Глава 3.

Быстрое дифференцирование, двойственность и обратное распространение ошибки

А.Н.Горбань

Вычислительный центр СО РАН в г.Красноярске[1]

В этой главе излагается материал, значение которого для вычислительной математики и всего того, что по-английски именуют «computer sciences», выходит далеко за пределы нейроинформатики. Со временем он безусловно будет включен в программы университетских курсов математического анализа.

Обсудим одну "очевидную" догму, без разрушения которой было бы невозможно эффективное обучение нейронных сетей. Пусть вычислительные затраты (оцениваемые временем, затраченным некоторым универсальным вычислительным устройством) на вычисление одного значения функции n переменных $H(x_1,...,x_n)$ примерно равны T. Сколько времени потребуется тому же устройству на вычисление grad$H$ (при разумном составлении программы)? Большинство математиков с университетским дипломом ответит:

$$T_{\mathrm{grad}H} \sim nT_H. \qquad (1)$$

Это неверно! Правильный ответ:

$$T_{\mathrm{grad}H} \sim CT_H. \qquad (2)$$

где C - константа, не зависящая от размерности n (в большинстве случаев $C \sim 3$).

Для всех функций многих переменных, встречающихся на практике, необходимые вычислительные затраты на поиск их градиента всего лишь в два-три раза превосходят затраты на вычисление одного значения функции. Это удивительно - ведь координатами вектора градиента служат n частных производных, а затраты на вычисление одной такой производной в общем случае примерно такие же, как и на вычисление значения функции. Почему же вычисление всех их вместе дешевле, чем по-отдельности?

«Чудо» объясняется довольно просто: нужно рационально организовать вычисление производных сложной функции многих переменных, избегая

[1] 660036, Красноярск-36, ВЦК СО РАН. E-mail: gorban@cc.krascience.rssi.ru

дублирования. Для этого необходимо подробно представить само вычисление функции, чтобы потом иметь дело не с «черным ящиком», преобразующим вектор аргументов в значение функции, а с детально описанным графом вычислений.

Поиск grad$H$ удобно представить как некоторый двойственный процесс над структурой вычисления H. Промежуточные результаты, появляющиеся при вычислении градиента, являются ни чем иным, как множителями Лагранжа. Оказывается, что если представить H как сложную функцию, являющуюся суперпозицией функций малого числа переменных (а по-другому вычислять функции многих переменных мы не умеем), и аккуратно воспользоваться правилом дифференцирования сложной функции, не производя по дороге лишних вычислений и сохраняя полезные промежуточные результаты, то вычисление всей совокупности $\partial H/\partial x_i$ (i=1,...,$\underline{n}$) немногим сложнее, чем одной из этих функций - они все собраны из одинаковых блоков.

Я не знаю, кто все это придумал первым. В нейроинформатике споры о приоритете ведутся до сих пор. Конец переоткрытиям положили две работы 1986 г.: Румельхарта, Хинтона и Вильямса [1,2] и Барцева и Охонина [3]. Однако первые публикации относятся к 70-м и даже 60-м годам нашего столетия. По мнению В.А.Охонина, Лагранж и Лежандр также вправе претендовать на авторство метода.

1. Обучение нейронных сетей как минимизация функции ошибки

Построение обучения как оптимизации дает нам универсальный метод создания нейронных сетей для решения задач. Если сформулировать требования к нейронной сети, как задачу минимизации некоторой функции - оценки, зависящей от части сигналов (входных, выходных, ...) и от параметров сети, то обучение можно рассматривать как оптимизацию и строить соответствующие алгоритмы, программное обеспечение и, наконец, устройства (hardware). Функция оценки обычно довольно просто (явно) зависит от части сигналов - входных и выходных. Ее зависимость от настраиваемых параметров сети может быть сложнее и включать как явные компоненты (слагаемые, сомножители,...), так и неявные - через сигналы (сигналы, очевидно, зависят от параметров, а функция оценки - от сигналов).

За пределами задач, в которых нейронные сети формируются по явным правилам (сети Хопфилда, проективные сети, минимизация аналитически заданных функций и т.п.) нам неизвестны случаи, когда требования к нейронной сети нельзя было бы представить в форме минимизации функции оценки. Не следует путать такую

постановку задачи и ее весьма частный случай - "обучение с учителем". Уже метод динамических ядер, описанный в предыдущей главе, показывает, каким образом обучение без учителя в задачах классификации может быть описано как минимизация целевой функции, оценивающей качество разбиения на классы.

Если для решения задачи не удается явным образом сформировать сеть, то проблему обучения можно, как правило, сформулировать как задачу минимизации оценки. Осторожность предыдущей фразы ("как правило") связана с тем, что на самом деле нам неизвестны и никогда не будут известны все возможные задачи для нейронных сетей, и, быть может, где-то в неизвестности есть задачи, которые несводимы к минимизации оценки.

Минимизация оценки - сложная проблема: параметров астрономически много (для стандартных примеров, реализуемых на PC - от 100 до 1000000), адаптивный рельеф (график оценки как функции от подстраиваемых параметров) сложен, может содержать много локальных минимумов, извилистых оврагов и т.п.

Наконец, даже для того, чтобы воспользоваться простейшими методами гладкой оптимизации, нужно вычислять градиент функции оценки. Здесь мы сталкиваемся с одной "очевидной" догмой, без разрушения которой было бы невозможно эффективное обучение нейронных сетей.

2. Граф вычисления сложной функции

Сложная функция задается с помощью суперпозиции некоторого набора «простых». Простые функции принадлежат исходно задаваемому конечному множеству *F*. Формально они выделены только принадлежностью к множеству *F* - никаких обязательных иных отличий этих функций от всех прочих в общем случае не предполагается.

В этом разделе мы изучим способы задания сложных функций, то есть формулы и соответствующие им графы. Свойства самих функций нас сейчас не интересуют (будем рассматривать ноты, а не слушать музыку).

Теория выражений, определяющих сложные функции, является простейшим разделом математической логики, в которой сами эти выражения называются *термами* [4].

Термы - это правильно построенные выражения в некотором формальном языке. Чтобы задать такой язык, необходимо определить его *алфавит*. Он состоит из трех множеств символов:

1) C - множество символов, обозначающих константы;

2) V - множество символов, обозначающих переменные;

3) F - множество функциональных символов, $F = \bigcup_{k=1}^{\infty} F_k$, где F_k - множество символов для обозначения функций *k* переменных.

Вводя различные множества символов, мы постоянно обращаемся к их *интерпретации* («... символы, обозначающие...»). Строго говоря, это не нужно - можно (а с чисто формальной точки зрения - и должно) описать все правила действия с символами, не прибегая к их интерпретации, однако ее использование позволяет сократить изложение формальных правил за счет обращения к имеющемуся содержательному опыту.

Термы определяются индуктивно:

1) любой символ из $C \cup V$ есть терм;

2) если $t_1,\ldots,t_k$ - термы и $f \in F_k$, то $ft_1\ldots t_k$ - терм.

