УДК 519.254 JEL C45; C61 DOI 10.26425/1816-4277-2022-11-48-55

Сравнение эффективности методов минимизации нулевого и первого порядка в нейронных сетях

Губарева Елена Алексеевна[1]

Канд. физ.-мат. наук, доц. каф. математики и информатики
ORCID: 0000-0002- 5135-6555, e-mail: ea_gubareva@guu.ru

Хашин Сергей Иванович[2]

Канд. физ.-мат. наук, доц. каф. информационных технологий и прикладной математики
ORCID: 0000-0002-6508-663X, e-mail: khash2@mail.ru

Шемякова Екатерина Сергеевна[3]

Ph.D., ассоциированный профессор департамента математики и статистики
ORCID: 0000-0002-3787-804X, e-mail: Ekaterina.Shemyakova@UToledo.Edu

[1]Государственный университет управления, г. Москва, Россия

[2]Ивановский государственный университет, г. Иваново, Россия

[3]Университет Толидо, г. Толидо, США

Аннотация

Для минимизации целевой функции в нейронных сетях обычно применяют методы первого порядка, предполагающие неоднократное вычисление градиента. Количество переменных в современных нейронных сетях может составлять многие тысячи и даже миллионы. Многочисленные эксперименты показывают, что время аналитического вычисления градиента функции N переменных примерно в $N/5$ раз больше времени вычисления самой функции. В статье рассматривается возможность использования для минимизации функции методов нулевого порядка. В частности, предлагается новый метод нулевого порядка для минимизации функции: спуск по двумерным пространствам. Проведено сравнение скоростей сходимости трех различных методов: стандартного градиентного спуска с автоматическим выбором шага, координатного спуска с выбором шага по каждой координате и спуска по двумерным подпространствам. Показано, что эффективность правильно организованных методов нулевого порядка в рассмотренных задачах обучения нейронных сетей не ниже градиентных.

Ключевые слова

Нейронные сети, целевая функция, минимизация, градиент, градиентный спуск, координатный спуск, скорость сходимости

Для цитирования: Губарева Е.А., Хашин С.И., Шемякова Е.С. Сравнение эффективности методов минимизации нулевого и первого порядка в нейронных сетях//Вестник университета. 2022. № 11. С. 48–55.

Comparison of the efficiency of zero and first order minimization methods in neural networks

Elena A. Gubareva[1]
Cand. Sci (Phys. and Math.), Assoc. Prof. at the Department of Mathematics and Computer Science
ORCID: 0000-0002- 5135-6555, e-mail: ea_gubareva@guu.ru

Sergey I. Khashin[2]
Cand. Sci (Phys. and Math.), Assoc. Prof. at the Information Technologies and Applied Mathematics Department
ORCID: 0000-0002-6508-663X, e-mail: khash2@mail.ru

Ekaterina S. Shemyakova[3]
Ph.D., Assoc. Prof. at the Department of Mathematics and Statistics
ORCID: 0000-0002-3787-804X, e-mail: Ekaterina.Shemyakova@UToledo.Edu

[1]State University of Management, Moscow, Russia

[2]Ivanovo State University, Ivanovo, Russia

[3]University of Toledo, Toledo, USA

Abstract

To minimize the objective function in neural networks, first-order methods are usually used, which involve the repeated calculation of the gradient. The number of variables in modern neural networks can be many thousands and even millions. Numerous experiments show that the analytical calculation time of an N variable function's gradient is approximately $N/5$ times longer than the calculation time of the function itself. The article considers the possibility of using zero-order methods to minimize the function. In particular, a new zero-order method for function minimization, descent over two-dimensional spaces, is proposed. The convergence rates of three different methods are compared: standard gradient descent with automatic step selection, coordinate descent with step selection for each coordinate, and descent over two-dimensional subspaces. It has been shown that the efficiency of properly organized zero-order methods in the considered problems of training neural networks is not lower than the gradient ones.

