ISSN 1512-1232

УДК 004.42

ЗАМЕЧАНИЯ К ОБУЧАЮЩЕМУ ПРАВИЛУ ОБРАТНОГО РАСПРОСТРАНЕНИЯ

Прангишвили А.И.[1], Намичейшвили О.М.[2]

[1] Грузинский технический университет, 0160, Грузия, Тбилиси, Костава 77
[2] Грузинский технический университет, 0160, Грузия, Тбилиси, Костава 77

Аннотация: *Алгоритм обратного распространения ошибки является одним из методов обучения многослойных нейронных сетей прямого распространения, называемых также многослойными персептронами. Многослойные персептроны успешно применяются для решения многих сложных задач. Однако нельзя утверждать, что алгоритм обратного распространения ошибки предлагает действительно оптимальное решение всех потенциально разрешимых проблем. Исторически относительно обучения многослойных машин он просто развеял пессимизм, который появился в нейроинформатике в семидесятых годах минувшего столетия, и не более. В связи с этим к названному методу и ранним работам касательно его вычислительной мощи, статья содержит определенные замечания, подтверждающие данную мысль.*

Ключевые слова: *прямое распространение, обратное распространение, многослойный персептрон, скорость обучения, параметр момента, сетевой паралич*

Введение

Как известно, однослойная сеть искусственных нейронов имеет серьёзные ограничения: класс задач, которые могут быть разрешены, очень узок. В данной статье мы сосредоточимся на сетях прямой передачи с несколькими слоями процессоров.

Мински и Паперт (Minsky и Papert, 1969 [1]) показали в 1969 году, что нейронная сеть прямой передачи с двумя слоями может преодолеть многие ограничения, но не представили решения проблемы корректировки весов от входа до скрытых процессоров (процессорных блоков обработки данных). Ответ на этот вопрос был дан Румельхартом, Хинтоном и Уильямсом в 1986 году (Rumelhart, Hinton, и Williams, 1986 [2]), и подобные решения, как оказалось, были получены и ранее (Werbos, 1974 [3]; Parker, 1985 [4]; Cun, 1985 [5]).

Центральная идея, стоящая за этим решением, заключается в том, что ошибки для процессоров скрытого слоя определяются обратным распространением ошибок процессоров выходного слоя.

По этой причине метод часто называют обучающим правилом обратного распространения.

Обратное распространение можно также рассматривать как обобщение дельта-правила для нелинейных функций активации и многослойных сетей.

Конечно, когда используются линейные функции активации, многослойная сеть не может обладать большей вычислительной мощностью, нежели однослойная сеть.

1 Многослойные сети прямой передачи (*прямого* распространения)

ISSN 1512-1232

Сеть прямой передачи (***прямого*** распространения), или сети с механизмом прогнозирования событий имеет многослойную структуру. Каждый слой состоит из процессоров, которые получают свой входной сигнал от процессоров нижнего слоя в прямом направлении и посылают свой выходной сигнал в прямом направлении процессорам верхнего слоя. В пределах слоя никаких связей нет. N_i входов поданы в первый слой $N_{h,1}$ *скрытых* процессоров.

Входные процессоры - просто «устройства разветвления выхода»; в этих устройствах никакая обработка не имеет место. Активация скрытого процессора является функцией F_i взвешенных входных сигналов и смещения. Она задается соответствующим уравнением.

Выходной сигнал скрытых процессоров распределён по следующему слою $N_{h,2}$ скрытых процессоров, вплоть до последнего слоя скрытых процессоров, выходы которых подаются в слой N_o *выходного процессора,* или *процессора вывода* (Рисунок: 1).

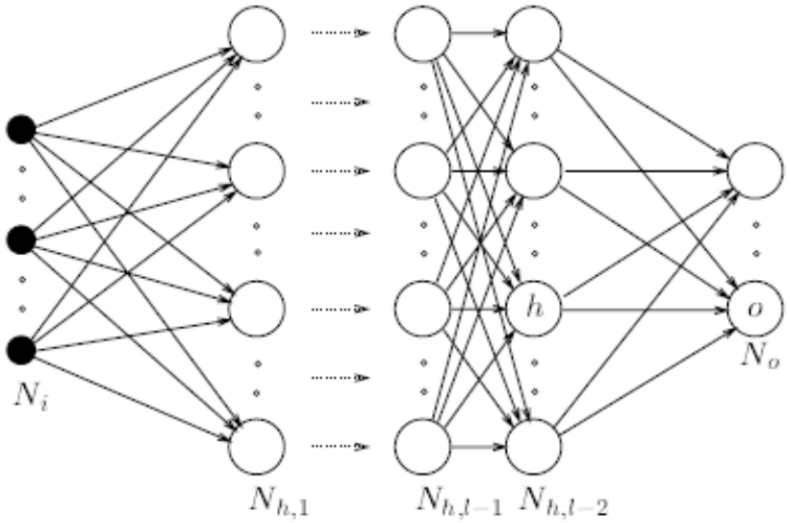


Рисунок 1. Многослойная сеть с l уровнями процессоров.

Хотя обратное распространение может быть применено к сетям с любым числом слоёв, для сетей с двоичными (бинарными) процессорами, как это показано (Hornik, Stinchcombe, & White, 1989 [6]; Funahashi, 1989 [7]; Cybenko, 1989 [8]; Hartman, Keeler, & Kowalski, 1990 [9]), только один слой скрытых процессоров достаточен, чтобы с произвольной точностью аппроксимировать любую функцию с конечным множеством разрывов, если функции активации скрытых процессоров нелинейны (универсальная теорема аппроксимации). В большинстве приложений сеть прямого распространения с единственным слоем скрытых процессоров используется с сигмоидальной функцией активации для процессоров.

2 Обобщённое дельта-правило

Так как теперь будут использоваться процессоры с нелинейными функциями активации, так называемое дельта-правило, которое известно для линейных функций, следует обобщить относительно набора нелинейных функций активации. Активация - дифференцируемая функция общего входного сигнала, задаваемая как

$$y_k^p = \mathsf{F}\left(s_k^p\right), \tag{1}$$

где

ISSN 1512-1232

$$s_k^p = \sum_j w_{jk} y_j^p + \theta_k . \quad (2)$$

Чтобы получить правильное обобщение хорошо известного из литературы дельта-правила,, следует установить

$$\Delta_p w_{jk} = -\gamma \frac{\partial E^p}{\partial w_{jk}} . \quad (3)$$

Мера E^p ошибки определяется как общая квадратичная ошибка для модели p в выходных процессорах:

$$E^p = \frac{1}{2} \sum_{o=1}^{N_o} \left(d_o^p - y_o^p \right)^2 , \quad (4)$$

где d_o^p - желаемый выходной сигнал для процессора o, когда модель p зафиксирована. Далее устанавливают $E = \sum_p E^p$ как *суммарную квадратичную ошибку*. Можем записать, что

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p} \frac{\partial s_k^p}{\partial w_{jk}} . \quad (5)$$

С помощью уравнения (2) мы видим, что второй сомножитель

$$\frac{\partial s_k^p}{\partial w_{jk}} = y_j^p \ . \quad (6)$$

Определив

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} , \quad (7)$$

мы получим правило обновления, которое является эквивалентным описанному в литературе дельта-правилу. Новое правило обновления приводит к градиентному спуску на поверхности ошибки, если изменения веса производят согласно соотношению:

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p . \quad (8)$$

Этим приёмом следует вычислить, какой должна быть величина δ_k^p для каждого процессора k в сети. Получаемый сейчас интересный результат состоит в том, что имеется простое рекурсивное вычисление величины δ для процессора, которое может быть осуществлено обратным распространением сигналов ошибки через сеть.