Множество термов *T* представляет собой объединение:

$$T = \bigcup_{i=0}^{\infty} T_i ,$$

где $T_0 = C \cup V$,

$$T_{i+1} = \bigcup_{k=1}^{\infty} \bigcup_{f \in F_k} \{ft_1\ldots t_k | t_1,\ldots,t_k \in \bigcup_{j=0}^{i} T_j\} \quad (T_1 \subseteq T_2 \subseteq \ldots \subseteq T_i \subseteq T_{i+1} \subseteq \ldots).$$

Удобно разбить T на непересекающиеся множества - слои $S_{i+1} = T_{i+1} \setminus T_i$ ($S_0 = T_0$). Элементы S_i будем называть термами глубины *i* или *i*-слойными термами. Множеству S_i принадлежат выражения, обозначающие те функции от термов предыдущих слоев $S_0,\ldots,S_{i-1}$, которые сами им не принадлежат[2].

Для оперирования с термами очень полезны две теоремы [4].

---

[2] Трудно удержаться от вольности речи - обращения к формально еще не введенной, но совершенно очевидной интерпретации («... обозначающие...»).

Теорема 1 (о построении термов). Каждый терм t единственным образом представляется в виде $ft_1 \ldots t_k$, где f - первый символ в t, $f \in F$, число k определяется по f ($f \in F_k$), а $t_1, \ldots, t_k$ - термы.

Эта теорема является точной формулировкой эквивалентности используемой бесскобочной и обычной записи.

Пусть u и v - выражения, то есть последовательности символов алфавита. Скажем, что u входит в v, если существуют такие выражения p и q (возможно, пустые), что v совпадает с puq.

Теорема 2 (о вхождении терма в терм). Пусть $f \in F_k$, $t_1, \ldots, t_k$ - термы, t представляется в виде $ft_1 \ldots t_k$, τ - терм и τ входит в t. Тогда или τ совпадает с t, или τ входит в одно из t_i $(i = 1, \ldots, k)$.

Доказываются эти теоремы элементарной индукцией по длине термов [4]. В доказательстве теоремы 2 выделяется лемма, представляющая и самостоятельный интерес.

Лемма 1. Каждое вхождение любого символа в терм τ начинает вхождение некоторого терма в τ.

Определим отношение между термами $t_1 \le t_2$ индуктивным образом «сверху вниз» - по глубине вхождения:

1) $t \le t$;

2) если t совпадает с $ft_1 \ldots t_k$, $f \in F_k$ и $t_1, \ldots, t_k$ - термы, то $t_1, \ldots, t_k \le t$;

3) если $t_1 \le t$ и $t \le t_2$, то $t_1 \le t_2$.

Согласно теореме 2, $t_1 \le t_2$ тогда и только тогда, когда t_1 входит в t_2.

Для каждого терма t определим множество входящих в него термов $S^t = \{\tau | \tau \le t\}$. Если $t \in S_i$, то при $0 \le k \le i$ непусты множества $S_k^t = S^t \cap S_k$. При этом множество S_i^t состоит из одного элемента - исходного терма t.

Свяжем с термом t ориентированный граф G_0^t с вершинами, взаимнооднозначно соответствующими термам из S^t. Будем одинаково обозначать вершины и соответствующие им термы. Пара вершин (τ_1, τ_2) образует ориентированное от τ_1 к τ_2

ребро графа G_0^t, если терм τ_2 имеет вид $ft_1\ldots t_k$, $f \in F_k$, $t_1,\ldots,t_k$ - термы и один из них t_i $(i=1,\ldots,k)$ совпадает с τ_1. Вершины графа G_0^t удобно располагать по слоям S_i^t.

Для произвольного графа G будем обозначать v(G) множество вершин, а e(G) - множество ребер G.

Возьмем для примера выражение для сложной функции

$$\varphi(x_1,x_2,x_3) = f_5(f_3(x_1,f_1(x_1,x_2)),f_4(f_1(x_1,x_2),f_2(x_2,x_3))). \qquad (3)$$

В принятой выше бесскобочной префиксной записи оно имеет вид

$$f_5f_3x_1f_1x_1x_2f_4f_1x_1x_2f_2x_2x_3, \qquad (3')$$

где все функциональные символы принадлежат F_2.

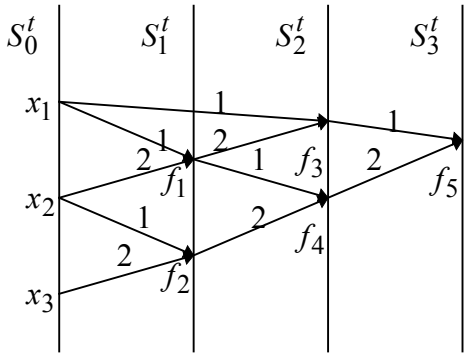

Рис. 1. Пример графа G_0^t. Около вершин и над ребрами указаны соответствующие метки

Граф G_0^t для этого терма изображен на рис. 1.

Для того, чтобы терм однозначно восстанавливался по графу, необходимы еще два дополнения.

1. Сопоставим каждой вершине $\tau \in$ v(G_0^t) метку $p(\tau)$ - символ алфавита. Если вершина принадлежит нулевому слою S_0^t, то ей соответствует терм, совпадающий с символом из $C \cup V$. Этот символ и сопоставляется вершине в качестве метки. Если вершина принадлежит S_i^t (i>0), то меткой служит функциональный символ: вершине τ сопоставляется $f \in F$, если τ имеет вид $ft_1\ldots t_k$, где $f \in F_k$, а $t_1,\ldots,t_k$ - термы.

2. Каждому ребру $(\tau',\tau) \in$ e(G_0^t), приходящему в вершину τ, сопоставим метку $P(\tau',\tau)$ - конечное множество натуральных чисел (номеров): пусть терм τ имеет вид $ft_1\ldots t_k$, где $f \in F_k$, а $t_1,\ldots,t_k$ - термы, тогда ребру (τ',τ) сопоставляется множество тех i ($1 \le i \le k$), для которых τ' совпадает с t_i. На практике в большинстве случаев эта метка состоит из одного номера, но возможны и другие варианты - так же, как функции вида $f(x,x)$. Для графических иллюстраций удобно ребра (τ',τ), имеющие в своей метке $P(\tau',\tau)$ больше одного номера, рисовать как пучок ребер, идущих от вершины τ' к вершине τ - по

одному такому ребру для каждого номера из $P(\tau',\tau)$; этот номер и будет меткой соответствующего ребра из пучка.

Граф G_0^t вместе со всеми метками будем обозначать G^t. На рис. 1 указаны соответствующие метки для разобранного примера.

Итак, для всякого терма t построен ориентированный граф G_0^t и две функции: первая сопоставляет каждой вершине $\tau \in \mathrm{v}(G_0^t)$ символ алфавита $p(\tau) \in C \cup V \cup F$, вторая (обозначим ее P) - каждому ребру $(\tau',\tau) \in \mathrm{e}(G_0^t)$ - конечное множество натуральных чисел $P(\tau',\tau)$. Отмеченный граф - набор (G_0^t, p, P) обозначаем G^t. Функции p и P удовлетворяют следующему ограничению:

А) если для данного $\tau \in S^t$ множество входящих ребер (τ',τ) непусто, то $p(\tau) = f^\tau \in F_k$ (является k-местным функциональным символом при некотором k) и семейство множеств

$$\{P(\tau',\tau) | (\tau',\tau) \in \mathrm{e}(G_0^t)\}$$

при фиксированном τ образует разбиение множества номеров $\{1,...,k\}$, то есть

$$P(\tau',\tau) \cap P(\tau'',\tau) = \varnothing \text{ при } \tau' \neq \tau'',$$

$$\forall \tau \qquad \bigcup_{\tau' :\, (\tau',\tau) \in \mathrm{e}(G_0^t)} P(\tau',\tau) = \{1,\ldots,k\}.$$

На этом завершается изложение основных формальных конструкций. Прежде, чем переходить к интерпретации, сформулируем теорему об эквивалентности графического и формульного представления термов.

