксированным значением γ ($0 < \gamma < 1$). Например, взять $\gamma=0,5$ – в этом случае угол между s и направлением наискорейшего спуска не превысит $\pi/6$.

Можно аддитивно подбирать γ по скорости обучения. Каждый шаг аддитивного подбора строится так. Задается некоторое натуральное k и вещественное $\varepsilon > 0$ ($\gamma > \varepsilon$). Начиная с некоторого вектора параметров α^0 $2k$ раз последовательно проводится спуск в направлениях $s(\gamma \pm \varepsilon)$ причем k раз – в направлении $s(\gamma + \varepsilon)$ и k раз – в направлении $s(\gamma - \varepsilon)$. Эти k плюсов и k минусов размещены в последовательности из $2k$ шагов случайно. Далее находится суммарное снижение оценки в направлениях $s(\gamma + \varepsilon)$ – H_+ и в направлениях $s(\gamma - \varepsilon)$ – H_- . Если $H_+ > H_-$, то новое значение $\gamma = \gamma + \varepsilon$, если $H_- > H_+$, то $\gamma = \gamma - \varepsilon$. Далее цикл повторяется с новым ε так, чтобы $\varepsilon \rightarrow 0$, но сумма ряда из ε расходилась, например, $\varepsilon = 1/\sqrt{n}$, n – номер цикла адаптации.

10.10. Одношаговый квазиньютоновский метод и сопряженные градиенты

У методов, основанных на back-back процедуре, есть серьезные конкуренты. Это квазиньютоновские методы с ограниченной памятью и метод сопряженных градиентов.

Идея квазиньютоновских методов с ограниченной памятью состоит в том, что поправка к направлению наискорейшего спуска ищется как результат действия матрицы малого ранга. Сама матрица не хранится, а ее действие на векторы строится с помощью скалярных произведений на несколько специально подобранных векторов.

Простейший и весьма эффективный метод основан на BFGS формуле (Бройден–Флетчер–Гольдфарб–Шанно) и использует результаты предыдущего шага. Для его описания обозначим: s_k – направление спуска на k -м шаге; d_k – величина k -го шага (k -й шаг – сдвиг на $d_k s_k$); g_k – градиент функции оценки в начальной

точке k -го шага; $y_k = g_{k+1} - g_k$ – изменение градиента в результате k -го шага. Естественно, конечная точка k -го шага является начальной для $k+1$ -го.

BFGS – формула для направления спуска на $k+1$ -м шаге:

$$s_{k+1} = -g_{k+1} + \frac{1}{(y_k, s_k)} [(s_k, g_{k+1}) y_k + (y_k, g_{k+1}) s_k] - d_k \frac{(s_k, g_{k+1})}{(y_k, s_k)} s_k - \frac{(y_k, y_k) (s_k, g_{k+1})}{(y_k, s_k)^2} s_k, \quad (131)$$

где (\cdot, \cdot) , как всегда, обозначает обычное скалярное произведение – сумму произведений координат.

Если одномерную оптимизацию в поиске шага проводить достаточно точно, то новый градиент g_{k+1} будет практически ортогонален предыдущему направлению спуска: $(s_k, g_{k+1}) = 0$. Формула (131) при этом упростится – получим

$$s_{k+1} = -g_{k+1} + \frac{(y_k, g_{k+1})}{(y_k, s_k)} s_k. \quad (132)$$

Это – формула весьма популярного метода сопряженных градиентов. Он предполагает достаточно аккуратную одномерную оптимизацию при выборе шага.

В описанных методах предполагается, что начальное направление спуска $p_0 = -g_0$. Время от времени после некоторой последовательности из k шагов целесообразно вновь возвращаться к наискорейшему спуску – производить рестарт. Если рестарт производится не с направления наискорейшего спуска, то формулу (132) надо изменить:

$$s_{k+1} = -g_{k+1} + \frac{(y_k, g_{k+1})}{(y_k, s_k)} s_k + \frac{(y_0, g_{k+1})}{(y_0, p_0)} s_0. \quad (133)$$

Здесь s_0 – направление рестарта. В качестве s_0 можно выбирать, например, последнее направление спуска перед рестартом и сразу переходить к s_1 . Он используется в тех случаях, когда очередное s_{k+1} – плохое направление спуска и движение вдоль него приводит к слишком маленькому шагу, либо вообще не приводит к улучшению.

Итак, описано три способа для выбора поправок к направлению наискорейшего спуска: одношаговый

квазиньютоновский метод (131), метод сопряженных градиентов (132) и метод сопряженных градиентов с рестартом (133).

Существует версия метода сопряженных градиентов, допускающая элегантную рекурсивную формулировку. Строятся две последовательности $\alpha^0, \alpha^1, \dots$; β^0, β^1, \dots . Вектор параметров β^0 получается из α^0 шагом наискорейшего спуска с одномерной оптимизацией. Вектор α^{i+1} получается из β^i той же последовательностью операций, что и β^i из α^0 . Вектор β^i ($i > 1$) получается спуском из α^0 в направлении $\alpha^i - \alpha^0$ с одномерной оптимизацией. Приведем для примера процедуру получения β^3 в скобочной записи

$$\alpha^0 [[[SS][SS]][[S][SS]]].$$

Вместо S может использоваться любая процедура спуска, например, R , $[RR]$, $[RS]$ и т.п. После k шагов метода проводится рестарт либо с точки α^k , либо с точки β^k .

10.11. Метод сопряженных направлений, не использующий производных

Обратное и, тем более, обратно-обратное функционирование предъявляют повышенные требования к возможностям сети. Возникает естественное желание – обойтись более простыми вариантами сети даже за счет увеличения времени обучения, если, конечно, это увеличение не будет слишком большим.

Одна группа методов, не использующих производные, описана в разделе 10.8. Здесь будет представлен еще один метод, который обладает некоторыми достоинствами метода сопряженных градиентов, но не использует производные.

Метод состоит из нескольких этапов. Сначала выделяется некоторое число перспективных направлений поиска. Потом идет минимизация в линейном пространстве, порождаемом этими направлениями. При замедлении обучения генерируется новое направление и т.д.

Обратимся сразу к ядру метода – работе с выделенными направлениями. Пусть s_1, \dots, s_k – линейно независимые направления в пространстве параметров. Метод состоит в том,

что сначала последовательно проводится одномерная минимизация в выбранных направлениях: из α^0 в направлении s^1 получаем α^1 , из α^1 в направлении $s_2 - \alpha^2$, ..., из α^{k-1} в направлении s_k получаем α^k . Далее выделяется направление наиболее удачного спуска: ищется

$$\Delta_m = \max_{i=1, \dots, k} \{H(\alpha^{i-1}) - H(\alpha^i)\}.$$

Соответствующее направление обозначим s^* . Потом проводится анализ – сравнение направления s_m и $s^* = \alpha^k - \alpha^0$. Заметим, что s^* – итоговое направление спуска. Если s^* оказывается "лучше", чем s_m , то s_m исключается из списка s_1, \dots, s_k , а s^* вставляется в список на последнее место, т.е. $s_k := s^*$.

Теперь опишем, что значит "лучше" – приведем критерии сравнения s_m и s^* . Обозначим

$$\Delta_1 = H(\alpha^{i-1}) - H(\alpha^i),$$

$$\Delta^* = H(\alpha^0) - H(\alpha^k),$$

$$\alpha^{k+1} = \alpha^k + s^*, \quad \Delta_{k+1} = H(\alpha^k) - H(\alpha^{k+1}).$$

Чтобы сравнить s_m и s^* , надо проверить два неравенства:

$$1) \quad H(\alpha^{k+1}) \geq H(\alpha^0);$$

$$2) \quad (\Delta^* - \Delta_{k+1})(\Delta^* - \Delta_m) \geq 0.5 \Delta_m (\Delta^* + \Delta_{k+1})^2. \quad (134)$$

Если выполнено хотя бы одно из неравенств (134), то старое направление s_m считается лучшим, чем новое s^* , и все повторяется сначала, с теми же направлениями s_1, \dots, s_k . В качестве начальной точки можно взять α^k , а можно (другая модификация алгоритма) провести из α^k одномерную оптимизацию в направлении s^* и результат взять в качестве начальной точки

Если оба неравенства (134) неверны, то s^* признается лучшим, чем s_m . Из списка s_1, \dots, s_k исключается s_m , а на последнее место ставится s^* и с этим новым набором векторов продолжается поиск. Начальную точку для него ищут одномерной оптимизацией из α^k в направлении s^* .

Если уменьшение H за цикл алгоритма меньше заранее заданного предела, то производится рестарт – снова порождаются s_1, \dots, s_k и начинается цикл алгоритма.

Выбор исходных векторов s_1, \dots, s_k для старта и рестарта может проводиться различными способами. Удобно при этом использовать раскраску сети. Число направлений k будет равно числу цветов. При этом гарантируется линейная независимость исходного набора направлений.

Пусть параметры сети раскрашены – назначено k цветов. Проведем последовательно для всех цветов по очереди цикл спуска по параметрам данного цвета:

$$\alpha^{i-1} [[\dots [R^1] [R^1] \dots [R^1]] \dots]. \quad (135)$$

Здесь i – номер цвета, при данном значении i меняются только параметры i -го цвета, остальные фиксированы, α^i получается из α^{i-1} процедурой (135).

Положим $s_i = \alpha^i - \alpha^{i-1}$. У вектора s_i ненулевыми могут быть только те компоненты, которые окрашены в i -й цвет. Порождая s_i с помощью (135), мы в то же время проводим первый цикл алгоритма. Далее можно искать s_i , сравнить его с s^* и т.д. При необходимости рестарта вновь обращаемся к процедуре (135).

Уже после первого цикла среди s_i ($i=1, \dots, k$) может появиться вектор с ненулевыми компонентами всех цветов. Конечно, вряд ли стоит запоминать его отдельно – достаточно запомнить коэффициенты его разложения по начальным s_i .

Два слова о смысле неравенств (134). Первое очевидно: спуск по направлению s^* из α^k не дает достаточно хороших результатов. Смысла второго станет ясен, если использовать квадратичное приближение для H . С помощью второго неравенства оценивается изменение определителя матрицы нормированных направлений s_i после замены s_i на s^* . Для принятия замены этот определитель должен возрастать.

Процедуру порождения начальных направлений (135) можно изменить многими способами. Можно, например, использовать в качестве s_i проекции градиента оценки на пространства параметров данного цвета. Главное здесь – построение к достаточно хороших и заведомо независимых направлений спуска.

Для функций, близких к квадратичным, хорошо работает следующая модификация метода. Пусть параметрам сети присвоено

k цветов, индексом i будем обозначать номер цвета, индекс i у символа процедуры означает, что меняются параметры только данного цвета.

Пусть задан начальный вектор параметров α^0 . С помощью процедуры (135) при $i=1$ находим α^1 и $s_1 = \alpha^1 - \alpha^0$.

С помощью процедуры (135) при $i=2$ находим новый вектор параметров α , обозначим его $\alpha^{0,1}$.

Проводим одномерную оптимизацию из $\alpha^{0,1}$ в направлении s_1 . Полученный вектор параметров обозначим $\alpha^{1,1}$. Проводим одномерную оптимизацию из α^1 в направлении $s_2 = \alpha^{1,1} - \alpha^1$. Результат обозначим α^2 . И так далее.

