Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Денис Першин

ОБЗОР НЕКОТОРЫХ ВИДОВ НЕЙРОННЫХ СЕТЕЙ

Препринт
63

Новосибирск 2000

В обзоре в научно-популярной форме рассмотрены основные виды нейронных сетей, алгоритмов обучения и создания сетей с оптимальной топологией.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

Denis Perchine

THE REVIEW OF SOME TYPES OF NEURAL NETWORKS

Preprint
63

Novosibirsk 2000

Some major type of neural networks, learning and net creation algorithms are reviewed in the article.

1. ВВЕДЕНИЕ

С давних времен человек задумывался о том, как он устроен. Ученые в своем рвении изучить человека достигли многого, но все же самой загадочной частью человеческого организма остается головной мозг.

Как человек думает? Почему он принимает такие решения, а не другие? Что за процессы происходят в его голове, когда он решает сложнейшие задачи современной науки? На все это вопросы до сих пор нет окончательного ответа.

Биологи стали разрабатывать модели головного мозга задолго до появления вычислительных машин. Стоит однако заметить, что исследование поведения этих моделей было затруднено из-за отсутствия вычислительных мощностей. Сейчас это не является такой сложной проблемой, как 20–30 лет назад.

Из сугубо биологической, задача моделирования головного мозга человека превратилась в прикладную задачу кибернетики.

Так возник раздел кибернетики, исследующий системы исскуственного интеллекта. К ее подразделу можно отнести нейронные сети.

На данный момент нейронные сети нельзя назвать хорошо изученными. Ученым известны в основном некоторые частные результаты, получаемые эмпирическим путем.

Данная работа является обзором, освещающим некоторые аспекты нейронных сетей.

2. НЕЙРОННЫЕ СЕТИ

2.1. Общие понятия

Нейронные сети (НС) представляют собой обобщенные модели головного мозга произвольной сложности.

Обычно такая модель исходит из предположения о том, что головной мозг человека состоит из *нейронов*, связанных между собой *синапсами* (связующими элементами). В теории нейронных сетей нейроны принято называть *узлами*, а синапсы — *связями*.

Также имеет смысл определить понятие *слоя*. *Слой* — это набор узлов, сгруппированных по какому-либо признаку. *Входные слои* — это набор узлов, которые получают входную информацию. Семантически они аналогичны нервным окончаниям, посредством которых человек чувствует, слышит, видит и т. д. *Выходные слои* — это набор узлов, ко-

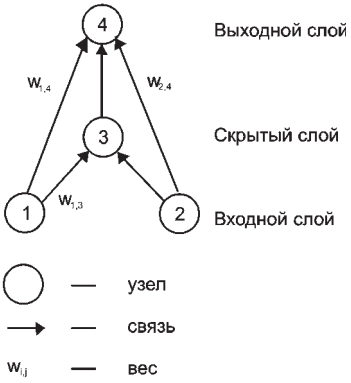

Рис. 1. Нейронная сеть

Рис. 2. Черный ящик

торые выдают выходную информацию. Их можно сравнить с нервными окончаниями, которые подают сигналы мышечным тканям человека. *Скрытые слои* — это слои, которые обычно находятся между входными и выходными и занимаются преобразованием информации. Их можно ассоциировать с головным и спинным мозгом (рис 1)

Если абстрагироваться от внутреннего представления нейронных сетей, то имеет смысл их рассматривать, как своеобразный «черный ящик», у которого есть некоторое количество входов и некоторое количество выходов (рис. 2). Информация, подаваемая на вход НС, преобразуется внутри «черного ящика» и на выходах представляет собой выходную информацию. Внутренние параметры самого ящика принято называть *весами*. Сами по себе веса являются параметрами связей между узлами. В дальнейшем они будут обозначаться w_{ij}, где j — номер узла, из которого выходит связь, а i — номер узла, к которому она

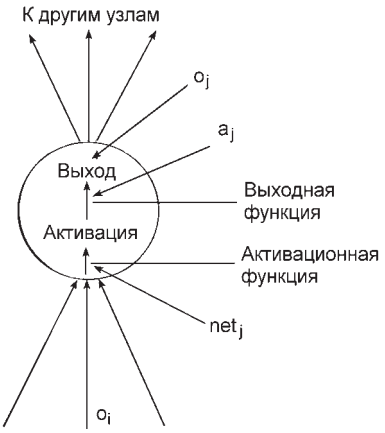

Рис. 3. Схематическое изображение узла

приходит. Выходное значение узла i будет обозначаться o_i, а net_j будет обозначать *чистое входное значение* узла, получаемое взвешенным суммированием значений, приходящих на вход узла с весами, соответствующими связям, по которым эти значения были получены. Или:

$$net_j = \sum_i w_{ij} o_i.$$

Информация поступает на вход узла, суммируется с весами и подвергается преобразованию при помощи *активационной функции* (рис. 3). *Активационная функция* (АФ) — это функция, которая вычисляет *активационное значение* из чистого входного значения и предыдущего активационного значения. Общая формула выглядит так:

$$a_j(t+1) = f_{act}(net_j(t+1), a_j(t)),$$

где $a_i(t)$ — предыдущее активационное значение узла j, f_{act} — активационная функция.