Пусть G - конечный ориентированный граф, не имеющий ориентированных циклов, и в G существует и единственна такая вершина τ^*, к которой от любой вершины ведет ориентированный путь. Пусть, далее, заданы две функции: p - на множестве вершин G со значениями в множестве символов алфавита и P - на множестве ребер G со значениями в множестве конечных наборов натуральных чисел и выполнено условие А.

Теорема 3. Существует и единственен терм t, для которого $G^t = (G,p,P)$.

Доказательство проводится в два этапа. Сначала в *G* устанавливается послойная структура: строится разбиение множества вершин *G*: $s_0 \cup s_1 \cup ... \cup s_k$. Множество s_0 состоит из тех вершин, к которым не ведет ни одного ребра - из-за отсутствия ориентированных циклов такие вершины существуют. Множество s_{i+1} состоит из тех вершин, к которым ведут ребра только из элементов $s_0 \cup s_1 \cup ... \cup s_i$. Последний непустой элемент в последовательности $s_0, s_1, ..., s_k$ состоит из одной вершины τ^*, все предшествующие элементы этой последовательности непусты, а объединение всех s_i содержит все вершины *G*.

Доказательство основного утверждения теоремы проводится индукцией по числу слоев *k*.

Интерпретация сопоставляет терму сложную функцию. Она строится так. Задается некоторое множество *D* - область интерпретации. Каждой константе *c*, входящей в интерпретируемый терм *t*, сопоставляется элемент из *D* ($c \in D$), каждому *k*-местному функциональному символу *f*, входящему в *t*, сопоставляется функция *k* переменных $f: D^k \to D$ (мы сохраняем одинаковое обозначение для символов и их интерпретации, это вполне соответствует интуиции и не должно приводить к путанице). Каждой переменной, входящей в интерпретируемый терм *t*, сопоставляется переменная, пробегающая *D*. В результате терму *t* сопоставляется функция *n* переменных $F^t: D^n \to D$, где *n* - число различных символов переменных, входящих в *t*. Эта «сложная» функция получается суперпозицией «простых» функций, соответствующих функциональным символам.

Если $t \in S_0$, то есть терм является константой или переменной, то вместо сложной функции F^t получаем константу или тождественную функцию id, переводящую значение переменной в него же. Если $t \in S_1$, то соответствующая функция является «простой». Истинные сложные функции появляются, начиная со слоя S_2.

Заметим, что заданная интерпретация терма *t* одновременно определяет интерпретацию каждого терма, входящего в *t*.

Представим процесс вычисления сложной функции F^t с помощью отмеченного графа G^t. Строится два представления: статическое (все результаты промежуточных вычислений располагаются на графе) и динамическое (шаг за шагом, слой за слоем).

Пусть задана интерпретация терма t и определены значения всех переменных, входящих в t. Тогда для любого терма τ, входящего в t, также задана интерпретация и определены значения всех функций F^τ $(\tau \le t)$. Каждой вершине τ, входящей в граф G_0^t, сопоставляется значение функции F^τ - элемент D. При этом вершинам нулевого слоя соответствуют значения переменных и констант, а единственной вершине последнего (выходного) слоя - значение F^t. Будем называть элементы D, соответствующие вершинам, *значениями* этих вершин и обозначать их $Z(\tau)$.

Для каждой вершины τ, принадлежащей ненулевому слою, можно выписать *уравнение функционирования*, связывающее значения вершин. Пусть $p(\tau)= f^\tau$, $f^\tau \in F_k$ и $\mathrm{in}(\tau)$ - совокупность входящих в τ ребер. Напомним, что совокупность меток ребер, оканчивающихся в τ,

$$\{P(\tau',\tau)|(\tau',\tau)\in \mathrm{in}(\tau)\}$$

образует разбиение множества $\{1,2,...,k\}$.

Уравнение функционирования для вершины τ, принадлежащей ненулевому слою, имеет вид

$$Z(\tau) = f^\tau(z_1,...,z_k),\ z_i = Z(\tau') \text{ при } i \in P(\tau',\tau),\ (\tau',\tau) \in \mathrm{in}(\tau). \qquad (4)$$

В силу уравнения функционирования (4), если для входящих в τ ребер $(\tau',\tau)\in \mathrm{in}(\tau)$ известны значения $Z(\tau')$ и задана интерпретация символа $p(\tau)= f^\tau$ - метки вершины, то можно найти значение вершины $Z(\tau)$. На основании этого (очевидного) замечания строится динамическое представление вычисления сложной функции.

С каждой вершиной графа τ, принадлежащей ненулевому слою, ассоциируется автомат, вычисляющий функцию $f^\tau(z_1,...,z_k)$, где $f^\tau \in F_k$ - метка вершины τ. Автоматы срабатывают по слоям в дискретные моменты времени (такты) - автоматы i-го слоя в i-й момент времени. В начальный момент сформированы значения вершин нулевого слоя - известны значения переменных и констант. Они поступают на входы автоматов первого слоя в соответствии с нумерацией аргументов. После i-го такта функционирования определены значения вершин, принадлежащих слоям $S_0,...,S_i$. На

i+1-м такте автоматы i+1-го слоя вычисляют значения вершин i+1-го слоя, получая входные сигналы с предыдущих слоев по правилу (4) - в соответствии с метками входящих ребер.

3. Вычисления на ориентированном графе

Для дальнейшего полезно обобщение изложенных конструкций, предназначенное для описания одновременного вычисления нескольких сложных функций.

Пусть G - связный (но не обязательно ориентированно связный) ориентированный граф без ориентированных циклов. Как и выше, множество вершин G обозначаем v(G), множество ребер - e(G). Пусть, далее, каждой вершине $\tau \in$ v(G) сопоставлена метка - символ алфавита $p(\tau)$, а каждому ребру $(\tau', \tau) \in$ e(G) сопоставляется метка - конечное множество натуральных чисел $P(\tau', \tau)$ и выполнено условие согласования A: если для данного $\tau \in$ v(G) множество входящих ребер (τ', τ) непусто, то $p(\tau) = f^{\tau} \in F_k$ (является k-местным функциональным символом при некотором k) и семейство множеств $\{P(\tau', \tau) | (\tau', \tau) \in e(G)\}$ при фиксированном τ образует разбиение множества номеров $\{1,...,k\}$.

С помощью транзитивного замыкания G устанавливаем на множестве его вершин v(G) отношения частичного порядка: $\tau \geq \tau'$, если существует ориентированный путь, ведущий от τ' к τ. Из-за отсутствия ориентированных циклов это отношение антисимметрично: если $\tau \geq \tau'$ и $\tau' \geq \tau$, то τ совпадает с τ'. Минимальные вершины (к которым не ведет ни одного ребра) называем *входными*, а максимальные (от которых не идет ни одного ребра) - *выходными*. Обозначим множество входных вершин v_{in}, а выходных - v_{out}. Метки входных вершин являются символами переменных или констант, метки остальных вершин - функциональные символы.