Для вычисления δ_k^p мы применяем цепное правило, чтобы записать эту частную производную как произведение двух сомножителей: один сомножитель отражает изменение в ошибке как функцию выхода процессора, а другой - изменение на выходе как функцию изменений на входе. Таким образом, имеем

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p} \frac{\partial y_k^p}{\partial s_k^p} . \quad (9)$$

Теперь вычислим второй сомножитель. Согласно уравнению (1), имеют соотношение

$$\frac{\partial y_k^p}{\partial s_k^p} = \mathrm{F}'\left(s_k^p \right), \quad (10)$$

ISSN 1512-1232

которое является просто производной ограниченной функции F с бесконечной областью определения для k-го процессора, оценённой по входу сети s_k^p к этому процессору. Чтобы вычислять первый сомножитель уравнения (9), мы рассматриваем два случая. Сначала предположим, что процессор k является выходным процессором сети $k = o$. В этом случае, как следует из определения E^p,

$$\frac{\partial E^p}{\partial y_o^p} = -\left(d_o^p - y_o^p\right), \tag{11}$$

что является известным результатом, получаемым при применении стандартного дельта-правила. Подставляя уравнения (11) и (10) в уравнение (9), мы приходим к выражению

$$\delta_o^p = \left(d_o^p - y_o^p\right) \cdot \mathsf{F}_o'\left(s_o^p\right) \tag{12}$$

для любого выходного процессора o. Во-вторых, если k не является выходным процессором и представляет собой всего лишь скрытый процессор $k = h$, мы не можем легко установить вклад процессора в выходную ошибку сети. Однако, мера ошибки может быть записана как функция входов сети от скрытого слоя к выходному слою: $E^p = E^p\left(s_1^p, s_2^p, \ldots, s_j^p, \ldots\right)$, и мы используем цепное правило к этой записи

$$\frac{\partial E^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} \cdot \frac{\partial s_o^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} \cdot \frac{\partial}{\partial y_h^p} \sum_{j=1}^{N_h} w_{ko} y_j^p = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} w_{ho} = -\sum_{o=1}^{N_o} \delta_o^p w_{ho}\,. \tag{13}$$

Подставляя это в уравнение (9), получают следующий результат:

$$\delta_h^p = \mathsf{F}'\left(s_h^p\right) \cdot \sum_{o=1}^{N_o} \delta_o^p w_{ho}\,. \tag{14}$$

Уравнения (12) и (14) дают рекурсивную процедуру для того, чтобы вычислить δ для всех процессоров в сети. Это используется для вычисления изменения веса, согласно уравнению (8). Данная процедура и составляет *обобщенное дельта-правило* для сети прямой передачи нелинейных процессоров.

2.1 Осмысление обратного распространения

Уравнения, полученные в предыдущем разделе, с математической точки зрения совершенно корректны, но что делает их на самом деле не заслуживающими серьёзного внимания? Разве есть способ понять обратное распространение иначе, не прибегая к этому описанию необходимых уравнений?

Ответ, конечно же, звучит утвердительно. Фактически, весь процесс обратного распространения интуитивно совершенно ясен. Смысл того, что происходит в вышеупомянутых уравнениях, состоит в следующем.

Когда обучающий образ зафиксирован, значения активации влияют на выходные процессоры, и фактический выход сети сравнивается с желаемыми выходными значениями; результат такого сравнения, обычно, оценивается ошибкой каждого из выходных устройств. Обозначим эту ошибку через e_o для отдельного выходного процессора o. Естественно, мы должны стремиться доставить этой ошибке e_o нулевое значение.

ISSN 1512-1232

Самый простой способ добиться этого - «жадный» алгоритм[1] (метод экономного продвижения), когда стремятся изменять связи в нейронной сети таким образом, чтобы в конце периода настройки ошибка e_o приблизительно стала нулевой для этого частного образа.

На основании дельта правила известно, что для уменьшения ошибки мы должны адаптировать входящие веса согласно соотношению

$$\Delta w_{ho} = \left(d_o - y_o\right) y_h . \tag{15}$$

Это первый шаг. Но этот единственный шаг недостаточен: когда мы применяем только данное правило, веса от входа к скрытым процессорам никогда не изменяются, и мы не имеем полного представления о возможностях сети прямой передачи как это утверждается универсальной теоремой аппроксимации.

Чтобы настроить (адаптировать) веса от входа к скрытым процессорам, нам вновь необходимо применить дельта правило. В этом случае, однако, мы не имеем значения величины δ для скрытых процессоров. Это решается в соответствии с цепным правилом, которое делает следующее: распределяет ошибку некоторого выходного процессора o между всеми скрытыми процессорами, которые с соответствующими весами связаны с этим процессором.

Иначе говоря, определённый скрытый процессор h получает небольшое изменение (приращение сигнала ошибки) от каждого выходного процессора o, равное небольшому изменению (приращению сигнала ошибки) этого выходного процессора, умноженному на вес связи между скрытым процессором и выходным процессором.

Символически сказанное выглядит следующим образом: $\delta_h = \sum_o \delta_o w_{ho}$. Впрочем, это не совсем так: мы забыли функцию активации скрытого процессора; эта функция F' должна быть применена к небольшому изменению (приращению сигнала ошибки), прежде, чем процесс обратного распространения может быть продолжен.

3 Работа с обратным распространением

Применение обобщенного дельта правила включает, таким образом, две стадии. На первой стадии предъявляется входной сигнал χ и распространяется вперед через сеть, чтобы вычислить выходные значения y_o^p для каждого выходного процессора. Этот выход сравнивается с его желаемым значением d_o, приводя к сигналу δ_o^p ошибки для каждого выходного процессора.

Вторая стадия предполагает обратный проход через сеть, в течение которого сигнал ошибки передаётся каждому процессору в сети, и рассчитываются соответствующие изменения веса.

Корректировки веса с сигмоидальной функцией активации. Результаты предыдущего раздела могут быть подытожены в трёх уравнениях:

[1] «Жадный» алгоритм (англ. *Greedy algorithm*) — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. Известно, что если структура задачи задается матроидом, тогда применение жадного алгоритма выдаст глобальный оптимум.