Если определены $\alpha^1, \dots, \alpha^{i-1}$, то α^i и s_i определяются так. С помощью (135) для данного значения i спускаемся от α^{i-1} к некоторому вектору $\alpha^{0,i}$. Далее, от $\alpha^{0,i}$ с помощью одномерной оптимизации в направлении s_i получаем $\alpha^{i,1}$, из $\alpha^{i,1}$, двигаясь в направлении $s_2 = \alpha^{i-1}$ и т.д. до $\alpha^{i-1,1-1}$. Используем α^{i-1} как начальную точку для одномерной оптимизации в направлении $s_i = \alpha^{i-1,1-1} - \alpha^{i-1}$. Полученный вектор обозначим α^i . Итак, получены s_i и α^i .

В этом методе движение происходит до $i=k$, после чего следует рестарт.

10.12. Использование случайных возмущений для ускорения обучения

Обучение можно уподобить спуску в долину по гористой местности. Все описанные методы спуска на каждом шаге ведут вниз. Это – достоинство, но оно может превращаться в недостаток. Если идти вниз и только вниз, то, быть может, придется долго следовать извилистыми ущельями и застревать даже в маленьких котловинах. Таких задержек можно избежать. Один из приемов: время от времени подниматься по ближайшему склону и искать оттуда направление наилучшего спуска.

Реализовать метафору можно с помощью случайных возмущений. Между циклами спуска вставляется операция "Вимп" – шлепок. Параметрам сети даются независимые случайные приращения. Вимп вставляется между циклами алгоритмов спуска.

После случайного сдвига следует рестарт – спуск продолжается.

Вимр ставит перед воспитателем сети несколько вопросов.

1. Как выбрать вид распределения для случайных сдвигов? Обычно выбирается равнораспределение в некотором интервале.
2. Как выбрать интервалы случайного сдвига для различных параметров? В частности, можно ли выбрать их одинаковыми?
3. Как изменять характеристики случайных сдвигов в ходе обучения?

Равноправие всех параметров в структуре сети – ситуация весьма специальная. Пример – полносвязная чисто коннекционистская сеть с одинаковыми нейронами. Но и эта симметрия разрушается случайной генерацией начальной карты, разделением нейронов на входные и выходные, заданием функции оценки. Поэтому естественно выбирать интервалы случайного сдвига для различных параметров по-разному.

Тут нужно учитывать показатели чувствительности и качество решений. Если же показатель чувствительности для данного параметра высок при хорошо решаемых примерах, значит к сдвигу этого параметра надо относиться осторожнее – интервал выбирать поуже. Если же при высоких показателях чувствительности примеры решаются плохо, то особой осторожности не требуется и интервал может быть выбран пошире. Также не следует особо сдерживать случайные сдвиги и для параметров с малыми показателями чувствительности, если только дополнительно не решается задача контрастирования.

Эти соображения можно различными способами перевести в число. Соревнование и селекция таких способов – дело будущего. Один из них будет описан далее. А пока несколько замечаний об общей величине шума и ее изменении в ходе обучения.

Если основная задача – ускорить обучение, то уровень шума естественнее снижать и стремить к нулю по мере приближения оценки к минимуму. А для создания запаса устойчивости к появлению в будущем новых примеров шум в ходе обучения полезно увеличивать.

Что может служить задаваемой извне характеристикой шума?

Один из вариантов – скачок оценки H_{pg} ("подъем"). Этот скачок можно задать в абсолютных величинах, а можно и в относительных – волях от текущей оценки. Если на данном шаге подъем превысил заданное значение, то нужно уменьшить уровень шума и повторить изменение (с той же начальной карты). И тк пока сохраняется превышение. Если на данном шаге подъем меньше заданного предела, то можно принять это случайное изменение, но на следующем шаге повысить уровень шума.

10.13. Температура сети и энергия активации параметров

При списании случайных возмущений сети удобно воспользоваться физическими аналогиями. Для этого вводятся новые величины: температура сети и энергия активации различных параметров.

Температура характеризует общий уровень шума, а энергия активации показывает, насколько данный параметр способен удерживать свое значение, то есть, какая температура необходима для того, чтобы он начал существенно флуктуировать.

Энергия активации определяется из следующих соображений. Рассмотрим для начала решение одного примера. При одном и том же показателе чувствительности α , чем ниже оценка H , тем выше энергия активации E – зависимость $E(H)$ при фиксированном α монотонно убывающая. При сравнительно малых H зависимость $E(\alpha)$ монотонно возрастающая. Хорошие результаты лучше не портить. При сравнительно больших H зависимость $E(\alpha)$ монотонно убывающая – плохие решения лучше разрушать, возмущая наиболее чувствительные элементы. Значение оценки, для которого E не зависит от α , обозначим H_0 . Описанным условиям удовлетворяет, например, функция

$$E(k, H) = \frac{H_0}{H} \left(E_0 + \frac{H_0 - H}{H} \alpha \right), \quad (136)$$

а также, конечно, многих других.

Итак, пусть для одного примера функция $E(\alpha, H)$ определена, например, в форме (136). Переходим к обсуждению

страницы примеров. Во-первых, величина H_0 для отдельного примера может определяться через среднее значение H на странице.

В частности, H_0 может быть равна средней по примерам оценке, либо среднему арифметическому между средней и минимальной и т.п. Величина H_0 – граница, отделяющая на данной странице хорошо решаемые примеры от плохо решаемых. Наконец, H_0 может задаваться заранее, как некоторый уровень приемлемости оценки.

Пусть H_0 определено. Тогда страница примеров порождает "страницу" энергий активации $E(\mathcal{X}, H)$ – для каждого примера и каждого параметра строится энергия активации. Как собрать из них окончательные значения энергий? Есть, как минимум, три стратегии.

1. *Авантюристическая* – "чем хуже – тем лучше": энергия активации параметра определяется как минимум его энергий активации по примерам. Параметры максимально подвижны.
2. *Консервативная*: энергия активации параметра определяется как максимум его энергии активации. Максимум берется по всем примерам страницы. Параметры максимально заморожены.
3. *Конформистская*: энергия активации параметра есть среднее арифметическое значение его энергий активации, вычисленных по отдельным примерам.

Мы предпочтаем консервативную стратегию.

Случайный сдвиг для параметра α_i определяется как

$$g_i A_i \exp\left(-\frac{E_i}{T}\right), \quad (137)$$

где g_i – случайная величина, равномерно распределенная в отрезке $[-1, 1]$, A_i – полуширина интервала приемлемых изменений α_i , E_i – энергия активации α_i , T – температура сети.

Температура подбирается из ограничений на подъем оценки. Эти ограничения со временем стоит менять и лучше вообще задавать их в относительных единицах $\Delta H/H$.

10.14. Использование случайных возмущений для формирования устойчивых навыков и для создания условий дальнейшего обучения

Этот раздел с длинным названием – один из самых коротких в книге.

Чтобы шум ускорял начальное обучение и не мешал на завершающих стадиях, его уровень надо снижать по мере уменьшения оценки.

Но одна из основных проблем в обучении нейронных сетей – создание навыков, которые не теряются при дальнейшем обучении. Как сделать так, чтобы старые навыки не вытеснялись новыми? Или, по крайней мере, вытеснялись не слишком быстро.

Средство, которое позволяет это сделать, если такая устойчивость вообще возможна, – использование шума, нарастающего в ходе обучения. Этот шум моделирует помехи от предстоящего обучения новым примерам, поэтому он должен в меньшей степени, чем ускоряющий шум, щадить параметры с высокими показателями чувствительности. Ведь дальнейшее обучение будет менять эти параметры наравне с остальными.

Предлагается устанавливать одинаковый интервал сдвига для параметров, симметрично участвующих в структуре сети. Так, для полносвязной сети с линейными синапсами и однопараметрическими нейронами есть три группы параметров: веса синапсов связи, веса синапсов памяти и параметры нейронов. Для слоистой сети параметры различных слоев структурно различаются и соответственно увеличивается количество групп.

Внутри группы структурно эквивалентных параметров выбирается один интервал случайного сдвига. Для каждой группы устанавливается также свой допустимый уровень подъема при сдвиге. Выпр проводится по очереди в разных группах с контролем подъема и коррекцией уровня по группам. Величина допустимых сдвигов со временем увеличивается до заданной границы. При этом либо достигается требуемый уровень устойчивости, либо, увы, не достигается приемлемо низкий уровень оценки.

10.15. Несколько слов о философии оптимизации

Уже ясно, что для обучения нейронной сети можно использовать любой алгоритм многомерной оптимизации, если он не требует слишком большой памяти. Для работы по этим алгоритмам сеть может предоставить учителю несколько основных процедур. Среди них вычисление значения целевой функции и ее градиента.

При организации алгоритма желательно использовать естественную структуру сети: параметры, градиенты и другие характеристики разбиты на много локальных массивов, действия в которых можно проводить параллельно. Кроме того, существует возможность раскрашивания сети.

В такой ситуации естественно использовать различные ускоряющие преобразования: одни – внутри локальных массивов, другие – между локальными массивами, трети – между цветами. Можно строить различные полупрямые произведения. Например, можно представить матрицу, модифицирующую направление спуска, с помощью набора матриц, преобразующих вектор спуска внутри каждого локального массива и матрицы, которая производит преобразование, смещающее векторы спуска из разных локальных массивов. Поясним это. Обозначим параметры a_{ij} , где i – номер локального массива, j – номер параметра в локальном массиве. Аналогично, $g_{ij} = \partial H_{pg} / \partial a_{ij}$. Пусть все локальные массивы имеют одинаковую длину n , а их число равно m . Направление спуска можно искать, например, в виде

$$s_{ij} = \sum_k b_{ik} a_{kj} g_{ki}. \quad (138)$$

Здесь a_{kj} – элементы $n \times n$ матрицы A_k , осуществляющей преобразование внутри k -го локального массива, а b_{ik} – элементы $m \times m$ – матрицы, смещающей массивы. Вместо $(mn)^2$ элементов преобразующая матрица (138) определяется с помощью $mn^2 + m^2$ различных чисел, что может быть значительно меньше, например, если $m=n=10^k$, то $(mn)^2 = 10^{4k}$, а $mn^2 + m^2 = 10^{3k} + 10^{2k} \approx 10^{3k}$ – в 10^k раз меньше. Кроме того, операции внутри локальных массивов могут проводиться параллельно.

Аналогичные приемы можно использовать и для назначенных цветов, а не только для исходно существующих локальных массивов.

Можно развивать теорию многомерной параллельной оптимизации, опираясь на представление (138) и идеологию квазиньютоновских методов и т.п. Это нужно сделать и это будет сделано, однако в предлагаемом кратком очерке, посвященном нейронным сетям, невозможно обсудить все варианты.

Остается, однако, неудовлетворенность подходом в целом. дело в том, что, грубо говоря, практически вся теория безусловной оптимизации посвящена минимизации квадратичных форм: $H(x) = (x, Qx) + (b, x) + c$. Конечно, постоянно делаются попытки уйти как можно дальше от функций, задаваемых квадратичными формами, но все же и метод Ньютона, и метод сопряженных градиентов и другие усовершенствованные методы спуска создавались применительно к квадратичным формам и тем функциям, которые с их помощью хорошо аппроксимируются.

Идеализированный модельный объект – путеводная звезда теории и ориентация на него неизбежна, какие бы слова при этом ни произносились. Удавшаяся теория выпуклой оптимизации потому и удалась, что выпуклая функция обладает многими важными свойствами положительно определенной квадратичной формы.

