УДК 519.852, 004.032.26 **DOI: 10.14529/cmse230402**

ИССЛЕДОВАНИЕ НЕЙРОСЕТЕВОГО МЕТОДА РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

© 2023 Н.А. Ольховский

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: olkhovskiina@susu.ru

Поступила в редакцию: 20.10.2023

В статье исследован метод определения вектора движения по гиперплоскостям, ограничивающим допустимый многогранник многомерной задачи линейного программирования на основе визуальных образов, подаваемых на вход нейронной сети прямого распространения. Алгоритм визуализации строит в окрестности точки, расположенной на ограничивающей гиперплоскости, рецептивное поле. Для каждой точки рецептивного поля вычисляется скалярное смещение до поверхности гиперплоскости. На основании вычисленного смещения каждой точке рецептивного поля присваивается скалярная величина. Полученный визуальный образ подается на вход нейронной сети прямого распространения, которая вычисляет на ограничивающей гиперплоскости направление максимального увеличения целевой функции. В статье предложена усовершенствованная форма крестообразного рецептивного поля. Описано построение обучающего множества на основе случайно сгенерированных ограничивающих гиперплоскостей и целевых функций в многомерных пространствах. Разработана масштабируемая архитектура нейронной сети с изменяемым числом скрытых слоев. Произведен подбор гиперпараметров нейронной сети. В вычислительных экспериментах подтверждена высокая (более 98%) точность работы крестообразного рецептивного поля. Исследована зависимость точности результатов нейронной сети от числа скрытых слоев и продолжительности обучения.

Ключевые слова: линейное программирование, метод поверхностного движения, искусственная нейронная сеть, глубокое обучение.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Ольховский Н.А. Исследование нейросетевого метода решения задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 4. С. 55–75. DOI: 10.14529/cmse230402.

Введение

Одной из фундаментальных задач современной прикладной математики является задача линейного программирования (ЛП) с большим числом параметров [1]. Оптимизационные модели, основанные на многопараметрической (многомерной) задаче ЛП встречаются в системах поддержки принятия решений в экономике [2, 3], в системах управления беспилотными летательными аппаратами [4], в управлении технологическими процессами [5–7], при построении логистических цепочек [8–10], в оперативном управлении и планировании [11, 12].

До сих пор одним из наиболее распространенных способов решения задач ЛП был класс алгоритмов, разработанных на основе симплекс-метода [13]. Было установлено, что симплекс-метод эффективен для решения большого класса задач ЛП. В частности, симплекс-метод эффективно использует преимущества любой гиперразреженности в задачах ЛП [14]. Однако симплекс-метод обладает некоторыми фундаментальными особенностями, которые ограничивают его использование для решения больших задач ЛП. Во-первых, в определенных случаях симплекс-метод должен выполнять итерации по всем вершинам симплекса, что соответствует экспоненциальной временной сложности [15–17]. Во-

вторых, в большинстве случаев симплекс-метод успешно решает задачи ЛП, содержащие до 50 000 переменных, однако при решении задач бóльших размерностей часто наблюдается потеря точности [18], которая не может быть компенсирована даже применением таких мощных вычислительных процедур, как «аффинное масштабирование» или «итеративное уточнение» [19]. В-третьих, в общем случае последовательный характер симплекс-метода затрудняет распараллеливание в многопроцессорных системах с распределенной памятью [20]. Были предприняты многочисленные попытки создать масштабируемую параллельную реализацию симплекс-метода, но все они оказались безуспешными [21]. Во всех случаях граница масштабируемости составляла от 16 до 32 процессорных узлов (см., например, [22]).

Хачиян доказал [23], используя вариант метода эллипсоидов (предложенный в 1970-х годах Шором [24], Юдиным и Немировским [25]), что задачи ЛП могут быть решены за полиномиальное время. Однако попытки применить этот подход на практике оказались безуспешными, поскольку в подавляющем большинстве случаев метод эллипсоида демонстрировал гораздо худшую эффективность по сравнению с симплекс-методом. Позже Кармаркар [26] предложил алгоритм внутренней точки с полиномиальным временем, который можно было использовать на практике. Этот алгоритм породил целую область современных методов внутренней точки [27], которые способны решать большие задачи ЛП с миллионами переменных и миллионами уравнений [28–32]. Более того, эти методы являются самокорректирующимися, а следовательно, обеспечивают высокую точность вычислений. Общим недостатком методов внутренней точки является необходимость найти некоторую допустимую точку, удовлетворяющую всем ограничениям задачи ЛП, перед началом вычислений. Нахождение такой внутренней точки может быть сведено к решению дополнительной задачи ЛП [33].

В работе [34] предложен новый подход к решению задачи ЛП, названный «метод поверхностного движения». В основе метода лежит идея о том, что точка максимального значения целевой функции (2) принадлежит границе допустимой области $\Gamma(M)$, и определить ее можно, двигаясь по поверхности многогранника постоянно в сторону максимального увеличения целевой функции. В общих чертах алгоритм поверхностного движения состоит из следующих шагов. В начале берется произвольная точка на границе допустимой области задачи ЛП. Сделать это можно, в частности, при помощи фейеровских отображений способом, описанным в статье [35]. Затем вычисляется вектор движения $\boldsymbol{d}$ по поверхности допустимой области, соответствующий направлению максимального возрастания целевой функции. После чего происходит движение в заданном направлении до ограничивающего ребра. В этой точке вычисляется новый вектор $\boldsymbol{d}$, и указанные шаги повторяются до тех пор, пока вектор $\boldsymbol{d}$ не окажется нулевым. Статья [34] содержит доказательство, что сделать это можно всегда за конечное число шагов. Проблема метода поверхностного движения состоит в том, что не представлен конструктивный способ нахождения вектора $\boldsymbol{d}$.

Решением проблемы с определением вектора $\boldsymbol{d}$ может стать сочетание искусственных нейронных сетей (ИНС) с оригинальным методом визуализации многомерных задач ЛП, предложенным в [36]. Известно, что нейронные сети прямого распространения демонстрируют впечатляющие результаты в задачах, связанных с распознаванием сложных визуальных образов. С другой стороны, в [36] приведен хорошо масштабируемый параллельный алгоритм, позволяющий за конечное, заранее прогнозируемое время, строить визуальные образы, доступные для обработки нейронными сетями. В настоящей статье исследован подход, объединяющий технологию глубоких нейронных сетей с возможностями параллельных

вычислений для нахождения вектора $\boldsymbol{d}$, указывающего направление максимального увеличения целевой функции.

Статья имеет следующую структуру. В разделе 1 кратко изложены основные положения метода визуализации задач ЛП и метода поверхностного движения, формирующие теоретическую основу настоящего исследования. В разделе 2 описана новая структура рецептивного поля, получившая название «крестообразная». Раздел 3 описывает разработку архитектуры фундаментальной нейросетевой модели, вычисляющей вектор движения $\boldsymbol{d}$ по гиперплоскостям, ограничивающим допустимую область. В разделе 4 приведены результаты вычислительных экспериментов. Заключение суммирует полученные результаты и рассматривает возможные направления дальнейших исследований.

1. Теоретические основы

Рассмотрим основные положения метода визуализации многомерных задач ЛП [36] и метода поверхностного движения [34]. В общем виде задачу линейного программирования можно представить следующим образом:

$$\bar{\boldsymbol{x}} = \arg\max\left\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle | A\boldsymbol{x} \leqslant \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n\right\}, \tag{1}$$

где $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$ и $\boldsymbol{c} \neq \boldsymbol{0}$. Причем ограничение $\boldsymbol{x} \geqslant \boldsymbol{0}$ входят в систему $A\boldsymbol{x} \leqslant \boldsymbol{b}$ в форме $-x_i \leqslant 0$ для всех $i \in \{1, 2, ..., n\}$. Здесь вектором $\boldsymbol{c}$ обозначен градиент целевой функции, максимум которой необходимо найти:

$$f(\boldsymbol{x}) = c_1x_1 + \ldots + c_nx_n. \tag{2}$$

Обозначим через $\mathcal{P}$ множество индексов, нумерующих строки матрицы A:

$$\mathcal{P} = \{1, \cdots, m\}. \tag{3}$$

Пусть $\boldsymbol{a}_i \in \mathbb{R}^n$ обозначает вектор, представляющий i-тую строку матрицы A. Мы предполагаем, что $\boldsymbol{a}_i \neq \boldsymbol{0}$ для всех $i \in \mathcal{P}$. Обозначим через $\hat{H}_i$ замкнутое полупространство, определяемое неравенством $\langle \boldsymbol{a}_i, \boldsymbol{x}\rangle \leqslant b_i$, а через H_i — ограничивающую его гиперплоскость:

$$\hat{H}_i = \left\{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle \leqslant b_i\right\}; \tag{4}$$

$$H_i = \left\{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i\right\}. \tag{5}$$

Целевая проекция точки $\boldsymbol{z} \in \mathbb{R}^n$ на гиперплоскость H_i вычисляется по формуле

$$\boldsymbol{\gamma}_i(\boldsymbol{z}) = \boldsymbol{z} + \beta_i(\boldsymbol{z})\boldsymbol{c}, \tag{6}$$

где

$$\beta_i(\boldsymbol{z}) = -\frac{\langle \boldsymbol{a}_i, \boldsymbol{z}\rangle - b_i}{\langle \boldsymbol{a}_i, \boldsymbol{c}\rangle} \|\boldsymbol{c}\|. \tag{7}$$

При этом гиперплоскость H_i не должна быть параллельной к вектору $\boldsymbol{c}$. Скалярная величина $\beta_i(\boldsymbol{z})$ называется целевым смещением точки $\boldsymbol{z}$ относительно гиперплоскости H_i.