Keywords

Neural networks, objective function, minimization, gradient, gradient descent, coordinate descent, convergence rate

For citation: Gubareva E.A., Khashin S.I., Shemyakova E.S. (2022) Comparison of the efficiency of zero and first order minimization methods in neural networks. *Vestnik universiteta*, no. 11, pp. 48–55.

ВВЕДЕНИЕ

За последние годы искусственные нейронные сети становятся все более важным инструментом исследований в самых разных областях знаний [1–3]. Большой обзор можно найти, например, в [4]. С ростом вычислительных возможностей современных компьютеров качество работы нейронных сетей постоянно растет. Уже с середины 2010-х гг. нейронные сети стали более надежно распознавать лицо, чем человек [5]. Однако такие достижения сопровождаются огромным ростом сложности нейросетей. Например, в работе [6] используется сеть с количеством параметров от 400 млн до свыше 500 млн.

В любом случае, процесс обучения нейронной сети сводится к нахождению минимума некоторой целевой функции f (w), где w – вектор внутренних параметров нейронной сети. При этом функция f является суммой отдельных слагаемых для каждого обучающего вектора, а вектор w является объединением внутренних параметров всех нейронов из нашей нейросети. Таким образом, в упомянутом выше примере [6] требуется найти минимум функции от нескольких сот миллионов переменных.

Использование градиентных методов в таких случаях сталкивается с различными трудностями, например, только один вектор градиента будет занимать несколько гигабайт памяти. Однако в настоящее время нейронные сети почти целиком ориентированы на применение градиентных методов [2–4]. Методы нулевого порядка, безградиентные [7–9], хотя и хорошо известны, традиционно считаются более медленными, и их рекомендуется использовать там, где аналитическое нахождение производных невозможно.

В нейронных сетях в задачах регрессии градиент целевой функции вычисляется по достаточно простой формуле. В задачах классификации наиболее естественная целевая функция – количество ошибок нейросети, принимает лишь целые значения и, следовательно, является кусочно-постоянной, то есть ее градиент почти всюду равен нулю. Поэтому задачи классификации, используя функцию softmax, тоже сводят к нахождению минимума некоторой функции, просто от чуть большего количества аргументов. Таким образом, на сегодняшний день при обучении нейросетей используются почти исключительно градиентные методы.

В настоящей работе мы хотим показать, что правильно организованные методы нулевого порядка (безградиентные) во многих случаях не менее эффективны, чем градиентные.

Сначала мы описываем применяемые инструменты, представляем служебную информацию о нахождении минимума квадратичной функции от двух переменных, описываем пробную целевую функцию, на которой мы будем сравнивать различные методы минимизации. Далее мы приводим описание трех сравниваемых методов, включая нами разработанный «спуск по двумерным подпространствам». В представленных таблицах приведены результаты численных экспериментов. На самом деле экспериментов было проведено значительно больше, в статье приведены лишь наиболее характерные результаты. На основании приведенных результатов проводится сравнение рассмотренных методов, выявляются их достоинства и недостатки.

ИНСТРУМЕНТАРИЙ

Для проведения численных экспериментов по проверке предложенного метода минимизации и его сравнения с другими методами был использован язык Python и его модули:

- Python (язык программирования) версии 3.95;
- Numpy (модуль работы с массивами и тензорами) версии 1.19.5;
- Matplotlib (модуль работы с графиками) версии 3.4.2;
- TensorFllow (модуль работы с нейронными сетями) версии 2.5.0;
- Keras (модуль работы нейронными сетями) версии 2.5.0;
- PyCharm (интегрированная среда разработки) версии 2019.3.5 (Community edition);
- Различные персональные компьютеры и версии Windows.