Определим послойную структуру: v(G)= $s_0 \cup s_1 \cup ... \cup s_k$. Нулевой слой состоит из минимальных вершин, первый слой из минимальных вершин графа, получаемого удалением из G нулевого слоя и выходящих из него ребер и т.д. - i+1-й слой состоит из минимальных вершин графа, получаемого удалением из G всех вершин объединения $s_0 \cup s_1 \cup ... \cup s_i$ и содержащих их ребер. Последний слой состоит только из выходных вершин. Предыдущие слои также могут содержать выходные вершины.

С каждой вершиной графа $\tau \in$ v(G) однозначно связан терм (который мы, следуя традиции предыдущего раздела, также будем обозначать τ). Для его построения удалим

из G все вершины, кроме тех, от которых к τ можно пройти по ориентированному пути, а также связанные с ними ребра. Полученный граф обозначим G^τ: $v(G^\tau)=\{\tau' \in v(G) | \tau \geq \tau'\}$. Граф G^τ удовлетворяет условиям теоремы 3, поэтому по нему единственным образом восстанавливается терм (и соответствующая сложная функция).

Пусть задано множество D - область интерпретации и указана интерпретация всех символов, отмечающих вершины графа, а также значения переменных, отвечающих входным вершинам. Тогда по уравнениям функционирования (4) (они полностью сохраняются и для рассматриваемых графов) можно определить значения $Z(\tau)$ для всех вершин графа. В результате определяются значения всех сложных функций, формулы для которых являются термами, соответствующими вершинам графа G.

Описанные общие графы могут появляться в различных ситуациях. Для дальнейшего важно то, что такие графы возникают, если из графа вычисления сложной функции удалить один или несколько последних слоев.

4. Двойственное функционирование, дифференциальные операторы и градиент сложной функции

А. Производная сложной функции одного переменного

Основную идею двойственного функционирования можно понять уже на простейшем примере. Рассмотрим вычисление производной сложной функции одного переменного. Пусть заданы функции одного переменного $f_1(A)$,$f_2(A)$,...,$f_n(A)$. Образуем из них сложную функцию

$F(x)=f_n(f_{n-1}(...(f_1(x))...))$. (1)

Можно представить вычисление $F(x)$ как результат работы n автоматов, каждый из которых имеет один вход и выдает на выходе значение $f_i(A)$, где A - входной сигнал (рис.2, а). Чтобы построить систему автоматов, вычисляющую $F'(x)$, надо дополнить исходные автоматы такими, которые вычисляют функции $f_i'(A)$, где A - входной сигнал (важно различать производную f_i по входному сигналу, то есть по аргументу функции f_i, и производную сложной функции $f_i(A(x))$ по x; $f_i'(A)$ - производные по A).

Для вычисления $F'(x)$ потребуется еще цепочка из n-1 одинаковых автоматов, имеющих по два входа, по одному выходу и подающих на выход произведение входов. Тогда формулу производной сложной функции

$$\frac{dF}{dx} = f_n'(f_{n-1}(\ldots(f_1(x)\ldots)) \times f_{n-1}'(\ldots(f_1(x)\ldots)\times\ldots\times f_1'(x)$$

можно реализовать с помощью сети автоматов, изображенной на рис. 2, б. Сначала по этой схеме вычисления идут слева направо: на входы f_1 и f_1' подаются значения x, после вычислений $f_1(x)$ это число подается на входы f_2 и f_2' и т.д. В конце цепочки оказываются вычисленными все f_i (f_{i-1} (...)) и f_i'(f_{i-1} (...)).

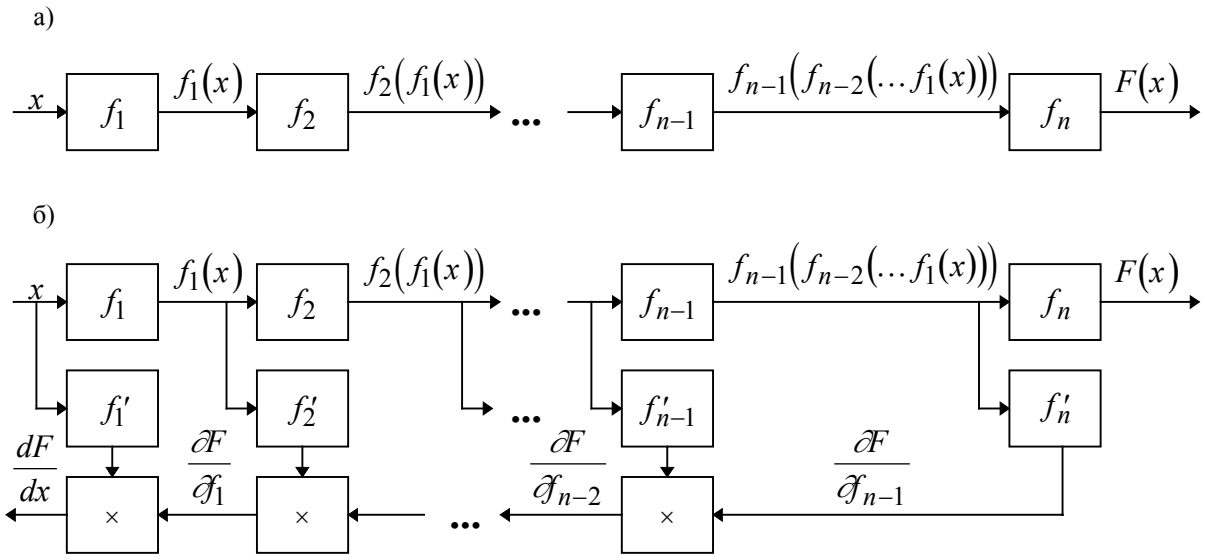

Рис.2. Схематическое представление вычисления сложной функции одного переменного и ее производных.

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 в обратном направлении - справа налево. Это обратное движение есть последовательное вычисление попарных произведений и передача их налево. В конце получаем dF/dx.

Б. Двойственное функционирование и быстрое дифференцирование

Вычисление градиента сложной функции многих переменных также будет представлено как некоторый вычислительный процесс, в ходе которого сигналы перемещаются в обратном направлении - от выходных элементов графа к входным. И так же, как при вычислении сложной функции, будет явно представлено прохождение каждого узла (подобно тому, как это сделано в уравнении функционирования (4)), а весь процесс в целом будет складываться из таких элементарных фрагментов за счет

структуры связей между узлами. Сама эта структура - та же самая, что и для прямого процесса.

Всюду в этом разделе область интерпретации - действительная прямая, а все функции дифференцируемы.

Основной инструмент при построении двойственного функционирования - формула для дифференцирования «двухслойной» сложной функции нескольких переменных

$$\frac{\partial f(A_1(x_1,\ldots,x_n),A_2(x_1,\ldots,x_n),\ldots,A_k(x_1,\ldots,x_n))}{\partial x_i}=\sum_{j=1}^{k}\frac{\partial f}{\partial A_j}\times\frac{\partial A_j}{\partial x_i}. \qquad (5)$$

При проведении индукции по числу слоев нам придется использовать графы с несколькими выходными вершинами, описанные в предыдущем разделе, поэтому сразу будем рассматривать этот более общий случай.