Определим допустимый многогранник

$$M = \bigcap_{i\in\mathcal{P}} \hat{H}_i, \tag{8}$$

представляющий множество допустимых точек задачи ЛП (1). Заметим, что M в этом случае будет замкнутым выпуклым множеством. Мы будем предполагать, что $M \neq \emptyset$, то есть задача ЛП (1) имеет решение. Обозначим через $\Gamma(M)$ множество граничных точек многогранника M. Под граничной точкой множества $M \subset \mathbb{R}^n$ понимается точка в $\mathbb{R}^n$, для которой любая открытая ее окрестность в $\mathbb{R}^n$ имеет непустое пересечение как с множеством M, так и с его дополнением.

Целевой гиперплоскостью $H_c(\boldsymbol{z})$, проходящей через точку $\boldsymbol{z}$, называется гиперплоскость, задаваемая формулой

$$H_c(\boldsymbol{z}) = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{c}, \boldsymbol{x} \rangle = \langle \boldsymbol{c}, \boldsymbol{z} \rangle\}. \tag{9}$$

Полупространство $\hat{H}_i$ называется рецессивным [37], если

$$\langle \boldsymbol{a}_i, \boldsymbol{c} \rangle > 0. \tag{10}$$

Определим

$$\mathcal{I} = \{i \in \mathcal{P} \,|\langle \boldsymbol{a}_i, \boldsymbol{c} \rangle > 0\}, \tag{11}$$

то есть $\mathcal{I}$ представляет множество индексов, для которых полупространство $\hat{H}_i$ является рецессивным. Поскольку допустимый многогранник M представляет собой ограниченное множество, имеем

$$\mathcal{I} \neq \emptyset. \tag{12}$$

Определим рецессивный многогранник

$$\hat{M} = \bigcap_{i \in \mathcal{I}} \hat{H}_i. \tag{13}$$

Очевидно, что $\hat{M}$ является выпуклым, замкнутым, неограниченным множеством. Из (8) и (11) следует

$$M \subset \hat{M}. \tag{14}$$

Обозначим через $\Gamma(\hat{M})$ множество граничных точек рецессивного многогранника $\hat{M}$. Согласно утверждению 3 в [37] имеем

$$\bar{\boldsymbol{x}} \in \Gamma(\hat{M}), \tag{15}$$

то есть решение задачи ЛП (1) лежит на границе рецессивного многогранника $\hat{M}$.

Целевая проекция $\hat{\boldsymbol{\gamma}}(\boldsymbol{z})$ точки $\boldsymbol{z} \in \mathbb{R}^n$ на границу $\Gamma(\hat{M})$ рецессивного многогранника $\hat{M}$ вычисляется по формуле

$$\hat{\boldsymbol{\gamma}}(\boldsymbol{z}) = \boldsymbol{z} + \hat{\beta}(\boldsymbol{z})\boldsymbol{c}, \tag{16}$$

где $\hat{\beta}(\boldsymbol{z}) = \min\{\beta_i(\boldsymbol{z}) \,|i \in \mathcal{I}\}$. Скалярная величина $\hat{\beta}(\boldsymbol{z})$ называется целевым смещением точки $\boldsymbol{z}$ относительно границы рецессивного многогранника. На основе целевых смещений строится цифровой образ задачи ЛП.

Метод поверхностного движения, описанный в [34], строит на поверхности допустимого многогранника путь из произвольной граничной точки $\boldsymbol{u}^{(0)} \in \hat{M} \cap \Gamma(M)$ до точки $\bar{\boldsymbol{x}}$, являющейся решением задачи ЛП (1). Перемещение по поверхности рецессивного многогранника происходит в направлении наибольшего увеличения целевой функции. Реализация метода поверхностного движения приведена в виде алгоритма 1 в [34].

Ключевым пунктом алгоритма является построение вектора движения $\boldsymbol{d}$ для текущего приближения $\boldsymbol{u}^{(k)} \in \hat{M} \cap \Gamma(M)$. Рассмотрим его подробнее. Сначала строится n-мерный

диск D, являющийся пересечением целевой гиперплоскости $H_c\left(\boldsymbol{u}^{(k)}\right)$, проходящей через точку $\boldsymbol{u}^{(k)}$, и n-мерного шара $V_r\left(\boldsymbol{u}^{(k)}\right)$ малого радиуса r с центром в точке $\boldsymbol{u}^{(k)}$.

$$D = H_c(\boldsymbol{u}^{(k)}) \cap V_r(\boldsymbol{u}^{(k)}). \tag{17}$$

На гипердиске вычисляется точка $\boldsymbol{v}$ с максимальным смещением относительно границы рецессивного многогранника $\hat{M}$.

$$\boldsymbol{v} = \arg\max\{\hat{\beta}(\boldsymbol{x}) \mid \boldsymbol{x} \in D\}. \tag{18}$$

Далее вычисляется точка $\boldsymbol{w}$, являющаяся целевой проекцией $\boldsymbol{v}$ на границу рецессивного многогранника.

$$\boldsymbol{w} = \hat{\boldsymbol{\gamma}}(\boldsymbol{v}). \tag{19}$$

Отметим, что, согласно утверждению 4 из [34], мы всегда можем подобрать такой радиус r для $V_r\left(\boldsymbol{u}^{(k)}\right)$, чтобы исходная точка $\boldsymbol{u}^{(k)}$ и целевая проекция $\boldsymbol{w}$ принадлежали одной и той же гиперплоскости. Вектор $\boldsymbol{d}$ движения по поверхности многогранника M, соответствующий направлению наибольшего увеличения целевой функции, определяется как разность целевой проекции $\boldsymbol{w}$ и точки $\boldsymbol{u}^{(k)}$.

$$\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{u}^{(k)}. \tag{20}$$

Для практической реализации метода поверхностного движения необходим эффективный способ вычисления вектора $\boldsymbol{d}$. Идея эффективного решения этой проблемы, предложенная в работе [36], состоит в следующем: построить на суперкомпьютере визуальный образ окрестности точки $\boldsymbol{u}^{(k)}$ и передать его на вход глубокой нейронной сети, обученной выдавать координаты вектора $\boldsymbol{d}$.

Для построения визуального образа задачи ЛП в [36] строится гиперкубическое рецептивное поля. Гиперкубическое рецептивное поле $\mathfrak{G}_{\text{cube}}(\boldsymbol{z}, \eta, \delta) \subset H_c$ плотности $\delta \in \mathbb{R}_{>0}$ с центром в точке $\boldsymbol{z} \in H_c$ и рангом $\eta \in \mathbb{N}$ представляет собой множество точек, являющихся узлами гиперкубической решетки размерности $n-1$ с фиксированным расстоянием δ между узлами, которая имеет 2η ячеек по каждому измерению. Общее количество точек в гиперкубическом рецептивном поле вычисляется по формуле

$$K_{\text{cube}} = (2\eta + 1)^{n-1}. \tag{21}$$

Пример гиперкубического рецептивного поля в пространстве $\mathbb{R}^3$ приведен на рис. 1а.

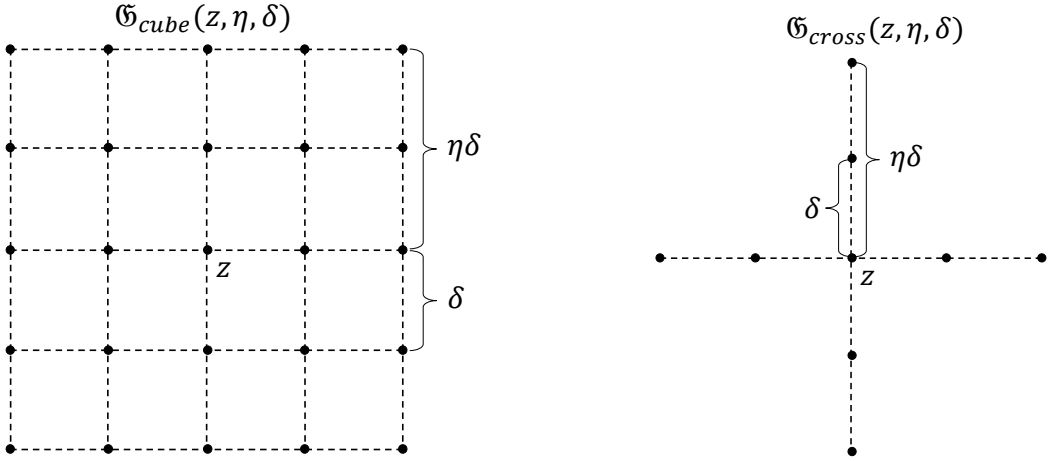
Образом $\mathfrak{I}(\boldsymbol{z}, \eta, \delta)$, порожденным рецептивным полем $\mathfrak{G}(\boldsymbol{z}, \eta, \delta)$, является упорядоченное множество вещественных чисел

$$\mathfrak{I}(\boldsymbol{z}, \eta, \delta) = \left\{ \hat{\beta}(\boldsymbol{g}) \middle| \boldsymbol{g} \in \mathfrak{G}(\boldsymbol{z}, \eta, \delta) \right\}. \tag{22}$$

Полученный образ подается на вход глубокой нейронной сети (DNN). Правильно DNN вычисляет координаты вектора $\boldsymbol{d}$.

$$\boldsymbol{d} = \text{DNN}\left(\mathfrak{I}(\boldsymbol{u}^{(k)}, \eta, \delta)\right). \tag{23}$$

Основная проблема гиперкубического рецептивного поля состоит в том, что с увеличением размерности пространства, число точек рецептивного поля в соответствии с формулой (21)



а) гиперкубическое б) крестообразное

Рис. 1. Две структуры рецептивного поля в пространстве $\mathbb{R}^3$

растет экспоненциально. В следующем разделе предлагается другая конфигурация рецептивного поля, называемая крестообразной. Количество точек в такой конфинурации растет линейно с ростом размерности.