МИНИМИЗАЦИЯ ФУНКЦИИ В ДВУМЕРНОМ ПРОСТРАНСТВЕ

Пусть $f(x, y)$ – функция двух переменных, минимум которой мы хотим найти. Будем аппроксимировать функцию квадратичным многочленом:

$$f(x,y) \approx a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2, \quad (1)$$

где a_i – постоянные коэффициенты, i = 1, 2, 3, 4, 5.

Вычислив значения функции в шести точках: (0, 0); (1, 0); (–1, 0); (0, 1); (0, –1) и (1, 1), определим коэффициенты a_i. Значение a_0 находится сразу: $a_0 = f(0, 0)$.

Введем обозначения: $g_1 = f(1, 0) - a_0$; $g_2 = f(-1, 0) - a_0$; $g_3 = f(0, 1) - a_0$; $g_4 = f(0, -1) - a_0$; $g_5 = f(1, 1) - a_0$. Тогда коэффициенты a_i находятся по формулам:

$$a_3 = \frac{g_1 + g_2}{2};\ a_1 = g_1 - a_3;\ a_5 = \frac{g_3 + g_4}{2};\ a_2 = g_3 - a_5;\ a_4 = g_5 - g_3 - g_1. \quad (2)$$

Зная коэффициенты функции $f(x, y)$, мы можем найти и точку ее экстремума:

$$\begin{cases} \dfrac{\partial f}{\partial x} = a_1 + 2a_3 x + a_4 y = 0; \\ \dfrac{\partial f}{\partial y} = a_2 + a_4 x + 2a_5 y = 0. \end{cases} \quad (3)$$

Если экстремум не является точкой минимума квадратичной функции, то возвратим ту из шести рассмотренных точек, в которой функция принимает наименьшее среди них значение.

ЦЕЛЕВАЯ ФУНКЦИЯ

Мы сравниваем только методы поиска локальных минимумов. С учетом инвариантности относительно сдвигов, не ограничивая общности, будем считать, что минимизируемая функция $f(\boldsymbol{x})$ от n аргументов имеет локальный минимум при $\boldsymbol{x} = \mathbf{0}$ и $f(\mathbf{0}) = 0$.

Функции, рассматриваемые в нейронных сетях, обычно являются гладкими (и даже бесконечно гладкими). Мы рассматриваем поведение функции в окрестности нуля, поэтому при анализе скорости сходимости мы можем аппроксимировать ее начальными членами ряда Тейлора. Так как точка $\boldsymbol{x} = \mathbf{0}$ является точкой локального экстремума функции $f(\boldsymbol{x})$, то ее ряд Тейлора в точке $\boldsymbol{x} = \mathbf{0}$ не может иметь членов 1-го порядка.

В принципе, эффективность методов можно сравнивать уже на квадратичных функциях вида $f(\boldsymbol{x}) = \sum a_{ij} x^i x^j$. Однако многие методы в этом случае могут сходиться очень быстро, в предельном случае – за один шаг. В реальных задачах такая ситуация обычно не встречается, и для более реалистичного сравнения методов надо брать более сложные функции. Для наших целей достаточно добавить два-три члена 4-го порядка. Примером такой функции для случая двух переменных будет хорошо известная и часто используемая в двумерном случае функция Розенброка [10], только со сдвигом координат:

$$f(x, y) = (1 - x)^2 + 100 \cdot (y - x^2)^2. \quad (4)$$

Брать члены 3-го порядка нежелательно, так как вновь предлагаемые методы не вполне локальны, и при некоторых условиях могут убегать из окрестностей локального минимума и переходить к другому. С точки зрения обучения нейронных сетей, это очень полезное свойство, но в нашей работе мы хотим сравнить именно скорость локальной оптимизации. Поэтому в настоящей статье пробная целевая функция была взята в виде суммы квадратичной формы полного ранга и четвертых степеней двух линейных функций.

Таким образом, исследуемая целевая функция имеет глобальный минимум в точке $\mathbf{0}$, равный нулю. Это позволяет сразу увидеть скорость сходимости метода.