Затем активационное значение попадает на вход *выходной функции*

$$o_i = f_{out}(a_i), \tag{1}$$

преобразующей информацию, поступившую с АФ, в информацию, которая будет выдана на выходы узла. Стоит заметить, что выходная

функция в большинстве НС выглядит как $f_{out}(a_i) = a_i$. Также часто используется выходная функция вида

$$f_{out}(a_i) = \begin{cases} 0, & a_i < 0, \\ 1, & a_i > 1, \\ a_i & \text{иначе.} \end{cases} \quad (2)$$

Как правило разделения на активационную и выходную функцию не делается и они называются *передаточной функцией*.

В абстрактной модели НС каждый узел можно соединить с любым другим. Но обычно для упрощения расчетов предполагается, что ни один узел не может быть связан с входными узлами (но входной узел может быть связан с другими узлами), и ни один выходной узел не может быть связан ни с каким другим узлом. Кроме того, вводится понятие *порядка слоев* и задается дополнительное условие, — что связи могут существовать только между двумя соседними слоями и только в направлении от меньшего к большему. Если все эти условия соблюдены, то получится *НС прямого распространения*. На рис. 1 изображен пример НС прямого распространения.

Топологией НС называется набор параметров, определяющих структуру НС и представляющих множество узлов, разделенных на слои, из которых выделяются входные и выходные узлы, активационные и выходные функции для каждого узла и множество связей между узлами.

2.2. Обучение нейронных сетей

После создания сети веса не определены, поэтому перед тем, как производить какие-либо действия, необходимо придать им некоторые первоначальные значения. Обычно эти значения случайны и равномерно распределены на отрезке $[0; 1]$, но некоторые виды сетей позволяют использовать другие алгоритмы для достижения заданных целей. Об этом будет упомянуто при рассмотрении конкретных реализаций.

Сама по себе, только что созданная сеть, с большой вероятностью, ничего разумного на выходе не даст, вследствие того, что ее весам были приданы случайные значения. Для того чтобы получить разумный результат, необходимо обучить НС.

Суть обучения состоит в подборе весов НС таким образом, чтобы ошибка была минимальной. Обучение происходит при помощи *обучающего алгоритма*, который позволяет подобрать веса так, чтобы достичь

минимума ошибки. Различные алгоритмы обучения будут рассмотрены ниже.

Для обучения НС необходимо иметь набор данных. Этот набор разбивается на три части: *обучающую*, *проверочную* и *тестирующую* выборки.

Обучающая выборка предназначена для обучения НС и используется для подбора весов при обучении (в большинстве случаев речь идет об итеративно-сходящихся алгоритмах).

Проверочная выборка предназначена для того, чтобы вовремя закончить обучение НС, так как иногда увеличение количества обучающих циклов (*эпох*) ведет к возникновению *эффекта переобучения*. При этом НС показывает превосходные результаты на обучающей выборке и все более ухудшающиеся результаты на проверочной выборке. В большинстве случаев это означает, что количество узлов в сети велико и сеть «запоминает» информацию вместо того, чтобы ее анализировать.

Тестирующая выборка предназначена для проверки качества прогноза окончательно обученной НС.

НС можно разделить по принципам обучения на два вида: *обучающиеся с учителем* и *самообучающиеся*. *Обучение с учителем* происходит обычно путем подачи на вход сети (на вход узлов входного слоя) входных данных и корректировки весов при помощи определенных алгоритмов обучения с учетом информации, которую выдала сеть на выходе (на выходе узлов выходного слоя). *Самообучение* отличается тем, что алгоритмы обучения не учитывают ошибку сети во время обучения.

Как было отмечено выше, обучение можно разделить на *эпохи*. *Эпоха* — это период обучения сети, за который сеть один раз была обучена при помощи каждого из наблюдений в обучающей выборке.

Стоит отметить, что в случаях, когда имеет смысл указывать зависимость значения некоторой переменной от номера эпохи (дискретного аналога времени), применяется обозначение (t), где t — номер эпохи.

2.3. Алгоритмы обучения нейронных сетей прямого распространения

2.3.1. Обратное распространение (backprop)

Стандартный алгоритм обратного распространения, изложенный в [13], является одним из самых первых алгоритмов, предназначенных для обучения сетей прямого распространения. Суть алгоритма состоит

в том, что на первом этапе происходит распространение активации от входного слоя к выходному (прямое распространение) и запоминаются значения на выходе каждого нейрона. Затем производится корректировка весов в обратном порядке в соответствии с формулами:

$$\begin{aligned} \Delta w_{ij} &= \eta \delta_j o_i, \\ \delta_j &= \begin{cases} f_j^{'}(net_j)(t_j - o_j), & \text{если узел} j - \text{выходной}, \\ f_j^{'}(net_j) \sum_k \delta_k w_{jk}, & \text{если узел} j - \text{скрытый}, \end{cases} \end{aligned} \tag{3}$$

где Δw_{ij} — изменение веса между i-ым и j-ым узлами, η — параметр, задающий скорость обучения, δ_j — направление и степень изменения веса, o_j — выходное значение узла j, $f_j^{'}$ — первая производная активационной функции узла j, net_j — чистое значение входа узла j, t_j — выходное значение j из тестовой выборки. Так как корректировка ошибки происходит от выходных слоев к входным, то всегда известна величина δ_k. Она показывает степень изменения веса для тех узлов, к которым идут выходы текущего нейрона, по которым и считается сумма, а w_{jk} — значение веса связи от узла j до узла k.

Свое название метод получил из-за того, что корректировка весов происходит в обратном направлении (в противоположность прямому распространению).

2.3.2. Расширенное обратное распространение (enhanced backprop or backprop momentum)

Расширенная версия обратного распространения использует понятия момента и выявления пологого места.