2. Крестообразное рецептивное поле

Построим набор взаимно ортогональных векторов, включающих в себя $\boldsymbol{c}$, следующим образом:

$$
\begin{aligned}
&\boldsymbol{c}^{(0)} = \boldsymbol{c} = (c_1, c_2, c_3, c_4, \ldots, c_{n-1}, c_n);\\
&\boldsymbol{c}^{(1)} = \begin{cases} \left(-\frac{1}{c_1}\sum_{i=2}^{n} c_i^2, c_2, c_3, c_4, \ldots, c_{n-1}, c_n\right), \text{ если } c_1 \neq 0;\\ (1,0,\ldots,0), \text{ если } c_1 = 0; \end{cases}\\
&\boldsymbol{c}^{(2)} = \begin{cases} \left(0, -\frac{1}{c_2}\sum_{i=3}^{n} c_i^2, c_3, c_4, \ldots, c_{n-1}, c_n\right), \text{ если } c_2 \neq 0;\\ (0,1,0,\ldots,0), \text{ если } c_2 = 0; \end{cases}\\
&\boldsymbol{c}^{(3)} = \begin{cases} \left(0,0, -\frac{1}{c_3}\sum_{i=4}^{n} c_i^2, c_4, \ldots, c_{n-1}, c_n\right), \text{ если } c_3 \neq 0;\\ (0,0,1,0,\ldots,0), \text{ если } c_3 = 0; \end{cases}\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\boldsymbol{c}^{(n-2)} = \begin{cases} \left(0,\ldots,0, -\frac{1}{c_{n-2}}\sum_{i=n-1}^{n} c_i^2, c_{n-1}, c_n\right), \text{ если } c_{n-2} \neq 0;\\ (0,\ldots,0,1,0,0), \text{ если } c_{n-2} = 0; \end{cases}\\
&\boldsymbol{c}^{(n-1)} = \begin{cases} \left(0,\ldots,0, -\frac{c_n^2}{c_{n-1}}, c_n\right), \text{ если } c_{n-1} \neq 0;\\ (0,\ldots,0,0,1,0), \text{ если } c_{n-1} = 0. \end{cases}
\end{aligned}
\tag{24}
$$

Легко видеть, что

$$\forall i, j \in \{0,1,\ldots,n-1\}, i \neq j : \left\langle \boldsymbol{c}^{(i)}, \boldsymbol{c}^{(j)} \right\rangle = 0.$$

В том числе

$$\forall i = 1,\ldots,n-1 : \left\langle \boldsymbol{c}, \boldsymbol{c}^{(i)} \right\rangle = 0. \tag{25}$$

Утверждение 4 из [36] показывает, что линейное подпространство размерности $(n-1)$, порождаемое ортогональными векторами $c_1, \ldots, c_{n-1}$, является гиперплоскостью, параллельной гиперплоскости H_c. Обозначим через E_c следующий ортонормированный базис:

$$E_c = \left\{ \boldsymbol{e}^{(i)} = \frac{\boldsymbol{c}^{(i)}}{\|\boldsymbol{c}\|} \middle| i \in \{1, \ldots, n-1\} \right\}. \tag{26}$$

Определение 1. Крестообразным рецептивным полем $\mathfrak{G}_{\text{cross}}(\boldsymbol{z}, \eta, \delta) \subset H_c$ плотности $\delta \in \mathbb{R}_{>0}$ с центром в точке $\boldsymbol{z} \in H_c$ и рангом $\eta \in \mathbb{N}$ будем называть конечное упорядоченное множество точек, расположенных на осях ортонормированного базиса E_c с центром координат в точке $\boldsymbol{z}$, определяющего линейное многообразие H_c, с фиксированным расстоянием δ между соседними точками. На каждой полуоси базиса располагается η точек. Центр координат также включается в крестообразное рецептивное поле.

Крестообразное рецептивное поле может быть построено с помощью алгоритма 1. Прокомментируем его шаги. На шаге 1 создается пустое множество точек $\mathfrak{G}$. На шагах 2–19 цикл **for** на каждой итерации заполняет точками одну ось базиса E_c. На шагах 3–18 цикл **for** создает одну точку. На шаге 4 инициализируется точка $\boldsymbol{s}$, все координаты которой заполнены нулями. Затем цикл **for** на шагах 5–16 вычисляет по очереди каждую координату s_j в базисе E_c и сохраняет ее в $\boldsymbol{s}$. Если номер координаты не совпадает с номером обрабатываемой оси, то такая координата равна 0 (шаги 6–7). Иначе s_j присваивается координата по оси $\boldsymbol{e}^{(j)}$ (шаги 8–14). Если точка расположена на отрицательной полуоси, то координате по очереди присваиваются значения $\{-\eta, \ldots, -1\}$ (шаги 9–10); на положительной полуоси координате присваиваются значения $\{1, \ldots, \eta\}$ (шаги 11–13). На шаге 15 вычисленная координата добавляется к $\boldsymbol{s}$. По окончании записи всех координат в $\boldsymbol{s}$, на шаге 17 новая точка со сдвигом $\boldsymbol{z}$ добавляется в множество точек рецептивного поля $\mathfrak{G}$. После того, как множество точек сформировано, на шаге 20 к нему добавляется центральная точка поля. Шаг 21 завершает работу алгоритма.

Пример крестообразного рецептивного поля в пространстве $\mathbb{R}^3$ приведен на рис. 1б. Общее количество точек крестообразного поля вычисляется по формуле

$$K_{\text{cross}} = 2\eta(n-1) + 1. \tag{27}$$

Действительно, алгоритм 1 в цикле **for** на шагах 2–19 строит точки на каждой из $n-1$ осей, задаваемых векторами из E_c. На каждой оси во вложенном цикле **for** (шаги 3–18) от нулевой точки строится η точек в положительном и η точек в отрицательном направлениях. Всего получается $2\eta(n-1)$ точек. В завершение к получившемуся множеству точек на шаге 20 добавляется центральная точка. В сумме получаем $2\eta(n-1)+1$.

Поскольку хранить постоянно весь массив точек нецелесообразно, на практике координаты точки можно вычислять динамически по ее номеру. Именно этот подход используется при построении образа рецептивного поля в алгоритме 2.

Дадим краткие комментарии по шагам этого алгоритма. Шаг 1 инициализирует пустое множество $\mathfrak{I}$, которое будет содержать целевые смещения точек рецептивного поля. Затем цикл **for** перебирает (шаги 2–19) все точки рецептивного поля, кроме центральной. На шаге 3 вычисляется номер оси, на которой расположена точка. На шаге 4 вычисляется номер точки на оси. На шагах 5–10 вычисляются координаты точки. На шаге 5 в качестве исходных принимаются координаты центральной точки рецептивного поля. В зависимости от того, расположена вычисляемая точка на отрицательной или положительной полуоси (шаг 6), к

Алгоритм 1 Построение крестообразного рецептивного поля $\mathfrak{G}_{\text{cross}}(z, \eta, \delta)$

Require: $\boldsymbol{z} \in H_c$, $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$

1: $\mathfrak{G} := \emptyset$
2: **for** $t = 1 \ldots n-1$ **do**
3: **for** $i = 1 \ldots 2\eta$ **do**
4: $\boldsymbol{s} := \boldsymbol{0}$
5: **for** $j = 1 \ldots n-1$ **do**
6: **if** $j \neq t$ **then**
7: $s_j = 0$
8: **else**
9: **if** $i \leqslant \eta$ **then**
10: $s_j := (i - \eta - 1)\delta$
11: **else**
12: $s_j := (i - \eta)\delta$
13: **end if**
14: **end if**
15: $\boldsymbol{s} := \boldsymbol{s} + s_j \boldsymbol{e}^{(j)}$
16: **end for**
17: $\mathfrak{G} := \mathfrak{G} \cup \{\boldsymbol{s} + \boldsymbol{z}\}$
18: **end for**
19: **end for**
20: $\mathfrak{G} := \mathfrak{G} \cup \{\boldsymbol{z}\}$
21: **stop**

исходным координатам добавляется перемещение в отрицательную сторону (шаг 7), либо в положительную (шаг 9). Центральная точка на этом этапе не обрабатывается, так как она для всех осей общая. Далее на шагах 11–18 с помощью формулы (16) вычисляется целевое смещение полученной точки рецептивного поля относительно границы рецессивного многогранника. После вычисления всех точек рецептивного поля, кроме центральной, на шаге 20 к образу добавляется еще одно смещение, равное нулю, так как центральная точка рецептивного поля всегда располагается на поверхности допустимого многогранника. Следующее утверждение дает оценку временной сложности описанного алгоритма.

Утверждение 1. Алгоритм 2 имеет временную сложностью $O(n^2 m)$.

