УДК 004.032.26
М.М. Поліщук, С.В. Гринюк, С.В. Дацюк
Луцький національний технічний університет

ПОРІВНЯННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ДЛЯ НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

М.М. Поліщук, С.В. Гринюк, С.В. Дацюк. Порівняння методів оптимізації для навчання нейронних мереж. Сучасні методи навчання нейронних мереж, полягають в знаходженні мінімуму деякої неперервної функції помилки. За останні роки були запропоновані різні алгоритми оптимізації, які використовують різні підходи для оновлення параметрів ваг моделі. Дана стаття описує найбільш поширені методи оптимізації, що застосовуються в задачах навчання нейронних мереж, також наведений порівняльний аналіз цих методів на прикладі навчання простої згорткової нейромережі на наборі даних MNIST. В процесі аналізу були розглянуті різні реалізації методу градієнтного спуску, імпульсні методи, адаптивні методи, узагальнені проблеми їх використання, а також виявлені основні переваги кожного з методів
Ключові слова: методи оптимізації, нейронні мережі, метод градієнтного спуску, стохастичний градієнт, tensorflow, машинне навчання, згорткові нейронні мережі

Н.Н. Полищук, С. Гринюк, С.В. Дацюк. Сравнение методов оптимизации для обучения нейронных сетей. Современные методы обучения нейронных сетей, заключаются в нахождении минимума некоторой непрерывной функции ошибки. За последнее года были предложены различные алгоритмы оптимизации, которые используют различные подходы для обновления параметров весов модели. Данная статья описывает наиболее распространенные методы оптимизации, применяемые в задачах обучения нейронных сетей, также приведен сравнительный анализ этих методов на примере обучения простой сверхточной нейросети на наборе данных MNIST. В процессе анализа были рассмотрены различные реализации метода градиентного спуска, импульсные методы, адаптивные методы, обобщенные проблемы их использования, а также выявлены основные преимущества каждого из методов.
Ключевые слова: методы оптимизации, нейронные сети, метод градиентного спуска, стохастический градиент, tensorflow, машинное обучение, сверточные нейронные сети.

M.M. Polishchuk S.V. Hrinyuk, S.V Datsyuk Comparison of optimization methods for neural networks training. Modern methods of training neural networks consist in finding the minimum of some continuous error function. Over the past years, various optimization algorithms have been proposed that use different approaches to update the parameters of the model weights. This article describes the most common optimization methods used in neural networks training process, also provides a comparative analysis of these methods on the example of learning simple convolutional neural network on the MNIST data set. Analysed various implementations of the gradient descent method, impulse methods, adaptive methods, generalized problems of their use.
Keywords: optimization methods, neural networks, gradient descent method, stochastic gradient, tensorflow, machine learning, convolutional neural networks.

Постановка проблеми. Сучасні нейромережеві методи відносяться до числа найбільш затребуваних алгоритмів машинного навчання, які безперервно розвиваються і застосовуються в різних сферах практичної діяльності. У зв'язку з широкою областю їх застосування формуються різноманітні напрямки, що відрізняються постановкою задачі і типами вхідних даних: розпізнавання зображень, синтаксичний аналіз текстів, діагностика захворювань та ін. Постійним вдосконалення існуючих нейромережевих алгоритмів відрізняються своїми властивостями і особливостями реалізації, часто виникає проблема визначення найбільш ефективного методу мінімізації функції помилки, що гарантує кращі результати при вирішенні конкретної задачі.

Аналіз досліджень. Основи неперервного зворотного поширення було виведено в контексті теорії керування Келл 1960 р. [1] та Брайсоном 1961 р. [2] з використанням принципів динамічного програмування. Дрейфус у 1962 році опублікував простіше виведення, яке основане лише на ланцюговому правилі [3]. Брайсон описав його, як метод багатоетапної оптимізації динамічних систем у 1969 році, але вже в 1970 р. А.Д. Ліннаінмаа остаточно опублікував загальний метод автоматичного диференціювання (АД) дискретних зв'язних мереж вкладених диференційованих функцій [6]. Він відповідає сучасному баченню зворотного поширення, яке є ефективним навіть коли мережі є розрідженими. Вже у 1973 р. Дрейфус застосував зворотне поширення для пристосування параметрів контролерів пропорційно градієнтам похибок [4]. Вербос у 1974 р. зазначив можливість застосування цього принципу до ШНМ та у 1982 р. застосував метод А.Д. Ліннаінмаа до нейронних мереж способом, який широко застосовується і сьогодні [5]. У 1986 році Румельхарт, Хінтон та Вільямс зазначили, що цей метод може породжувати корисні внутрішні представлення вхідних даних в прихованих шарах нейронних мереж, а у 1993 році він став першим переможцем міжнародного змагання з розпізнавання образів за допомогою зворотного поширення.

Метою даного дослідження є аналіз основних особливостей стратегій оптимізації для подальшого обґрунтованого їх вибору, узагальнення експериментальних результатів для вирішення конкретних завдань машинного навчання.