Понятие момента вводит зависимость текущего изменения весов от их предыдущих изменений. Это позволяет избежать проблем с осцилляцией, обычных для алгоритмов обратного распространения, когда поверхность ошибки имеет очень узкую зону минимума. Формула изменения весов для расширенного обратного распространения выглядит как

$$\Delta w_{ij}(t+1) = \eta \delta_j o_i + \alpha \delta_j w_{ij}(t), \tag{4}$$

где α — константа, задающая степень влияния момента, а $t+1$ — номер текущей эпохи.

Эффект этого изменения состоит в том, что пологое место на поверхности ошибки достигается достаточно быстро, большими шагами,

а когда поверхность становится более пологой, шаги уменьшаются. Этот подбор размера шага сильно увеличивает скорость обучения.

2.3.3. Пакетное обратное распространение (batch backprop)

Пакетное обратное распространение является модификацией алгоритмов обратного распространения, относящейся ко времени обновления весов. Разница состоит в том, когда происходит обновление весов. В обратном распространении обновление весов происходит после каждого наблюдения. В пакетном обратном распространении все изменения суммируются, и обновление происходит только после того, как все наблюдения обучающего набора будут использованы. Этот способ обновления крайне полезен при параллельном обучении, когда стоимость обмена данными очень высока.

Стоит отметить, что данный метод, как и метод, описанный в следующем параграфе, может применяться к различным алгоритмам обратного распространения.

2.3.4. Фрагментарное обратное распространение (backprop with chunkwise update)

Алгоритм разрабатывался как некоторый компромисс между обновлением после каждого шага и обновлением после каждой эпохи.

Суть модификации состоит в том, что обновления весов происходят через каждые n наблюдений, где n — размер фрагмента. В исходной постановке считается, что $0 > n > N$, где N – количество наблюдений в обучающей выборке. При такой постановке фрагментарное обратное распространение имеет смысл использовать в тех случаях, когда обратное распространение будет слишком дорого, а пакетное обратное распространение будет производить обновления слишком редко.

Можно сказать, что данный алгоритм является обобщением подхода к проблеме обновления. Изменяя размер фрагмента, после которого происходит обновление, можно получить как пакетное обратное распространение (когда $n = N$), так и обратное распространение с обновлением после каждого шага (когда $n = 1$).

2.3.5. Обратное распространение с угасанием весов (backprop with weight decay)

Угасание весов было предложено П. Вербосом [17]. Оно состоит в уменьшении веса в процессе обучения обратным распространением. Вдобавок к обычному изменению весов значение веса уменьшается на некоторую часть d старого значения веса. Получившаяся формула выглядит так:

$$\Delta w_{ij}(t+1) = \eta \delta_j o_i - d w_{ij}(t). \tag{5}$$

Эффект алгоритма состоит в том, что в процессе обучения незначимые веса все время уменьшаются и в конце концов станут равными нулю. Более подробное описание можно найти в [14].

2.3.6. Ускоренное обратное распространение (quickprop)

Одним из способов ускорить обучение является использование информации о кривизне поверхности ошибки. Это требует вычисления второй производной функции ошибки. Ускоренное обратное распространение предполагает, что поверхность ошибок локально является квадратичной и пытается переместиться за один раз прямо в минимум параболы.

Quickprop вычисляет производные каждого веса. После вычисления градиента обычным способом обратного распространения перемещение производится по следующей формуле:

$$\Delta w_{ij}(t+1) = \frac{S_{ij}(t+1)}{S_{ij}(t) - S_{ij}(t+1)} \Delta w_{ij}(t), \tag{6}$$

где $\Delta w_{ij}(t+1)$ — текущее изменение веса между узлами i и j. $S_{ij}(t+1)$ — частная производная функции ошибки по w_{ij}, $S_{ij}(t)$ — предыдущая частная производная функции ошибки по w_{ij}, $\Delta w_{ij}(t+1)$ — предыдущее изменение веса между узлами i и j.

Более подробно Quickprop описан в [4].

2.3.7. Rprop

Основной принцип Rprop состоит в выделении влияния знака частной производной на размер шага. В результате учитывается только знак

частной производной для того, чтобы указать направление шага. Размер шага однозначно определяется при помощи специфичного для каждого веса, так называемого «значения обновления» $\Delta_{ij}(t+1)$:

$$\Delta w_{ij}(t+1) = \begin{cases} -\Delta_{ij}(t+1), & \text{если } \frac{\partial E(t+1)}{\partial w_{ij}} > 0, \\ +\Delta_{ij}(t+1), & \text{если } \frac{\partial E(t+1)}{\partial w_{ij}} < 0 \\ 0 & \text{иначе,} \end{cases} \tag{7}$$

где $\frac{\partial E(t+1)}{\partial w_{ij}}$ обозначает суммарный градиент по всем наблюдениям в обучающей выборке.

Необходимо заметить, что если заменить $\Delta_{ij}(t+1)$ константой Δ, уравнение (7) превращается в так называемое правило обновления «Манхаттэн».