Доказательство. Рассмотрим алгоритм 2. Шаги 5, 7, 9, 11 и 13 имеет временную сложность $O(n)$. Остальные шаги, за исключением операторов цикла **for**, имеют вычислительную сложность $O(1)$. Количество итераций внутреннего цикла **for** (шаг 12) можно оценить как $O(m)$. Следовательно, шаги 12–17 имеют суммарную временную сложность $O(nm)$. Количество итераций внешнего цикла **for** (шаг 2) может быть оценено как $O(n)$. Таким образом, временная сложность алгоритма 2 в целом может быть оценена как $O(n^2 m)$. □

Алгоритм 2 Построение крестообразного образа $\mathfrak{I}_{\text{cross}}(\boldsymbol{z}, \eta, \delta)$

Require: $\boldsymbol{z} \in H_c$, $\eta \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$

$\mathfrak{I} := \emptyset$
for $k = 1 \ldots 2\eta(n-1)$ **do**
  $l := \lfloor (k-1)/2\eta \rfloor + 1$
  $p := (k-1) \mod 2\eta + 1$
  $\boldsymbol{g} := \boldsymbol{z}$
  if $p \leqslant \eta$ **then**
    $\boldsymbol{g} := \boldsymbol{g} + (p - \eta - 1)\delta \boldsymbol{e}^{(l)}$
  else
    $\boldsymbol{g} := \boldsymbol{g} + (p - \eta)\delta \boldsymbol{e}^{(l)}$
  end if
  $\hat{\beta} := -\dfrac{\langle \boldsymbol{a}_1, \boldsymbol{g} \rangle - b_1}{\langle \boldsymbol{a}_1, \boldsymbol{c} \rangle} \|\boldsymbol{c}\|$
  for $i = 2 \ldots m$ **do**
    $\beta_i := -\dfrac{\langle \boldsymbol{a}_i, \boldsymbol{g} \rangle - b_i}{\langle \boldsymbol{a}_i, \boldsymbol{c} \rangle} \|\boldsymbol{c}\|$
    if $\beta_i \leqslant \hat{\beta}$ **then**
      $\hat{\beta} := \beta_i$
    end if
  end for
  $\mathfrak{I} := \mathfrak{I} \cup \{\hat{\beta}\}$
end for
$\mathfrak{I} := \mathfrak{I} \cup \{0\}$
stop

3. Построение нейронной сети для движения по гиперплоскости

Пусть в $\mathbb{R}^n$ заданы случайный вектор $\boldsymbol{a}$ и гиперплоскость $H_{\boldsymbol{a}}$, ортогональная вектору $\boldsymbol{a}$. Без ограничения общности мы можем полагать, что гиперплоскость $H_{\boldsymbol{a}}$ проходит через нулевой вектор:

$$H_{\boldsymbol{a}} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \langle \boldsymbol{a}, \boldsymbol{x} \rangle = \boldsymbol{0}\}. \tag{28}$$

Зададим произвольный вектор $\boldsymbol{c}$ такой, что

$$\langle \boldsymbol{a}, \boldsymbol{c} \rangle > 0. \tag{29}$$

Гиперплоскость $H_{\boldsymbol{a}}$ представляет собой случайную грань некоторого допустимого многогранника M, а вектор $\boldsymbol{c}$ — градиент случайной целевой функции. При этом полупространство $\hat{H}_{\boldsymbol{a}}$, задаваемое формулой

$$\hat{H}_{\boldsymbol{a}} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \langle \boldsymbol{a}, \boldsymbol{x} \rangle \leqslant 0\}, \tag{30}$$

является рецессивным по отношению к вектору $\boldsymbol{c}$. Очевидно, что в контексте задачи ЛП $M \subset \hat{H}_{\boldsymbol{a}}$ и $H_{\boldsymbol{a}} \cap \Gamma(M) \neq \emptyset$.

Для построения цифрового образа гиперплоскости $H_{\boldsymbol{a}}$, используем целевую гиперплоскость в точке $\mathbf{0}$:

$$H_{\boldsymbol{c}}(\mathbf{0}) = \{\boldsymbol{x} \in \mathbb{R}^n \mid \langle \boldsymbol{c}, \boldsymbol{x} \rangle = \mathbf{0}\}. \tag{31}$$

Воспользовавшись (24) и (26), сформируем базис $E_{\boldsymbol{c}} \subset H_{\boldsymbol{c}}(\mathbf{0})$. С помощью алгоритма 4 из [36] построим гиперкубический образ $\mathfrak{I}_{\text{cube}}(\mathbf{0}, \eta, \delta)$, и с помощью алгоритма 2, описанного выше, построим крестообразный образ $\mathfrak{I}_{\text{cross}}(\mathbf{0}, \eta, \delta)$. Далее мы будем обозначать эти образы как $\mathfrak{I}_{\text{cube}}$ и $\mathfrak{I}_{\text{cross}}$. С помощью функции

$$\varphi_{\mathfrak{I}}(\rho) = 511(\rho - \min(\mathfrak{I}))/(\max(\mathfrak{I}) - min(\mathfrak{I})) - 256 \tag{32}$$

выполним нормализацию обоих образов:

$$\bar{\mathfrak{I}}_{\text{cube}} = \left\{\varphi_{\mathfrak{I}_{\text{cube}}}(\rho) \mid \rho \in \mathfrak{I}_{\text{cube}}\right\}, \tag{33}$$

$$\bar{\mathfrak{I}}_{\text{cross}} = \left\{\varphi_{\mathfrak{I}_{\text{cross}}}(\rho) \mid \rho \in \mathfrak{I}_{\text{cross}}\right\}. \tag{34}$$

Для построения обучающего множества необходимо сопоставить нормализованному образу правильный вектор движения $\boldsymbol{d_a}$ по гиперплоскости $H_{\boldsymbol{a}}$. Обозначим через $\pi_{\boldsymbol{a}}(\boldsymbol{c})$ ортогональную проекцию вектора $\boldsymbol{c}$ на гиперплоскость $H_{\boldsymbol{a}}$:

$$\pi_{\boldsymbol{a}}(\boldsymbol{c}) = \boldsymbol{c} - \frac{\langle \boldsymbol{a}, \boldsymbol{c} \rangle}{\|\boldsymbol{a}\|^2}\boldsymbol{a}. \tag{35}$$

Очевидно, что ортогональная проекция $\pi_{\boldsymbol{a}}(\boldsymbol{c})$ указывает направление максимального увеличения значения целевой функции на гиперплоскости $H_{\boldsymbol{a}}$. Таким образом

$$\boldsymbol{d_a} = \pi_{\boldsymbol{a}}(\boldsymbol{c}). \tag{36}$$

В качестве правильного ответа вместо n координат вектора $\boldsymbol{d_a}$ в $\mathbb{R}^n$ будем указывать $n-1$ координату в базисе $E_{\boldsymbol{c}}$. Обозначим через $\boldsymbol{g_a} \in H_{\boldsymbol{c}}$ ортогональную проекцию вектора $\boldsymbol{d_a}$ на гиперплоскость $H_{\boldsymbol{c}}$:

$$\boldsymbol{g_a} = \boldsymbol{d_a} - \frac{\langle \boldsymbol{c}, \boldsymbol{d_a} \rangle}{\|\boldsymbol{c}\|^2}\boldsymbol{c}. \tag{37}$$

Используя (36) и (35), данную формулу можно преобразовать к виду

$$\boldsymbol{g_a} = \frac{\langle \boldsymbol{c}, \boldsymbol{a} \rangle}{\|\boldsymbol{c}\|^2}\boldsymbol{c} - \boldsymbol{a}. \tag{38}$$

Коэффициентом угла наклона вектора $\boldsymbol{g_a}$ к базисному вектору $\boldsymbol{e}^{(i)}$ назовем косинус угла между векторами $\boldsymbol{g_a}$ и $\boldsymbol{e}^{(i)}$:

$$\cos\alpha_i = \frac{\left\langle \boldsymbol{e}^{(i)}, \boldsymbol{g_a} \right\rangle}{\|\boldsymbol{g_a}\|}. \tag{39}$$

Взятые вместе, коэффициенты углов наклона образуют вектор правильного ответа

$$\boldsymbol{y_a} = \left(\frac{\left\langle \boldsymbol{e}^{(1)}, \boldsymbol{g_a} \right\rangle}{\|\boldsymbol{g_a}\|}, \ldots, \frac{\left\langle \boldsymbol{e}^{(n-1)}, \boldsymbol{g_a} \right\rangle}{\|\boldsymbol{g_a}\|} \right). \tag{40}$$

Для настройки оптимальных гиперпараметров нейронной сети была разработана гипермодель, представленная на рис. 2. Эта гипермодель включает входной слой, блок скрытых

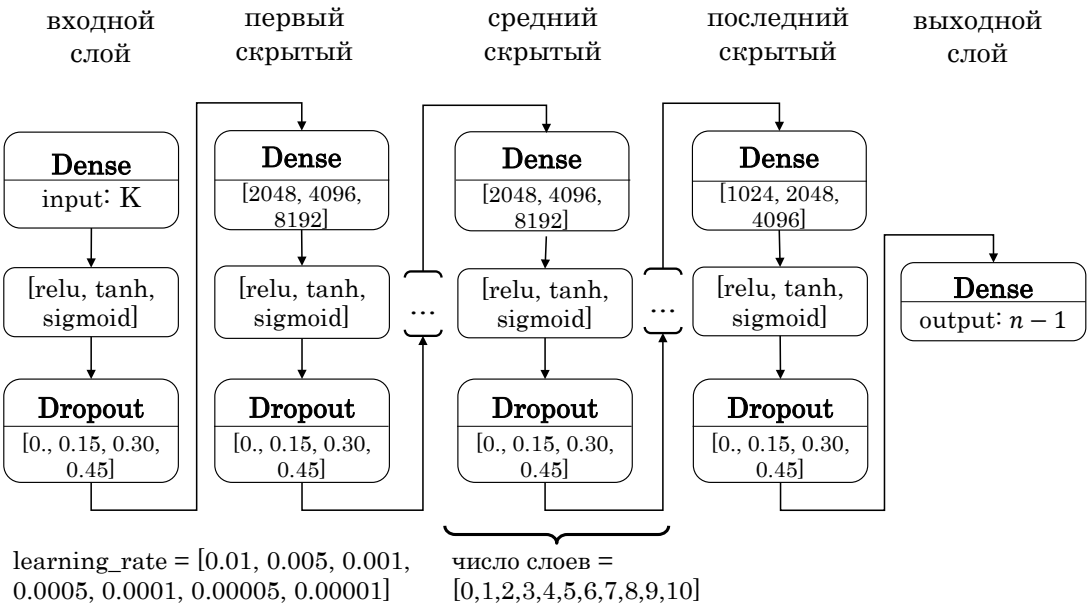


Рис. 2. Архитектура гипермодели

Таблица 1. Параметры рецептивного поля

Параметр	Семантика	Значение
n	размерность пространства	10
η	ранг рецептивного поля	5
δ	расстояние между соседними точками	1
K_{cube}	число точек рецептивного поля	91

слоев и выходной слой. Блок скрытых слоев состоит из первого скрытого слоя, переменного числа средних скрытых слоев и последнего скрытого слоя. Все слои являются полностью связанными (dense connection). Выбор функции активации для всех слоев осуществлялся из набора $\{ReLU, sigmoid, tanh\}$. Входной слой имеет K нейронов, соответствующих точкам используемого рецептивного поля. Выходной слой имеет $n-1$ нейронов, соответствующих числу коэффициентов углов наклона вектора $\boldsymbol{g_a}$.