ISSN 1512-1232

- Вес связи уточняется величиной, пропорциональной произведению сигнала δ ошибки на процессоре k, получающем входной и выходной сигнал процессора j, который посылает этот сигнал по данной связи:

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p . \tag{16}$$

- Если процессор является выходным процессором, сигнал ошибки даётся выражением

$$\delta_o^p = \left(d_o^p - y_o^p\right) \cdot \mathsf{F}'\left(s_o^p\right). \tag{17}$$

За функцию активации F можно принять «сигмоидальную» функцию, как она определена ниже:

$$y^p = \mathsf{F}\left(s^p\right) = \frac{1}{1+e^{-s^p}}. \tag{18}$$

В этом случае производная равна

$$\begin{aligned} \mathsf{F}'\left(s^p\right) &= \frac{\partial}{\partial s^p}\frac{1}{1+e^{-s^p}} = -\frac{1}{\left(1+e^{-s^p}\right)^2}\left(-e^{-s^p}\right) \\ &= \left(\frac{1}{1+e^{-s^p}}\right)^2 \cdot e^{-s^p} = \left(y^p\right)^2 \frac{1-y^p}{y^p} = y^p\left(1-y^p\right) \end{aligned} \tag{19}$$

из условия, чтобы сигнал ошибки для выходного процессора мог быть записан в следующем виде:

$$\delta_o^p = \left(d_o^p - y_o^p\right) y_o^p \left(1-y_o^p\right). \tag{20}$$

- Сигнал ошибки для некоторого скрытого процессора определяется рекурсивно в терминах сигналов ошибки процессоров, с которыми он непосредственно соединяется, и весов связей с ними. Для сигмоидальной функции активации:

$$\delta_h^p = \mathsf{F}'\left(s_h^p\right) \cdot \sum_{o=1}^{N_o} \delta_o^p w_{ho} = y_h^p\left(1-y_h^p\right) \cdot \sum_{o=1}^{N_o} \delta_o^p w_{ho} . \tag{21}$$

Скорость обучения и параметр момента. Процедура обучения требует, чтобы изменение в весе было пропорционально величине $\partial E^p / \partial w$. Истинный градиентный спуск требует, чтобы были предприняты бесконечно малые шаги. Константа пропорциональности характеризует скорость (темп) обучения. Для практических целей мы выбираем скорость обучения, которая, являясь по возможности большой, не ведёт к колебаниям. Один из способов избегнуть колебания в целом состоит в том, чтобы делать изменение в весе зависящим от прошлого изменения веса добавлением параметра момента (предотвращающего преждевременное прекращение процесса самообучения нейронной сети)

$$\Delta w_{jk}(t+1) = \gamma \cdot \delta_k^p \cdot y_j^p + \alpha \cdot \Delta w_{jk}(t), \tag{4.22}$$

ISSN 1512-1232

где t - порядковый номер изменения в весе, а α - параметр момента - константа, которая определяет эффект предыдущего изменения веса.

Роль параметра момента демонстрируется на рисунке 2. Когда никакой параметр момента не используется, требуется длительное время прежде, чем минимум будет достигнут с низкой скоростью обучения, тогда как для высоких скоростей обучения минимум никогда не будет достигнут из-за колебаний. При добавлении параметра момента минимум будет достигнут быстрее.

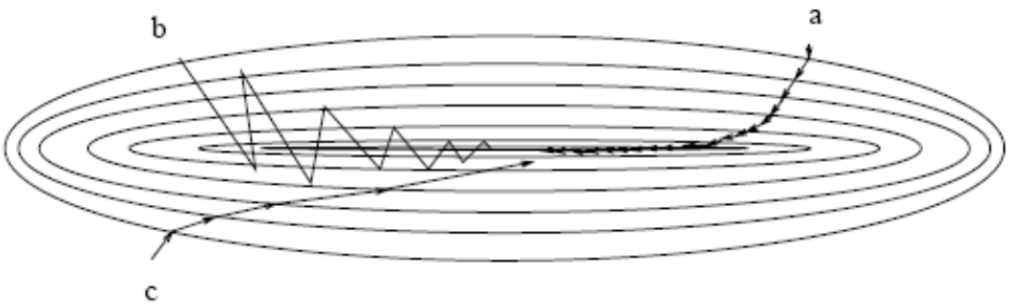


Рисунок 2. Спуск в весовом пространстве:
a) при малой скорости обучения;
b) при большой скорости обучения (с проявлением колебания);
c) при большой скорости обучения (с добавленным параметром момента).

Обучение с помощью образа. Хотя, теоретически, алгоритм обратного распространения осуществляет градиентный спуск на поверхности суммарной ошибки, если только веса корректируются после того, как был представлен полный набор обучающих образов, очень часто, почти всегда, правило обучения применяется к каждому образу отдельно, то есть, используется образ p, рассчитывается E^p, и приводятся в соответствие (адаптируются) веса $(p = 1, 2, \ldots, P)$.

В литературе существует эмпирическое указание на то, что это приводит к более быстрой сходимости. Однако следует проявлять осторожность к порядку, в котором должны представляться обучающие образы.

Например, при использовании одной и той же обучающей последовательности много раз сеть может сфокусироваться на первых нескольких образцах.

Эта проблема может быть преодолена использованием пермутационного, или перестановочного метода обучения.

4 Пример

Сеть прямого распространения (сеть с механизмом прогнозирования событий) может использоваться, чтобы аппроксимировать функцию обучающих образов.

Предположим, что мы имеем систему (например, химический процесс или финансовый рынок), относительно которой желательно знать некоторые характеристики.

Входной сигнал системы дается двумерным вектором χ , а выходной сигнал даётся одномерным вектором d. Мы хотим оценить зависимость $d = f(\chi)$ на 80 примерах $\{\chi^p, d^p\}$, изображенных на рисунке 3 (в верхнем левом углу).

ISSN 1512-1232

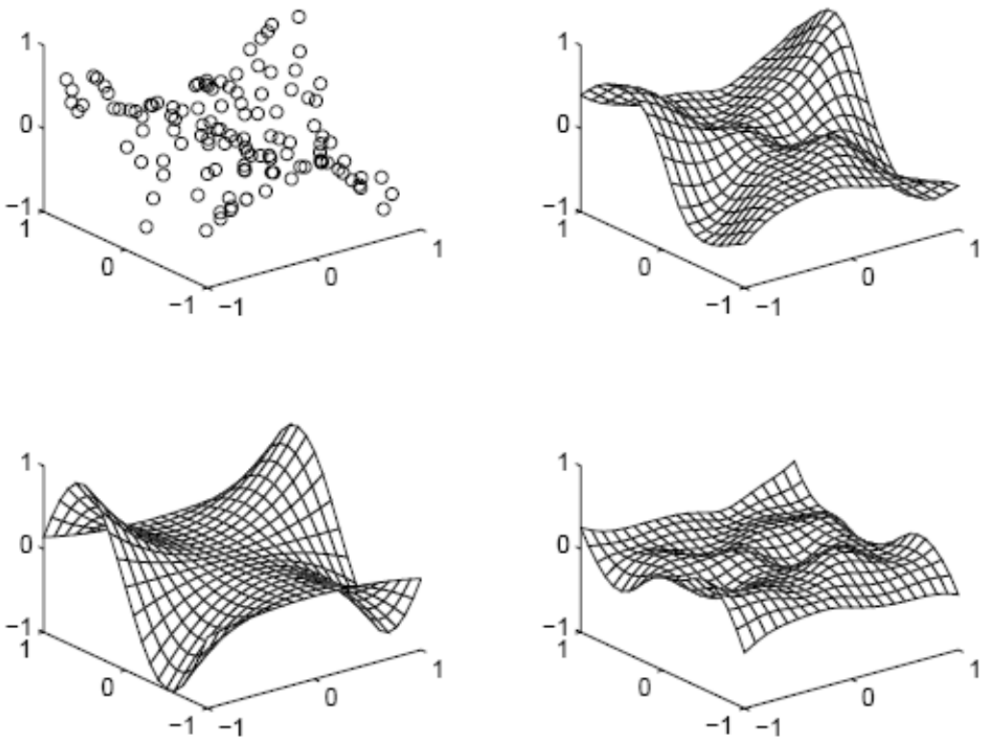


Рисунок 3. Пример аппроксимации функции сетью прямого распространения. Верхний левый угол: исходные обучающие образы; верхний правый угол: аппроксимация сетью; левый нижний угол: функция, которая сгенерирована обучающими образами; правый нижний угол: ошибка в аппроксимации.