Вторым шагом Rprop является определение новых значений обновления $\Delta_{ij}(t+1)$. Основой для этого является процесс приспособления, зависящий от знака:

$$\Delta_{ij}(t+1) = \begin{cases} \eta^+ \Delta_{ij}(t), & \text{если } \frac{\partial E(t)}{\partial w_{ij}} \frac{\partial E(t+1)}{\partial w_{ij}} > 0, \\ \eta^- \Delta_{ij}(t), & \text{если } \frac{\partial E(t)}{\partial w_{ij}} \frac{\partial E(t+1)}{\partial w_{ij}} < 0, \\ \Delta_{ij}(t) & \text{иначе,} \end{cases} \tag{8}$$

где $0 < \eta^- < 1 < \eta+$.

Иными словами Rprop работает так: каждый раз, когда частная производная соответствующего веса $w_{ij}(t+1)$ изменяет свой знак, что обозначает тот факт, что последнее обновление весов было слишком большим и локальный минимум был пройден мимо, значение обновления $\Delta_{ij}(t+1)$ уменьшается на коэффициент η^-. Если производная не меняет знак, то шаг увеличивается на величину η^+, чтобы быстрее подойти к минимуму.

Наиболее хорошие результаты получаются при $\eta^- = 0,5$ и $\eta^+ = 1,2$.

Более подробно Rprop обсуждается в [10], [11], [12].

2.3.8. RpropMAP (Rprop с с разложением весов)

Расширенная версия Rprop работает так же, как и обратное распространение с угасанием весов, за исключением того, что параметр λ вычисляется автоматически при помощи подхода Байеса. Широкое обсуждение принципов обучения Байеса можно найти в [2].

В подходе Байеса используется теорема Байеса для определения posteriori распределения весов $p(w|D)$ при предварительном предположении о весах $p(w)$ и шума в данных соответственно вероятности $p(D|w)$:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)}. \tag{9}$$

Можно показать, что разложение весов соответствует предположению о том, что веса распределены со средним 0. Мы минимизируем функцию ошибки $E = E_D + \lambda E_w$, где E_D — ошибка нейронной сети (например, сумма квадратов ошибок) и E_w относится к разложению весов. Используя данный подход можно подправлять λ время от времени в течение процесса обучения. Предположим, что веса имеют нормальное распределение с нулевым средним и дисперсией $\frac{1}{\alpha}$, и ошибка также имеет нормальное распределение с вариацией $\frac{1}{\beta}$. Тогда можно найти эти два параметра. Будем устанавливать $\lambda = \frac{\alpha}{\beta}$ каждые несколько эпох, тогда параметры изменятся следующим образом: $\alpha_{new} = W/\sum w_i^2$ и $\beta_{new} = N/E_D$, где W — количество весов и N — количество наблюдений.

Стоит отметить, что данному методу нет необходимости иметь проверочную выборку, так как проблема переобучения не возникает.

2.4. Алгоритмы создания сети с оптимальной топологией

Одной из основных проблем при использовании НС является нахождение оптимальной топологии сети.

Алгоритмы каскадной корреляции являются по своей сути мета-алгоритмами, сочетающими в себе обычные алгоритмы обучения, такие как: обратного распространения, быстрого распространения и т. д. и алгоритмы создания НС с оптимальной топологией. Вы можете спросить: «почему не нахождения оптимальной топологии?» Ответ на этот вопрос прост: это семейство алгоритмов строит сеть по определенным правилам и каждый раз проверяет, оптимально ли еще увеличивать сеть каким-либо из доступных алгоритму путей. Если больше нет оптимального пути, то сеть считается оптимальной. На самом деле это не всегда так, поскольку идея таких алгоритмов состоит в том, что сеть с полученной в результате топологией считается оптимальной «по построению».

2.4.1. Каскадная корреляция

Каскадная корреляция была разработана Скотом Фалманом [5], [3], [6] с целью облегчить процесс выбора топологии сети.

Каскадная корреляция относится к создающему классу алгоритмов. Это класс алгоритмов, который определяет не только веса НС, но и ее внутреннюю структуру.

Каскадная корреляция (КК) — это архитектура обучения с «учителем», которая строит топологию с наименьшим количеством узлов многослойной НС прямого распространения. У КК существуют два преимущества: исследователю нет необходимости заботиться о топологии НС а также она существенно быстрее обычных алгоритмов обучения.

Каскадная корреляция объединяет две идеи: первая — каскадная топология, в которой скрытые слои добавляются один раз и остаются неизменными, вторая — обучающий алгоритм, который создает и добавляет новые скрытые узлы. Для каждого нового скрытого узла алгоритм пытается максимизировать значение корреляции между выходом нового узла и ошибкой НС.

Алгоритм реализуется указанным ниже способом.

1. КК начинает с минимальной НС, состоящей только из входного и выходного слоев. Оба слоя — *полносвязные*, то есть существуют все возможные связи между всеми узлами при учете ограничений, налагаемых на сети прямого распространения.

2. Обучаются обычным алгоритмом все связи, ведущие к выходному слою до тех пор, пока ошибка не перестает уменьшаться.

3. Создаются так называемые узлы-кандидаты. Каждый узел-кандидат связан со всеми входными слоями и со всеми существующими скрытыми узлами. Между узлами-кандидатами и выходными узлами связь отсутствует.

4. Производится попытка максимизировать корреляцию между задействованием узлов-кандидатов и остаточной ошибкой сети, обучая связи, ведущие к узлам-кандидатам. Обучение происходит при помощи обычных алгоритмов. Обучение прекращается, когда показатели корреляции перестают улучшаться.

5. Выбирается узел-кандидат с максимальной корреляцией, фиксируются веса его входных связей, и он добавляется в сеть. Для превращения узла-кандидата в скрытый узел создаются связи между ним и выходными узлами. Возвращаемся к пункту 2.