ВРЕМЯ ВЫЧИСЛЕНИЯ ЦЕЛЕВОЙ ФУНКЦИИ

Функции, минимизация которых выполняется при обучении нейронных сетей, обычно являются весьма громоздкими. Поэтому вычисление функции занимает основную часть времени минимизации.

Будем считать, что однократное вычисление функции f от N переменных занимает время T. Вектор градиента $grad\, f$ можно приближенно найти численно:

$$\operatorname{grad} f_i \approx \frac{f(\boldsymbol{x} + \varepsilon \boldsymbol{e}_i) - f(\boldsymbol{x})}{\varepsilon}, \quad (5)$$

где ε – некоторая константа; $\boldsymbol{e}_i$ – i-й координатный вектор, $i = 1, \ldots, N$, при этом значение функции f требуется вычислить $N + 1$ раз. Учитывая достаточную гладкость функции, величину ε можно зафиксировать, например, положить $\varepsilon = 10^{-6}$.

При работе с нейронными сетями градиент целевой функции можно найти аналитически, это и более точно, и более быстро. Численные эксперименты, проведенные как на C++, так и на Python/TensorFlow/Keras, показывают вполне ожидаемый результат: время аналитического нахождения градиента примерно равно $0{,}25NT$, точнее говоря, колеблется от $0{,}2NT$ до $0{,}3NT$, то есть в 3–5 раз быстрее численного нахождения градиента ($\approx NT$). Поэтому в дальнейшем, при сравнении методов нулевого порядка (безградиентных) с методами первого порядка (использующих градиент), будем полагать, что время аналитического вычисления градиента функции от N переменных примерно в $N/5$ раз больше времени вычисления самой функции.

СРАВНИВАЕМЫЕ МЕТОДЫ

1. *Градиентный спуск* [2; 4; 11; 12].

Имеется множество разновидностей алгоритма градиентного спуска (NAG, Adagrad, RMSProp и др.). Однако в наши планы не входит их исследование и сравнение – мы рассматриваем лишь один базовый алгоритм: сдвиг в направлении антиградиента с автоматическим подбором скорости обучения LR: если шаг был неудачный, LR умножается на 0,7, если удачный – то на 1,1.

2. *Координатный спуск* [12–14].

В рассматриваемой версии этого алгоритма мы имеем вектор координатного сдвига, в котором запоминается величина последнего успешного сдвига по каждой координате. Если сдвиг по этой координате был неудачным в обе стороны, то его величина умножается на 0,5, в противном случае – на 1,3.

3. *Спуск по двумерным подпространствам* (предлагаемый нами метод).

Пусть **w** – вектор, полученный на текущем этапе минимизации целевой функции. В начале процедуры минимизации выбираем вектор координатного сдвига ***dw***, все его значения на первом шаге полагаем равными заданной скорости обучения *LR*, обычно это 0,1. Выбор этого значения не слишком важен, так как вектор будет постоянно корректироваться во время работы.

В имеющемся N-мерном пространстве выбираем последовательно двумерные координатные подпространства. Пусть (i, j) – выбранная пара координат. Рассмотрим функцию двух переменных

$$g(x, y) = f(\boldsymbol{w} + x \cdot dw_i \cdot \boldsymbol{e_i} + y \cdot dw_j \cdot \boldsymbol{e_j}), \tag{6}$$

где $\boldsymbol{e_i}$, $\boldsymbol{e_j}$ – i-й и j-й базисные векторы соответственно; dw_i, dw_j – i-я и j-я координаты вектора сдвига ***dw*** соответственно.

С помощью описанной выше процедуры найдем точку предполагаемого минимума функции $g(x, y)$. Для повышения устойчивости алгоритма, в случае если найденные значения (x, y) слишком велики (больше 10 по модулю), приводим их к отрезку от –10 до 10.