Сеть прямого распространения была запрограммирована как устройство, имеющее: два входа, десять скрытых процессоров с сигмоидальной функцией активации и некоторый выходной процессор с линейной функцией активации.

Легко выяснить, как уравнение (20) должно быть адаптировано для линейной функции активации вместо сигмоидальной. Веса сети имеют малые начальные значения, а сеть обучена на 5 000 обучающих итерациях по обучающему правилу обратного распространения, описанному в предыдущем разделе.

Зависимость между χ и d, представленная сетью, показана на рисунке 3 (в верхнем правом углу), в то время как функция, которую сгенерировала обучающая выборки, дается на рисунке 3 (в левом нижнем углу). Ошибка в аппроксимации изображена на рисунке 3 (в правом нижнем углу).

Мы видим, что ошибка выше на границах той области, в пределах которой были сгенерированы обучающие образы. Сеть значительно лучше при интерполяции, чем при экстраполяции.

5 Другие функции активации

Хотя сигмоидальные функции весьма часто используются как функции активации, другие функции также могут привлечь внимание. В некоторых случаях это ведет к формуле, которая известна из традиционных теорий аппроксимации (приближения) функций.

ISSN 1512-1232

Например, из Фурье-анализа известно, что любая периодическая функция (при определённых условиях) может быть записана как бесконечная сумма синусных и косинусных членов (т.е. как ряд Фурье):

$$f(x) = \sum_{n=0}^{\infty} \left(a_n \cos nx + b_n \sin nx \right). \tag{23}$$

Мы можем переписать это и в альтернативной форме - как суммирование только синусных членов

$$f(x) = a_0 + \sum_{n=1}^{\infty} c_n \sin(nx + \theta_n), \tag{24}$$

где $c_n = \sqrt{a_n^2 + b_n^2}$ и $\theta_n = \arctan(b_n / a_n)$. Это может рассматриваться как сеть прямого распространения с единственным входным процессором для x; с единственным выходным процессором для $f(x)$ и скрытыми процессорами с функцией активации $\mathsf{F} = \sin(s)$. Коэффициент (фактор) a_0 соответствует смещению выходного процессора, коэффициенты c_n соответствуют весам со скрытого процессора к выходному процессору; фазовый коэффициент θ_n соответствует элементу (величине) смещения скрытых процессоров, а коэффициент n соответствует весам между входным и скрытым слоями.

Основное различие между подходом Фурье и подходом обратного распространения то, что в Фурье подходе «веса» между входом и скрытыми процессорами (т.е. коэффициенты n) являются фиксированными целыми числами, которые аналитически определены, тогда как в подходе обратного распространения эти веса могут принять любое значение и, обычно, познаются с помощью эвристического обучения.

Чтобы иллюстрировать использование других функций активации, мы обучили сеть прямого распространения (т.е. сеть с механизмом прогнозирования событий) с одним выходным процессором, четырьмя скрытыми процессорами, и одним входом с десятью образами (эталонами), извлечёнными из функции $f(x) = \sin(2x) \cdot \sin(x)$. Результат изображен на Рисунке 4. Та же самая функция (хотя с другими тренирующими точками) обучена сетью из восьми (!) сигмоидальных скрытых процессоров (см. Рисунок: 5). Данные рисунки убедительно свидетельствует, что лучший результат, относящийся к первому случаю, вполне оправдывает использование всего имеющегося под рукой знания относительно задачи.

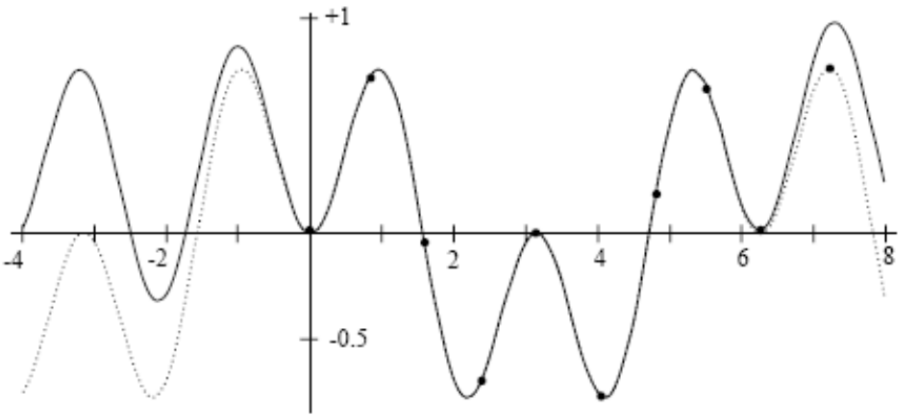


Рисунок 4. периодическая функция $f(x) = \sin(2x)\sin(x)$, аппроксимированная в условиях синусной функции активации. (Адаптировано из работы Dastani, 1991 [10]).

ISSN 1512-1232

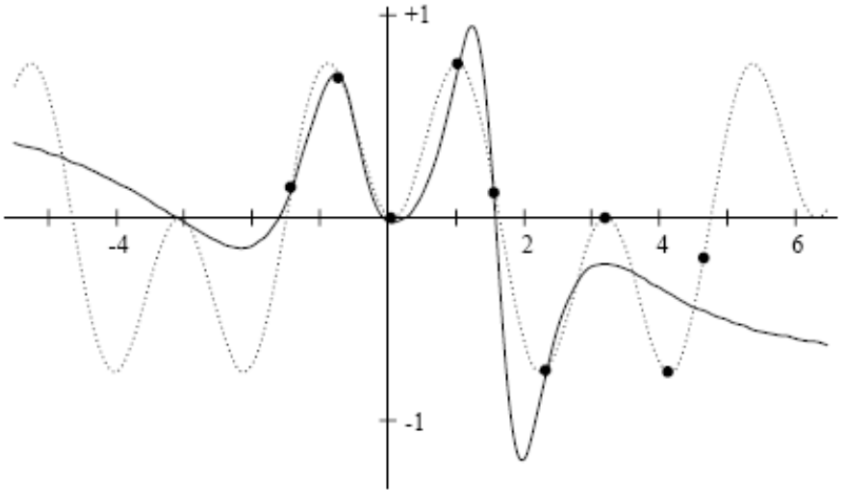


Рисунок 5. периодическая функция $f(x) = \sin(2x)\sin(x)$, аппроксимированная в условиях сигмоидальной функции активации. (Адаптировано из работы Dastani, 1991 [10]).

6 Недостатки обратного распространения

Несмотря на очевидный успех обучающего алгоритма обратного распространения, есть некоторые аспекты, которые мешают этому алгоритму гарантированно приносить универсальную пользу.

Наиболее неприятный аспект - продолжительность процесса обучения. Это может быть результатом неоптимальности скорости обучения и параметра момента. Многие усовершенствованные алгоритмы, основанные на обучении методом обратного распространения, содержат в себе тот или иной оптимизирующий метод для адаптации этой скорости обучения, как будет выяснено в следующем разделе.

Вообще говоря, неудачи обучения напрямую обусловлены двумя источниками, а именно: сетевым параличом и локальными минимумами.