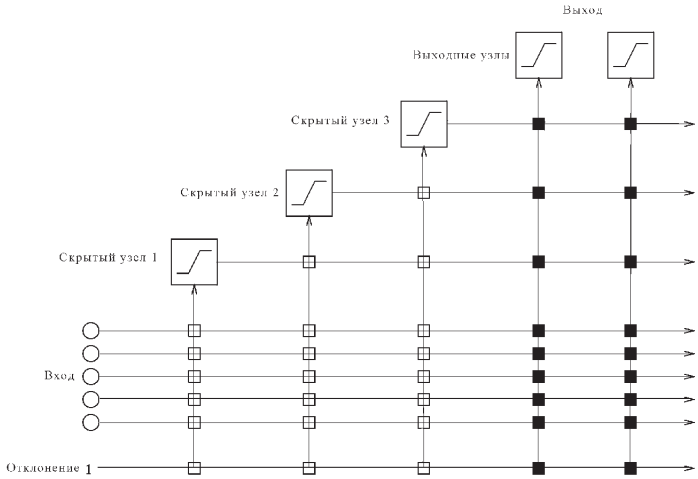

Рис. 4. Нейронная сеть, обученная при помощи каскадной корреляции после того, как было добавлено 3 скрытых узла. Связи с незакрашенными пересечениями зафиксированы, с черными — постоянно обучаются

Алгоритм повторяется до тех пока ошибка не упадет до заданного уровня.

При обучении выходных элементов мы пытаемся минимизировать сумму квадратов ошибок E:

$$E = \sum_p \frac{1}{2} \sum_j (o_{pj} - t_{pj})^2, \tag{10}$$

где t_{pj} — реальное значение и o_{pj} — наблюдаемый выход выходного узла j для наблюдения p. Ошибка E минимизируется методом градиентного спуска, используя:

$$e_{pj} = (o_{pj} - t_{pj}) f'(net_j), \tag{11}$$

$$\frac{\partial E}{\partial w_{ij}} = \sum_p e_{pj} I_{ip}, \tag{12}$$

где f' — производная активационной функции выходного узла j, I_{ip} — значение входного узла или скрытого узла i для наблюдения p, w_{ij} вы-

ражает связь между входным или скрытым узлом i и выходным узлом j.

После обучения параметры узлов-кандидатов изменяются так, чтобы корреляция C между значением o_{pj} узла-кандидата и остаточной ошибкой e_{pj} выходного узла стала максимальной. Фалман записывал корреляцию так:

$$\begin{aligned} C &= \sum_j \left| \sum_p (o_{pj} - \overline{o_j})(e_{pj} - \overline{e_j}) \right| \\ &= \sum_j \left| \sum_p o_{pj} e_{pj} - \overline{e_j} \sum_p o_{pj} \right| \\ &= \sum_j \left| \sum_p o_{pj} (e_{pj} - \overline{e_j}) \right|, \end{aligned} \tag{13}$$

где $\overline{o_j}$ — математическое ожидание наблюдаемого выхода узла-кандидата и $\overline{e_j}$ — средняя ошибка выходного узла по всем наблюдениям p. Максимизация C производится методом градиентного спуска, используя:

$$\delta_p = \sum_j \sigma_j (e_{pj} - \overline{e_j}) f', \tag{14}$$

$$\frac{\partial C}{\partial w_i} = \sum_p \delta_p I_{pi}, \tag{15}$$

где σ_j — знак корреляции между выходом узла-кандидата и остаточной ошибкой на выходе j.

2.4.2. Модификации каскадной корреляции

Одна из основных проблем каскадной корреляции — это топология получаемой сети. Так как каждый скрытый узел связан с каждым другим скрытым узлом, очень сложно распараллелить сеть. Возникшие позже модификации призваны облегчить решение этой проблемы.

2.4.3. Каскадная корреляция вида «сверстник/потомок»

Эта модификация была предложена С. Балуйя и С. Е. Фалманом [15]. Узлы-кандидаты разбиваются на две группы:

— узлы-«потомки»: эти узлы получают данные от всех входных узлов и всех существовавших до этого скрытых слоев, так что узел увеличивает количество слоев в сети на единицу;

— узлы-«сверстники»: эти узлы связаны с входными узлами и со всеми скрытыми узлами из созданных ранее слоев, но не с последним скрытым слоем. Когда узел-«сверстник» добавляется в сеть, он становится составляющим последнего скрытого слоя.

Во время обучения узлов-кандидатов узлы-«сверстники» и узлы-«потомки» конкурируют друг с другом. Если C остается неизменной, в большинстве случаев узлы-«потомки» имеют лучшую корреляцию и поэтому будут добавлены. Это ведет к углублению сети, как в случае с каскадной корреляцией. Поэтому коэффициент корреляции C для узлов-«потомков» умножается на коэффициент $\lambda \leq 1.0$. Например, если $\lambda = 0.5$, узел-«потомок» будет выбран только тогда, когда его коэффициент корреляции как минимум в два раза больше, чем лучший коэффициент корреляции для узлов-«сверстников». $\lambda \to 0$ ведет к тому, что сеть будет состоять из одного скрытого слоя.

2.4.4. Каскадная корреляция со случайным выбором слоя.