Подбор гиперпараметров выполнялся на обучающем множестве, сгенерированном в пространстве $\mathbb{R}^{10}$ с конфигурацией рецептивного поля, представленной в табл. 1. Для генерации случайных координат векторов $\boldsymbol{a}$ и $\boldsymbol{c}$ использовалось стандартное нормальное распределение. Число нейронов для скрытых слоев подбиралось из набора $\{1024, 2048, 4096, 8192\}$. Количество средних скрытых слоев подбиралось из набора $\{0, 1, ..., 8\}$. Заметим, что максимально возможное число скрытых слоев при таком выборе параметров равнялось 10, что соответствует заданной размерности пространства $n = 10$.

Основываясь на представленной гипермодели, был выполнен байесовский поиск оптимального набора гиперпараметров с использованием платформы W&B [38]. При обучении был использован оптимизатор *RMSProp* [39]. Размер блока обучающих прецедентов (batch size) равнялся 128. В качестве функции потерь (loss) была использована косинусная мера (cosine similarity)

$$\text{CS} = -\frac{\sum_{k=1}^{n-1}(\boldsymbol{\alpha}_k \cdot \boldsymbol{y}_k)}{\sqrt{\sum_{k=1}^{n-1}\boldsymbol{\alpha}_k^2} \cdot \sqrt{\sum_{k=1}^{n-1}\boldsymbol{y}_k^2}}. \tag{41}$$

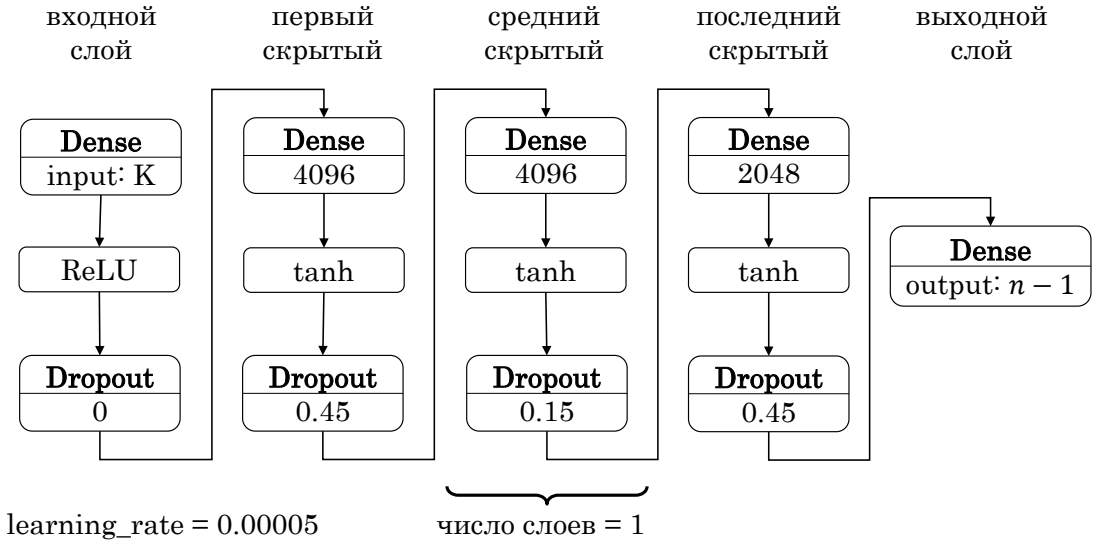


Рис. 3. Архитектура нейронной сети

Здесь $\boldsymbol{\alpha}_k$ — это значения коэффициентов наклона, предсказанные нейронной сетью, $\boldsymbol{y}_k$ — коэффициенты наклона, рассчитанные на основе формулы (40). Набор обучающих данных из 100 000 прецедентов был разделен следующим образом:

- обучающая выборка: 80000 элементов;
- тестовая выборка: 15000 элементов;
- валидационная выборка: 5000 элементов.

Для обучения и тестирования нейронных сетей использовался комплекс «Нейрокомпьютер» Южно-Уральского государственного университета [40], оборудованный графическими процессорами nVidia Tesla V100. Обучение производилось при помощи библиотек `keras` и `TensorFlow`. В результате была получена глубокая ИНС, архитектура которой представлена на рис. 3. Для оценки качества работы нейронных сетей была использована оригинальная модификация средней абсолютной ошибки, получившая название «средняя абсолютная нормализованная ошибка» MANE (mean absolute normalized error):

$$\mathrm{MANE} = \frac{1}{n} \cdot \sum_{i=1}^{n} \left| \frac{y_i}{\|\boldsymbol{y}\|} - \frac{\alpha_i}{\|\boldsymbol{\alpha}\|} \right|. \tag{42}$$

Создание обучающего множества и обучение искусственной нейронной сети производилось по схеме, представленной на рис. 4. Сначала программа генерации случайных гиперплоскостей, приняв в качестве входного параметра число Q, генерирует Q пар $\{\boldsymbol{a}, \boldsymbol{c}\}$, где каждая пара состоит из случайного вектора $\boldsymbol{a}$, определяющего гиперплоскость (28) и случайного вектора целевой функции $\boldsymbol{c}$. Массив сгенерированных данных помещается в файл, передаваемый программе визуализации. Визуализатор принимает входные параметры, определяющие положение точек рецептивного поля:

- ранг рецептивного поля η;
- плотность рецептивного поля δ;
- форма рецептивного поля `cube` (гиперкубическое) или `cross` (крестообразное).

В результате работы визуализатора формируется файл прецедентов, содержащий Q строк. В каждой строке записаны K нормализованных по формуле (32) коэффициентов, форми-

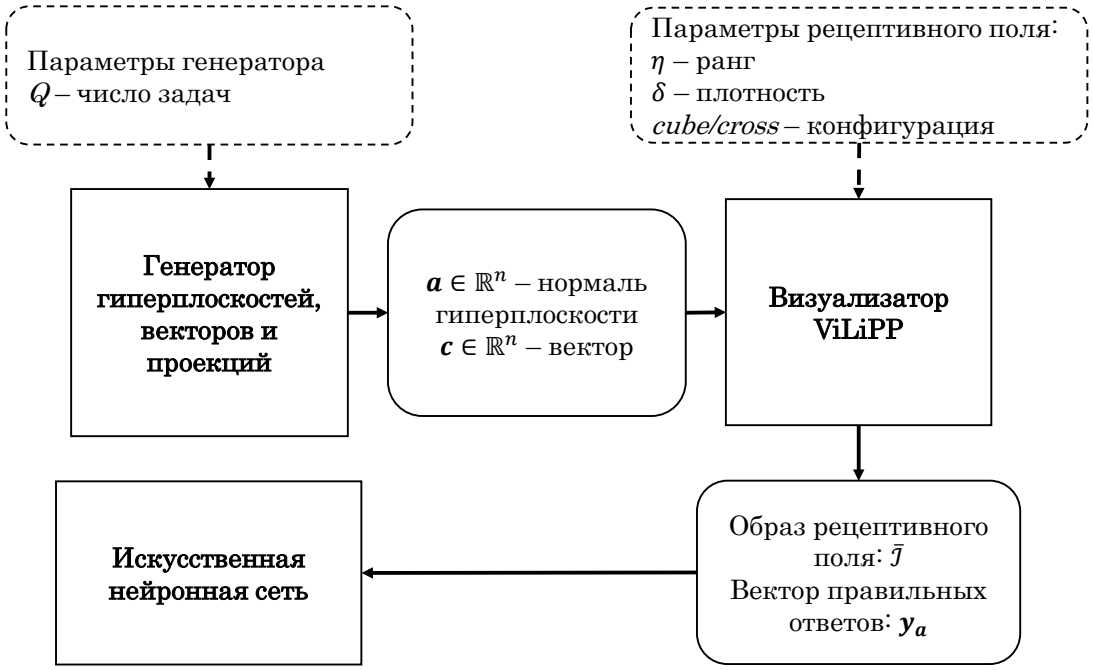


Рис. 4. Архитектура программного комплекса для построения обучающего множества

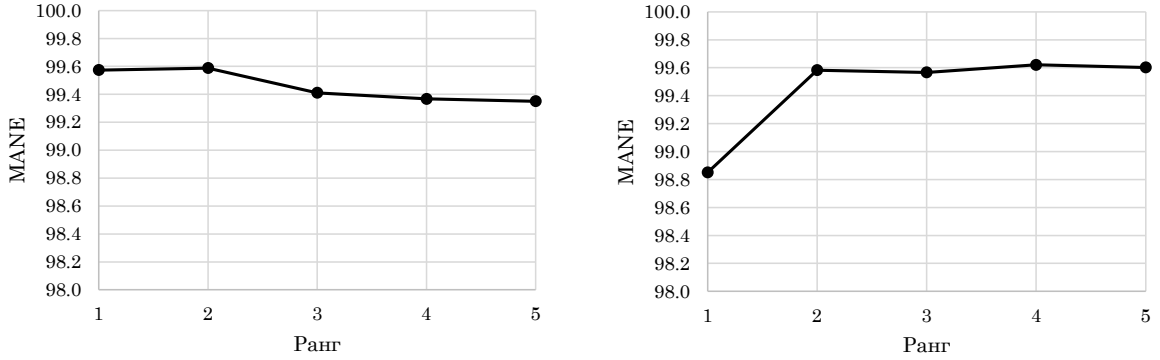
рующих образ $\bar{\jmath}$, и $n-1$ коэффициентов угла наклона, формирующих вектор правильного ответа $\boldsymbol{y_a}$.