Сетевой паралич. Как сетевая обучающаяся последовательность, веса могут быть откорректированы и приведены к очень большим значениям.Общий входной сигнал скрытого процессора или выходного процессора может поэтому достигнуть очень высоких (или положительных, или отрицательных) значений, и из-за сигмоидальной функции активации процессор будет иметь очень близкую к нулю или очень близкую к единице активацию.

Как ясно из уравнений (20) и (21), корректировки веса, которые пропорциональны величинам $y_k^p\left(1-y_k^p\right)$, будут близки к нулю, и процесс обучения может прийти к фактической остановке.

Локальные минимумы. Поверхность ошибки сложной сети полна холмов и долин. Из-за градиентного спуска, сеть может быть заманена в ловушку в локальном минимуме, когда поблизости есть намного более глубокий минимум.

Вероятностные методы могут помочь избегнуть этой западни, но они имеют тенденцию быть медленными.

Другая предложенная возможность состоит в том, чтобы увеличить число скрытых процессоров. Хотя в этом случае и придется работать при более высокой размерности

ISSN 1512-1232

пространства ошибки, тем не менее, шанс заманить сеть в ловушку становится меньшим. Представляется, что есть некоторый верхний предел количества скрытых процессоров, который, когда он превышен, снова приводит к системе, заманиваемой в ловушку в локальных минимумах.

7 Улучшенные алгоритмы

Многие исследователи разработали усовершенствования и расширения основного алгоритма обратного распространения, описанного выше. Однако пока было бы слишком рано давать им исчерпывающую оценку: некоторые из этих методов, могут оказаться фундаментальными, другие же могут просто исчезнуть. Несколько методов обсуждаются в этом разделе.

Возможно, наиболее очевидное усовершенствование сводится к тому, чтобы заменить довольно примитивный метод наискорейшего спуска методом минимизации с заданием направления, например, минимизацией сопряженным градиентом.

Обратите внимание, что минимизация вдоль направления $\mathbf{u}$ приводит функцию f в место, где его градиент перпендикулярен $\mathbf{u}$ (иначе минимизация вдоль $\mathbf{u}$ не закончена). Взамен отслеживания градиента на каждом шагу, строится (создаётся) набор n направлений, все из которых сопряжены друг с другом так, что минимизация по одному из этих направлений $\mathbf{u}_j$ не портит минимизацию по другому из более ранних направлений $\mathbf{u}_i$, то есть, направления не смешиваются.

Таким образом, одна минимизация в направлении $\mathbf{u}_i$ бывает достаточной при условии что n минимизаций в системе с n степенями свободы приводят эту систему к минимуму (если система квадратичная).

К этому и сводится отличие от градиентного спуска, который непосредственно минимизирует в направлении (по линии) наиболее крутого спуска (Press, Flannery, Teukolsky, & Vetterling, 1986 [11]).

Предположим, что функция, которая будет минимизирована, аппроксимирована своим рядом Тэйлора

$$f(\boldsymbol{\chi}) = f(\mathbf{p}) + \sum_i \left.\frac{\partial f}{\partial x_i}\right|_{\mathbf{p}} x_i + \tfrac{1}{2}\sum_{i,j} \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{p}} x_i x_j + \cdots \approx \tfrac{1}{2}\boldsymbol{\chi}^T \mathbf{A}\boldsymbol{\chi} - \mathbf{b}^T\boldsymbol{\chi} + c$$

где T обозначает транспонирование, и

$$c \equiv f(\mathbf{p}) \quad \mathbf{b} \equiv -\nabla f\big|_{\mathbf{p}} \quad [\mathbf{A}]_{ij} \equiv \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{p}}. \tag{25}$$

$\mathbf{A}$ является симметричной положительно определенной матрицей размера $n \times n$, Гессианом преобразования f в $\mathbf{p}$.

Матрицу $\mathbf{A}$ называют *положительно определенной,* если $\forall y \neq 0$:

$$\mathbf{y}^T \mathbf{A}\mathbf{y} > 0. \tag{26}$$

Градиент f, задаваемый выражением

$$\nabla f = \mathbf{A}\boldsymbol{\chi} - \mathbf{b}, \tag{27}$$

ISSN 1512-1232

таков, что изменение $\boldsymbol{\chi}$ приводит к изменению и самого градиента:

$$\delta(\nabla f) = \mathbf{A}(\delta\boldsymbol{\chi}). \tag{28}$$

Теперь предположим, что f была минимизирована по направлению $\mathbf{u}_i$ к точке, где градиент $-\mathbf{g}_{i+1}$ функции f перпендикулярен к $\mathbf{u}_i$, то есть,

$$\mathbf{u}_i^T \mathbf{g}_{i+1} = 0, \tag{29}$$

и разыскивается новое направление $\mathbf{u}_{i+1}$. Для уверенности, что движение вдоль $\mathbf{u}_{i+1}$ не портит минимизации вдоль $\mathbf{u}_i$, потребуем, чтобы градиент f оставался перпендикулярным $\mathbf{u}_i$, то есть,

$$\mathbf{u}_i^T \mathbf{g}_{i+2} = 0; \tag{30}$$

в противном случае мы еще раз должны были бы осуществить минимизацию в направлении, которое имеет компонента $\mathbf{u}_i$. Объединяя (29) и (30), приходим к следующему:

$$0 = \mathbf{u}_i^T (\mathbf{g}_{i+1} - \mathbf{g}_{i+2}) = \mathbf{u}_i^T \delta(\nabla f) = \mathbf{u}_i^T \mathbf{A}\mathbf{u}_{\mathbf{i+1}}. \tag{31}$$

Когда для двух векторов $\mathbf{u}_i$ и $\mathbf{u}_{i+1}$ выполняется уравнение (31), то говорят, что эти векторы являются *сопряженными*.

Теперь, начиная с некоторой точки $\mathbf{p}_0$, первое направление минимизация $\mathbf{u}_0$ берётся равным $\mathbf{g}_0 = -\nabla f(\mathbf{p}_0)$ и заканчивается в новой точкой $\mathbf{p}_1$. Для $i \geq 0$ вычисляется направление

$$\mathbf{u}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{u}_i, \tag{32}$$

где γ_i выбирается так, чтобы выполнить условие $\mathbf{u}_i^T \mathbf{A}\mathbf{u}_{\mathbf{i+1}} = 0$ и сделать последующие градиенты перпендикулярными, то есть,

$$\gamma_i = \frac{\mathbf{g}_{i+1}^T \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i^T \cdot \mathbf{g}_i}, \text{ где } \mathbf{g}_k = -\nabla f\big|_{\mathbf{p}_k} \text{ для всех } k \geq 0. \tag{33}$$

Затем, вычисляют $\mathbf{p}_{i+2} = \mathbf{p}_{i+1} + \lambda_{i+1}\mathbf{u}_{i+1}$, где λ_{i+1} выбрано так, чтобы минимизировать $f(\mathbf{p}_{i+2})$. Это далеко не тривиальная проблема (см. (Press и другие., 1986).) Однако существуют линейные методы минимизации со сверхлинейной сходимостью.

Метод, как известно, сходится линейно если $E_{i+1} = cE_i$ где $c < 1$. Методы, которые сходятся с более высокой степенью, то есть, $E_{i+1} = c(E_i)^m$, где $m > 1$, называют *сверхлинейными* (или *суперлинейными*).

Можно показать, что все $\mathbf{u}$, созданные таким образом, являются взаимно сопряженными (например, см. (Stoer и Bulirsch, 1980 [12])). Процесс, описанный выше, известен как метод *Fletcher-Reeves*, но есть много вариантов, которые работают более или менее в том же самом духе (Hestenes и Stiefel, 1952 [13]; Polak, 1971 [14]; Powell, 1977 [15]).