Данная модификация использует идею, близкую к каскадной корреляции вида «сверстник/потомок». Каждый узел-кандидат случайным образом ассоциируется со скрытым слоем существующей сети либо с новым слоем. Например, если существует 4 узла-кандидата и 6 скрытых слоев, то узлы-кандидаты ассоциируются со слоями 1, 3, 5 и 6 соответственно. Узлы-кандидаты соединяются так, как будто они находятся в ассоциированных слоях.

Корреляция C изменяется следующим образом:

$$C^{'} = C \times f^{1+l-x}, \tag{16}$$

где C — первоначальная корреляция, l — количество слоев, x — количество ассоциированных слоев, f — параметр, который имеет смысл задавать в интервале $[1.0; 2.0]$. Значения $f > 2.0$ ведут к тому, что сеть будет состоять из двух скрытых слоев.

2.4.5. Статические алгоритмы

Алгоритм называется статическим, если решение о создании связи между узлами i и j может быть принято без обучения сети. В действительности может использоваться любая функция вида $\mathbf{N} \to \{0, 1\}$. В основном, данный подход используется в многослойных сетях. Это сети, у которых узел i получает данные от узла j тогда и только тогда,

когда узел i находится в слое, лежащем ниже, чем слой, в котором находится j. Тогда необходимо только определить веса, принадлежащие слоям.

В качестве примера можно использовать функцию вида:

$$h_k = \max\left(1, \left\lfloor b \times e^{-(k-1)d} + \tau \times \Delta b \right\rfloor\right), \tag{17}$$

где τ — случайное число в диапазоне $[-1; 1]$, b, d и Δb — параметры функции.

2.4.6. Экспоненциальная каскадная корреляция (ЭКК)

Это просто небольшая модификация. Узел j получает данные с узла i тогда и только тогда, когда $i \leq m \times j$. Эта модель создает сеть с экспоненциально возрастающим количеством узлов на один скрытый слой. Например, если $m = \frac{1}{2}$, то каждый последующий скрытый слой будет в два раза больше предыдущего.

2.4.7. Упрощающая каскадная корреляция (УКК)

Основная цель упрощающей каскадной корреляции состоит в том, чтобы уменьшить *ожидаемую* ошибку на тестовых наблюдениях вместо ошибки во время обучения. УКК пытается определить оптимальное количество скрытых узлов и удалить ненужные связи после того, как был добавлен новый скрытый узел. Как было показано Верфритцем [16], критерий выбора, как он используется в stopped обучении, может быть применен для удаления ненужных связей.

Алгоритм работает следующим образом (шаги, относящиеся к КК, выделены):

1. *Обучаются обычным алгоритмом все связи, ведущие к выходному слою до тех пор, пока ошибка не перестает уменьшаться.*

2. Вычисляется критерий выбора.

3. *Обучаются узлы-кандидаты.*

4. *Выбирается узел-кандидат с максимальной корреляцией, фиксируются веса его входных связей, и он добавляется в сеть. Для превращения узла-кандидата в скрытый узел создаются связи между ним и выходными узлами. Возвращаемся к пункту 2.*

5. Вычисляется критерий выбора.

6. Вес каждой связи поочередно устанавливается в ноль, и вычисляется критерий выбора; если существует хотя бы одна связь, удаление

которой уменьшит критерий выбора, то удаляется связь, удаление которой приведет к наибольшему уменьшению критерия выбора.

7. Вычисляется критерий выбора; если он больше, чем вычисленный перед вставкой нового слоя, то сеть становится слишком большой и пора прекращать процесс.

Обычно используются три критерия выбора: Байесов критерий Швартца (БКШ), информационный критерий Акаика [1] (ИКА) и консервативная средне-квадратическая ошибка предсказания (КСКОП). БКШ — это более консервативный критерий, нежели ИКА, то есть сети, полученные при его помощи, обычно бывают меньшего размера. Имеет смысл заметить, что БКШ и ИКА — критерии выбора для линейных моделей. Хотя КСКОП не опирается ни на какую статистическую теорию, он неплохо работает на практике. Иногда его можно применять для нелинейных моделей, например, в случае, когда выборка достаточно велика.

2.5. Сети Кохонена

Автор специально выделил сети, изобретенные Тойво Кохоненом, и подобные им, в отдельный раздел, чтобы подчеркнуть значимость его вклада в развитие нейронных сетей.

Первая разработка Кохонена — обучающаяся классификация векторов (Learning Vector Quantisation — LVQ), затем были разработаны всемирно известные самоорганизующиеся карты Кохонена (self organized Kohonen's maps — SOM). Также в данной главе приводится описание DLVQ — усовершенствованной версии алгоритма LVQ, разработанной в Штуттгартском университете.

2.5.1. LVQ

Идея алгоритма состоит в том, чтобы найти естественные группировки в наборе данных [7]. Каждое наблюдение ассоциируется с точкой в d-мерном пространстве. Предполагается, что наблюдения $\overrightarrow{x}$ одного класса образуют облако или кластер в пространстве. Алгоритм предполагает, что вектора $\overrightarrow{x}$, принадлежащие к одному и тому же классу i, распределены нормально с вектором средних $\overrightarrow{\mu_i}$, который представляется набором весов, входящих в выходной узел, описывающий класс i (в дальнейшем будет использоваться обозначение $\overrightarrow{w_i}$; термин узел можно рассматривать как вектор; имя класса будет использоваться в качестве

индекса узла), и что все входные наблюдения — нормализованы.

$$w_i = \begin{pmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{in} \end{pmatrix}. \tag{18}$$