На основе полученной пары чисел (x, y) изменим векторы ***w*** и ***dw***.

При разработке метода были проверены различные схемы выбора пар координат (i, j) (в языке Python нумерация элементов массивов начинается с 0 – *примеч. авторов*). Наиболее эффективной была признана простейшая схема: $(i, j) = (0, 1)$, затем $(2, 3)$, затем $(4, 5)$ и т.д. Никакие другие, более сложные алгоритмы выбора пар, не дали никакого заметного улучшения.

Были проверены и схемы с перекрытием: $(i, j) = (0, 1); (1, 2); (2, 3); \ldots$ и т.д. Но они оказались существенно менее эффективными.

РЕЗУЛЬТАТЫ ЧИСЛЕННЫХ ИСПЫТАНИЙ

Мы провели сравнение скорости сходимости трех различных методов:

- стандартный градиентный спуск с автоматическим выбором шага (grad);
- координатный спуск с выбором шага по каждой координате (coord);
- спуск по двумерным подпространствам (2dim).

Результаты численных экспериментов, проведенных авторами, приведены в таблицах 1–3. В них представлены данные о результатах рассматриваемых методов оптимизации для размерностей пространства 10, 100 и 1 000 соответственно. Для каждого метода указано количество вычислений целевой функции (графа #Nf в таблицах) и достигнутое значение минимума целевой функции *S*. При этом одно вычисление градиента приравнивается к вычислениям целевой функции (N – размерность пространства).

Таблица 1

Результаты вычислений для размерности пространства, равной 10

Grad		Coord		2Dim	
#Nf	***S***	**#Nf**	***S***	**#Nf**	***S***
122	0,001431	115	0,001526	112	0,00378
242	$5{,}735 \cdot 10^{-4}$	240	$5{,}889 \cdot 10^{-6}$	217	$1{,}47 \cdot 10^{-4}$
482	$1{,}986 \cdot 10^{-4}$	534	$1{,}795 \cdot 10^{-7}$	322	$3{,}93 \cdot 10^{-6}$
602	$1{,}329 \cdot 10^{-4}$	599	$1{,}333 \cdot 10^{-7}$	392	$3{,}23 \cdot 10^{-7}$
962	$4{,}204 \cdot 10^{-5}$	647	$1{,}049 \cdot 10^{-7}$	427	$9{,}31 \cdot 10^{-8}$
1 082	$2{,}953 \cdot 10^{-5}$	660	$9{,}861 \cdot 10^{-8}$	-	-
1 562	$8{,}653 \cdot 10^{-6}$	-	-	-	-
2 042	$3{,}131 \cdot 10^{-6}$	-	-	-	-
2 642	$1{,}059 \cdot 10^{-6}$	-	-	-	-