Хотя для квадратичной системы с n степенями свободы необходимыми являются только n итераций, вследствие того, что мы не минимизируем квадратичные системы, так

ISSN 1512-1232

же как и результат ошибок округления, n направлений должны быть отслежены несколько раз (см. рисунок 6).

Powell-ом введены некоторые усовершенствования, чтобы исправить поведение в неквадратичных системах. Результирующая сложность составляет $O(n)$, что значительно лучше, чем линейная сходимость метода наискорейшего спуска.

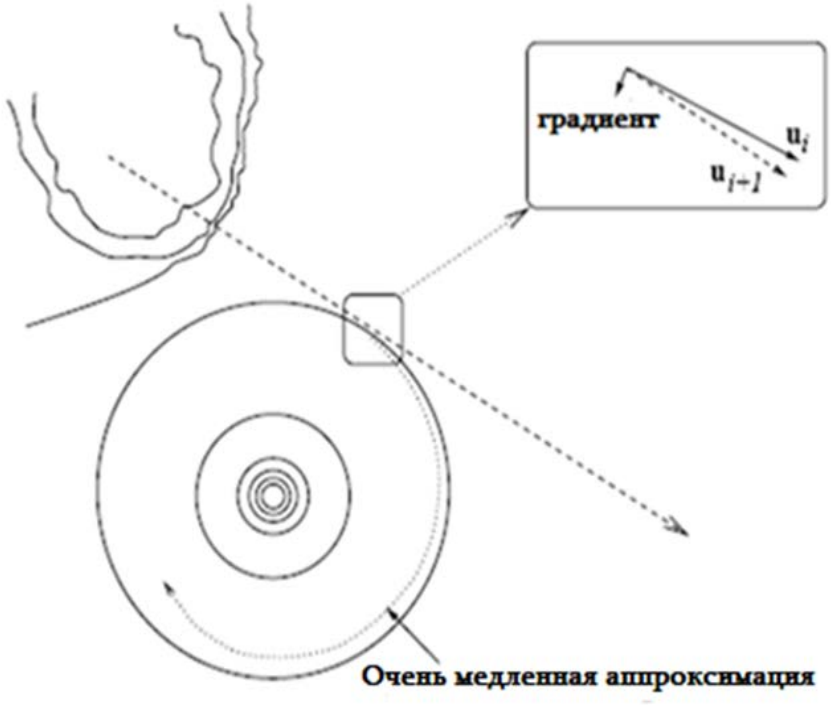


Рисунок 6. Медленное уменьшение с сопряженным градиентом в неквадратичных системах. Холмы слева очень круты, приводя к большому вектору поиска $\mathbf{u}_i$. Когда квадратичная часть введена, новое направление поиска создаётся из предыдущего направления и градиента, приводя к минимизации движением по спирали. Эта проблема может быть преодолена, обнаруживая подобную минимизацию спирального характера и перезапуская алгоритм с $\mathbf{u}_0 = -\nabla f$.

Некоторые усовершенствования метода обратного распространении были представлены как основывающиеся на независимом адаптивном параметре скорости обучения для каждого веса.

Авторами Van den Boomgaard и Smeulders (Boomgaard и Smeulders, 1989 [16]) показано, что для сети прямого распространения без скрытых процессоров инкрементальная процедура для нахождения оптимальной матрицы веса W нуждается в настройке весов согласно соотношению

$$\Delta \boldsymbol{W}(t+1) = \gamma(t+1)\big(\mathbf{d}(t+1) - \boldsymbol{W}(t)\boldsymbol{\chi}(t+1)\big)\boldsymbol{\chi}(t+1), \qquad (34)$$

в котором γ - не константа, а изменяющаяся (варьируемая) матрица размера $(N_i+1)(N_i+1)$, зависящая от входного вектора. Используя *априорное* знание о входном сигнале, требования памяти для γ могут быть уменьшены.

ISSN 1512-1232

Silva и Almeida (Silva & Almeida, 1990 [17]) также показывают преимущества независимого размера шага для каждого веса в сети. В их алгоритме скорость обучения адаптируется после каждого обучающего образа:

$$\gamma_{jk}(t+1)=\begin{cases} u\gamma_{jk}(t) & \text{если } \dfrac{\partial E(t+1)}{\partial w_{jk}} \text{ и } \dfrac{\partial E(t)}{\partial w_{jk}} \text{ имеют одинаковые признаки} \\ d\gamma_{jk}(t) & \text{если } \dfrac{\partial E(t+1)}{\partial w_{jk}} \text{ и } \dfrac{\partial E(t)}{\partial w_{jk}} \text{ имеют одинаковые признаки} \end{cases}, \quad (35)$$

где u и d - положительные константы со значениями, которые немного выше и ниже единицы, соответственно. Смысл состоит в том, чтобы уменьшить скорость обучения в случае колебаний.

8 Насколько хороши многослойные сети прямого распространения?

Из примера, показанного на рисунке 3, ясно, что аппроксимация, осуществляемая сетью, не совершенна. На завершение аппроксимации с ошибкой влияют:

1. Алгоритм обучения и количество итераций. Это определяет, насколько хорошо ошибка на обучающей выборке минимизирована.

2. Число обучающих образов. Это определяет, насколько хорошо образы обучения представляют фактическую функцию.

3. Число скрытых процессоров. Это определяет «вычислительную мощность» сети. Для «гладких» функций необходимо только небольшое количество скрытых процессоров, для функции со значительным колебанием, будет необходимо большее число скрытых процессоров.

В предыдущих разделах мы обсуждали правила обучения, такие как обратное распространение и другие алгоритмы обучения, базирующиеся на градиенте, а также проблему достижения минимальной ошибки. В этом же разделе мы в частности обращаемся к влиянию количества обучающих образов и влиянию числа скрытых процессоров.

Сначала мы должны определить адекватную меру ошибки. Все алгоритмы обучения нейронной сети стараются минимизировать ошибку выборки обучающих образов, которые доступны для тренинга сети. Средняя ошибка на обучающий образ определяется как *коэффициент ошибки обучения*:

$$E_{\text{learning}}=\frac{1}{P_{\text{learning}}}\sum_{p=1}^{P_{\text{learning}}}E^p,$$

в котором E^p является разницей между желаемым значением выхода и фактическим выходом сети для обучающих образцов:

$$E^p=\tfrac{1}{2}\sum_{o=1}^{N_o}\left(d_o^p-y_o^p\right)^2.$$

Это именно та ошибка, которая измеряется в продолжение процесса обучения. Очевидно, что фактическая ошибка сети будет отличаться от ошибки из-за выбора расположения обучающих образов. Разность между желаемым значением выхода и

ISSN 1512-1232

фактическим выходом сети должна быть проинтегрирована по всей области значений входа, чтобы дать более реалистическую меру ошибки. Этот интеграл может быть оценен, если у нас имеется большая выборка образов - тестирующая выборка. Теперь определяем коэффициент ошибки тестирования как среднюю ошибку тестирующей выборки:

$$E_{\text{test}} = \frac{1}{P_{\text{test}}} \sum_{p=1}^{P_{\text{test}}} E^p .$$

В следующих разделах мы увидим, как эти меры ошибки зависят от размера обучающей выборки и количества скрытых процессоров.