Для классификации наблюдения $\overrightarrow{x}$ измеряется евклидово расстояние $\|\overrightarrow{w_i} - \overrightarrow{x}\|^2$ от $\overrightarrow{x}$ до всех выходных узлов $\overrightarrow{w_i}$, и $\overrightarrow{x}$ считается принадлежащим к тому классу, до среднего которого евклидово расстояние наименьшее. Но что произойдет, если наблюдение $\overrightarrow{x_A}$, принадлежащее к классу A, будет считаться принадлежащим к другому классу B? Тогда для неверно классифицированного наблюдения веса узлов $\overrightarrow{w_A}$ и $\overrightarrow{w_B}$ будут изменены следующим образом:

— узел $\overrightarrow{w_A}$, к которому должно было быть приписано наблюдение $\overrightarrow{x_A}$, передвигается ближе к $\overrightarrow{x_A}$,

— узел $\overrightarrow{w_B}$, к которому ошибочно было приписано наблюдение $\overrightarrow{x_A}$, отодвигается от него,

— веса узлов изменяются, используя правило:

$$w_{ij} = w_{ij} + \eta(o_i + w_{ij}), \tag{19}$$

где w_{ij} — вес между выходом o_i входного узла i и выходного узла j, η — параметр обучения. Выбирая его больше либо меньше нуля, мы задаем направление перемещения.

Алгоритм работы LVQ выглядит представленным ниже образом:

1. Загрузить нормализованную обучающую выборку и для каждого класса вычислить вектор средних μ. Создать сеть: для каждого класса создать узел, у которого установить веса, соответствующие μ.

2. Попытаться сопоставить каждое наблюдение с классом. Если произошло неверное сопоставление наблюдения $\overrightarrow{x_A}$, принадлежащего классу A, с классом B, то

a) узел $\overrightarrow{w_A}$ передвинуть ближе к вектору $\overrightarrow{x_A}$,

b) узел $\overrightarrow{w_B}$ отодвинуть в сторону от $\overrightarrow{x_A}$.

3. Вернуться к шагу 2. (До тех пор, пока количество верно классифицированных векторов не перестанет уменьшаться.)

2.5.2. DLVQ

Идея алгоритма возникла независимо от работ Тойво Кохонена в процессе работы над проектом SNNS (Stuttgart Nueral Network Simulator или Штутгарттский симулятор нейронных сетей) в Штуттгартском университете. Эта идея аналогична идее LVQ, за одним исключением: алгоритм начинает создание сети с минимального набора скрытых слоев и добавляет новые по мере необходимости, а LVQ необходима уже построенная топология.

Алгоритм работы DLVQ описан ниже.

1. Загрузить нормализованную обучающую выборку, и для каждого класса вычислить вектор средних μ. Создать сеть: для каждого класса создать узел, у которого установить соответствующие веса.

2. Попытаться сопоставить каждое наблюдение с классом. Если произошло неверное сопоставление вектора $\overrightarrow{x_A}$, принадлежащего классу $\overrightarrow{w_A}$, с классом $\overrightarrow{w_B}$, то

a) вектор $\overrightarrow{\mu_A}$ передвинуть ближе к вектору $\overrightarrow{x_A}$,

b) вектор средних $\overrightarrow{\mu_B}$ отодвинуть в сторону от $\overrightarrow{x_A}$.

3. Вернуться к шагу 2. (До тех пор, пока количество верно классифицированных векторов не перестанет уменьшаться.)

4. Если результат не удовлетворяет, то создать новый скрытый слой, иначе закончить обучение. Теперь для неверно классифицированных векторов вычислить новый вектор средних μ. Для каждого класса создать новый узел в новом скрытом слое с весом μ. Перейти к шагу 2. Более подробное описание LVQ можно найти в [7].

2.5.3. SOM

В противоположность большинству других сетей, SOM использует обучение без «учителя». Основное применение данного класса сетей — классификация данных, получение двухмерного изображения входных данных.

SOM состоит из двух слоев: одномерный входной слой и двухмерный слой из соревнующихся узлов, организованный как двухмерная решетка из узлов. Этот слой нельзя назвать ни скрытым, ни выходным. Каждый соревнующийся узел имеет вектор весов W_i, который после обучения описывает некоторый класс. Обучающий алгоритм для SOM объединяет два важных принципа:

— кластеризацию входных данных,

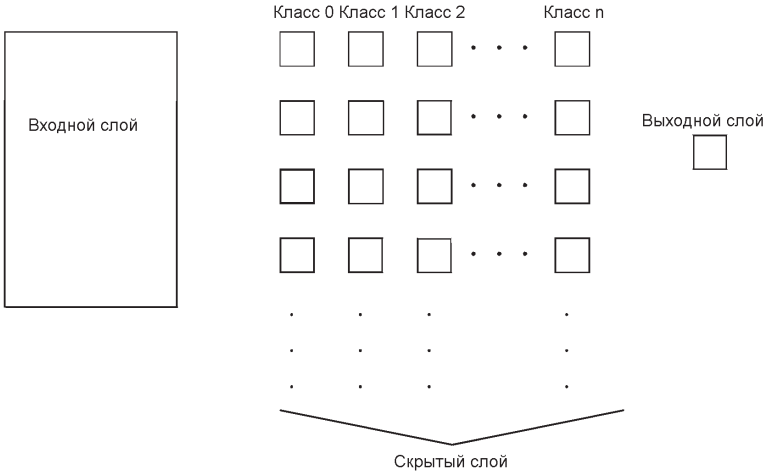

Рис. 5. Топология сети, обученной при помощи DLVQ

— такую пространственную организацию карты, чтобы похожие входные наблюдения активизировали узлы, находящиеся рядом на решетке.