Составлено авторами по материалам исследования

Таблица 2

Результаты вычислений для размерности пространства, равной 100

Grad		Coord		2Dim	
#Nf	***S***	**#Nf**	***S***	**#Nf**	***S***
$20 \cdot 10^3$	$4{,}26 \cdot 10^{-4}$	$61 \cdot 10^3$	0,003804	$7 \cdot 10^3$	$1{,}12 \cdot 10^{-4}$
$40 \cdot 10^3$	$1{,}54 \cdot 10^{-4}$	$22 \cdot 10^3$	0,001522	$14 \cdot 10^3$	$2{,}69 \cdot 10^{-5}$
$61 \cdot 10^3$	$8{,}59 \cdot 10^{-5}$	$242 \cdot 10^3$	$4{,}436 \cdot 10^{-4}$	$21 \cdot 10^3$	$1{,}17 \cdot 10^{-5}$
$102 \cdot 10^3$	$4{,}13 \cdot 10^{-5}$	$303 \cdot 10^3$	$2{,}541 \cdot 10^{-4}$	$28 \cdot 10^3$	$6{,}46 \cdot 10^{-6}$
$183 \cdot 10^3$	$1{,}78 \cdot 10^{-5}$	$424 \cdot 10^3$	$8{,}655 \cdot 10^{-5}$	$35 \cdot 10^3$	$4{,}05 \cdot 10^{-6}$
$285 \cdot 10^3$	$9{,}33 \cdot 10^{-6}$	$605 \cdot 10^3$	$1{,}836 \cdot 10^{-5}$	$42 \cdot 10^3$	$2{,}71 \cdot 10^{-6}$
$489 \cdot 10^3$	$4{,}06 \cdot 10^{-6}$	$1\,028 \cdot 10^3$	$6{,}907 \cdot 10^{-7}$	$56 \cdot 10^3$	$1{,}35 \cdot 10^{-6}$
$591 \cdot 10^3$	$2{,}96 \cdot 10^{-6}$	$1\,331 \cdot 10^3$	$1{,}060 \cdot 10^{-7}$	$70 \cdot 10^3$	$7{,}30 \cdot 10^{-7}$
$816 \cdot 10^3$	$1{,}67 \cdot 10^{-6}$	-	-	$105 \cdot 10^3$	$1{,}91 \cdot 10^{-7}$
$1\,040 \cdot 10^3$	$1{,}03 \cdot 10^{-6}$	-	-	$123 \cdot 10^3$	$9{,}99 \cdot 10^{-8}$

Составлено авторами по материалам исследования

Таблица 3

Результаты вычислений для размерности пространства, равной 1 000

Grad		Coord		2Dim	
#Nf	***S***	**#Nf**	***S***	**#Nf**	***S***
$100 \cdot 10^3$	0,00166	$2 \cdot 10^3$	5,103763	$70 \cdot 10^3$	0,00113
$200 \cdot 10^3$	$5{,}76 \cdot 10^{-4}$	$138 \cdot 10^3$	$2{,}832 \cdot 10^{-4}$	$140 \cdot 10^3$	$4{,}27 \cdot 10^{-4}$
$300 \cdot 10^3$	$3{,}11 \cdot 10^{-4}$	$259 \cdot 10^3$	$2{,}028 \cdot 10^{-4}$	$210 \cdot 10^3$	$2{,}34 \cdot 10^{-4}$
$401 \cdot 10^3$	$2{,}02 \cdot 10^{-4}$	$380 \cdot 10^3$	$1{,}659 \cdot 10^{-4}$	$280 \cdot 10^3$	$1{,}49 \cdot 10^{-4}$
$601 \cdot 10^3$	$1{,}10 \cdot 10^{-4}$	$622 \cdot 10^3$	$1{,}279 \cdot 10^{-4}$	$350 \cdot 10^3$	$1{,}04 \cdot 10^{-4}$
$1\,002 \cdot 10^3$	$5{,}12 \cdot 10^{-5}$	$1\,227 \cdot 10^3$	$8{,}833 \cdot 10^{-5}$	$420 \cdot 10^3$	$7{,}60 \cdot 10^{-5}$
$1\,603 \cdot 10^3$	$2{,}53 \cdot 10^{-5}$	$1\,832 \cdot 10^3$	$6{,}982 \cdot 10^{-5}$	$490 \cdot 10^3$	$5{,}81 \cdot 10^{-5}$
$2\,004 \cdot 10^3$	$1{,}81 \cdot 10^{-5}$	$2\,438 \cdot 10^3$	$5{,}850 \cdot 10^{-5}$	$700 \cdot 10^3$	$3{,}02 \cdot 10^{-5}$
$3\,006 \cdot 10^3$	$9{,}90 \cdot 10^{-6}$	$3\,649 \cdot 10^3$	$4{,}476 \cdot 10^{-5}$	$1\,050 \cdot 10^3$	$1{,}37 \cdot 10^{-5}$
$4\,008 \cdot 10^3$	$6{,}43 \cdot 10^{-6}$	$4\,861 \cdot 10^3$	$3{,}645 \cdot 10^{-5}$	$1\,400 \cdot 10^3$	$7{,}65 \cdot 10^{-6}$
6 012·103	3,50·10-6	6 074·103	3,070·10-5	2 100·103	3,28·10-6
8 016·103	2,28·10-6	9 713·103	2,030·10-5	3 430·103	1,21·10-6
9 920·103	1,65·10-6	12 140·103	1,613·10-5	-	-