Адаптивный ландшафт нейронной сети – график зависимости H_{pg} от параметров. Труд ли он похож на чашу – график положительной квадратичной формы. Скорее всего, он напоминает ландшафт горного района без особо острых пиков и пропастей – нечто вроде Карпатских гор.

То же можно сказать и про многие другие функции. Вывод – нужен новый модельный объект, который бы дал другую идеологическую установку для теории оптимизации.

За общностью математических теорий обычно спрятаны лежащие в основе и подразумеваемые семейства модельных объектов.

В теории безусловной оптимизации неоднократно вводились дополнительные претенденты на роль модельных объектов. Именно поэтому большую популярность приобрели стандартные тестовые

неквадратичные задачи – такие как минимизация функции Розенброка

$$F(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Однако для многомерной оптимизации вряд ли возможно строить дополнительный набор тестовых задач, собранный "поштучно", как это было для малых размерностей. Как поступить в этом случае?

Сказать трудно, но можно предположить, что модельным должен быть какой-нибудь достаточно простой класс случайных функций. Для него нужно будет строить алгоритмы и оценивать скорость сходимости. Эти алгоритмы будут хорошо работать с теми функциями, которые близки к типичным реализациям случайных функций данного класса. Можно привлекать также аналогии из статистической физики. Теории такого типа пока нет, но уже предварительные соображения позволяют предложить новые алгоритмы. Перейдем к описанию одного из них.

10.16. Метод виртуальных частиц

Основная идея метода – использовать случайные сдвиги аргумента и суммирование полученных значений функции для усреднения. Ожидается, что в результате сотрутся (станут относительно меньше) случайные или, лучше сказать, псевдослучайные детали аддитивного ландшафта и откроется более прямой путь к искомой котловине минимума.

Метод виртуальных частиц может надстраиваться практически над любым методом оптимизации. Он состоит в том, что к точке, спускающейся по аддитивному ландшафту (частице), добавляется несколько других, траектории которых получаются из траектории частицы сдвигом на случайный вектор. Эти "виртуальные" частицы время от времени уничтожаются и рождаются новые. Спуск строится так, чтобы уменьшалась взвешенная сумма высот.

Вот одна реализация алгоритма. Параметры сети разбиваются на группы структурно эквивалентных. Для каждой группы задается свой интервал случайных сдвигов. Определяется число виртуальных частиц n и генерируется $n-1$ случайный

вектор g_1, \dots, g_{n-1} . Их координаты независимо и равномерно распределены в данных интервалах. Начальное положение частицы α^0 , начальное положение виртуальных частиц $\alpha^0 + g_i$. Виртуальная частица с номером n строится так:

$$g_n = -\frac{1}{\sqrt{n}} \sum_{i=1}^{n-1} g_i.$$

До следующего порождения виртуальных частиц их положения задаются так

$$\alpha + g_i \quad (i=1, \dots, n),$$

где α – вектор параметров, характеризующий положение частицы.

Всем частицам, кроме n -й, присваивается одинаковый вес W ($0 < W < 1$), n -я получает вес $W_n = W/\sqrt{n}$. Далее минимизируется функция

$$H_w(\alpha) = H_{pg}(\alpha) + W \sum_{i=1}^{n-1} H_{pg}(\alpha + g_i) + W_n H_{pg}(\alpha + g_n). \quad (140)$$

Градиент каждого из слагаемых (140) может быть вычислен самой сетью с помощью двойственного функционирования. То же относится и к величинам, получаемым с помощью back-back процедуры. Алгоритм может быть выбран любой – от наискорейшего спуска и партан-методов до метода сопряженных градиентов.

Выбор g_n в виде (139) и $W_n = W/\sqrt{n}$ определяется двумя обстоятельствами: для каждой координаты g_n дисперсия будет совпадать с дисперсией соответствующих координат g_i ($i=1, \dots, n-1$) и, кроме того, для квадратичных $H_{pg}(\alpha)$ точки минимума $H_{pg}(\alpha)$ и $H_w(\alpha)$ совпадут. Действительно, если

$$H_{pg}(\alpha) = (a, \alpha) + (b, \alpha) + c', \text{ то } H_w(\alpha) = (a, \alpha) + (b, \alpha) + c',$$

где c' – новая константа, зависящая от g_n .

В методе виртуальных частиц возникает важный вопрос: когда уничтожать имеющиеся виртуальные частицы и порождать новые? Есть три существенно различающихся варианта.

1. Функция (140) минимизируется до тех пор, пока скорость обучения не упадет ниже критической. После этого вновь производят случайные сдвиги частицы и обучение продолжают.
2. Порождение новых частиц производят после каждого цикла базового алгоритма оптимизации – при рестартах. Например,

после каждого шага метода наискорейшего спуска, после партан-шага итерационного партан-метода и т.п.

3. При каждом вычислении оценок и градиентов.

Третий способ вносит случайный процесс внутрь базового алгоритма, в результате возможны и колебания при одномерной оптимизации, и некоторые другие неприятности. Его преимущество - экономия памяти, так как не надо хранить g_i , а только текущее значение и накапливаемую для вычисления g_n сумму.

Первый способ наиболее консервативен. Он долго сохраняет все достоинства и недостатки предшествующего спуска, хотя направление движения может существенно измениться при порождении новых виртуальных частиц.

Наиболее перспективным пока кажется второй способ: он, с одной стороны, не разрушает базового алгоритма, а с другой - за счет многократного порождения виртуальных частиц хорошо "прощупывает" аддитивный ландшафт.

Что же касается экономии памяти, то она может быть достигнута за счет двухэтапного порождения g_i : генерируется небольшое число случайных сдвигов в каждой группе структурно эквивалентных параметров, а потом, используя разные сочетания этих сдвигов между группами, порождается намного больше векторов g_i для всего набора параметров. Так, если групп k , то, порождая в каждой по два сдвига, получаем в результате комбинирования 2^k различных g_i .

Два вопроса - об амплитуде сдвига, количестве виртуальных частиц а также - о динамике этих величин в ходе обучения - требуют специального исследования для каждого класса задач. Тем не менее, уже ясно, что если виртуальные частицы используются для спрямления пути к минимуму, то амплитуду g со временем следует уменьшать.

Но метод виртуальных частиц - еще одно средство для решения важнейшей проблемы нейрокомпьютинга - устойчивости приобретенных навыков к обучению новых задачам. Чтобы

добиться этого, и амплитуду, и количество частиц со временем желательно увеличивать.

Метод виртуальных частиц легко и без всяких дополнительных затрат сочетается со случным поиском: достаточно перед порождением новых виртуальных частиц выбрать в качестве основной частицы ту, для которой H_{pq} меньше. Ею может оказаться старая основная частица либо одна из старых виртуальных.

Затраты времени на виртуальные частицы можно оценить так: для квадратичных форм, когда они не дают никаких дополнительных преимуществ (и вообще не меняют ответа), время увеличивается в $n+1$ раз, где n - число виртуальных частиц. Стоит ли выигрыш в трудных случаях таких затрат в легких случаях - вопрос особый, тем более, что часто заранее нельзя решить, будет ли случай легким (почти что квадратичная форма) или трудным.

Впрочем, можно избежать и таких затрат времени, если в большом нейрокомпьютере разместить сразу $n+1$ нейронную сеть - одну для частицы и n для виртуальных частиц. Тогда увеличение времени будет связано только с обменом данными между сетями. Остальные операции - прямое и обратное функционирование и т.д. все сети могут выполнять параллельно.

10.17. Монотонные сети, двужзначная оценка и правило Хебба

До сих пор мы свободно использовали гладкую и точно заданную функцию оценки. Даже весьма малые изменения в ответах оценивались как улучшение или ухудшение. Поэтому можно было вычислять градиенты, строить алгоритмы спуска, ускорять их и т.д.

Возможность оценить сколь угодно малый сдвиг - благо¹,

¹ Впрочем, представьте, что Ваше малейшее движение получает оценку. . . Я бы в таком мире не выжил, хотя поначалу, быть может, чему-нибудь быстро научился.

которого живые существа лишены. Наиболее примитивные базовые составляющие большинства теорий обучения живых существ можно обозначить так:

- 1) поисковая активность,
- 2) поощрение успеха,
- 3) наказание неуспеха.

Тут могут быть различные акценты, различные и противоречащие друг другу дальнейшие разработки. Для нас это пока не важно. Значимо лишь выделение составляющих 1-3 и то, что градации оценки немного: обычно две (успех-неуспех) или три (успех-нейтральный исход-неудача). В такой ситуации уже нет гладкого адаптивного ландшафта и потому бессильны методы спуска.

Основная идея, заменяющая спуск, состоит в формализации базовых компонент (активность-поощрение-наказание) и построении на этой основе алгоритмов обучения. Формализация, как всегда, не однозначна. Здесь будет разработана одна группа методов, основанная на обобщенном правиле Хебба.

Это правило дает формализацию наказаний и поощрений. Кратко идея такова: наказание состоит в усилении тормозных и ослаблении возбуждающих связей, поощрение же, напротив, - в усилении возбуждающих и ослаблении тормозных.

Первое уточнение - не стоит модифицировать все связи, достаточно только те, которые вносили существенный вклад в данный успех или неудачу. Здесь мы возвращаемся к идеи показателей чувствительности. В чисто коннекционистских сетях обычно за показатель чувствительности для синаптического веса принимается модуль переданного по нему сигнала.

Почему ожидается, что правило Хебба будет работать? Предполагаем, что наказание разрушит ошибочные действия, а поощрение усилит успешные.

Наказание именно разрушает, а не исправляет - нет направления спуска, неясно, куда "лучше". Тенденция неправильно действовать ослабляется. За счет поисковой активности появляются удачи, которые фиксируются поощрением. После некоторого времени обучения поисковая активность уже может быть не совсем случайной - появляется ряд запретов, с

одной стороны, и преимущественных направлений - с другой.

С неожиданной сложностью мы сталкиваемся при формализации понятий "тормозные и возбуждающие связи". Для одного нейрона это еще можно определить именно через изменения: усиление торможения ведет к уменьшению выходного сигнала, усиление возбуждения - к его увеличению.

Но увеличение выходного сигнала для одного нейрона легко может привести к уменьшению для другого: первый возбудился, и на следующем временном такте затормозил другого. Поэтому трудно разделить параметры и сигналы на возбуждающие и тормозящие. Уже при нескольких тактах функционирования может возникнуть путаница: первый возбуждает одного, тормозит другого, а потом в результате возбуждается (или тормозится?) четвертый.

Невозможность четкой формализации в общем случае приводит к мысли: надо вводить сети специальной структуры, для которых естественно возникает разделение воздействий на возбуждающие и тормозящие.

Дальнейшее изложение строится так. Сначала вводится понятие монотонного функционального элемента и обсуждаются алгоритмы обучения таких элементов. далее определяются монотонные связи между монотонными элементами - и возбуждающие, и тормозящие и строятся монотонные сети слоистой структуры. Наконец, конструируются алгоритмы обучения монотонных сетей.

Чтобы определить возбуждение и торможение в тех случаях, когда параметров и выходных сигналов несколько, надо использовать понятие порядка в векторных пространствах. Нужно ответить на вопрос: что может означать для двух векторов x , у отношения $x \geq y$?

