Лекция 5

Постановка и возможные пути решения задачи обучения нейронных сетей

Частичная задача обучения

Пусть у нас есть некоторая нейросеть N. В процессе функционирования эта нейронная сеть формирует выходной сигнал $\bar{y} \in Y$ в соответствии с входным сигналом $\bar{x} \in X$, реализуя некоторую функцию $g : X \to Y, \quad \bar{y} = g(\bar{x})$. Если архитектура сети задана, то есть, заданы количество и вид нейронов сети, а также структура связи между ними, то вид функции g определяется значениями синаптических весов, смещений сети и параметров функций активации нейронов. Обозначим буквой G множество всех возможных функций g, соответствующих заданной архитектуре сети.

Пусть решение некоторой задачи - функция $r : X \to Y$. Рассмотрим случай, когда функция r задана парами входных-выходных векторов $(\bar{x}^1, \bar{y}^1), (\bar{x}^2, \bar{y}^2), \mathbf{K}, (\bar{x}^M, \bar{y}^M)$, для которых $\bar{y}^m = r(\bar{x}^m)$, $(m = 1,2,\mathbf{K},M)$.

Определение. Набор пар $(\bar{x}^1, \bar{y}^1), (\bar{x}^2, \bar{y}^2), \mathbf{K}, (\bar{x}^M, \bar{y}^M)$ таких, что $\bar{y}^m \in X$, $\bar{x}^m \in Y$ и $\bar{y}^m = r(\bar{x}^m)$, будем называть обучающей выборкой функции $r : X \to Y$.

Определим функцию ошибки $D_r : G \to R$. Эта функция, показывает для каждой из функций g степень близости к r. Функцию ошибки также часто называют целевой функцией обучения нейронной сети.

Пример 5.1. Пусть $g(x)$ и $r(x)$ непрерывные, интегрируемые функции определённые на отрезке $[a,b]$.

Тогда определим $D_r(g) = \int_a^b (g(t) - r(t))^2 dt$.

Пример 5.2. Пусть $g(\bar{x})$ - некоторая непрерывная на R^n функция, а $r(\bar{x})$ задана обучающей выборкой $(\bar{x}^i, \bar{y}^i), i = 1, 2, ..., M$. Тогда целевую функцию можно определить как

$$D_r(g) = \frac{1}{2} \sum_{i=1}^{M} (g(\bar{x}^i) - r(\bar{x}^i))^2 .$$

Решить поставленную задачу с помощью нейронной сети заданной архитектуры - это значит построить (синтезировать) функцию $g' \in G$, подобрав параметры нейронов (как

правило это синаптические веса и смещения) таким образом, чтобы функционал D обращался в минимум для всех пар $(\bar{x}^m, \bar{y}^m)$:

$$D_r(g') = \min_{g \in G} D_r(g)$$

Задача обучения определяется совокупностью четырёх элементов: $\langle N, r, G, D_r \rangle$, где

N - нейронная сеть, реализующая функцию $g \in G$;

$r : X \to Y$ определяет желаемый результат обучения (часто определяется обучающей выборкой);

G – множество функций $g : X \to Y$. Это множество полностью определяется архитектурой сети;

$D_r(g)$ - функция ошибки, показывающая для каждой сети N степень близости к r.

Необходимо найти нейронную сеть N' определяющую $g' \in G$, минимизирующую функцию *D:*

$$D_r(g') = \min_{g \in G} D_r(g).$$

Обучение – это, как правило, итерационная процедура. На каждой итерации происходит уменьшение функции ошибки. Обучение требует длительных вычислений.

В этой лекции мы рассматриваем только частичную задачу обучения, то есть задачу, в условиях которой однозначно задана архитектура сети. Дело в том, что если расширить исходные условия и предположить, что мы не знаем количества нейронов, их вида и структуры связей, то задача обучения переводится из класса задач отыскания экстремума линейно-параметризированного нелинейного отображения в класс задач многомерной нелинейной оптимизации с множеством решений.

Различают три основных вида стратегии обучения: «с учителем», «без учителя», смешанную. В первом случае, нейросеть настраивают некоторым алгоритмом по заданной обучающей выборке. Во втором случае обучающей выборки не требуется, а сеть в процессе обучения настраивается в соответствии с некоторым правилом. В третьем случае часть параметров сети настраивается по заданной обучающей выборке, а другая – без использования знаний о правильных ответах.