8.1 Влияние числа обучающих образов

Здесь как пример используется простая задача: функция $y = f(x)$ должна быть аппроксимирована нейронной сетью прямого распространения. Нейронная сеть имеет один вход, 5 скрытых процессоров с сигмоидальной функцией активации и линейный процессор выхода. Предположим, что мы имеем только небольшое количество обучающих образов (например, 4) и сети обучены этими образами. Обучение завершается, когда ошибка больше не уменьшается.

Оригинальная (желаемая) функция показана на рисунке 7A как пунктирная линия. Обучающие образы и аппроксимация сети показаны на том же самом рисунке.

Мы видим, что в этом случае ошибка обучения (Elearning) является небольшой (выход сети отлично работает благодаря обучающим образам), однако ошибка тестирования (Etest) сети является большой. Аппроксимация, достигаемая на 20 обучающих образах, показана на рисунке 7B. Ошибка обучения сети (Elearning) больше чем в случае 5 обучающих образов, но ошибка тестирования сети (Etest) меньше.

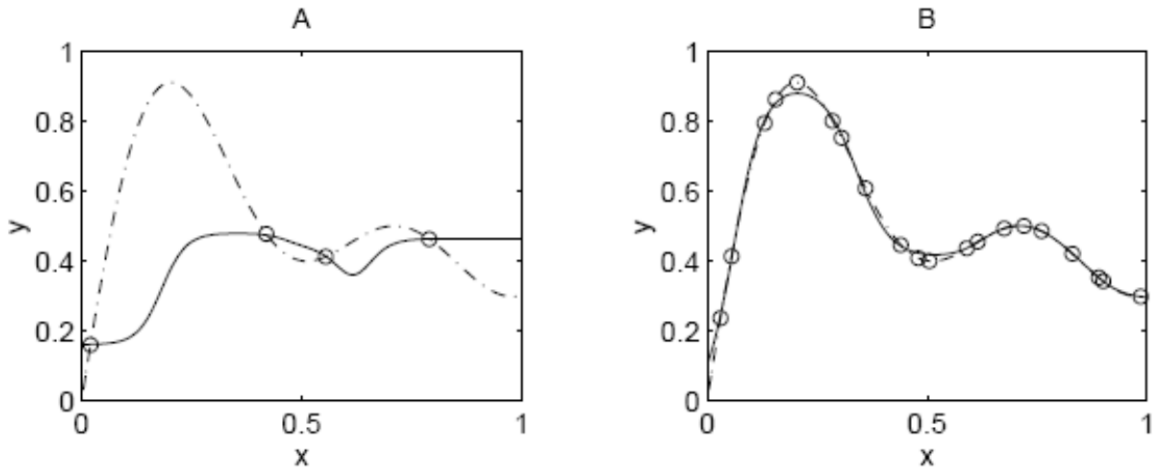


Рисунок 7. Влияние обучающей выборки на общую характеристику. Пунктирная линия дает желаемую функцию, обучающие образы изображены кружочками, а аппроксимацию сетью показывает сплошная линия. Используются 5 скрытых процессоров: a) 4 обучающие образы; b) 20 обучающих образов.

Этот эксперимент был проведён и с другими размерами обучающих выборок, причём для каждого размера обучающей выборки он повторялся 10 раз.

ISSN 1512-1232

Средние значения коэффициента (частоты) ошибок обучения и коэффициента (частоты) ошибок тестирования, как функции размера обучающей выборки, даются на рисунке 8.

Обратите внимание, что ошибка обучения увеличивается с ростом размера обучающей выборки, а ошибка тестирования уменьшается с увеличением размера обучающей выборки.

Следовательно, никакая низкая ошибка обучения на (малой) обучающей выборке не может стать гарантией качественной работы сети.

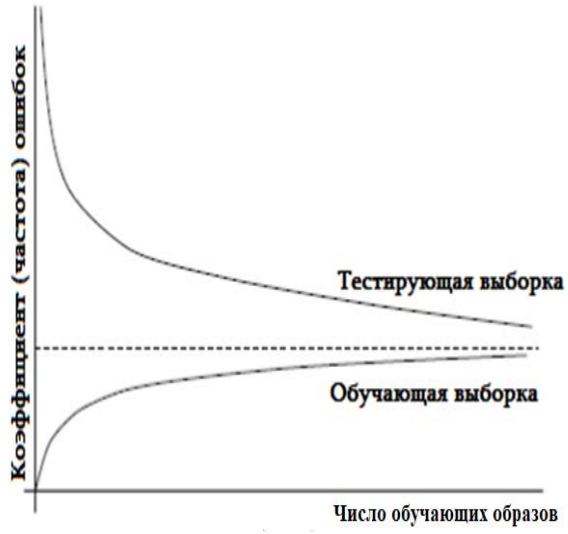


Рисунок 8. Влияние объёма обучающей выборки на коэффициент (частоту появления) ошибки. Среднее значение коэффициента (частоты появления) ошибок тестирования и обучения как функция числа обучающих образов.

С увеличением количества обучающих образов два коэффициента ошибки сходятся к одному и тому же значению. Это значение зависит от репрезентативной способности сети, оцениваемой выдачей оптимальных весов и качественностью аппроксимации, равно как и от количества скрытых процессоров и функции активации.

Если коэффициент ошибки обучения не сходится к коэффициенту ошибки тестирования, то это означает, что процедура обучения не нашла глобальный минимум.

8.2 Влияние количества скрытых процессоров

Используется та же самая функция, что и в предыдущем разделе, но теперь число скрытых процессоров отличается.

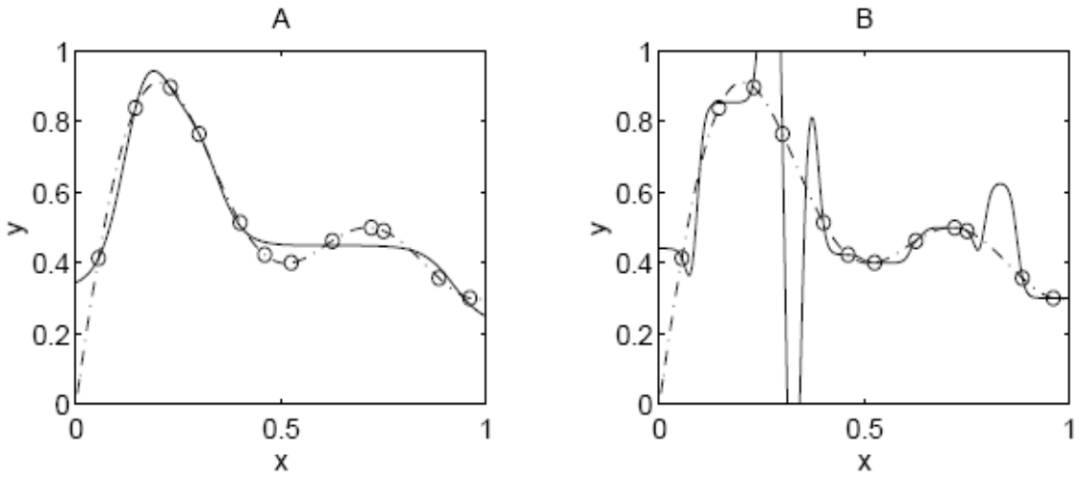
Оригинальная (т.е. желаемая) функция, обучающие образы и аппроксимация сети показана на рисунке 9A для 5 скрытых процессоров и на рисунке 9B для 20 скрытых процессоров.