Итак, пусть G - связный (но не обязательно ориентированно связный) ориентированный граф без ориентированных циклов. Как и выше, v(G) - множество вершин G, e(G) - множество ребер. Пусть, далее, каждой вершине $\tau\in$v(G) сопоставлена метка - символ алфавита $p(\tau)$, а каждому ребру $(\tau',\tau)\in$e(G) сопоставляется метка - конечное множество натуральных чисел $P(\tau',\tau)$ и выполнено условие согласования A: если для данного $\tau\in$v(G) множество входящих ребер (τ',τ) непусто, то $p(\tau)=f^{\tau}\in F_k$ (является k-местным функциональным символом при некотором k) и семейство множеств $\{P(\tau',\tau)|(\tau',\tau)\in$e$(G)\}$ при фиксированном τ образует разбиение множества номеров $\{1,...,k\}$. Для каждой вершины $\tau\in$v(G) обозначим множество входящих в нее ребер in(τ), а выходящих из нее - out(τ).

Пусть указана интерпретация всех символов, отмечающих вершины графа, значения переменных, отвечающих входным вершинам и уравнениям функционирования (4) определены значения $Z(\tau)$ для всех вершин графа. В результате определены значения всех сложных функций, формулы для которых являются термами, соответствующими вершинам графа G. Процесс вычисления $Z(\tau)$ будем называть *прямым* в противовес *обратному* или *двойственному*, к описанию которого мы переходим.

При заданных $Z(\tau)$ для каждой вершины и каждого ребра строятся *переменные двойственного функционирования* (или, более кратко, *двойственные переменные*). Будем обозначать их $\mu(\tau)$ для вершин и $\mu(\tau',\tau)$ - для ребер.

Для выходных вершин $\mu(\tau)$ являются независимыми переменными. Пусть их значения заданы. Для вершины τ, не являющейся выходной, значение $\mu(\tau)$ есть сумма значений двойственных переменных, соответствующих выходящим из τ ребрам:

$$\mu(\tau) = \sum_{(\tau,\tau')\in \mathrm{out}(\tau)} \mu(\tau,\tau') . \tag{6}$$

Для ребра (τ',τ) значение $\mu(\tau',\tau)$ определяется согласно формуле (5):

$$\mu(\tau',\tau) = \mu(\tau) \sum_{i\in P(\tau',\tau)} \frac{\partial f^{\tau}(t_1,\ldots,t_k)}{\partial t_i} . \tag{7}$$

В формуле (7) $f^{\tau} = p(\tau) \in F_k$ - метка ребра τ, $t_1,\ldots,t_k$ - аргументы «простой» функции f^{τ}, а производные f^{τ} берутся при условиях, определяемых прямым процессом:

$$t_j = Z(\theta) \text{ при } \theta \in \mathrm{in}(\tau),\ j \in P(\theta,\tau) . \tag{8}$$

Для каждого $j\in\{1,\ldots,k\}$ такое θ существует и единственно в силу того, что метки входящих в вершину τ ребер образуют разбиение множества номеров $\{1,\ldots,k\}$.

В самом распространенном случае все метки ребер $P(\tau',\tau)$ содержат по одному элементу. В этом случае формула (7) приобретает особенно простой вид

$$\mu(\tau',\tau) = \mu(\tau)\frac{\partial f^{\tau}(t_1,\ldots,t_k)}{\partial t_{i(\tau',\tau)}} \text{ где } P(\tau',\tau) = \{i(\tau',\tau)\} . \tag{7'}$$

Используя (6)-(8), можно записать правила вычисления двойственных переменных для вершин, не использующие двойственных переменных для ребер:

$$\mu(\tau)\left(= \sum_{(\tau,\theta)\in \mathrm{out}(\tau)} \mu(\tau,\theta)\right) = \sum_{(\tau,\theta)\in \mathrm{out}(\tau)} \mu(\theta) \sum_{i\in P(\tau,\theta)} \frac{\partial f^{\theta}(t_1,\ldots,t_k)}{\partial t_i}, \tag{9}$$

где производные $f^{\theta} = p(\theta) \in F_k$ берутся при условиях

$$t_j = Z(\eta) \text{ при } \eta \in \mathrm{in}(\theta),\ j \in P(\eta,\theta) . \tag{10}$$

Греческими буквами τ, θ, η здесь обозначены вершины графа.

Опять же, в распространенном случае, когда все $P(\tau',\tau)$ одноэлементны, применяем (7′) вместо (7) и получаем

$$\mu(\tau)\left(= \sum_{(\tau,\theta)\in \mathrm{out}(\tau)} \mu(\tau,\theta)\right) = \sum_{(\tau,\theta)\in \mathrm{out}(\tau)} \mu(\theta)\frac{\partial f^{\theta}(t_1,\ldots,t_k)}{\partial t_{i(\tau,\theta)}} \text{ где } P(\tau,\theta) = \{i(\tau,\theta)\} \tag{9'}$$

Напомним, что каждой вершине τ, принадлежащей ненулевому слою графа G, соответствуют терм τ и сложная функция F^τ от независимых переменных и констант, отмечающих вершины нулевого слоя G.

Процесс вычисления двойственных переменных организуется послойно от выходных вершин к входным и часто называется процессом *обратного распространения* (back propagation) или просто обратным процессом.

Теорема 5. Пусть задана интерпретация всех символов, отмечающих вершины графа G, определены значения независимых переменных (а также констант), соответствующих вершинам входного слоя v_{in} и значения независимых переменных двойственного функционирования $\mu(\tau)$ для вершин выходного слоя $\mathrm{v}_{\mathrm{out}}$. Тогда для любого $\theta \in \mathrm{v}_{\mathrm{in}}$

$$\mu(\theta) = \sum_{\tau \in \mathrm{v}_{\mathrm{out}}} \mu(\tau) \frac{\partial F^\tau}{\partial x_\theta}. \qquad (11)$$

где F^τ - соответствующая вершине τ функция от независимых переменных и констант, отмечающих вершины нулевого слоя G, x_θ - соответствующая вершине $\theta \in \mathrm{v}_{\mathrm{in}}$ переменная или константа, а производные в (11) берутся при фиксированных значениях прочих переменных и констант.

Доказательство проводится индукцией по числу слоев. Для графов из двух слоев - нулевого и первого - теорема очевидна. Спуск от i+1-слойных графов к i-слойным производится с помощью формулы 5.

Прохождение вершины графа и прилегающих к ней ребер при прямом и обратном процессах проиллюстрировано на рис. 3.

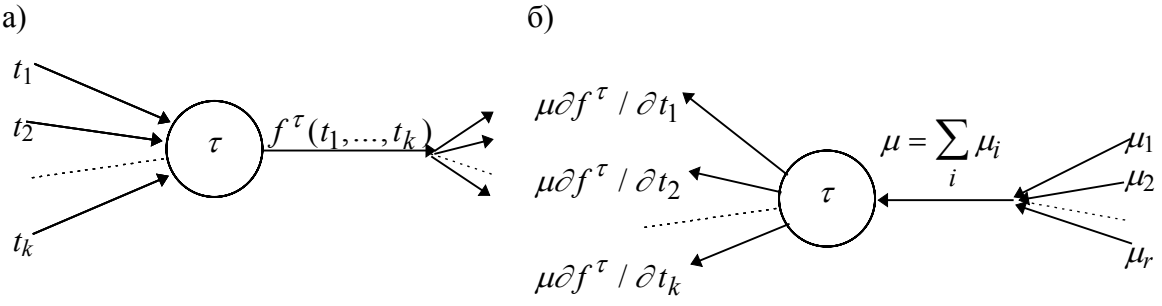

Рис. 3. Прохождение вершины τ в прямом (а) и обратном (б) направлении.