Если выбраны множество обучающих примеров - пар $(\bar{x}^m, \bar{y}^m)$ $(m = 1,2,\mathbf{K},M)$ - и способ вычисления функции ошибки D, обучение нейронной сети при априорно заданной архитектуре – это задача отыскания экстремума линейно-параметризированного

нелинейного отображения. Размерность задачи зависит от вида функции r (количества пар обучающей выборки) и архитектуры нейросети. Для сетей небольшой размерности (порядка нескольких сот нейронов) и количества пар обучающей выборки не более ста - количество итераций обучения может быть от нескольких тысяч до 10^8.

Функция D может иметь произвольный вид. Поэтому обучение в общем случае - многоэкстремальная невыпуклая задача оптимизации.

Для решения этой задачи могут быть использованы следующие алгоритмы:

- алгоритмы локальной оптимизации с вычислением частных производных первого порядка,
- алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка,
- алгоритмы прямого вычисления значений параметров сети по известным исходным данным,
- стохастические алгоритмы и алгоритмы глобальной оптимизации.

К первой группе относятся метод скорейшего спуска, методы тяжелого шарика, методы с одномерной и двумерной оптимизацией целевой функции в направлении антиградиента, метод сопряженных градиентов.

Ко второй группе относятся метод Ньютона, методы оптимизации с разреженными матрицами Гессе, квазиньютоновские методы, метод Гаусса-Ньютона, метод Левенберга-Марквардта.

К третьей группе относятся алгоритмы использующие методы решения систем линейных уравнений.

Стохастическими алгоритмами являются поиск в случайном направлении, имитация отжига, метод Монте-Карло (численный метод статистических испытаний), эволюционные (генетические) алгоритмы, а также алгоритмы перебора значений переменных, от которых зависит целевая функция.

Задача аппроксимации функции в стандартной постановке

Для некоторой функции $r(x)$, заданной обучающей выборкой $(\bar{x}^i, \bar{d}^i), i = 1,..., N$, необходимо найти вектор параметров $\bar{w}'$ такой, что НС реализующая функцию $\bar{y} = f(\bar{w}', \bar{x})$ наилучшим образом аппроксимирует функцию r, т. е. верно

$$D_r(f(\bar{w}',\bar{x})) = \sum_{n=1}^{N} e_n = \min_{\bar{w}} D_r(f(\bar{w},\bar{x})),$$

где e_n - ошибка НС на n-й паре выборки.

Как правило, для вычисления ошибки на одной паре применяют формулу

$$e_n = \frac{1}{2}\left\|\bar{y}^n - f(\bar{w},\bar{x}^n)\right\|^2.$$

Если функцию $f(\bar{w}',\bar{x})$ можно представить как суперпозицию функций $f_1, f_2, ..., f_l$,

$$\bar{y}^m = \sum_{j=1}^{l} w_j f_j(\bar{x}^m) = \bar{w}\cdot\bar{f},$$

то задача аппроксимации сводится к решению системы уравнений

$$\begin{pmatrix} f_1(\bar{x}^1) & \mathbf{K} & f_l(\bar{x}^1) \\ \mathbb{M} & \mathbf{O} & \mathbb{M} \\ f_1(\bar{x}^M) & \mathbf{L} & f_l(\bar{x}^M) \end{pmatrix}\begin{pmatrix} w_1 \\ \mathbb{M} \\ w_l \end{pmatrix} = \begin{pmatrix} \bar{y}^1 \\ \mathbb{M} \\ \bar{y}^M \end{pmatrix},$$

или, в сокращенной записи,

$$F\cdot\bar{w} = \bar{y}.$$

Решение такой системы имеет вид

$$\bar{w} = (F^T F)^{-1} F^T \bar{y}.$$

Это решение существует при условии невырожденности матрицы $(F^T F)$. Кроме того, при росте количества элементов обучающей выборки, растёт размерность системы и значительно увеличиваются вычислительные затраты на решение этой системы. Поэтому, как правило, задача обучения решается методами, использующими стандартные алгоритмы оптимизации.

Сравнительный анализ алгоритмов обучения

Эффективность алгоритмов обучения проверяется на определенных тестах, соответствующих принятым стандартам. При этом алгоритмы сравниваются по количеству итераций обучения, количеству расчётов целевой функции, количеству операций умножения и сложения, требуемых для одной итерации алгоритма.