Виклад основного матеріалу та аналіз результатів експерименту. Сучасні пакети для машинного навчання використовують різні варіації класичного методу градієнтного спуску, що володіють більш високою продуктивністю і точністю в реальних практичних завданнях за рахунок реалізованих в них механізмів розв'язання перерахованих вище проблем. Однак найчастіше користувач уже використовує вбудовані оптимізаційні алгоритми не володіючи достатньою інформацією про особливості поведінки доступних для використання методів на даному наборі даних. Наприклад, класифікатор MLPClassifier з популярної бібліотеки Scikit-learn машинного навчання на мові Python надає користувачеві вибір з декількох методів:

- SGD (стохастичний градієнт);
- Adam (метод адаптивної оцінки моментів);
- L-BFGS (квазіньютоновскій алгоритм Бройде - Флетчера - Гольдфарба - Шанно з обмеженим використанням пам'яті).

Найпопулярніший на сьогодні фреймворк для машинного навчання tensorflow, що містить реалізації алгоритмів SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Momentum, Nesterov momentum, RMSProp. Причому користувач повинен не тільки вибрати алгоритм оптимізації, а й відрегулювати значення настроювальних параметрів алгоритму, що важко зробити без розуміння особливостей цих методів.

Найбільш використовуваним методом навчання нейронних мереж є алгоритм зворотного поширення помилки (Backpropagation algorithm) в якому мінімізація цільової функції проводиться методом пакетного градієнтного спуску (Batch gradient descent), тобто на кожній ітерації навчання моделі зміна ваг відбувається за формулою:

$$\omega_{N+1} = \omega_N - \alpha \nabla_\omega E(\omega) \quad (1)$$

де E – цільова функція помилки, що залежить від параметрів;
w – ваги коефіцієнтів нейронної мережі;
α – швидкість навчання.

В даному методі ваги нейронної мережі оновлюються в напрямку, протилежному напрямку градієнта цільової функції з кроком, який обумовлений швидкістю навчання. В якості переваг методу пакетного градієнтного спуску можна виділити простоту реалізації і той факт, що метод гарантовано збігається до глобального або локального мінімуму для опуклих і неопуклих функцій відповідно. Однак існує безліч недоліків даного методу, внаслідок чого їх рідко застосовується в реальній практиці:

- метод градієнтного спуску може бути дуже повільним на великих наборах даних, тому що на кожній ітерації обчислюється градієнт для всіх векторів навчального набору;
- не дозволяє оновлювати модель «на льоту » і додавати в процесі навчання нові приклади навчальної вибірки через те, що оновлення ваг цільової функції проводиться відразу для всього вихідного набору даних;
- для неопуклих функцій існує проблема потрапляння в локальні мінімуми, тому що метод гарантує знаходження розв'язку лише для опуклих цільових функцій помилки;
- вибір оптимальної швидкості навчання може виявитися складною проблемою. Занадто маленька швидкість навчання може привести до дуже повільної збіжності і навпаки, велика швидкість навчання може перешкоджати збіжності, і як наслідок функція помилок буде коливатися навколо мінімуму і не досягне його;
- рівномірне оновлення всіх параметрів з однаковою швидкістю навчання призводить до погіршення якості навчання в разі, якщо вихідний набір даних не є збалансованим, якщо в вибірці існують класи, представлені меншим числом об'єктів.

Стохастичний градієнтний спуск (SGD - Stochastic gradient descent) на відміну від попереднього методу виконує оновлення параметрів для кожного навчального прикладу x(i) та мітки y(i):

$$\theta = \theta - \eta * \nabla\theta_j(\theta; x_i; y_i) \quad (2)$$

Пакетний градієнтний спуск виконує надлишкові обчислення для великих наборів даних, оскільки перераховує градієнти для аналогічних прикладів перед кожним оновленням параметрів. SGD припиняє цю надмірність, виконуючи одночасно одне оновлення, що зазвичай набагато швидше та який можна використовувати для навчання в Інтернеті.

SGD виконує часті оновлення з високою дисперсією, що призводить до значних коливань цільової функції, як показано на Рис. 1.

Хоча пакетний градієнтний спуск збігається до мінімуму басейну, в які поміщаються параметри, коливання SGD, з одного боку, дає йому можливість перейти до нових і потенційно кращих локальних мінімумів. З іншого боку, це в кінцевому рахунку ускладнює зближення до точного мінімуму,. Проте, якщо коли ми повільно зменшуємо швидкість навчання, SGD демонструє таку ж поведінку збіжності, як пакетний градієнтий спуск, майже напевно зближуючись з локальним або глобальним мінімумом для невипуклої та опуклої функції