Перед началом обучающего процесса необходимо задать всем соревнующимся элементам случайные нормализованные веса. Входные вектора наблюдений подаются на вход соревнующимся узлам параллельно, и ближайший узел объявляется победителем. Так как вектора нормализованы, сходство между нормализованным входным вектором $X = (x_i)$ и весами $W_i = (w_{ij})$ может быть вычислено с использованием евклидового расстояния:

$$net_J(t) = \|X(t) - W_J(t)\| . \tag{20}$$

Вектор W_c[1], наиболее похожий на X, является вектором с наименьшим евклидовым расстоянием до X:

$$net_c(t) = \min_j \{\|X(t) - W_j(t)\|\} . \tag{21}$$

Пространственная организация карты достигается за счет того, что изменяется не только узел, который оказался наиболее близким, но и

[1]Здесь и далее c будет использоваться для обозначения выигравшего узла.

его ближайшее окружение[2] N_c (в отличии от простых обучающих алгоритмов типа LVQ):

$$\begin{array}{ll} \Delta w_{ij}(t) & = e_j(t)(x_i(t) - w_{ij}(t)) \quad \text{для } j \in N_c \\ \Delta w_{ij}(t) & = 0 \qquad\qquad\qquad\qquad\quad \text{для } \notin N_c \end{array} \tag{22}$$

где $e_j(t) = h(t)e^{-(d_j/r(t))^2}$ (функция Гаусса), d_j — расстояние между W_j и победителем W_c, $h(t)$ — адаптационный параметр во время t такой, что $0 \leq h(t) \leq 1$, $r(t)$ — радиус ближайшего окружения N_c в зависимости от t.

Обычно адаптационный параметр и радиус ближайшего окружения уменьшаются во времени для обеспечения конечности процесса.

Более подробно теория SOM описана в [8], [9].

СПИСОК ЛИТЕРАТУРЫ

1. Akaike, H. Fitting autoregressive models for prediction. *Annals of the institute of statistical mathematics 21* (1969).
2. Bishop, C. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.
3. Fahlman, S. E., and Lebiere, C. The cascade-correlation learning architecture. Tech. Rep. CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, August 1991.
4. Fahlman, S. E. Faster-learning variations on back-propagation: An empirical study. In *1988 Connectionist Models Summer School* (San Mateo, CA, 1988), T. J. S. G. E. Hinton and D. S. Touretzky, Eds., Morgan Kaufmann.
5. Fahlman, S. E. The recurrent cascade-correlation architecture. Tech. Rep. CMU-CS-91-100, School of Computer Science, Carnegie Mellon University, 1991.
6. Hoefeld, M., and Fahlman, S. E. Learning with limited numerical precision using the cascade-correlation algorithm. Tech. Rep. CMU-CS-91-130, School of Computer Science, Carnegie Mellon University, 1991.
7. Kohonen, T., Kangas, J., Laaksoonen, J., and Torkkola, K. Lvq_pak learning vector quantization program package. Tech. rep., Laboratory of Computer and Information Science Rakentajanaukio 2 C, 1991 - 1992.
8. Kohonen, T. *Self-Organization and Associative Memory.* Springer-Verlag, 1988.
9. Kohonen, T. *Self-Organization and Associative Memory, Third Edition.* Springer Verlag 1989, 1989.
10. M. Riedmiller, H. Rprop: A fast and robust backpropagation learning strategy. In *Proc. of the ACNN* (1993).

[2]Ближайшее окружение определяется как набор узлов, попадающих в окружность определенного радиуса с победителем в центре. Таким образом, $N(1)$ будет состоять из восьми ближайших соседей на двухмерной решетке; $N(2)$ будет $N(1)$ плюс 16 ближайших соседей, и т.д.

11. Riedmiller, M., and Braun, H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)* (1993).
12. Riedmiller, M. Untersuchungen zu konvergenz und generalisierungsverhalten überwachter lernverfahren mit dem SNNS. In *Proceedings of the SNNS 1993 workshop* (1993).
13. Rumelhart, D., and McClelland, J. *Parallel Distributed Processing*, vol. 1. MIT Press, 1986.
14. Schreiner, T. Ausdünnungsverfahren für Neuronale Netze. Diplomarbeit 1140, IPVR, Universität Stuttgart, 1994.
15. S. Baluja, S. F. Reducing network depth in the cascade-correlation learning architecture. Tech. Rep. CMU-CS-94-209, Carnegie Mellon University, 1994.
16. Wehrfritz, C. Neuronale netze als statistische methode zur erklärung von klassifikationen. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Statistik 1, 1994.
17. Werbos, P. Backpropagation: Past and future. In *Proceedings of the IEEE International Conference on Neural Networks* (1988), IEEE Press, pp. 343–353.

Денис Першин

ОБЗОР НЕКОТОРЫХ ВИДОВ НЕЙРОННЫХ СЕТЕЙ

Препринт
63

Рукопись поступила в редакцию 28.07.1999
Рецензент Ф. А. Мурзин
Редактор З. В. Скок

Подписано в печать 18.01.2000
Формат бумаги 60×84 1/16 Объем 1,5 уч.-изд.л., 1,6 п.л.
Тираж 50 экз.

ЗАО РИЦ "Прайс-курьер", 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6