5. Сложность вычисления функции и ее градиента

Подсчитаем теперь число операций, необходимых для вычисления всех двойственных переменных $\mu(\tau)$ для вершин и $\mu(\tau',\tau)$ - для ребер.

Во-первых, нужно вычислить все частные производные

$$\frac{\partial f^{\theta}(t_1,\ldots,t_k)}{\partial t_i}$$

для всех вершин θ и всех k аргументов «простой» функции, соответствующей каждой вершине. Число таких производных равно сумме числа аргументов для всех функциональных символов, соответствующих вершинам графа, то есть следующей величине E:

$$E = \sum_{(\tau,\theta)\in e(G)} P(\tau,\theta).$$

Договоримся в этом разделе отображать ребра (τ',τ), имеющие в своих метках $P(\tau',\tau)$ больше одного номера, как пучки ребер, идущих от вершины τ' к вершине τ - по одному такому ребру для каждого номера из $P(\tau',\tau)$. Число E просто равно числу ребер в графе. Число необходимых умножений и число сложений также не превосходят E.

Количество вычислений «простых» функций при вычислении сложной равно числу вершин графа. Обозначим его V. Отношение E/V дает представление об отношении вычислительных затрат на вычисление градиента к затратам на вычисление функции. Чтобы последовательно использовать эту оценку, а также искать те функции, для которых вычисление градиента еще проще, необходимо зафиксировать исходные предположения. Будем обозначать T_f затраты на вычисление f.

1. Для каждой вершины графа, соответствующей ей функции f, и любого аргумента этой функции x справедлива оценка $T_f \cong T_{\partial f/\partial x}$;
2. Для функций $f = f^{\tau}(z_1,\ldots,z_k)$, соответствующих вершинам графа, $T_f \geq T_{\Sigma}$, $\Sigma = \sum_{i=1}^{k} z_i$, то есть сумма переменных - простейшая функция;
3. Умножение и сложение имеют примерно одинаковую сложность.

В предположениях 1-3 зафиксирован тот уровень точности, с которым будут делаться оценки. При формальном использовании отношения «a примерно равно b» неизбежны парадоксы типа «куча»: один камень не куча, если n камней - не куча, то и n+1 - тоже, следовательно... . Чтобы избежать этого и сделать рассуждения более наглядными, поступим следующим образом. Сложность «простой» функции k

переменных и любой ее частной производной оценим как ck, где c - некоторая константа, $c \geq 1$; сложность суммы k слагаемых (и произведения k сомножителей) определим как k-1. Последнее вычитание единицы имеет смысл при рассмотрении большого числа сумм, содержащих мало слагаемых.

Пусть, как и выше, E- число ребер графа, V- число вершин, V_{out} - число выходных вершин (им не соответствует ни одной суммы в (6)). Сложность прямого прохождения графа (вычисления функций) оценивается как cE.

Обратное прохождение графа при вычислении градиентов складывается из трех слагаемых:

1. Вычисление всех частных производных простых функций, соответствующих вершинам графа. Необходимо вычислить E таких производных. Сложность вычисления одной такой производной для вершины τ, имеющей $|\text{in}(\tau)|$ входящих ребер, оценивается как $c|\text{in}(\tau)|$. Оценка сложности вычисления всех этих производных дается следующей суммой $T_{\text{Dif}} = c\sum_{\tau} |\text{in}(\tau)|^2$ $(\leq cE^2)$.
2. Вычисление всех произведений (7) на ребрах - E произведений (в связи с тем, что мы в данном разделе передачу сигнала на разные входы автомата, вычисляющего $f^{\tau}(z_1,\ldots,z_k)$, обозначаем различными ребрами, уравнения (7), сохраняя прежнюю внешнюю форму, преобразуются так, что в суммах остается по одному слагаемому, остальное суммирование переносится к вершинам (6)).
3. Вычисление всех сумм (6) - сложность равна E-(V- V_{out}).

Итого, полная сложность обратного прохождения сигналов оценивается как

$$T = T_{\text{Dif}} + 2E - (V - V_{\text{out}}) = c\sum_{\tau} |\text{in}(\tau)|^2 + 2E - (V - V_{\text{out}}).$$

Основную роль в последней сумме играет первое слагаемое - именно им определяется сложность обратного распространения по графу (если все $|\text{in}(\tau)|=1$, то $T_{\text{Dif}} = cE$, а уже в случае, когда $|\text{in}(\tau)|=2$, мы получаем $T_{\text{Dif}} = 2cE$). Поэтому можно положить $T \approx T_{\text{Dif}}$ (строго говоря, $3T_{\text{Dif}} \geq T \geq T_{\text{Dif}}$, однако различия в два-три раза для нас сейчас несущественны).

Если характерное число переменных у простых функций, соответствующих вершинам графа, равно m (то есть $|\text{in}(\tau)|=m$), то для вычисляемой по графу сложной

функции F можно оценить отношение затрат на вычисление F и grad$F$ следующим образом:

$$T_{\mathrm{grad}F} \cong c\frac{E}{k}k^2 = ckE,\ T_F \cong cE,\ \frac{T_{\mathrm{grad}F}}{T_F} \cong k\,.$$

Эта оценка, конечно, лучше, чем полное число переменных n, но все же искомое отношение оценивается примерно как отношение числа ребер к числу вершин E/V (с точностью до менее значимых слагаемых). Если нас интересуют графы с большим числом связей, то для сохранения эффективности вычисления градиентов нужно ограничиваться специальными классами простых функций. Из исходных предположений 1-3 наиболее существенно первое ($T_f \cong T_{\partial f/\partial x}$). Зададимся вопросом: для каких функций f вычисление частных производных вообще не требует вычислений?

Ответ очевиден: общий вид таких функций

$$f(z_1,\ldots z_n) = \sum_{i\in P_1} z_i + \sum_{j\in P_2, k\in P_3} z_j z_k\,, \qquad (12)$$

где множества индексов P_1, P_2, P_3 не пересекаются и, кроме того, P_2, P_3 имеют одинаковое число элементов. Значения всех частных производных таких функций уже известны, а источники (адреса) этих значений указываются связями графа. Какие-то значения z_k во второй сумме могут быть константами (значения нулевого слоя), какие-то - независимыми переменными (также нулевой слой) или значениями, найденными при промежуточных вычислениях.

В общем случае функции (12) билинейны. Их частный случай - линейные функции: если индексам из P_2 в (12) соответствуют константы, то функция f - линейная.

И функции вида (12), и соответствующие им вершины будем называть *квазилинейными*.