Порядок в конечномерном векторном пространстве E будем задавать с помощью замкнутого выпуклого телесного (т.е. с непустой внутренностью) конуса Q . По определению $x \geq 0$, если $x \in Q$. Стандартный пример для пространств с выделенной системой координат: Q состоит из векторов, у которых все координаты неотрицательны. Другой, менее тривиальный пример:

$$Q = \{x \in E \mid x_1 \geq x_2 \geq \dots \geq x_n \geq 0\}$$

Q состоит из векторов, координаты которых составляют невозрастающую последовательность неотрицательных чисел. Конечно, множество возможных способов задания порядка в данном E бесконечно. Элементы Q называются положительными векторами. Предполагается, что $Q \cap (-Q) = \{0\}$, то есть $x \geq 0 \& x \leq 0 \Rightarrow x = 0$.

Для определения возбуждающих и тормозящих связей нам в дальнейшем понадобится понятие монотонного оператора. Пусть заданы два пространства E_1, E_2 с конусами положительных векторов Q_1, Q_2 . Линейное отображение $A : E_1 \rightarrow E_2$ называется положительным, если $A(Q_1) \subset Q_2$ – для любого $x \in E_1$ если $x \geq 0$, то и $Ax \geq 0$ в E_2 . Положительные операторы, отображающие E_1 в E_2 , образуют замкнутый телесный конус в пространстве линейных отображений $L(E_1, E_2)$.

Для двух векторов $x, y \in E$ полагают $x \geq y$, если $x - y \geq 0$. Аналогично, для двух линейных операторов $A, B \in L(E_1, E_2)$ полагают $A \geq B$, если $A - B$ – положительный оператор.

Отображение F , определенное в области $U \subset E_1$ и принимающее значение в E_2 , называется монотонно возрастающим, если для любых $x, y \in U$

$$x \geq y \Rightarrow F(x) \geq F(y), \quad (141)$$

и монотонно убывающим, если

$$x \geq y \Rightarrow F(y) \geq F(x). \quad (142)$$

Если отображение F дифференцируемо в U , то его дифференциал $DF(x)$ для каждого $x \in U$ является линейным оператором из $L(E_1, E_2)$. Если F – монотонно возрастающее отображение, то для любого $x \in U$ оператор $DF(x)$ положителен. Если же F – монотонно убывающее, то для каждого $x \in U$ положителен оператор $-DF(x)$. Для выпуклых областей U верно и обратное: если для любого $x \in U$ оператор $DF(x)$ положителен, то F – монотонно возрастающее отображение, а если для любого $x \in U$ положителен оператор $-DF(x)$, то F – монотонно убывающее.

Определим теперь монотонные функциональные элементы для того, чтобы строить потом из них монотонные сети. Функциональный элемент f по вектору входов A и вектору параметров α вычисляет вектор выходов $f(A, \alpha)$. Отличие от предыдущего изложения пока состоит в том, что выход –

векторный. Здесь мы не будем разделять элемент с векторным выходом на отдельные компоненты, каждый из которых вычисляет одну координату вектора выхода.

Элемент f называется монотонным по входам, если в пространстве входов и в пространстве выходов $E_{f \text{ in}}$ и $E_{f \text{ out}}$ определены векторные порядки с конусами положительных элементов $Q_{f \text{ in}}$ и $Q_{f \text{ out}}$ и для любого вектора параметров α $f(\cdot, \alpha)$ – монотонно возрастающее отображение из $E_{f \text{ in}}$ в $E_{f \text{ out}}$.

Аналогично, f называется монотонным по параметрам, если в пространстве параметров $E_{f \text{ p}}$ и пространстве выходов $E_{f \text{ out}}$ определены векторные порядки с конусами положительных элементов $Q_{f \text{ p}}$ и $Q_{f \text{ out}}$ и для любого вектора входов A $f(\cdot, A)$ – монотонно возрастающее отображение из $E_{f \text{ p}}$ в $E_{f \text{ out}}$.

Наконец, f называется монотонным, если он монотонен и по входам, и по параметрам.

Приведем несколько примеров. Нейрон с характеристикой вход–выход

$$f(A, \alpha) = \frac{A}{\alpha + |A|} \quad (\alpha > 0)$$

является монотонным по входу, но не по параметру, если на действительной прямой принят обычный порядок. Пороговый нейрон

$$f(A, \alpha) = \begin{cases} 0, & \text{если } A < \alpha; \\ 1, & \text{если } A \geq \alpha; \end{cases} \quad (143)$$

является монотонным по входу и обладает тем свойством, что с ростом α выходной сигнал f для всех A не возрастает. После замены α на $1/\alpha$ пороговый нейрон

$$f(A, \alpha) = \begin{cases} 0, & \text{если } A < 1/\alpha; \\ 1, & \text{если } A \geq 1/\alpha; \end{cases} \quad (143)$$

$(\alpha > 0)$ становится уже монотонным. Линейный синапс, получающий на входе сигнал A и выдающий на выходе αA , является монотонным при ограничениях $\alpha > 0, A > 0$.

Линейный сумматор, выдающий на выходе сумму входных сигналов, является монотонным элементом. Он монотонен по входам A_i , если задать $Q_{i \text{ in}}$ так:

$$Q_{i \text{ in}} = \{A | A_i \geq 0 \text{ для всех } i\}, \quad (144)$$

а параметров у линейного сумматора нет.

Наконец, примером монотонного элемента является нейронная сеть из пороговых нейронов (143) с линейными синапсами, имеющими неотрицательные веса, с линейными сумматорами перед входами нейронов и подачей внешних входных сигналов на нейроны.

В пространствах E_{in} , E_{out} , E_p задаются стандартные конусы покоординатно неотрицательных векторов (144). Параметры сети – обратные пороги нейронов α (143) и веса синапсов. Все они должны быть неотрицательными (обратные пороги – положительными). Этот пример интересен еще тем, что вычисляется функция F – монотонная, недифференцируемая и поэтому градиентные методы использовать нельзя.

Именно эти монотонные сети из пороговых нейронов будут служить нам стандартным примером монотонного функционального элемента. Обучение такого элемента рассмотрим в следующем порядке. Сначала – режим с предъявлением по одному примеру: предъявление примера – оценка функционирования – коррекция параметров (с повторными пробными циклами или без них) – поисковое изменение параметров – предъявление нового примера. Потом – работа со страницей примеров.

Итак, пусть монотонная сеть пороговых нейронов получила входные сигналы, проработала заданное число тактов и выдала выходные сигналы. Этот цикл функционирования получает внешнюю оценку. Оценка двузначна: "хорошо" или "плохо". Если "хорошо", то необходимо поощрение, если "плохо", то наказание.

И поощрение, и наказание заключаются в специальных изменениях параметров. Предварительно, однако, надо решить, какие параметры менять при поощрениях и наказаниях. Полученный в результате функционирования выходной сигнал в разной степени зависит от различных параметров. Желательно, чтобы изменения были точно направлены на то, что нужно, и не затрагивали лишних параметров.

Здесь мы сталкиваемся с одним явственным преимуществом нейросетевой структуры перед многими другими. Это преимущество не связано с высоким уровнем параллелизма. Оно не зависит от способа реализации процесса. Состоит это

преимущество в том, что для нейросетевой структуры по циклу функционирования нетрудно оценить значимость каждого параметра в получении данного ответа.

Частично мы уже касались этого в разделах, посвященных анализу чувствительности: величина прошедшего через синапс сигнала может служить показателем чувствительности для синаптического веса. Однако там можно было пользоваться частными производными оценки – более информативными показателями. Здесь же, когда нет ни гладкой оценки, ни дифференцируемости функции вход-выход, преимущества нейронных сетей проявляется особенно четко.

Пусть некий нейрон в результате цикла функционирования выдал наружу единичный выходной сигнал. Как определить те параметры, которые за это нужно поощрить (в случае успеха) или наказать (в случае неуспеха). Для этого используется своеобразное "обратное функционирование". В ходе процесса отмечаются те параметры, которые подлежат поощрению (повышению) или наказанию (понижению).

Обратное функционирование надстраивается над прямым и использует результаты каждого его такта. Начинается оно с того нейрона, который выдал выходной сигнал. Первым в список отмеченных параметров включается обратный порог этого нейрона. Далее отмечаются веса всех синапсов, по которым на этот нейрон пришли ненулевые сигналы с предыдущего такта. Далее – обратные пороги тех нейронов, от которых на предыдущем такте поступали сигналы на отмеченные синапсы. И так далее.

Опишем общее правило для составления списка значимых параметров. Пусть прямое функционирование занимает θ тактов: $T=1, 2, \dots, \theta$, T – логарифмическое время – номер такта. Параметры отмечаются в обратном порядке – от $T=\theta$ до $T=1$. Список отмеченных параметров разбивается на θ списков, соответствующих различным значениям T .

Для каждого T сначала отмечаются параметры нейронов, потом синапсов, по которым эти нейроны в такт прямого функционирования T получили ненулевые сигналы с такта $T-1$.

Если список составлен для $T=\theta, \theta-1, \dots, t+1$, то для $T=t$ в

него включаются обратные пороги тех нейронов, от которых выходные сигналы с такта прямого функционирования $T=t$ поступили на синапсы, отмеченные на предыдущем такте обратного функционирования $T=t+1$.

В начале обратного функционирования при $T=0$ отмечается один или несколько обратных порогов выходных нейронов – тех нейронов, которые дали на выходе ненулевые сигналы.

Список, построенный таким образом, может оказаться слишком большим. Поэтому дадим еще одно правило обратного функционирования – обратное функционирование с сортировкой.

Заметим сначала, что наиболее значительное увеличение числа отмеченных параметров происходит, когда отмечаются синапсы – на один нейрон может приходить много сигналов и отмечали мы все синаптические веса для синапсов, приносящих ненулевые сигналы. А можно отмечать не все, но только те, которые приносят самые большие сигналы. Так как на синапс пороговые нейроны могут передавать либо 0, либо 1, то ненулевой переданный через синапс сигнал просто равен синаптическому весу.

Итак, на каждом шаге обратного функционирования T выделяется некоторое множество нейронов – так же, как и раньше, обозначим их число n_T . Потом на том же шаге T выделяются те синапсы, которые при прямом функционировании приносили на эти нейроны ненулевые сигналы с такта $T-1$. Веса этих синапсов расположим в n_T массивов – по числу нейронов. В один массив включаем веса синапсов, передающих сигналы одному нейрону.

Теперь из этих массивов нужно выбросить лишние элементы. Для этого есть, по крайней мере, два способа.

1. По предельному количеству значимых синапсов, ведущих к одному нейрону. Задается некоторое число k . В каждом из n_T массивов выбирается k наибольших синаптических весов. Эти параметры включаются в список, остальные отбрасываются. Если в каком-либо из указанных n_T массивов параметров меньше, чем k , то отмечаются все параметры массива.
2. По достаточности переданного сигнала для возбуждения нейрона. В каждом из n_T указанных массивов выбирается

наибольший синаптический вес, к нему прибавляется второй по величине и так до тех пор, пока сумма не станет больше, чем порог возбуждения соответствующего нейрона. Параметры массива, вошедшие в сумму, оставляются в списке значимых величин, остальные – отбрасываются.

Таким образом, описано три способа выделения значимых параметров. Теперь можно определить поощрение и наказание.