Влияние количества скрытых процессоров, обнаруживаемое на рисунке 9B, называют, *переобучением*.

Сеть в точности соответствует обучающим образам, но из-за большого количества скрытых процессоров функция, которая фактически представлена сетью, существенно отклонена от желаемой.

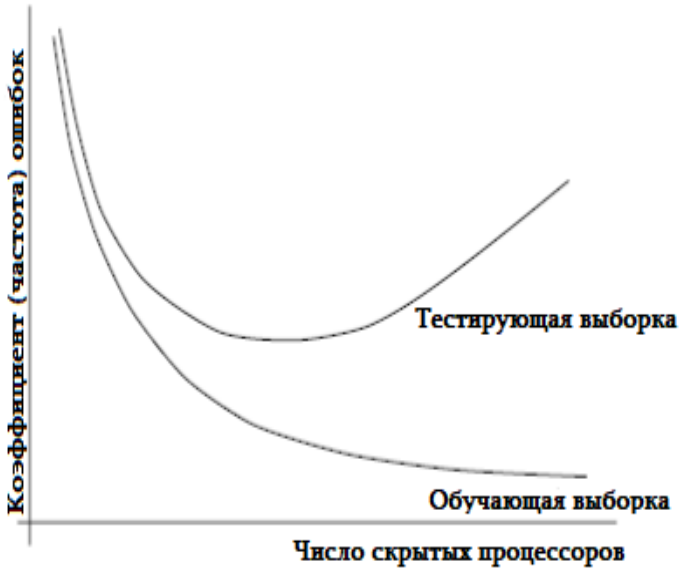
ISSN 1512-1232

В частности, в случае обучающих образов, которые содержат определённое количество шума (всегда имеющегося в реальных данных), сеть будет «соответствовать шуму» обучающих образов, вместо того, чтобы осуществить гладкую аппроксимацию.



]Рисунок 9. Влияние количества скрытых процессоров на работу сети. Пунктирная линия дает желаемую функцию, кружочки обозначают обучающие образы, а сплошная линия дает аппроксимацию сетью. Используются 12 обучающих образов: a) 5 скрытых процессоров; b) 20 скрытых процессоров.

Эти примеры показывают, что большое количество скрытых процессоров ведет к малой ошибке на обучающей выборке, но не обязательно ведет к малой ошибке на тестирующей выборке. Добавление скрытых процессоров будет всегда вести к уменьшению ошибки обучения (Elearning). Однако, добавление скрытых процессоров будет сначала вести к уменьшению ошибки тестирования (Etest), но затем приведёт к увеличению ошибки тестирования (Etest). Этот эффект называют эффектом образования резких максимумов (англ. peaking). Среднее значение коэффициента ошибки обучения и коэффициента ошибки тестирования как функций размера обучающей выборки даются на рисунке 10.



ISSN 1512-1232

Рисунок 10. Среднее значение коэффициента (частоты появления) ошибок обучения и тестирования как функция размера обучающей выборки (то есть числа скрытых процессоров).

9 Применения

Обратное распространение было применено к широкому кругу разнообразных исследовательских приложений. Sejnowski и Rosenberg (1986) (Sejnowski и Rosenberg, 1986 [18]) добились значительного успеха с NETtalk - системой, которая конвертирует (преобразовывает) печатный английский текст в весьма разборчивую речь.

Сеть с прямым распространением и с одним слоем скрытых процессоров была описана Gorman и Sejnowski (1988) (Gorman и Sejnowski, 1988 [19]) как классифицирующая машина для звуковых сигналов.

Другое приложение многослойной сети с прямым распространением и обучающим алгоритмом обратного распространения может состоять в изучении неизвестной функции между сигналами входа и выхода при предъявления примеров. Полагают, что сеть способна к обобщению даже в том случае, когда входные значения, которые не представлялись при обучении на образах, всё-таки приводят к правильным выходным значениям. Пример - работа Josin (Josin, 1988 [20]). Этот автор использовал двуслойную сеть с прямой передачей и обратным распространением обучения, чтобы реализовать инверсное кинематическое преобразование, которое необходимо контроллеру руки робота.

Настоящая работа подготовлена к намечающейся в Грузинском техническом университете (Тбилиси, Грузия) аккредитации магистерской программы «Информатика» и в некоторой мере представляет собой адаптацию материалов интересного исследования Ben Kröse и Patrick van der Smagt (1996) (Ben Kröse and Patrick van der Smagt, 1996 [21]).

Список цитированной литературы:

[1] Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. The MIT Press.

[2] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by backpropagating errors. *Nature, 323*, 533-536.

[3] Werbos, P. W. (1990). A menu for designs of reinforcement learning over time. In W. T. M. III, R. S. Sutton, & P. J. Werbos (Eds.), *Neural Networks for Control*. MIT Press/Bradford.

[4] Parker, D. B. (1985). *Learning-Logic* (Tech. Rep. Nos. TR{47). Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

[5] Cun, Y. L. (1985). Une procédure d'apprentissage pour réseau a seuil asymétrique. Proceedings of Cognitiva, 85, 599-604.

[6] Hornik, K., Stinchcombe, M., &White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359-366.

[7] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. Neural Networks, 2(3), 193-192.

[8] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems, 2(4), 303-314.

[9] Hartman, E. J., Keeler, J. D., & Kowalski, J. M. (1990). Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation, 2*(2), 210-215.

[10] Dastani, M. M. (1991). Functie-Benadering met Feed-Forward Netwerken. Unpublished master's thesis, Universiteit van Amsterdam, Faculteit Computer Systemen.

ISSN 1512-1232

[11] Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1986). *Numerical Recipes: The Art of Scientific Computing*. Cambridge: Cambridge University Press.

[12] Stoer, J., & Bulirsch, R. (1980). *Introduction to Numerical Analysis*. New York-Heidelberg-Berlin: Springer-Verlag.

[13] Hestenes, M. R., & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Nat. Bur. Standards J. Res., 49*, 409-436.

[14] Polak, E. (1971). *Computational Methods in Optimization*. New York: Academic Press.

[15] Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming, 12*, 241-254.

[16] Boomgaard, R. van den, & Smeulders, A. (1989). Self-learning image processing using a-priori knowledge of spatial patterns. In T. Kanade, F. C. A. Groen, & L. O. Hertzberger (Eds.), Proceedings of the I.A.S. Conference (p. 305-314). Elsevier Science Publishers.

[17] Silva, F. M., & Almeida, L. B. (1990). Speeding up backpropagation. In R. Eckmiller (Ed.), *Advanced Neural Computers* (pp. 151-160). North-Holland.

[18] Sejnowski, T. J., & Rosenberg, C. R. (1986). *NETtalk: A Parallel Network that Learns to Read Aloud* (Tech. Rep. Nos. JHU/EECS-86/01). The John Hopkins University Electrical Engineering and Computer Science Department.

[19] Gorman, R. P., & Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. Neural Networks, 1(1), 75-89.

[20] Josin, G. (1988). Neural-space generalization of a topological transformation. *Biological Cybernetics, 59*, 283-290.

[21] Kröse, B. J. A., & Smagt, P. van der (1996). *An introduction to Neural Networks*, Eighth edition, Amsterdam (Netherlands): University of Amsterdam, Wessling (Germany): German Aerospace Research Establishment, 135 pages.

Article received: 2018-04-08