4. Вычислительные эксперименты

В первой серии экспериментов исследовалось влияние конфигурации и числа точек рецептивного поля на точность работы ИНС. На основе архитектуры, представленной на рис. 3, были построены 10 нейронных сетей, отличающихся числом входных нейронов. Число входных нейронов соответствовало гиперкубической и крестообразной конфигурациям рецептивного поля. Для каждой конфигурации перебиралось значение ранга η от 1 до 5. Результаты представлены на рис. 5. На графиках видно, что крестообразное рецептивное поле незначительно уступает в точности гиперкубическому. При этом для размерности $n = 4$ наилучшие результаты крестообразное поле демонстрирует, начиная с ранга 2 (рис. 5б). Гиперкубическое рецептивное поле для этой размерности демонстрирует наилучший результат также при ранге 2, однако дальнейшее увеличение ранга приводит к ухудшению точности (рис. 5а). Основываясь на полученных результатах, для дальнейших экспериментов с задачами больших размерностей было выбрано крестообразное рецептивное поле ранга 5.

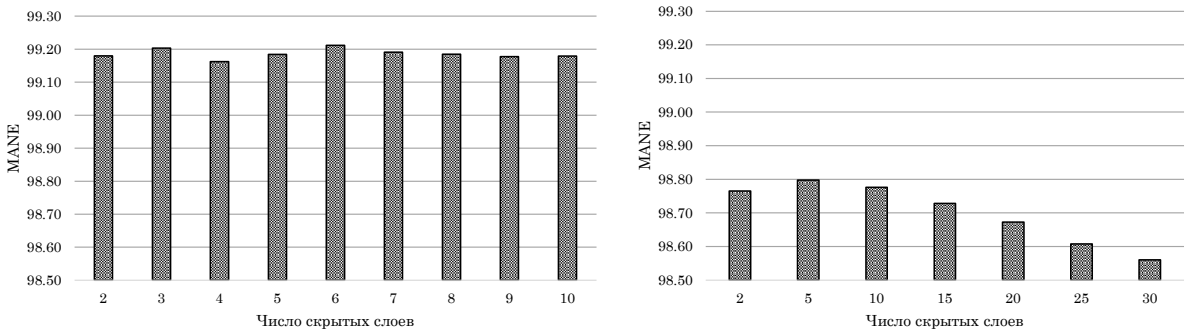
Во второй серии вычислительных экспериментов была исследована зависимость точности ИНС от числа скрытых слоев. Для пространств $\mathbb{R}^{10}$ и $\mathbb{R}^{30}$ был построен набор нейронных сетей на основе архитектуры, представленной на рис. 3, для которых число скрытых слоев варьировалось от 2 до n. Результаты представлены на рис. 6. Для размерности 10 число скрытых слоев практически не влияет на точность работы ИНС (рис. 6а). Однако для размерности 30 лучший результат показывает ИНС с пятью скрытыми слоями (рис. 6б).

В третьей серии экспериментов была предпринята попытка улучшить точность работы ИНС за счет увеличения времени обучения. Были протестированы ИНС для пространств $\mathbb{R}^{10}$ и $\mathbb{R}^{30}$. Обучение нейронной сети в $\mathbb{R}^{10}$ происходило на протяжении 900 эпох, в про-



а) гиперкубическое б) крестообразное

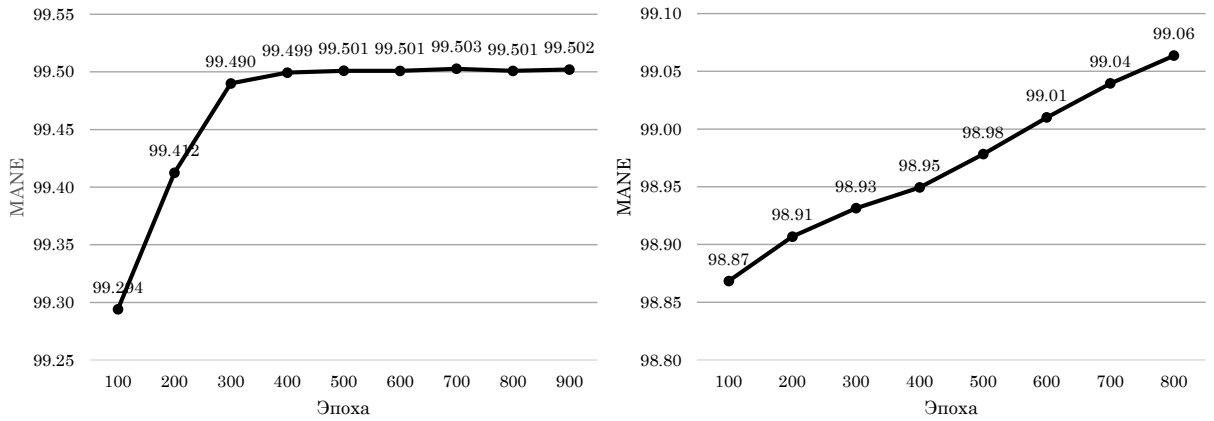
Рис. 5. Зависимость точности от конфигурации и ранга рецептивного поля в $\mathbb{R}^4$



а) в пространстве $\mathbb{R}^{10}$ б) в пространстве $\mathbb{R}^{30}$

Рис. 6. Зависимость точность ИНС от числа скрытых слоев

странстве $\mathbb{R}^{30}$ — 800 эпох. И в том и в другом случае время обучения составило 60 мин. Результаты представлены на рис. 7. Обращает на себя внимание тот факт, что в $\mathbb{R}^{10}$ максимальная точность достигается на 400 эпохе и далее существенно уже не меняется. В $\mathbb{R}^{30}$ точность увеличивалась практически линейно вплоть до 800 эпохи. Таким образом, можно сделать предположение, что с ростом размерности пространства время обучения ИНС существенно увеличивается.



а) в пространстве $\mathbb{R}^{10}$ б) в пространстве $\mathbb{R}^{30}$

Рис. 7. Зависимость точность ИНС от количество эпох обучения

Заключение

В настоящей работе исследован метод, позволяющий при помощи глубокой нейронной сети прямого распространения вычислять направление движения по гиперплоскостям, ограничивающим допустимый многогранник задачи ЛП. Вычисляемое направление движения соответствует максимальному увеличению значений целевой функции. Улучшен ранее разработанный метод визуализации поверхности многогранника. Представлена новая крестообразная конфигурация рецептивного поля, позволяющая избежать экспоненциального роста числа точек визуализации при увеличении размерности задачи ЛП. Для крестообразной конфигурации рецептивного поля разработан метод генерации обучающих прецедентов, по которому созданы обучающие наборы данных, визуализирующие положение случайных гиперплоскостей и целевых векторов в пространствах размерности до 30 включительно. С использованием обучающих наборов разработана масштабируемая архитектура глубокой ИНС с изменяемым числом скрытых слоев и проведены несколько серий вычислительных экспериментов. Результаты экспериментов свидетельствуют о том, что предложенная крестообразная конфигурация рецептивного поля с точки зрения точности результатов, выдаваемых ИНС, практически не отличается от гиперкубической конфигурации. Испытания крестообразного рецептивного поля показали, что точность основанной на нем ИНС не зависит от числа скрытых слоев. Точность более 98% по метрике MANE достигается в $\mathbb{R}^{10}$ и $\mathbb{R}^{30}$ ИНС с двумя скрытыми слоями на 50 эпохе. Установлено, что для крестообразной конфигурации наилучшая точность достигается при ранге 3 и с дальнейшим увеличением ранга существенно не изменяется. Скорость обучения зависит от размерности пространства. В $\mathbb{R}^{10}$ наилучший результат достигается на 400 эпохе обучения и в дальнейшем не изменяется. В $\mathbb{R}^{30}$ до 800 эпохи точность продолжает расти линейно и достигает значения 99.06%.

В качестве направления дальнейших исследований отметим следующие шаги, реализация которых позволит построить принципиально новый способ решения многомерных задач линейного программирования.

- Разработка нейросетевой модели для вычисления вектора $\boldsymbol{d}$ в точке, лежащей на ребре — многообразии размерности $n-k$, образованном пересечением k гиперплоскостей.
- Разработка ансамбля нейросетевых моделей для вычисления вектора $\boldsymbol{d}$ в любой точке допустимого многогранника размерности n.
- Исследование возможности применения сверточных нейронных сетей при распознавании сложных структур на поверхности допустимого многогранника.

Исследование выполнено при финансовой поддержке РНФ (проект № 23-21-00356).

Литература

1. Соколинская И.М., Соколинский Л.Б. О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии – XI международная конференция (ПаВТ'2017): Короткие статьи и описания плакатов, Казань, 3–7 апреля, 2017. Челябинск: Издательский центр ЮУрГУ, 2017. С. 471–484. URL: http://omega.sp.susu.ru/pavt2017/short/014.pdf.

2. Brogaard J., Hendershott T., Riordan R. High-Frequency Trading and Price Discovery // Review of Financial Studies. 2014. Vol. 27, no. 8. P. 2267–2306. DOI: 10.1093/rfs/hhu032.