Поощрение состоит в увеличении значимых параметров, наказание – в их уменьшении. Остается "всего лишь" выбрать величины этих увеличений и уменьшений. Они могут зависеть от:

- 1) величины параметра;
- 2) момента обратного функционирования T , на котором параметр был включен в список значимых;
- 3) места соответствующего элемента в структуре сети (в том числе, конечно, от того, является параметр синаптическим весом или обратным порогом нейрона).

Сравнение с методом спуска показывает, что поощрение и наказание сильно различаются. Если поощрение служит фиксации полезного навыка и во многом аналогично спуску, то наказание уменьшает параметры, определяющие неправильный ответ, но, строго говоря, не ведет к улучшению. Оно разрушает неправильные навыки с тем, чтобы их места заняли случайно найденные и поощренные правильные.

Для поощрения существует аналог одномерной оптимизации с повторными решениями примера. Опишем его. Пусть выбраны начальные сдвиги значимых параметров. Обозначим вектор этих сдвигов $s (s>0)$. Прибавим s_k к значимым параметрам и снова дадим сети решить этот пример. Если оценка снова "хорошо", увеличим сдвиг в g раз, $g>1$, и снова протестируем – и так до тех пор, пока не будет получена оценка "плохо" или все значимые параметры не выйдут на заранее определенные максимальные предельные значения.

Пусть, наконец, получена оценка "плохо". Переходим к дроблению шага – делением пополам или методом золотого сечения находим точку смены "хорошо" на "плохо" с наперед заданной точностью ϵ : сдвиг на ds – "хорошо", сдвиг на $(d+\epsilon)s$ – "плохо".

Но это еще не все. После сдвига на $(d+\varepsilon)s$ при прямом функционировании возбуждаются некоторые "лишние" нейроны – те, которые не возбуждаются после сдвига на ds . Определим для возбуждения этих лишних нейронов значимые параметры. Среди них окажутся и некоторые из тех параметров, что были значимы ранее для хорошей оценки – иначе при сдвиге на $(d+\varepsilon)s$ оценка бы не портилась. Удалим из s соответствующие компоненты (положим их равными нулю). Получим вектор сдвига s' . Продолжим оптимизацию из точки, полученной сдвигом параметров на ds . Теперь вектор сдвига s' , начальный шаг ε .

И так продолжаем, пока в векторе сдвига остаются положительные координаты или пока все параметры, которые надо увеличивать, не выйдут на максимальные предельные значения.

При работе с одним примером для наказания нет аналога одномерной оптимизации. Значимые параметры уменьшаются на некоторую величину. Уменьшение может производиться за один шаг, например, умножением всех значимых параметров на q ($0 < q < 1$). Оно может проводиться в несколько шагов с повторным решением примера – вдруг по дороге будет получена оценка "хорошо". Наконец, оно может быть радикальным – все значимые параметры обращаются в ноль.

После наказания следует шаг поисковой активности – случайный сдвиг параметров. При определении величины сдвига нужна память о предыдущих поощрениях и наказаниях – сдвиг недавно поощренных или наказанных параметров должен быть меньше, чем остальных. Это организуется, например, так. Для каждого параметра вводится показатель r ($0 < r \leq 1$), на который умножается случайный сдвиг стандартной амплитуды. После поощрения данного параметра его показатель умножается на r_+ ($0 < r_+ < 1$), после наказания – на r_- ($0 < r_- < 1$). После очередного случайного сдвига r заменяется на ur . Начальные значения всех r равны 1.

Можно также воспользоваться идеей температуры сети и энергии активации параметров (см. раздел 10.13), повышая энергию активации после каждого поощрения или наказания и понижая ее после очередной "встряски".

Работа сразу со страницей примеров позволяет ввести

аналог одномерной оптимизации для наказаний и существенно ускоряет обучение.

Итак, пусть задана страница примеров. Оценка страницы H_{pg} – число хорошо решаемых примеров данной страницы. Величина H_{pg} – функция вектора параметров сети α : $H_{pg}(\alpha)$. Алгоритм обучения строится так, чтобы величина $H_{pg}(\alpha)$ со временем не уменьшалась. Пусть в результате k -го цикла обучения принимается вектор параметров α^k . Тогда последовательность $H_{pg}(\alpha^0), H_{pg}(\alpha^1), \dots, H_{pg}(\alpha^k)$ – неубывающая.

Более того, алгоритм должен работать так, что если на каком-либо пробном шаге величина H_{pg} возросла, то далее на принимаемых значениях α она не должна уменьшаться – еще раз встречаемся с "принципом максимального термометра".

Если в начальный момент все примеры решаются плохо, то для каждого из них составляется список значимых параметров, эти списки объединяются для всех примеров страницы, все параметры объединенного списка уменьшаются, после чего производится случайный сдвиг всех параметров сети. Как уже говорилось, интервал сдвига для только что наказанных параметров выбирается меньшим, чем для остальных. Результаты сдвига оцениваются – вычисляется $H_{pg}(\alpha)$. Если снова все примеры решаются плохо, то есть $H_{pg}(\alpha)=0$, то цикл повторяется. И так до тех пор, пока не появятся примеры, решаемые хорошо, после чего переходим к следующей процедуре.

Пусть на данной странице есть и хорошо, и плохо решаемые примеры. Цикл обучения начинается с наказания. Для всех примеров страницы выделяются значимые параметры. Для плохо решаемых они объединяются в общий список наказываемых параметров. Параметры, входящие в этот список, уменьшаются до тех пор, пока дальнейшее уменьшение любого из них на ε не приведет к уменьшению H_{pg} или не выведет за заранее установленные ограничения ($\varepsilon > 0$ – заданное малое число). Если в ходе уменьшения сеть стала решать какой-либо пример хорошо, то он тут же исключается из списка плохо решаемых и дальнейшее уменьшение параметров не должно разрушить и его правильного решения наряду с правильными решениями других

примеров.

Алгоритм наказания строится так. По списку наказываемых параметров и первоначальному вектору их значений строится вектор начального сдвига $-s$. Отрицательные координаты $-s$ соответствуют наказываемым параметрам. Делается пробный шаг - начальный вектор параметров α^0 заменяется на $\alpha^0 - s$. Этот шаг оценивается - сеть решает все примеры страницы при новых значениях параметров. Возможно несколько вариантов:

- а) в результате сдвига на $-s$ множество хорошо решаемых примеров не изменилось, тогда шаг увеличивается в g раз ($g > 1$) и переходим к оценке для вектора параметров $\alpha^0 - gs$;
- б) множество хорошо решаемых примеров увеличилось, тогда запоминаем, какие примеры теперь решаются хорошо, и тоже увеличиваем шаг в направлении $-s$;
- в) некоторые примеры, за которые сеть получала раньше оценку "хорошо", стали решаться плохо, пусть это произошло в первый раз при шаге rd , то есть при значениях параметров $\alpha^0 - rds$, а при шаге d все еще хорошо.

В последнем случае методом деления пополам или золотого сечения находим на отрезке $[d, rd]$ такую величину шага δ , что сеть с параметрами $\alpha^0 - \delta s$ хорошо решает все примеры, за которые она раньше получала оценку "хорошо", а сеть с параметрами $\alpha^0 - (\delta + \varepsilon)s$ некоторые из них уже решает плохо.

Таким образом, выделяется группа примеров (один или несколько), для которых при переходе от параметров $\alpha^0 - \delta s$ к $\alpha^0 - (\delta + \varepsilon)s$ оценка меняется с "хорошо" на "плохо". Для таких примеров выделяются значимые параметры и вектор s изменяется на s' : s' отличается от s тем, что координаты s' , соответствующие этим значимым параметрам, полагаются нулевыми. Далее из точки $\alpha^0 - \delta s$ начинается движение в направлении s' до тех пор, пока это возможно без ухудшений, потом s' заменяется на новый вектор - еще часть координат обращается в ноль - и так до тех пор, пока движение возможно.

После наказания следует поощрение. Применяется уже описанный ранее аналог одномерной оптимизации. Постепенно увеличивая значимые параметры для хорошо решаемых примеров доходит до того момента, когда дальнейшее увеличение этих

параметров более, чем на ε , приводит к ухудшениям. Если по дороге при пробных функционировании на странице появляются новые хорошо решаемые примеры, то для них определяются значимые параметры, которые тут же начинаем поощрять. Процесс строится так, чтобы сохранять возникающие улучшения.

Далее - снова наказание, потом поощрение и т.д. Процесс останавливается либо тогда, когда все примеры страницы решены хорошо, либо после заданного числа наказаний и поощрений. Последним выполняется поощрение. Если не все примеры страницы решены хорошо, то проводится случайный поиск - генерируются случайные сдвиги параметров. Принимаются только те из них, которые не ухудшают решений. Потом снова наказание и поощрение и т.д.

Когда правильно решены все примеры страницы или заметна их часть, переход к следующей странице желательно проводить специальным образом - включая в новую страницу хорошо решаемые примеры с предыдущей. Это проводится для того, чтобы не терять приобретенных навыков при дальнейшем обучении. При описанном способе наказаний и поощрений необходимо такое включение всех или хотя бы части хорошо решаемых примеров с предыдущих страниц.

Есть менее радикальный способ наказаний и поощрений - найдя предельные значения сдвигов, отступить потом назад к α^0 по правилу

$$\alpha = \lambda \alpha^0 + (1 - \lambda) \alpha^f,$$

где $0 < \lambda < 1$, α^f - вектор параметров, на котором закончилась операция наказания или поощрения, начавшаяся в точке α^0 . Число λ - еще один параметр алгоритма.

При наличии такого "шага назад" остается возможность увеличивать и уменьшать любые параметры сети, не сразу натыкаясь на разрушение выработанных ранее навыков.

Заметим, что для описанного способа обучения можно использовать либо общее правило выделения значимых параметров без сортировки, либо с сортировкой и отбрасыванием лишних параметров по достаточности переданного сигнала. Правило, использующее предельное количество значимых синапсов, не подходит: если по значимым синапсам не набирается достаточный

для возбуждения сигнал, то в алгоритме поощрений и наказаний замена s на s' может оказаться недостаточной для дальнейшего движения.

Обучение общих монотонных функциональных элементов можно проводить таким же способом, если имеется алгоритм выделения наиболее значимых параметров для решения отдельного примера.

Рассмотрим теперь сети из монотонных элементов с монотонными связями. Элементы будем обозначать f_1, \dots, f_n . Каждому f_i сопоставляются пространства: входов $E_{i, in}$, выходов $E_{i, out}$ и параметров $E_{i, p}$. Во всех этих пространствах определены конусы положительных элементов $Q_{i, in}$, $Q_{i, out}$ и $Q_{i, p}$. Для каждой пары f_i, f_j определен линейный оператор связей $M_{ij} \in L(E_{i, out}, E_{j, in})$. Связь может быть возбуждающей или тормозящей. По определению, для возбуждающей связи M_{ij} положительный оператор, а для тормозящей положителен $-M_{ij}$. На вход f_j в момент T поступает вектор сигнала

$$A_{ij} = \sum_{t=1}^T M_{ij} f_{it-1},$$

где f_{it-1} – вектор выхода i -го элемента в момент $T-1$.

Напомним, что поощрение параметров состоит в том, что к вектору параметров добавляется положительный вектор, а наказание – в вычитании положительного вектора. Аналогично, поощрение возбуждающей связи состоит в добавлении к M_{ij} положительного оператора, а наказание – в вычитании положительного оператора. Для тормозящей связи, наоборот, назовем поощрением вычитание, а наказанием – прибавление положительного оператора. При этом предполагается, что поощряющие и наказывающие добавки выбираются так, что характер связей не меняется: возбуждающая остается возбуждающей, а тормозящая – тормозящей.