Составлено авторами по материалам исследования

ОБСУЖДЕНИЕ РЕЗУЛЬТАТОВ

1. Градиентный спуск показал заметно худшие результаты, чем методы нулевого порядка. На самом деле, среди большого разнообразия градиентных методов можно подобрать гораздо более эффективные в данном случае. Однако нашей задачей было продемонстрировать эффективность именно методов нулевого порядка. Приведенные данные показывают, что они как минимум не хуже более сложных, градиентных.

2. Имеется много методов нулевого порядка: Нелдера–Мида, Хука–Дживса, Розенброка, Пауэлла и др. Однако в случае большой размерности пространства они очень легко вырождаются (симплексы сплющиваются и т.п.).

3. Как ни странно, использование на следующем шаге вектора предыдущего сдвига не дает эффекта. Точнее говоря, величина сдвига в предыдущем направлении оказывается очень мала (асимптотически меньше 0,01 для случая больших размерностей). Это же относится к методу Хука–Дживса.

4. Были проверены и схемы выбора пар координат с перекрытием: $(i, j) = (0, 1); (1, 2); (2, 3); \ldots$ и т.д. Но они оказались существенно менее эффективными, по той же самой причине: x-координата, то есть величина сдвига в предыдущем направлении асимптотически оказывается слишком мала.

5. При разработке метода были проверены различные схемы выбора пар координат (i, j). Наиболее эффективной была признана простейшая схема: $(i, j) = (0, 1)$, затем (2, 3), затем (4, 5) и т.д. Никакие другие, более сложные алгоритмы не дали никакого заметного улучшения.

ЗАКЛЮЧЕНИЕ

На основании проведенного сравнения скорости сходимости различных методов минимизации функции от большого количества переменных можно сделать следующие выводы.

1. Эффективность методов нулевого порядка минимизации функции при правильной их организации может быть не меньше, чем у методов первого порядка.

2. Предложен метод минимизации функции «Спуск по двумерным подпространствам». Он показал эффективность лучшую, чем у градиентного метода для случая большой размерности (сотни и тысячи).

3. При увеличении размерности пространства поведение методов минимизации может существенно ухудшаться. Сравнение в этом случае надо проводить отдельно, не надеяться на аналогичные результаты при меньших размерностях.

4. Так как для методов нулевого порядка не требуется вычисления производной, их область применения может быть значительно шире, чем для методов первого порядка. В частности, их можно применять для минимизации дискретных величин. Например, в нейронных сетях один из наиболее естественных показателей – количество ошибок, то есть количество неправильных ответов, выдаваемым нейронной сетью в задаче распознавания. Методы первого порядка в этом случае принципиально не применимы, так как градиент дискретной целевой функции почти всюду равен нулю.