Далее представлено описание и проведен сравнительный анализ алгоритмов обучения, которые при реализации на персональном компьютере требуют строго меньше 2*N вспомогательных переменных, где N - это число параметров сети, настраиваемых в процессе обучения (синаптических весов и смещений). Это следующие алгоритмы:

1. обобщенный градиентный алгоритм обучения,
2. градиентный алгоритм обучения с автоматическим определением длины шага (автономный градиентный алгоритм обучения),

3. алгоритм поиска в случайном направлении,
4. градиентный алгоритм обучения с одномерной оптимизацией,
5. градиентный алгоритм обучения с одномерной оптимизацией и с автоматическим определением длины шага.

В обобщенном градиентном алгоритме на каждой итерации выполняется вычисление градиента функции ошибки (определяются значения частных производных по синаптическим весам и смещениям) и делается шаг в направлении антиградиента. Величина шага задается пользователем.

Обобщенный градиентный алгоритм обучения в отличие от традиционного метода обратного распространения ошибки дает возможность обучать многослойные сети с произвольным числом слоев. Использование двойственных переменных ускоряет процесс обучения.

Градиентный алгоритм с автоматическим определением длины шага определяется следующим набором параметров:

- начальное значение шага,
- количество итераций, через которое происходит запоминание данных сети (синаптических весов и смещений),
- величина (в процентах) увеличения шага после запоминания данных сети, и величина уменьшения шага в случае увеличения функции ошибки.

В начале обучения с помощью автономного градиентного алгоритма записываются на диск значения весов и смещений сети. Затем происходит заданное число итераций обучения с заданным шагом. Если после завершения этих итераций значение функции ошибки не возросло, то шаг обучения увеличивается на заданную величину, а текущие значения весов и смещений записываются на диск. Если на некоторой итерации произошло увеличение функции ошибки, то с диска считываются последние запомненные значения весов и смещений, а шаг обучения уменьшается на заданную величину.

При использовании автономного градиентного алгоритма происходит автоматический подбор длины шага обучения в соответствии с характеристиками адаптивного рельефа.

Замечено, что в начале обучения шаг должен быть порядка 0.1, а в конце обучения - от 10^5 до 10^6. Автономный алгоритм по сравнению с обобщенным градиентным

алгоритмом имеет преимущество в том, что шаг обучения автоматически увеличивается, подстраиваясь под адаптивный рельеф. Тем самым существенно сокращается количество шагов, которое требуется для обучения сети.

В алгоритмах с одномерной оптимизацией на каждом шаге обучения выполняется два пробных шага: один в направлении антиградиента, другой - в противоположном направлении. По трем точкам (включая исходную точку итерации) строится парабола. Для параболы с лучами, направленными вверх, шаг делается в вершину параболы. Если лучи параболы направлены вниз, то выполняется шаг в сторону антиградиента на заданную величину. Таким образом, а этих алгоритмах адаптивный рельеф вдоль антиградиента функции ошибки аппроксимируется параболой.

В программной реализации алгоритма с одномерной оптимизацией исключено увеличение значения функции ошибки. Значения этой функции запоминаются для четырех точек (исходная точка, два пробных шага и минимум параболы). Шаг делается в точку с минимальным значением функции ошибки.

В алгоритме с одномерной оптимизацией и автоматическим определением длины шага выполняются действия, аналогичные автономному градиентному алгоритму. Величина пробных шагов подбирается под адаптивный рельеф. Данный способ исключения шагов с увеличением функции ошибки позволяет уменьшить количество вычислений на каждом шаге алгоритма.

В алгоритме поиска в случайном направлении на каждой итерации делается шаг, направление которого задается случайным образом. Если данный шаг приводит к увеличению функции ошибки, то происходит возврат к исходным значениям синаптических весов и смещений и выполняется шаг в другом случайном направлении.

Компьютерные эксперименты по обучению нейронных сетей показали, что наибольшей скоростью сходимости среди всех описанных алгоритмов обладает автономный градиентный алгоритм.

На основе экспериментальных данных, можно сделать следующие выводы:

Стохастические методы не могут быть использованы для обучения нейронных сетей большой размерности. Даже метод поиска в случайном направлении, который требует для обучения меньше шагов, чем другие стохастические методы, сходился катастрофически медленно.

Градиентный алгоритм с одномерной оптимизацией потребовал для обучения сети меньше итераций, чем обобщенный и градиентный алгоритмы. Однако большой объем вычислений на каждой итерации привел к тому, что этот алгоритм имеет меньшую

эффективность по сравнению с алгоритмами, которые не используют одномерную оптимизацию.