3. Deng S., Huang X., Wang J., *et al.* A Decision Support System for Trading in Apple Futures Market Using Predictions Fusion // IEEE Access. 2021. Vol. 9. P. 1271–1285. DOI: 10.1109/ACCESS.2020.3047138.
4. Seregin G. Lecture Notes on Regularity Theory for the Navier-Stokes Equations. WORLD SCIENTIFIC, 2014. DOI: 10.1142/9314.
5. Demin D. Synthesis of optimal control of technological processes based on a multialternative parametric description of the final state // Eastern-European Journal of Enterprise Technologies. 2017. Vol. 3, no. 4 (87). P. 51–63. DOI: 10.15587/1729-4061.2017.105294.
6. Kazarinov L.S., Shnayder D.A., Kolesnikova O.V. Heat load control in steam boilers // 2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM). IEEE, 2017. P. 1–4. DOI: 10.1109/ICIEAM.2017.8076177.
7. Zagoskina E., Barbasova T., Shnaider D. Intelligent Control System of Blast-furnace Melting Efficiency // 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON). IEEE, 2019. P. 0710–0713. DOI: 10.1109/SIBIRCON48586.2019.8958221.
8. Fleming J., Yan X., Allison C., *et al.* Real-time predictive eco-driving assistance considering road geometry and long-range radar measurements // IET Intelligent Transport Systems. 2021. Vol. 15, no. 4. P. 573–583. DOI: 10.1049/itr2.12047.
9. Scholl M., Minnerup K., Reiter C., *et al.* optimization of a Thermal Management System for Battery Electric Vehicles // 2019 Fourteenth International Conference on Ecological Vehicles and Renewable Energies (EVER). IEEE, 2019. P. 1–10. DOI: 10.1109/EVER.2019.8813657.
10. Meisel S. Dynamic Vehicle Routing // Anticipatory Optimization for Dynamic Decision Making. Vol. 51. New York, NY: Springer, 2011. P. 77–96. Operations Research/Computer Science Interfaces Series. DOI: 10.1007/978-1-4614-0505-4_6.
11. Cheng A.M.K. Real-Time Scheduling and Schedulability Analysis // Real-Time Systems. Wiley, 2002. P. 41–85. DOI: 10.1002/0471224626.ch3.
12. Kopetz H. Real-Time Scheduling // Real-Time Systems. Boston, MA: Springer, 2011. P. 239–258. Real-Time Systems Series. DOI: 10.1007/978-1-4419-8237-7_10.
13. Dantzig G.B. Linear programming and extensions. Princeton university press, 1998. 656 p.
14. Hall J.A.J., McKinnon K.I.M. Hyper-Sparsity in the Revised Simplex Method and How to Exploit it // Computational Optimization and Applications. 2005. Vol. 32, no. 3. P. 259–283. DOI: 10.1007/s10589-005-4802-0.
15. Klee V., Minty G.J. How good is the simplex algorithm? // Inequalities — III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969 / ed. by O. Shisha. Academic Press, 1972. P. 159–175.
16. Jeroslow R. The simplex algorithm with the pivot rule of maximizing criterion improvement // Discrete Mathematics. 1973. Vol. 4, no. 4. P. 367–377. DOI: 10.1016/0012-365X(73)90171-4.
17. Zadeh N. A bad network problem for the simplex method and other minimum cost flow algorithms // Mathematical Programming. 1973. Vol. 5, no. 1. P. 255–266. DOI: 10.1007/BF01580132.
18. Bartels R.H., Stoer J., Zenger C. A Realization of the Simplex Method Based on Triangular Decompositions // Handbook for Automatic Computation. Berlin, Heidelberg: Springer Berlin Heidelberg, 1971. P. 152–190. DOI: 10.1007/978-3-642-86940-2_11.

19. Tolla P. A Survey of Some Linear Programming Methods // Concepts of Combinatorial Optimization / ed. by V.T. Paschos. 2nd ed. Hoboken, NJ, USA: Wiley, 2014. P. 157–188. DOI: 10.1002/9781119005216.ch7.

20. Hall J.A.J. Towards a practical parallelisation of the simplex method // Computational Management Science. 2010. Vol. 7, no. 2. P. 139–170. DOI: 10.1007/s10287-008-0080-5.

21. Mamalis B., Pantziou G. Advances in the Parallelization of the Simplex Method // Algorithms, Probability, Networks, and Games. Vol. 9295 / ed. by C. Zaroliagis, G. Pantziou, S. Kontogiannis. Springer, 2015. P. 281–307. Lecture Notes in Computer Science. DOI: 10.1007/978-3-319-24024-4_17.

22. Lubin M., Hall J.A.J., Petra C.G., Anitescu M. Parallel distributed-memory simplex for large-scale stochastic LP problems // Computational Optimization and Applications. 2013. Vol. 55, no. 3. P. 571–596. DOI: 10.1007/s10589-013-9542-y.

23. Khachiyan L. Polynomial algorithms in linear programming // USSR Computational Mathematics and Mathematical Physics. 1980. Vol. 20, no. 1. P. 53–72. DOI: 10.1016/0041-5553(80)90061-0.

24. Shor N.Z. Cut-off method with space extension in convex programming problems // Cybernetics. 1977. Vol. 13, no. 1. P. 94–96. DOI: 10.1007/BF01071394.

25. Юдин Д.Б., Немировский А.С. Информационная сложность и эффективные методы решения выпуклых экстремальных задач // Экономика и математические методы. 1976. № 2. С. 357–369.

26. Karmarkar N. A new polynomial-time algorithm for linear programming // Combinatorica. 1984. Vol. 4, no. 4. P. 373–395. DOI: 10.1007/BF02579150.

27. Gondzio J. Interior point methods 25 years later // European Journal of Operational Research. 2012. Vol. 218, no. 3. P. 587–601. DOI: 10.1016/j.ejor.2011.09.017.

28. Fathi-Hafshejani S., Mansouri H., Peyghami M.R., Chen S. Primal–dual interior-point method for linear optimization based on a kernel function with trigonometric growth term // Optimization. 2018. Vol. 67, no. 10. P. 1605–1630. DOI: 10.1080/02331934.2018.1482297.

29. Asadi S., Mansouri H. A Mehrotra type predictor-corrector interior-point algorithm for linear programming // Numerical Algebra, Control & Optimization. 2019. Vol. 9, no. 2. P. 147–156. DOI: 10.3934/naco.2019011.

30. Yuan Y. Implementation tricks of interior-point methods for large-scale linear programs // International Conference on Graphic and Image Processing (ICGIP 2011). Vol. 8285. International Society for Optics, Photonics, 2011. P. 828502. DOI: 10.1117/12.913019.

31. Kheirfam B., Haghighi M. A full-Newton step infeasible interior-point method for linear optimization based on a trigonometric kernel function // Optimization. 2016. Vol. 65, no. 4. P. 841–857. DOI: 10.1080/02331934.2015.1080255.

32. Xu Y., Zhang L., Zhang J. A full-modified-Newton step infeasible interior-point algorithm for linear optimization // Journal of Industrial and Management Optimization. 2015. Vol. 12, no. 1. P. 103–116. DOI: 10.3934/jimo.2016.12.103.

33. Roos C., Terlaky T., Vial J.-P. Interior Point Methods for Linear Optimization. Springer-Verlag, 2005. 500 p. DOI: 10.1007/b100325.

34. Ольховский Н.А., Соколинский Л.Б. О новом методе линейного программирования // Вычислительные методы и программирование. 2023. Т. 24, № 4. С. 408–429. DOI: 10.26089/NumMet.v24r428.

35. Соколинский Л.Б., Соколинская И.М. Об одном итерационном методе решения задач линейного программирования на кластерных вычислительных системах // Вычислительные методы и программирование. 2020. Т. 21, № 3. С. 329–340. DOI: 10.26089/NUMMET.V21R328.

36. Ольховский Н.А., Соколинский Л.Б. Визуальное представление многомерных задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. Т. 11, № 1. С. 31–56. DOI: 10.14529/cmse220103.

37. Соколинский Л.Б., Соколинская И.М. О новой версии апекс-метода для решения задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 2. С. 5–46. DOI: 10.14529/cmse230201.

38. Biewald L. Experiment Tracking with Weights and Biases // Software available from wandb.com. 2020. January.

39. Goodfellow I., Bengio Y., Courville A. Deep Learning (Adaptive Computation and Machine Learning). Vol. 8. 2016.

40. Биленко Р.В., Долганина Н.Ю., Иванова Е.В., Рекачинский А.И. Высокопроизводительные вычислительные ресурсы Южно-Уральского государственного университета // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. Т. 11, № 1. С. 15–30. DOI: 10.14529/cmse220102.

Ольховский Николай Александрович, младший научный сотрудник, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse230402

STUDY OF NEURAL NETWORK MODELS FOR LINEAR PROGRAMMING

© 2023 N.A. Olkhovsky

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: olkhovskiina@susu.ru

Received: 20.10.2023

The article explores a method for determining the motion vector from hyperplanes that bound the feasible polytop of a multidimensional linear programming problem. The method is based on visual images fed to the input of a feedforward neural network. The visualization algorithm constructs a receptive field in the vicinity of a point located on the bounding hyperplane. For each point of the receptive field, the scalar bias to the hyperplane surface is calculated. Based on the calculated bias, each receptive field point is assigned with a scalar value. The resulting visual image is fed to the input of a feed-forward neural network, which calculates the direction of maximum increase in the objective function on the bounding hyperplane. The article proposes an improved form of the cross-shaped receptive field. The construction of a training set based on randomly generated bounding hyperplanes and objective functions in multidimensional spaces is described. A scalable neural network architecture with a variable number of hidden layers has been developed. The hyperparameters of the neural network were selected. Computational experiments confirmed the high (more than 98%) accuracy of the cross-shaped receptive field. The dependence of the accuracy of the neural network results on the number of hidden layers and the duration of training was studied.

Keywords: linear programming, surface movement method, artificial neural network, deep learning.