Само формирование поощряющих и наказывающих добавок связано с определением наиболее значимых для данного примера параметров. Именно они поощряются или наказываются.

Как уже говорилось в начале раздела, при произвольной структуре возбуждающих и тормозящих связей нельзя по общей оценке "хорошо" или "плохо" определить, какие элементы и связи наказывать, а какие поощрять. Чтобы можно было

однозначно распространить оценку на элементы и связи, требуется специальная структура сети.

Прежде, чем описать эту структуру, сделаем одно замечание, которое позволит сократить обозначения. Пусть в сети существует такая группа элементов, внутри которой связи только возбуждающие или нулевые, а для любого элемента f_i , не принадлежащего к группе, верно следующее: если от f_i к одному элементу группы ведет возбуждающая (или тормозящая) связь, то такая же (возбуждающая или тормозящая) или нулевая связь ведет от f_i ко всякому другому элементу группы; аналогично и для связей, ведущих от группы к f_i .

Тогда всю группу можно считать одним монотонным функциональным элементом. Пространства E_{in} , E_{out} , E_p и соответствующие положительные конусы для этого объединенного элемента получаются как прямые суммы пространств и конусов по элементам группы

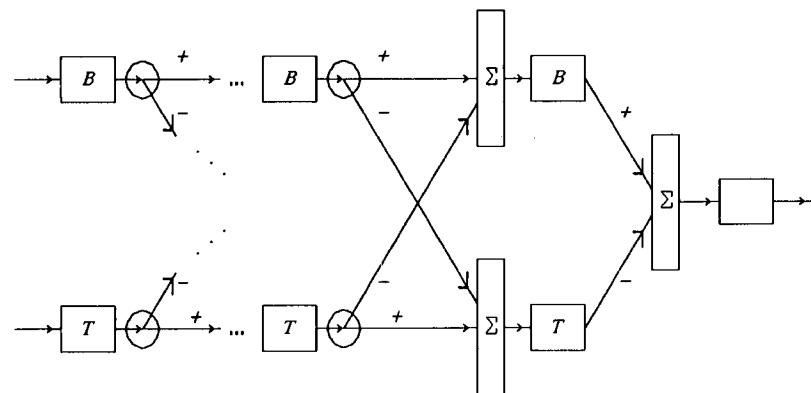


Рис. 23. Сеть, допускающая однозначное распределение двузначной оценки, поощрения и наказания на элементы и связи. Возбуждающие элементы обозначены буквой B , тормозящие – T , возбуждающие связи помечены знаком "+", тормозящие – знаком "-".

Предполагаем, что описанное укрупнение выполнено всюду, где это возможно. Тогда сети, допускающие однозначное распространение оценки на элементы и связи, выглядят весьма

просто (рис. 23). Это слоистые сети, состоящие из монотонных элементов. На последнем - выходном слое один элемент, на остальных - по два. Один элемент слоя называется возбуждающим, другой - тормозящим. Знаки связей, передающих сигналы следующему слою, распределены так - между одноименными элементами - возбуждающие, между разноименными - тормозящие (см. рис. 23). На последний слой от тормозящего элемента предпоследнего слоя идет тормозящая связь, от возбуждающего - возбуждающая. С этим и связаны их названия.

Поощрения и наказания распределяются по элементам и связями так.

Поощрение: поощряются все возбуждающие элементы, связи, ведущие от возбуждающих элементов, в том числе и тормозящие связи, идущие от возбуждающих элементов к тормозящим. В то же время наказываются тормозящие элементы и все связи, идущие от тормозящих элементов. Кроме того, поощряется выходной элемент.

Наказание: наказываются выходной элемент, все возбуждающие элементы и связи, ведущие от возбуждающих элементов. В то же время поощряются тормозящие элементы и связи, ведущие от тормозящих элементов.

После того, как определены наказание и поощрение, а также задан алгоритм выделения значимых параметров, обучение сетей строится так же, как и для монотонных функциональных элементов. Подчеркнем, что каждый элемент может являться целой нейронной сетью.

10.18. Четыре типа устойчивости

Навыки обученного нейрокомпьютера должны быть устойчивы к возмущениям разных типов. Это очевидно, однако придать точный смысл понятию "устойчивость" непросто. В теории динамических систем и дифференциальных уравнений известны десятки определений устойчивости.

Разработчики нейрокомпьютеров находятся в несколько лучшей ситуации - для практических целей не нужно учитывать нюансы, возникающие из-за наличия в теории бесконечной оси времени. достаточно описать различия между содержательно

разными типами устойчивости и создать средства для выработки устойчивых навыков. мы использовали представления о четырех типах устойчивости. Тип устойчивости определяется по типу возмущающих воздействий.

1. Устойчивость к случайным возмущениям входных сигналов.
2. Устойчивость к флуктуациям параметров сети.
3. Устойчивость к разрушению части элементов сети.
4. Устойчивость к обучению новым примерам.

В конкретных ситуациях необходимо определять возмущения, по отношению к которым нужно выработать устойчивость. Так, например, при распознавании визуальных образов можно выделить несколько разновидностей возмущений входного сигнала: прибавление случайного сигнала (шум фона), затенение части исходного изображения, искажение изображения некоторым преобразованием.

В существенной конкретизации нуждается также четвертый тип устойчивости: требуется указать, что это за новые примеры. Трудно даже представить себе устойчивость к обучению любому новому примеру. Если же совокупность новых примеров задана, то их можно включить в задачник. Тем самым выработка устойчивости к обучению этим примерам заменяется просто обучением.

Если принять гипотезу, что обучение новым примерам будет действовать на старые навыки так же, как случайный сдвиг параметров, то получим, что выработка устойчивости 2-го типа является средством для получения устойчивости 4-го типа.

другое средство для этого - выработка устойчивости к обучению "до предела" отдельным примерам, уже входящим в задачник. Это свойство 1-устойчивости состоит в том, что обучение до минимума оценки по любому (одному) из обучающих примеров не разрушает навыка решения остальных. Возмущение здесь состоит в изменении процесса обучения.

для выработки 1-устойчивости примеры предъявляются сети не постстранично, а по одному, и сеть учится каждому из них до предела. Это изменение, надо сказать, было бы неразумным, если бы речь шла о скорейшем приобретении навыка. Однако для выработки важнейшей устойчивости 4-го типа такая периодически

производимая порча процесса обучения может быть полезной.

Проще интерпретировать возмущения, входящие в описания 2-го и 3-го типов устойчивости. Для 2-го типа это - случайное изменение каждого параметра на величину, лежащую в заданных пределах. Для 3-го типа это удаление из сети заданной части элементов каждого типа: определяется число удаляемых нейронов, синапсов и т.п., после чего случайным образом выбираются удаляемые элементы.

Для выработки устойчивости первых трех типов полезны генераторы случайных искажений. Для устойчивости 1-го типа генератор искажений производит возмущение входных сигналов и тем самым преобразует обучающий пример. Для устойчивости 2-го типа генератор искажений меняет случайным образом параметры сети в заданных пределах, а для устойчивости 3-го типа - удаляет случайно выбранную часть сети из заданного количества элементов.

Следует различать обратимые и необратимые искажения. Для обратимых исходная информация о входных сигналах и параметрах сети сохраняется - просто дополнительно генерируется искаженный пример или искаженная сеть. Для необратимых исходная информация теряется. Случайные разрушения, производимые для обучения (3-й тип), должны быть всегда обратимы. Возмущения 1-го и 2-го типов могут быть как обратимыми, так и необратимыми.

Выбор распределений для случайных искажений не столь существенен, как установление их пределов, и не будет здесь обсуждаться.

Можно предложить несколько способов обучения с искажениями.

1. Искажения вводятся заново при каждом предъявлении каждого обучающего примера.
2. Каждая страница примеров искажается, а потом ведется обучение по этой странице, далее вносятся новые искажения, либо производится переход к следующей страницы.
3. Каждая страница дополняется набором искаженных примеров и ведется обучение по таким расширенным страницам.

Опыт показывает, что третий подход - обучение по расширенным страницам - более эффективен. Поэтому обсудим его детальнее. Пусть задана исходная страница обучающих примеров. В расширенной странице каждый пример исходной сохраняется. Кроме него появляется ряд примеров с тем же ответом, но искаженными входными сигналами. Это - для выработки устойчивости 1-го типа.

Для выработки устойчивости 2-го типа и 3-го типа нужно вводить новый вид примеров: сеть получает оценку за решение данного примера другой сетью! Эта другая сеть получается из исходной случайным сдвигом параметров (2-й тип) или удалением части элементов (3-й тип). Возмущения обратимы - исходные значения параметров сохраняются, случайные сдвиги и разрушения производятся каждый раз заново из исходного состояния.

Такая организация страниц для выработки устойчивости второго типа близка к методу виртуальных частиц - там тоже в общую оценку входит оценка решения другой (виртуальной) сетью.

Оценки за решение исходных и искаженных примеров по-разному должны входить в H_{pg} для расширенной страницы. Чем больше искажение - тем меньше вес оценки соответствующего примера.

Одна из вышнейших проблем нейрокомпьютинга - выработка устойчивости 4-го типа. Нужно, чтобы новые навыки не разрушали старых. Выработка устойчивости 2-го и 3-го типов способствует повышению стабильности навыков. Однако этого может оказаться недостаточно. Тогда на помощь приходит еще один прием - выработка 1-устойчивости, устойчивости к обучению каждому примеру "до конца" - последовательная минимизация индивидуальных оценок за решение отдельных примеров.

С самого начала обучения делать это нельзя. Процесс чаще всего просто расходится. Необходимо сначала учить по всему задачнику, страница за страницей, пока не будут достигнуты удовлетворительные результаты по всем примерам, и лишь потом переходить к индивидуальному "переучиванию", чередуя его с

постраничным обучением. Другим способом выработать 1-устойчивость за разумное время нам почти никогда не удавалось.

Опыт показывает, что обучение может выработать устойчивость к весьма сильным возмущениям. Так, в решавшихся нами задачах распознавания визуальных образов уровень шума на входе мог в несколько раз превосходить общую интенсивность сигнала, случайный сдвиг параметров - достигать 0.2-0.3 их предельной величины, разрушение 10-20% элементов. И все равно обученная сеть делает не более 10% ошибок!

11. ОПЫТ РЕАЛИЗАЦИИ

11.1. Краткая справка о том, как обучение было ускорено в 10000 раз

для отработки и сравнения различных алгоритмов обучения был создан эмулятор нейрокомпьютера для персональных компьютеров типа IBM PC. Затраты времени на обучение сравнивались в тактах сети.

Опыт обучения накапливался на двух классах задач: распознавание визуальных образов и бинарная классификация. Программы бинарной классификации использовались для различных прикладных задач: медицинская диагностика, социально-психологическая диагностика (подбор команды), прогнозирование результата выборов и др.

для сравнения использовались стандартные алгоритмы обратного распространения ошибки с предъявлением примеров по одному. Описанные в предыдущих разделах усовершенствования позволили ускорить обучение в 10^4 - 10^5 раз. В результате эмулятор на IBM PC приобрел не только демонстрационное, но и прикладное значение, так как обучение многим прикладным задачам требует теперь всего 10-15 минут. Время срабатывания обученной сети - от долей секунды до нескольких секунд - в зависимости от числа нейронов.