Пусть интерпретация функциональных символов такова, что в графе вычислений присутствуют только вершины двух сортов: квазилинейные или с одной входной связью (соответствующие простым функциям одного переменного). Обозначим количества этих вершин V_q и V_1 соответственно. Оценка сложности вычисления градиента для таких графов принципиально меняется. Обратное функционирование в этом случае требует следующих затрат:

1) Поиск всех производных функций одного переменного, соответствующих вершинам графа (число таких производных равно V_1), сложность поиска одной производной оценивается как c.
2) Вычисление всех произведений (7) на ребрах - E произведений.
3) Вычисление всех сумм (6) - сложность равна E-(V- V_{out}).

Итого, суммарная сложность обратного прохождения сигналов оценивается как

$$T_{\text{grad}F} = cV_1+2E-(V- V_{\text{out}}).$$

Оценим сложность вычислений при прямом распространении:

1) Для любой квазилинейной вершины τ сложность вычисления функции оценивается как $|\text{in}(\tau)|$-1,
2) Для любой вершины с одной входной связью сложность оценивается как c.

Итак, для прямого распространения сложность оценивается как

$$T_F=cV_1+(E- V_1- V_q).$$

С ростом числа связей $\dfrac{T_{\text{grad}F}}{T_F} \to 2$!!!

Графы вычислений (с заданной интерпретацией функциональных символов), в которых присутствуют только вершины двух сортов: квазилинейные или с одной входной связью (соответствующие простым функциям одного переменного) играют особую роль. Будем называть их ***существенно квазилинейными***. Для функций, вычисляемых с помощью таких графов, затраты на вычисление вектора градиента примерно вдвое больше, чем затраты на вычисление значения функции (всего!). При этом число связей и отношение E/V могут быть сколь угодно большими. Это достоинство делает использование существенно квазилинейных графов весьма притягательным во всех задачах оптимизации. Их частным случаем являются нейронные сети, для которых роль квазилинейных вершин играют адаптивные линейные сумматоры.

Таким образом, обратное прохождение адаптивного сумматора требует таких же затрат, как и прямое. Для других элементов стандартного нейрона обратное распространение строится столь же просто: точка ветвления при обратном процессе превращается в простой сумматор, а нелинейный преобразователь $x \mapsto f(x)$ в линейную связь $\mu \mapsto f'(x)\mu$ с коэффициентом связи $f'(x)$. Поэтому для нейронных

сетей обратное распространение выглядит особенно просто. Детали можно найти в различных руководствах, например [5,6]

Отличие общего случая от более привычных нейронных сетей состоит в том, что можно использовать билинейные комбинации (12) произвольных уже вычисленных функций, а не только линейные функции от них с постоянными коэффициентами-весами.

В определенном смысле квазилинейные функции (12) вычисляются линейными сумматорами с весами 1 и z_j ($j \in P_2$) и аргументами z_i ($i \in P_1$) и z_k ($k \in P_3$), только веса не обязательно являются константами, а могут вычисляться на любом слое.

Естественно возникает вопрос о вычислительных возможностях сетей, все вершины которых являются квазилинейными. Множество функций, вычисляемых такими сетями, содержит все координатные функции и замкнуто относительно сложения, умножения на константу и умножения функций. Следовательно оно содержит и все многочлены от любого количества переменных. Любая непрерывная функция на замкнутом ограниченном множестве в конечномерном пространстве может быть сколь угодно точно приближена с их помощью.

6. Двойственность по Лежандру и смысл двойственных переменных

Просматривая текст предыдущих разделов, я убедился, что подробное изложение и элементы стиля математической логики могут даже очевидное сделать сложным. В данном разделе описывается связь двойственного функционирования сетей автоматов с преобразованием Лежандра и неопределенными множителями Лагранжа. Это изложение ориентировано на читателя, знакомившегося с лагранжевым и гамильтоновым формализмами и их связью (в классической механике или вариационном исчислении). Фактически на двух страницах будет заново изложен весь предыдущий материал главы.

Переменные обратного функционирования μ 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, встречавшихся нам в предыдущих разделах и главах, можно выделить три группы переменных:

внешние входные сигналы $x_{...}$,

переменные функционирования - значения на выходах всех элементов сети $f_{...}$,

переменные обучения $a_{...}$

(многоточиями заменяются различные наборы индексов).

Объединим их в две группы - вычисляемые величины $y_{...}$ - значения $f_{...}$ и задаваемые - $b_{...}$ (включая $a_{...}$ и $x_{...}$). Упростим индексацию, перенумеровав f и b натуральными числами: $f_1,...,f_N$; $b_1,...,b_M$.

Пусть функционирование системы задается набором из N уравнений

$$\psi_i(y_1,...,y_N,b_1,...,b_M)=0 \ (i=1,...,N). \qquad (13)$$

Для послойного вычисления сложных функций вычисляемые переменные - это значения вершин для всех слоев, кроме нулевого, задаваемые переменные - это значения вершин первого слоя (константы и значения переменных), а уравнения функционирования имеют простейший вид (4), для которого

$$\psi_i = Z(\tau) - f(z_1,...,z_k),$$
$$\text{где } z_i = Z(\tau') \text{ при } i \in P(\tau',\tau).$$

Предполагается, что система уравнений (13) задает способ вычисления y_i .

Пусть имеется функция (лагранжиан) $H(y_1,...,y_N,b_1,...,b_M)$. Эта функция зависит от b и явно, и неявно - через переменные функционирования y. Если представить, что уравнения (13) разрешены относительно всех y ($y=y(b)$), то H можно представить как функцию от b:

$$H=H_1(b)=H(y_1(b),...,y_N(b),b). \qquad (14)$$

где b - вектор с компонентами b_i .

Для задачи обучения требуется найти производные $D_i=\partial H_1(b)/\partial b_i$. Непосредственно и явно это сделать трудно.

Поступим по-другому. Введем новые переменные $\mu_1,...,\mu_N$ (множители Лагранжа) и производящую функцию W:

$$W(y,b,\mu) = H(y,b) + \sum_{i=1}^{N\mu} \mu_i \psi_i(y,b).$$

В функции W аргументы y, b и μ - независимые переменные.

Уравнения (13) можно записать как

$$\frac{\partial W}{\mu_i} = 0,\ (i = 1,\ldots,N). \tag{15}$$

Заметим, что для тех y, b, которые удовлетворяют уравнениям (13), при любых μ

$$W(y,b,\mu) \equiv H(y,b). \tag{16}$$

Это означает, что для истинных значений переменных функционирования y при данных b функция $W(y,b,\mu)$ совпадает с исследуемой функцией H.

Попытаемся подобрать такую зависимость $\mu_i(b)$, чтобы, используя (16), получить для $D_i=\partial H_1(b)/\partial b_i$ наиболее простые выражения . На многообразии решений (15)

$$D_i = \frac{\partial H(y(b),b)}{\partial b_i} = \frac{\partial W(y(b),b,\mu(b))}{\partial b_i}.$$

Поэтому

$$\begin{aligned} D_i &= \frac{\partial H(y,b)}{\partial b_i} + \sum_{j=1}^{N} \frac{\partial H(y,b)}{\partial y_j} \times \frac{\partial y_j(b)}{\partial b_i} = \\ &= \frac{\partial W(y,b,\mu)}{\partial b_i} + \sum_{j=1}^{N} \frac{\partial W(y,b,\mu)}{\partial y_j} \times \frac{\partial y_j(b)}{\partial b_i} + \sum_{j=1}^{N} \frac{\partial W(y,b,\mu)}{\partial \mu_j} \times \frac{\partial \mu_j(b)}{\partial b_i} = \\ &= \sum_{j=1}^{N} \left(\frac{\partial H(y,b)}{\partial y_j} + \sum_{k=1}^{N} \mu_k \frac{\partial \psi_k(y,b)}{\partial y_j} \right) \times \frac{\partial y_j(b)}{\partial b_i} + \sum_{j=1}^{N} \mu_j \frac{\partial \psi_j(y,b)}{\partial b_i} + \frac{\partial H(y,b)}{\partial b_i}\ . \end{aligned} \tag{17}$$

Мы всюду различаем функцию $H(y,b)$, где y и b - независимые переменные, и функцию только от переменных b $H(y(b),b)$, где $y(b)$ определены из уравнений (13). Аналогичное различение принимается для функций $W(y,b,\mu)$ и $W(y(b),b,\mu(b))$.