FOR CITATION

Olkhovsky N.A. Study of Neural Network Nodels for Linear Programming. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2023. Vol. 12, no. 4. P. 55–75. (in Russian) DOI: 10.14529/cmse230402.

References

1. Sokolinskaya I.M., Sokolinsky L.B. On the Solution of Linear Programming Problems in the Age of Big Data. Parallel Computational Technologies. Vol. 753 / ed. by L.B. Sokolinsky, M.L. Zymbler. Cham: Springer, 2017. P. 86–100. Communications in Computer and Information Science. DOI: 10.1007/978-3-319-67035-5_7.
2. Brogaard J., Hendershott T., Riordan R. High-Frequency Trading and Price Discovery. Review of Financial Studies. 2014. Vol. 27, no. 8. P. 2267–2306. DOI: 10.1093/rfs/hhu032.
3. Deng S., Huang X., Wang J., *et al.* A Decision Support System for Trading in Apple Futures Market Using Predictions Fusion. IEEE Access. 2021. Vol. 9. P. 1271–1285. DOI: 10.1109/ACCESS.2020.3047138.
4. Seregin G. Lecture Notes on Regularity Theory for the Navier-Stokes Equations. WORLD SCIENTIFIC, 2014. DOI: 10.1142/9314.
5. Demin D. Synthesis of optimal control of technological processes based on a multialternative parametric description of the final state. Eastern-European Journal of Enterprise Technologies. 2017. Vol. 3, no. 4 (87). P. 51–63. DOI: 10.15587/1729-4061.2017.105294.
6. Kazarinov L.S., Shnayder D.A., Kolesnikova O.V. Heat load control in steam boilers. 2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM). IEEE, 2017. P. 1–4. DOI: 10.1109/ICIEAM.2017.8076177.
7. Zagoskina E., Barbasova T., Shnaider D. Intelligent Control System of Blast-furnace Melting Efficiency. 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON). IEEE, 2019. P. 0710–0713. DOI: 10.1109/SIBIRCON48586.2019.8958221.
8. Fleming J., Yan X., Allison C., *et al.* Real-time predictive eco-driving assistance considering road geometry and long-range radar measurements. IET Intelligent Transport Systems. 2021. Vol. 15, no. 4. P. 573–583. DOI: 10.1049/itr2.12047.
9. Scholl M., Minnerup K., Reiter C., *et al.* optimization of a Thermal Management System for Battery Electric Vehicles. 2019 Fourteenth International Conference on Ecological Vehicles and Renewable Energies (EVER). IEEE, 2019. P. 1–10. DOI: 10.1109/EVER.2019.8813657.
10. Meisel S. Dynamic Vehicle Routing. Anticipatory Optimization for Dynamic Decision Making. Vol. 51. New York, NY: Springer, 2011. P. 77–96. Operations Research/Computer Science Interfaces Series. DOI: 10.1007/978-1-4614-0505-4_6.
11. Cheng A.M.K. Real-Time Scheduling and Schedulability Analysis. Real-Time Systems. Wiley, 2002. P. 41–85. DOI: 10.1002/0471224626.ch3.

12. Kopetz H. Real-Time Scheduling. Real-Time Systems. Boston, MA: Springer, 2011. P. 239–258. Real-Time Systems Series. DOI: 10.1007/978-1-4419-8237-7_10.

13. Dantzig G.B. Linear programming and extensions. Princeton university press, 1998. 656 p.

14. Hall J.A.J., McKinnon K.I.M. Hyper-Sparsity in the Revised Simplex Method and How to Exploit it. Computational Optimization and Applications. 2005. Vol. 32, no. 3. P. 259–283. DOI: 10.1007/s10589-005-4802-0.

15. Klee V., Minty G.J. How good is the simplex algorithm?. Inequalities — III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969 / ed. by O. Shisha. Academic Press, 1972. P. 159–175.

16. Jeroslow R. The simplex algorithm with the pivot rule of maximizing criterion improvement. Discrete Mathematics. 1973. Vol. 4, no. 4. P. 367–377. DOI: 10.1016/0012-365X(73)90171-4.

17. Zadeh N. A bad network problem for the simplex method and other minimum cost flow algorithms. Mathematical Programming. 1973. Vol. 5, no. 1. P. 255–266. DOI: 10.1007/BF01580132.

18. Bartels R.H., Stoer J., Zenger C. A Realization of the Simplex Method Based on Triangular Decompositions. Handbook for Automatic Computation. Berlin, Heidelberg: Springer Berlin Heidelberg, 1971. P. 152–190. DOI: 10.1007/978-3-642-86940-2_11.

19. Tolla P. A Survey of Some Linear Programming Methods. Concepts of Combinatorial Optimization / ed. by V.T. Paschos. 2nd ed. Hoboken, NJ, USA: Wiley, 2014. P. 157–188. DOI: 10.1002/9781119005216.ch7.

20. Hall J.A.J. Towards a practical parallelisation of the simplex method. Computational Management Science. 2010. Vol. 7, no. 2. P. 139–170. DOI: 10.1007/s10287-008-0080-5.

21. Mamalis B., Pantziou G. Advances in the Parallelization of the Simplex Method. Algorithms, Probability, Networks, and Games. Vol. 9295 / ed. by C. Zaroliagis, G. Pantziou, S. Kontogiannis. Springer, 2015. P. 281–307. Lecture Notes in Computer Science. DOI: 10.1007/978-3-319-24024-4_17.

22. Lubin M., Hall J.A.J., Petra C.G., Anitescu M. Parallel distributed-memory simplex for large-scale stochastic LP problems. Computational Optimization and Applications. 2013. Vol. 55, no. 3. P. 571–596. DOI: 10.1007/s10589-013-9542-y.

23. Khachiyan L. Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics. 1980. Vol. 20, no. 1. P. 53–72. DOI: 10.1016/0041-5553(80)90061-0.

24. Shor N.Z. Cut-off method with space extension in convex programming problems. Cybernetics. 1977. Vol. 13, no. 1. P. 94–96. DOI: 10.1007/BF01071394.

25. Yudin D., Nemirovsky A. Information complexity and efficient methods for solving convex extremal problems. Economics and mathematical methods (Ekonomika i matematicheskie metody). 1976. No. 2. P. 357–369. (in Russian).

26. Karmarkar N. A new polynomial-time algorithm for linear programming. Combinatorica. 1984. Vol. 4, no. 4. P. 373–395. DOI: 10.1007/BF02579150.

27. Gondzio J. Interior point methods 25 years later. European Journal of Operational Research. 2012. Vol. 218, no. 3. P. 587–601. DOI: 10.1016/j.ejor.2011.09.017.

28. Fathi-Hafshejani S., Mansouri H., Peyghami M.R., Chen S. Primal–dual interior-point method for linear optimization based on a kernel function with trigonometric growth term. Optimization. 2018. Vol. 67, no. 10. P. 1605–1630. DOI: 10.1080/02331934.2018.1482297.

29. Asadi S., Mansouri H. A Mehrotra type predictor-corrector interior-point algorithm for linear programming. Numerical Algebra, Control & Optimization. 2019. Vol. 9, no. 2. P. 147–156. DOI: 10.3934/naco.2019011.

30. Yuan Y. Implementation tricks of interior-point methods for large-scale linear programs. International Conference on Graphic and Image Processing (ICGIP 2011). Vol. 8285. International Society for Optics, Photonics, 2011. P. 828502. DOI: 10.1117/12.913019.

31. Kheirfam B., Haghighi M. A full-Newton step infeasible interior-point method for linear optimization based on a trigonometric kernel function. Optimization. 2016. Vol. 65, no. 4. P. 841–857. DOI: 10.1080/02331934.2015.1080255.

32. Xu Y., Zhang L., Zhang J. A full-modified-Newton step infeasible interior-point algorithm for linear optimization. Journal of Industrial and Management Optimization. 2015. Vol. 12, no. 1. P. 103–116. DOI: 10.3934/jimo.2016.12.103.

33. Roos C., Terlaky T., Vial J.-P. Interior Point Methods for Linear Optimization. Springer-Verlag, 2005. 500 p. DOI: 10.1007/b100325.

34. Olkhovsky N.A., Sokolinsky L.B. A new method of linear programming. Numerical Methods and Programming (Vychislitel'nye Metody i Programmirovanie). 2023. Vol. 24, no. 4. P. 408–429. (in Russian) DOI: 10.26089/NUMMET.V24R428.

35. Sokolinskaya I.M., Sokolinsky L.B. On an iterative method for solving linear programming problems on cluster computing systems. Numerical Methods and Programming (Vychislitel'nye Metody i Programmirovanie). 2020. Vol. 21, no. 3. P. 329–340. DOI: 10.26089/NumMet.v21r328.

36. Olkhovsky N.A., Sokolinsky L.B. Visualizing Multidimensional Linear Programming Problems. Parallel Computational Technologies. Vol. 1618 / ed. by L.B. Sokolinsky, M.L. Zymbler. Cham: Springer, 2022. P. 172–196. Communications in Computer and Information Science. DOI: 10.1007/978-3-031-11623-0_13.

37. Sokolinsky L.B., Sokolinskaya I.M. Apex Method: A New Scalable Iterative Method for Linear Programming. Mathematics. 2023. Vol. 11, no. 7. P. 1654. DOI: 10.3390/math11071654.

38. Biewald L. Experiment Tracking with Weights & Biases. Software available from wandb.com. 2020. January.

39. Goodfellow I., Bengio Y., Courville A. Deep Learning (Adaptive Computation and Machine Learning). Vol. 8. 2016.

40. Bilenko R.V., Dolganina N.Y., Ivanova E.V., Rekachinsky A.I. High-performance Computing Resources of South Ural State University. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2022. Vol. 11, no. 1. P. 15–30. DOI: 10.14529/cmse220102.