В качестве примера предлагается описание пакета CLAB (КЛАБ - классификатор бинарный). В нем используется один алгоритм обучения, достаточный для многих прикладных задач медицинской, социально-психологической и технической диагностики.

11.2. Пакет CLAB: пакет программ для создания нейросетевого бинарного классификатора на базе ЭВМ IBM PC/AT¹

11.2.1. Нейросеть

Нейросеть состоит из нейронов и синапсов. Через синапсы нейроны посылают сигналы. Это происходит так:

- 1) в текущий момент времени нейрон передает свой текущий сигнал на все выходящие из него синапсы;
- 2) одновременно он получает сигналы со всех входящих в него синапсов и, кроме того, внешний входной сигнал;
- 3) полученные сигналы суммируются и преобразуются согласно характеристической функции (в пакете CLAB вид этой функции $Y=x/(c+abs(x))$);
- 4) результат этой операции берется в качестве сигнала для следующего момента времени.

При прохождении через синапс сигнал умножается на вес синапса. Задача обучения нейросети заключается в подборе таких весов синапсов, чтобы по предъявлении определенных входных сигналов нейросеть после заданного числа шагов выдала с выходных нейронов требуемый ответ.

В пакете CLAB обучение производится путем минимизации суммарной целевой функции модифицированным партан-методом, основанным на принципе двойственности.

11.2.2. Задача бинарной классификации

Бинарной классификацией мы называем один частный случай распознавания образов, а именно, тот случай, когда образы могут принадлежать двум классам, условно называемым здесь "красные" и "синие".

В пакете CLAB образы представляются векторами входных сигналов. В таком виде они подаются нейросети для обучения или распознавания. Мы их называем примерами. Совокупность примеров, предназначенных для обучения, составляет обучающую выборку. Обучающая выборка хранится в файле, называемом

задачником.

Выходные сигналы для всех примеров снимаются с двух последних нейронов. Они интерпретируются как две координаты на плоскости, а ответ нейросети изображается крестиком с этими координатами. На этой же плоскости указываются две точки, изображаемые курсорами соответствующего цвета. Ответ считается правильным, если крестик оказывается вблизи нужного курсора.

11.2.3. Как создать новую сеть?

Для создания новой сети в пакете имеется программа NetGeneg, которая сначала требует указать ей имя файла для хранения параметров сети и карты синапсов, а затем запрашивает требуемые параметры:

SIZE - количество нейронов сети;
TIME - число тактов времени от получения входных сигналов до выдачи результата;
CH - характеристику, то есть параметр характеристической функции нейронов;
Level - уровень начальных значений синапсов (которые генерируются датчиком псевдослучайных чисел в диапазоне от $-Level/2$ до $+Level/2$).

Наш опыт показывает, что параметры SIZE и TIME лучше выбирать по возможности меньшими, а CH - от 0,1 до 0,8.

11.2.4. Как создать задачник?

В принципе задачник можно создать, пользуясь любым штатным редактором, установленным на вашей РС. Для этого надо только выполнить правила, перечисленные ниже.

1. Задачник организован по страницам. В начале каждой страницы в отдельной строке следует указать количества "красных" и "синих" примеров (сначала "красных", потом "синих").
2. Каждое из этих чисел не должно превышать двадцати.
3. В конце каждого примера на отдельной строке набирается

¹Авторы пакета CLAB и его описания - С.Е.Гилев и Е.М.Миркес.

- буква R для "красного" или L для "синего" примеров.
4. Пример можно описывать в нескольких строках, в каждой строке перечисляются через запятые номера нейронов, на которые подаются входные сигналы. После последнего нейрона ставится двоеточие, а "затем через пробелы" - значения входных сигналов в том же порядке, в котором указывались номера соответствующих нейронов.

В пакете CLAB имеется специальный редактор Editor, который берет на себя часть функций по контролю правильности задачника.

Этот редактор высвечивает на экране два окна, одно из которых - NEURON предназначено для имен входных нейронов, а другое - VALUE - для соответствующих сигналов. Редактор софдует одновременно номер нейрона и сигнал с нулевым значением, которое тотчас же предлагает отредактировать. Примеры с неопределенной принадлежностью тому или иному классу редактор не включает в задачник. Точно так же в задачник не включаются страницы с нулевым числом примеров.

Вы можете задать свои имена для входных нейронов. Для этого Вам надо создать файл с расширением .ptn. Если такого файла у Вас нет, то Editor предложит воспользоваться стандартным ptn-файлом, в котором в качестве имен входов используются их номера. В ptn-файле указываются также имена (каждое не более десяти символов) двух классов - сначала "красного", а затем "синего". Для каждого вновь вводимого имени вначале указывается количество входных нейронов, объединяемых этим именем. Если это число 0, то соответствующая строка будет выводиться на экран в окне NEURON, но Editor не затребует для него входных значений. Такие "имена" можно использовать в качестве комментариев или для определения алиниых имен, например, вопросов в социологических задачах.

Ptn-файл может содержать дополнительную информацию для этапа обучения, а именно, можно пометить звездочками те входы, получение информации на которые затруднительно и при использовании классификатора эта информация зачастую будет отсутствовать. Для обучения нейросети задаче с такими "дырами" в векторе входов в программе Teacher (см. ниже)

имеется соответствующее средство - "дырокол".

В следующих ниже таблицах 1, 1' перечислены все функции редактора. Следует лишь пояснить, что курсор изображается в виде выделенной цветом пары нейрон-значение. Если курсор не виден, это означает, что он находится за последней парой в строке или за последней строкой в примере. Вставки производятся перед текущими строкой или парой. При редактировании номера нейрона или значения сигнала клавиша Esc означает отказ от редактирования данного числа, а Enter - конец редактирования.

Таблица 1. Help Редактора

Help	
F1	- Help
F2	- Save
Arrows	- Move cursor
R or L	- Set a class of the current example
Enter	- Switch to Editor mode / / Insert edited value
P	- Split the current page (before the current example)
E	- Make a new example (before the current one)
PgUp or PgDn	- Go to another example
^PgUp or ^PgDn	- Go to another page
Home	- Go to the 1-st example at the current page
End	- Go to the last example at the current page
^Home	- Go to the 1-st page
^End	- Go to the last page
Esc	- Exit from Editor mode / Quit
Press any Key to leave Helper	

Таблица 1'. Help Редактора

Help	
F1	- Help
F2	- Сохранить
Arrows	- Двигать курсор
R or L	- Задать класс текущего примера
Enter	- Переход в режим редактирования / / Ввод отредактированного значения
P	- Разделить текущую страницу (перед текущим примером)
E	- Создать новый пример (перед текущим)
PgUp or PgDn	- Перейти к другому примеру
^PgUp or ^PgDn	- Перейти к другой странице
Home	- Перейти к 1-му примеру на текущей странице
End	- Перейти к последнему примеру на текущей странице
^Home	- Перейти к 1-й странице
^End	- Перейти к последней странице
Esc	- Выйти из режима редактирования / / Закончить
Нажмите любую клавишу, чтобы выйти из Helper'a	

11.2.5. Как обучать нейросеть

Самый простой способ обучения нейросети состоит в следующем:

- 1) запустить программу Teacher, при этом программа запросит дорожку или имя файла, содержащего описание сети, а затем - дорожку и имя файла задачника;
- 2) ждать, пока все примеры на первой странице не обучатся до приемлемых оценок, после чего перейти к следующей странице.

Процесс повторять, пока не обучатся примеры со всех страниц. При этом возможно, что цикл по страницам придется повторять неоднократно. Однако вмешательство оператора может существенно ускорить этот процесс. В пакете CLAB имеются средства для вмешательства в процесс обучения. Они рассматриваются ниже.

11.2.5.1. Курсоры

Изменение положения курсоров меняет вид целевой функции. Это может способствовать выходу из тупиковой ситуации, когда на протяжении нескольких циклов подряд оценка заметно не изменяется.

Кроме того, наш опыт показывает, что на конечном этапе обучения небольшое улучшение оценки на какой-либо странице задачника может привести к ухудшению ситуации на других страницах. Поэтому на этапе доучивания следует ослабить жесткость требований к разделенности классов. Этого можно добиться сближением курсоров.

Еще один эффект, достигаемый изменением положения курсоров, состоит в том, что приближение курсора к крестику, соответствующему примеру с наименьшей оценкой, приводит к уменьшению его вклада в суммарный градиент. Следовательно, увеличивается влияние примеров с худшими оценками. В результате крестики, соответствующие примерам из одного класса, собираются более плотными группами.

11.2.5.2. Веса

Изменения относительного вклада градиентов от разных примеров можно добиться также изменением их весов. Веса в нашем пакете можно задавать в интервале от нуля, что соответствует полному исключению примера, до максимального значения, равного девяти. Следует с осторожностью задавать большой разброс в весах, так как в этом случае при небольшом улучшении оценки примера с большим весом могут сильно ухудшиться оценки остальных примеров. Наш опыт показывает, что в большинстве случаев достаточно придать двум-трем примерам с наихудшими оценками веса, равные двум.

11.2.5.3. BUMP

Для вывода сети из локального минимума оценки мы включили в обучающую программу пакета CLAB процедуру, которую мы назвали BUMP. Эта процедура вносит в карту

синапсов случайный вклад, величина которого задается при вызове процедуры.

При обучении сети с частым применением BUMP'а наблюдался интересный эффект - сеть постепенно приобретала помехоустойчивость, то есть со временем уровень BUMP'ов можно было повышать без существенного ухудшения оценки. Этот эффект легко объясняется, если представить функцию оценки как рельеф в многомерном пространстве - пространстве карт синапсов. Понятно что BUMP легко вышибает сеть из узких ям и оврагов, в то время как для вывода из широких долин его уровень может оказаться недостаточным. В результате такого "естественного отбора" сеть закрепляется на широкой долине. Существует много принципиально различных стратегий для выбора уровня BUMP'а в ходе обучения. Поэтому обучение с автоматическим применением процедуры BUMP нами не включено в состав пакета CLAB. Оператор, обучающий сеть, должен сам выбрать стратегию.

11.2.5.4. "Дырокол"

Если задача такова, что для некоторых примеров часть информации может отсутствовать (например, из-за ее труднодоступности), то Ptn-файл может содержать, как это указывалось в разделе 11.2.4, звездочки при именах соответствующих входных нейронов. Пакет CLAB позволяет создавать варианты примера, заменяя сигналы на отмеченных звездочками входах нулевыми значениями. Индивидуальная оценка каждого примера представляется при этом суммой оценок по всем вариантам.

Следует иметь в виду, что при обучении предъявляются все варианты расстановки "дыр" по всем отмеченным звездочками позициям. Поэтому, чтобы таких вариантов было не слишком много (их предъявление для обучения требует времени), не стоит задавать слишком большим (>3) число входов, на которые одновременно могут быть поданы неопределенные значения (это число запрашивается

программой).