Библиографический список

1. Гафаров Ф.М., Галимянов А.Ф. *Искусственные нейронные сети и приложения: учеб. пособие.* Казань: Изд-во Казан. ун-та; 2018. 121 с.
2. Николенко С., Кадурин А., Архангельская Е. *Глубокое обучение: серия «Библиотека программиста».* СПб.: Питер; 2018. 480 с.
3. Chollet F. *Deep Learning with Python*. 2nd Ed. Shelter Island: Manning Publications Co.; 2020. 450 p.
4. Будума Н. *Основы глубокого обучения.* М.: Манн, Иванов и Фербер; 2020. 306 с.
5. Schroff F., Kalenichenko D., Philbin J. FaceNet: A unified embedding for face recognition and clustering, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015; P. 815–823. https://doi.org/10.1109/CVPR.2015.7298682
6. Yanping Huang et al. *GPipe: Efficient training of giant neural networks using pipeline parallelism.* arXiv:1811.06965 [cs.CV]. https://doi.org/10.48550/arXiv.1811.06965
7. Банди Б. *Методы Оптимизации. Вводный курс.* Пер. с англ. М.: Радио и связь; 1988. 127 с.
8. Kochenderfer M.J., Wheeler T.A. *Algorithms for Optimization.* MIT Press; 2019. 520 p.
9. Avriel M. *Nonlinear Programming: Analysis and Methods.* Dover Publishing; 2003. 512 p.
10. Rosenbrock H.H. An automatic method for finding the greatest or least value of a function. *The Computer Journal.* 1960;3:175–184.

11. Городецкий С.Ю., Гришагин В.А. *Нелинейное программирование и многоэкстремальная оптимизация*. Нижний Новгород: Изд-во Нижегородского гос. ун-та им. Н.И. Лобачевского; 2007. 489 с.
12. Nocedal J., Wright S.J. *Numerical Optimization*. 2nd ed. NY: Springer New York; 2006. 651 p. https://doi.org/10.1007/978-0-387-40065-5
13. Акулич И.Л. М*атематическое программирование в примерах и задачах*. М.: Высшая школа; 1986. 352 с.
14. Прокопенко Н.Ю. *Методы оптимизации*. Н. Новгород: ННГАСУ; 2018. 118 с.

References

1. Gafarov F.M., Galimyanov A.F. *Artificial neural networks and applications: textbook. Allowance.* Kazan: Kazan Publishing House. Un-ta; 2018. (In Russian).
2. Nikolenko S., Kadurin A., Arkhangelskaya E. *Deep Learning. series "Programmer's Library"*. St. Petersburg: Peter; 2018. (In Russian).
3. Chollet F. *Deep Learning with Python*. 2nd Ed. Shelter Island: Manning Publications Co.; 2020. 450 p.
4. Buduma N. *Fundamentals of Deep Learning*. M.: Mann, Ivanov and Ferber; 2020. 306 p. (In Russian).
5. Schroff F., Kalenichenko D., Philbin J. FaceNet: A unified embedding for face recognition and clustering, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015; P. 815–823. https://doi.org/10.1109/CVPR.2015.7298682
6. Yanping Huang et al. *GPipe: Efficient training of giant neural networks using pipeline parallelism*. arXiv:1811.06965 [cs.CV]. https://doi.org/10.48550/arXiv.1811.06965
7. Bunday B.D. *Basic optimization methods*. Trans. from Eng. M.: Radio i svyaz'; 1988. 127 p. (In Russian).
8. Kochenderfer M.J., Wheeler T.A. *Algorithms for Optimization*. MIT Press; 2019. 520 p.
9. Avriel M. *Nonlinear programming: analysis and methods*. Dover Publishing; 2003. 512 p.
10. Rosenbrock H.H. An automatic method for finding the greatest or least value of a function. *The Computer Journal*. 1960;3:175–184.
11. Gorodeckii S.Y., Grishagin V.A. *Nonlinear programming and multi-extreme optimization*. N. Novgorod: N.I. Lobachevsky State University of Nizhny Novgorod Publ. House; 2007. 489 c. (In Russian).
12. Nocedal J., Wright S.J. *Numerical Optimization*. 2nd ed. NY: Springer New York; 2006. 651 p. https://doi.org/10.1007/978-0-387-40065-5
13. Aculich I.L. *Mathematical programming in examples and tasks*. Moscow: Vyshaya shkola; 1986. 352 p. (In Russian).
14. Prokopenko M.Y. *Optimization methods*. N. Novgorod: NNGASU; 2018. 118 p. (In Russian).