Произвол в определении $\mu(b)$ надо использовать наилучшим образом - все равно от него придется избавляться, доопределяя зависимости. Если выбрать такие μ, что слагаемые в первой сумме последней строки выражения (17) обратятся в нуль, то формула для D_i резко упростится. Положим поэтому

$$\frac{\partial H(y,b)}{\partial y_j} + \sum_{k=1}^{N} \mu_k \frac{\partial \psi_k(y,b)}{\partial y_j} = 0\,. \tag{18}$$

Это - система уравнений для определения μ_k (k=1,...,N). Если μ определены согласно (18), то

$$D_i = \frac{\partial H(y,b)}{\partial b_i} + \sum_{j=1}^{N} \mu_j \frac{\partial \psi_j(y,b)}{\partial b_i}$$

это - в точности те выражения, которые использовались при поиске производных сложных функций. В наших вычислениях мы пользовались явным описанием функционирования. Метод множителей Лагранжа допускает и менее явные описания сетей.

В математическом фольклоре бытует такой тезис: сложная задача оптимизации может быть решена только при эффективном использовании двойственности. Методы быстрого дифференцирования являются яркой иллюстрацией этого тезиса.

7. Оптимизационное обучение нейронных сетей

Когда можно представить обучение нейронных сетей как задачу оптимизации? В тех случаях, когда удается *оценить* работу сети. Это означает, что можно указать, хорошо или плохо сеть решает поставленные ей задачи и оценить это «хорошо или плохо» количественно. Строится *функция оценки*. Она, как правило, явно зависит от выходных сигналов сети и неявно (через функционирование) - от всех ее параметров. Простейший и самый распространенный пример оценки - сумма квадратов расстояний от выходных сигналов сети до их требуемых значений:

$$H = \frac{1}{2} \sum_{\tau \in \mathrm{v}_{\mathrm{out}}} (Z(\tau) - Z^*(\tau))^2 ,$$

где $Z^*(\tau)$ - требуемое значение выходного сигнала.

Другой пример оценки- качество классификации в сетях Кохонена. В этом случае ответы заранее неизвестны, но качество работы сети оценить можно.

Устройство, вычисляющее оценку, надстраивается над нейронной сетью и градиент оценки может быть вычислен с использованием описанного принципа двойственности.

В тех случаях, когда оценка является суммой квадратов ошибок, значения независимых переменных двойственного функционирования $\mu(\tau)$ для вершин выходного слоя $\mathrm{v}_{\mathrm{out}}$ при вычислении градиента H устанавливаются равными

$$\mu(\tau) = \frac{\partial H}{\partial Z(\tau)} = (Z(\tau) - Z^*(\tau)) \text{ -} \qquad (19)$$

на вход при обратном функционировании поступают ошибки выходных сигналов! Это обстоятельство настолько впечатлило исследователей, что они назвали метод

вычисления градиента оценки методом обратного распространения ошибок. Впрочем, после формулы Уидроу, описанной в главе 2, формула (19) должна быть очевидной.

Для обучения используется оценка, усредненная по примерам с известным ответом.

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. Вообще говоря, желательно иметь алгоритмы, которые требуют затрат памяти порядка Kn, K=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 нейрокомпьютерной 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 отличаются 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 особо существенной, а качественные изменения должны быть исключены. Это означает, что в достаточно большой окрестности найденной точки общего минимума оценок значения этих функций незначительно отличаются от минимальных. Мало того, что должна быть найдена точка общего минимума, так она еще должна лежать в достаточно широкой низменности, где значения всех минимизируемых функций близки к минимуму. Для решения этой задачи нужны специальные средства.

Итак, имеем четыре специфических ограничения, выделяющих обучение нейрокомпьютера из общих задач оптимизации: астрономическое число параметров, необходимость высокого параллелизма при обучении, многокритериальность решаемых задач, необходимость найти достаточно широкую область, в которой значения всех минимизируемых функций близки к минимальным. В остальном - это просто задача оптимизации и многие классические и современные методы достаточно естественно ложатся на структуру нейронной сети.

Заметим, кстати, что если вести оптимизацию (минимизацию ошибки), меняя параметры сети, то в результате получим решение задачи аппроксимации. Если же ведется минимизация целевой некоторой функции и ищутся соответствующие значения переменных, то в результате решаем задачу оптимизации (хотя формально это одна и та же математическая задача и разделение на параметры и переменные определяется логикой предметной области, а с формальной точки зрения разница практически отсутствует).

Значительное число публикаций по методам обучения нейронных сетей посвящено переносу классических алгоритмов оптимизации (см., например, [7,8]) на нейронные сети или поиску новых редакций этих методов, более соответствующих описанным ограничениям - таких, например, как метод виртуальных частиц [5,6]. Существуют обширные обзоры и курсы, посвященные обучению нейронных сетей (например, [9,10]). Не будем здесь останавливаться на обзоре этих работ - если найден градиент, то остальное приложится.

Работа над главой была поддержана Красноярским краевым фондом науки, грант 6*F*0124.

Литература

1.Rumelhart D.E., Hinton G.E., Williams R.J. Learning internal representations by error propagation. // Parallel Distributed Processing: Exploration in the Microstructure of Cognition, D.E.Rumelhart and J.L.McClelland (Eds.), vol. 1, Cambridge, MA: MIT Press, 1986. PP. 318 - 362.

2.Rummelhart D.E., Hinton G.E., Williams R.J. Learning representations by back-propagating errors // Nature, 1986. V. 323. P. 533-536.

3.Барцев С.И., Охонин В.А. Адаптивные сети обработки информации. Препринт ИФ СО АН СССР, Красноярск, 1986, №59Б, 20 с.

4.Шенфилд Дж. Математическая логика. М.: Наука, 1975. 528 с.

5.Горбань А.Н. Обучение нейронных сетей. М.": изд. СССР-США СП "ПараГраф", 1990. 160 с. (English Translation: AMSE Transaction, Scientific Siberian, A, 1993, Vol. 6. Neurocomputing, PP. 1-134).

6.Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. Новосибирск: Наука (Сиб. отделение), 1996. 276 с.

7.Химмельблау Д. Прикладное нелинейное программирование. М.: Мир, 1975. 534 с.

8.Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. М.: Мир,1985. 509 с.

9.Zurada J. M. Introduction to artificial neural systems. PWS Publishing Company, 1992. 785 pp.

10.Haykin S. Neural networks. A comprehensive foundations. McMillan College Publ. Co. N.Y., 1994. 696 pp.