Таблица 2. Help при обучении

	Help	Help
F 1	-	= Сохранить
F 2	- Save	= Переместить курсоры
F 3	- Replace cursors	= Сменить веса
F 4	- Change weights	= Сменить страницу
F 5	- Change page	BUMP
F 6	-	= Создать "дыры"
F 7	- Make "holes"	= Двигать курсор
Arrows	- Move a cursor	= Поменять курсор
Tab	- Change a cursor	= Продолжать
Enter	- Continue	= Ввести новые веса
Numeric Keys	- Insert new weights	= Закончить
Esc	- Quit	
		Нажмите любую клавишу для выхода из Helper'a

12. УРОВНИ ОТЧУЖДЕНИЯ НЕЙРОННОЙ СЕТИ

до сих пор обсуждение велось так, как будто вся жизнь нейронной сети от конструирования структуры и обучения до решения прикладных задач происходит на одном нейрокомпьютере. Это совсем не обязательно. Более того, разумно иметь нейрокомпьютеры разной сложности и универсальности и передавать нейронную сеть от одних к другим по мере ее обучения. Можно определить разные уровни отчуждения обученной сети от универсального нейрокомпьютера. Выделим сначала предельные случаи.

Нулевой уровень отчуждения – нейронная сеть существует на универсальном нейрокомпьютере, имеется сложно организованный задачник, богатый набор команд и большая библиотека обучающих алгоритмов.

Третий уровень отчуждения – полное отчуждение от универсального учителя. Может выполняться только одна команда – прямое функционирование. Обучение невозможно, зато возможны очень простая техническая реализация и наиболее высокое быстродействие.

Второй уровень отчуждения – возможно доучивание по отдельному примеру. Кроме прямого функционирования могут еще выполняться один или несколько шагов какого-нибудь алгоритма обучения. Число и, возможно, величина шагов фиксируются заранее. Алгоритм выбирается попроще.

Первый уровень отчуждения допускает доучивание по одной небольшой (сравнительно со всем задачником) странице примеров. Алгоритм, опять же, один, однако число шагов, их величина и другие параметры могут подбираться в ходе обучения, примерам страницы могут назначаться веса, список примеров может обновляться.

Чем дальше от универсального учителя, тем труднее приобрести новый сложный навык, но тем проще техническая реализация и дешевле устройство.

13. ЧТО ОСТАЛОСЬ ЗА РАМКАМИ КНИГИ?

1. Бионическое направление в нейрокомпьютинге, идущее от прямых аналогий с работой мозга.
2. Описание конкретных задач, решаемых нейрокомпьютерами. Некоторые классы задач, например, распознавание образов, заслуживают отдельных книг.
3. Проблемы конструирования нейрокомпьютеров в "железе".
4. Адаптивные среды – непрерывные системы, способные к обучению. В них уже нет отдельных нейронов или синапсов.
5. Большего внимания заслуживает ассоциативная память, а также гибридные нейрокомпьютеры, в которых ассоциативная память используется для предварительной обработки информации, подаваемой потом обучаемой сети.

ЛИТЕРАТУРА

Предлагаемый список литературы заведомо неполон. В него включено несколько подобных руководств по методам оптимизации, работы по обучающимся системам и нейронным сетям. Предпочтение отдавалось монографиям и обзорам. За современным состоянием исследований в области обучения нейронных сетей можно следить по журналу "Neural Networks" (Pergamon Press, Oxford), издаваемому с 1988 года.

1. Айзerman М.А., Браверман Э.М., Розенбаум А.И. Метод потенциальных функций в теории обучения машин. М.: Наука, 1970. 383 с.
2. Барцев С.И., Гилев С.Е., Охонин В.А. Принцип двойственности в организации адаптивных сетей обработки информации //Динамика химических и биологических систем. Новосибирск: Наука, 1989. С. 6-55.
3. Дунин-Барковский В.Л. Нейронные схемы ассоциативной памяти //Моделирование возбудимых структур. Пущино: изд. ЦБИ, 1975. С. 90-141.
4. Пшеничный Б.Н., Данилин Ю.М. Численные методы в экстремальных задачах. М.: Наука, 1975. 319 с.
5. Соколов Е.Н., Вайткевич Г.Г. Нейроинтеллект: от нейрона к нейрокомпьютеру. М.: Наука, 1989. 238 с.
6. Фролов А.А., Муравьев И.П. Нейронные модели ассоциативной памяти. М.: Наука, 1987. 160 с.
7. Цыпкин Я.З. Основы теории обучающихся систем. М.: Наука, 1970. 252 с.
8. The Adaptive Brain/Ed. by S. Grossberg. North-Holland, 1987. V.1. Cognition, Learning, Reinforcement, and Rhythm. 498 p. V.2. Vision, Speech, Language, and Motor Control. 514 p.
9. Alexander S. Th. Adaptive Signal Processing. Theory and Applications. Springer. 1986. 179 p.
10. Amari Sh., Maginu K. Statistical Neurodynamics of Associative Memory//Neural Networks, 1988. V.1. N1. P. 63-74.
11. Arbib M.A. Brains, Machines, and Mathematics. Springer,

1987. 202 p.

12. Baba N. New Topics in Learning Automate Theory and Applications. Springer, 1985. 131 p. (Lec. Not. Control and Information, N71).
13. The Computer and the Brain. Perspectives of Human and Artificial Intelligence/Ed. by J.R. Brinc, C.R. Haden, C. Burava. North-Holland, 1989. 300 p.
14. Connectionism in Perspective/Ed. by R. Pfeifer, Z. Schreter, F. Fogelman-Soulie and L. Steels. North-Holland, 1989. 518 p.
15. Dennis J.E.Jr., Schnabel R.B. Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, 1983. = Деннис Дж. мл., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений. М.: Мир, 1988. 440 с.
16. Disordered Systems and biological Organization/Ed. by Bienenstock, F. Fogelman-Soulie, G. Weisbuch. Springer, 1986. 405 p.
17. Fukushima K. Neocognitron: A Hierarchical Neural Network Capable of Networks, 1988. V.1. N2. P. 119-130.
18. Gecseg F. Products of Automata. Springer, 1986. 107 p.
19. George F.H. Brain as a computer. Pergamon Press, 1961. = Ажорад Ф. Мозг как вычислительная машина. М. Изд-во иностр. лит., 1963. 528 с.
20. Gill Ph.E., Murray W., Wright M. Practical optimization. Academic Press, 1981. = Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. М.: Мир, 1985. 509 с.
21. Grossberg S. Nonlinear Neural Networks: Principles, Mechanism and Architectures//Neural Networks, 1988. V.1. N1. P. 17-62.
22. Hecht-Nielsen R. Neurocomputing: Picking the Human Brain//IEEE Spectrum, 1988. March. P. 36-41.
23. Himmelblau D.M. Applied nonlinear programming. McGraw-Hill, 1972. = Химмельблау А. Прикладное нелинейное программирование. М.: Мир, 1975. 534 с.
24. Hopfield J.J. Neural Networks and physical systems with emergent collective computational abilities//Proc. Nat. Sci.

USA. 1982. V. 79. P. 2554-2558.

25. Kohonen T. An Introduction to Neural Computing // Neural Networks, 1988. V. 1. N1. P. 3-16.
26. Kohonen T. Self-Organization and Associative Memory. Springer, 1989. 312 p.
27. Narendra K.S., Amnasway A.M. A stable Adaptive Systems. Prentice-Hall, 1988. 350 p.
28. Neural Computers/Ed. by R. Eckmiller, Ch. Malsburg. Springer, 1989. 556 p.
29. Real Brains, Artificial Minds/Ed. by J.L. Casti, A. Karlqvist. Norton-Holland, 1987. 226 p.
30. Rosenblatt F. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Washington, D.C.: Spartan Books, 1961. = Розенблатт Ф. Принципы нейродинамики. Перцептрон и теория механизмов мозга. М.: Мир, 1965. 480 с.
31. Rummelhart D.E., Hinton G.E., Williams R.J. Learning representations by back-propagating errors//Nature, 1986. V. 323. P. 533-536.
32. Widrow B., Stearns S. Adaptive Signal Processing. Prentice-Hall, 1985. = Уидроу Б., Стирнз С. Адаптивная обработка сигналов. М.: Мир, 1989. 440 с.

ОГЛАВЛЕНИЕ

Предисловие	3
1. Перечень элементов	8
2. Нейрон	9
2.1. Функционирование нейрона	9
2.2. Обучение нейрона	10
2.3. Сложный нейрон как сеть простых	11
3. Сумматоры	14
4. Синапс	15
4.1. Функционирование синапса	15
4.2. Обучение синапсов и коннекционизм	17
5. Входы, выходы и функционирование	20
5.1. Входы и выходы	20
5.2. Сети периодического функционирования	21
5.3. Сети непрерывного функционирования	22
6. Оценки и примеры	23
6.1. Оценки одного акта функционирования	23
6.2. Оценки для нескольких обучающих примеров	24
7. Градиенты и двойственность	26
7.1. Обучение как оптимизация	26
7.2. Двойственное функционирование сетей автоматов при вычислении градиентов сложных функций	26
7.3. Двойственное функционирование сетей автоматов. Обобщения	37
7.4. Смысъл двойственных переменных	47
7.5. Обратное функционирование нейронных сетей	49
7.6. Back-back процедура	63
7.7. Несколько заключительных замечаний о двойственности	67
8. Анализ чувствительности	69
8.1. Зачем нужен анализ чувствительности	69
8.2. Определение чувствительности по производным функциям оценки	70
8.3. Анализ чувствительности по прямому функционированию	71

8.4. Системы, элементы которых линейны по параметрам	72	10.15. Несколько слов о философии оптимизации	118
8.5. Формирование списка параметров, к изменениям которых система наиболее чувствительна	74	10.16. Метод виртуальных частиц	120
9. Контрастирование сети	77	10.17. Монотонные сети, двузначная оценка и правило Хебба	123
9.1. Контрастирование для интерпретации навыков сети	77	10.18. Четыре типа устойчивости	138
9.2. Контрастирование для технической реализации нейрокомпьютера	77	11. Опыт реализации	143
9.3. Как проводить контрастирование	78	11.1. Краткая справка о том, как обучение было ускорено в 10000 раз	143
9.4. Сглаживание характеристик - средство для улучшения интерполяционных и экстраполяционных навыков	80	11.2. Пакет CLAB: пакет программ для создания нейросетевого бинарного классификатора на базе ЭВМ IBM PC/AT	144
10. Алгоритмы обучения	83	11.2.1. Нейросеть	144
10.1. Обучение нейронных сетей и алгоритмы оптимизации	83	11.2.2. Задача бинарной классификации	144
10.2. Способ предъявления примеров и принцип постраничного обучения	84	11.2.3. Как создать новую сеть?	145
10.3. Две идеологии в построении обучения сетей: популяционная и генетическая	87	11.2.4. Как создать задачник?	145
10.4. Операции, выполняемые сетью	89	11.2.5. Как обучать нейросеть	148
10.5. Учет ограничений при обучении	95	11.2.5.1. Курсоры	149
10.6. Одномерная оптимизация	96	11.2.5.2. Веса	149
10.7. Авумерная оптимизация	100	11.2.5.3. BUMP	149
10.8. Наискорейший спуск, случайный поиск и партан-методы	103	11.2.5.4. "Дырокол"	150
10.9. Использование back-back процедуры для ускорения обучения	106	12. Уровни отчуждения нейронной сети	152
10.10. Одношаговый квазиньютоновский метод и сопряженные градиенты	108	13. Что осталось за рамками книги?	153
10.11. Метод сопряженных направлений, не использующий производных	110	Литература	154
10.12. Использование случайных возмущений для ускорения обучения	113		
10.13. Температура сети и энергия активации параметров	115		
10.14. Использование случайных возмущений для формирования устойчивых навыков и для создания условий дальнейшего обучения	117		