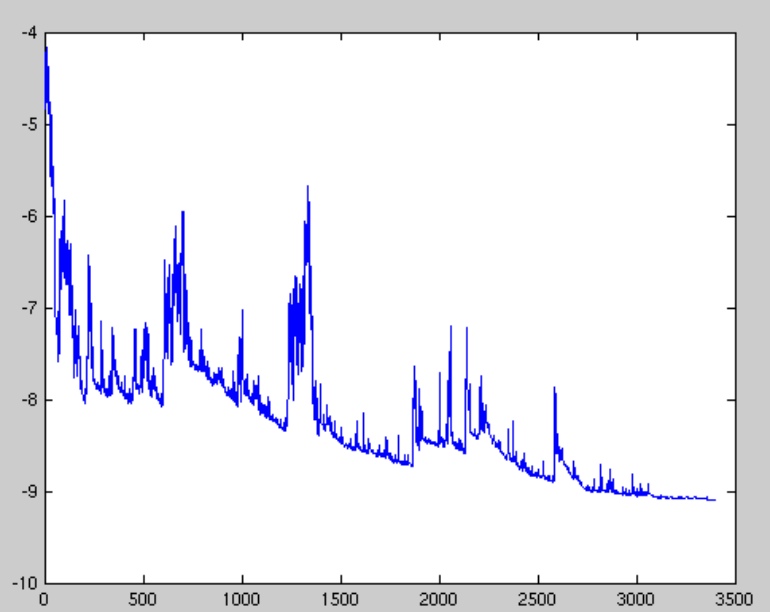


Рис. 1. Коливання цільової функції в методі SGD

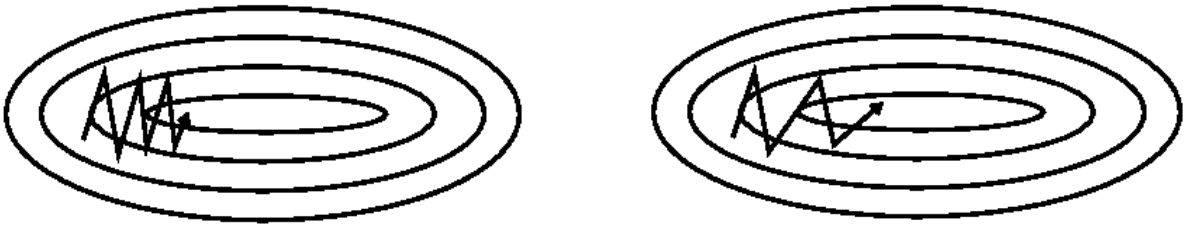
Міні-пакетний градієнтний спуск наслідує найкраще з двох вищеописаних методів і виконує оновлення для кожного міні-пакету n прикладів навчання:

$$\theta = \theta - \eta * \nabla\theta_j(\theta; x_{i:i+n}; y_{i:i+n}) \tag{3}$$

Таким чином, він зменшує дисперсію оновлень параметрів, що може призвести до більш стабільної конвергенції; він може використовувати високооптимізовані матриці, реалізації яких є майже у всіх сучасних бібліотеках глибокого навчання, які роблять міні-пакетні обчислення градієнта дуже ефективними. Загальні розміри міні-пакетів варіюються від 50 до 256, але можуть бути різними для різних застосувань.

Алгоритми імпульсної оптимізації градієнтного спуску.

Momentum. SGD має проблеми з навігацією по ярах, тобто на ділянках, де поверхня кривих набагато крутіша в одному вимірі, ніж в іншому, які є спільними навколо місцевих оптимумів. У цих сценаріях SGD коливається по схилах мінімуму (рис. 2 а).



а) б)

Рис. 2. SGD (стохастичний градієнт): а – без Momentum; б – із застосуванням Momentum

Momentum – це метод, який допомагає прискорити SGD у відповідному напрямку і гасить коливання (рис. 2 б.) Це досягається за рахунок додаванням дробу γ вектора оновлення минулого кроку часу до поточного вектора оновлення (4), в результаті отримуємо вираз для оновлення параметрів ваг (5):

$$vt = \gamma v_{t-1} + \eta \nabla J(\theta) \quad (4)$$

$$\theta = \theta - vt \quad (5)$$

Adagrad – це алгоритм для оптимізації на основі градієнта, який виконує адаптацію швидкості навчання до параметрів, виконуючи менші оновлення (тобто низькі швидкості навчання) для параметрів, пов'язаних з ознаками, які часто зустрічаються і більші оновлення (тобто високі швидкості навчання) для параметрів, пов'язаних з рідкісними ознаками. З цієї причини цей оптимізатор добре підходить для роботи з розрідженими даними. Dean et al. [10] виявили, що Адаград значно поліпшив надійність SGD і використовував його для навчання великих нейронних мереж в Google, які серед іншого навчилися розпізнавати кішок у відео Youtube. Pennington et al. використовували Адаград для підготовки вбудованих слів GloVe, оскільки рідкісні слова вимагають набагато більших оновлень, ніж часто вживані [6].

Раніше у всіх методах виконувалось оновлення для всіх параметрів θ відразу після кожної ітерації параметра θ(i) з однаковою швидкістю навчання η. Однак Adagrad використовує різну швидкість навчання для кожного параметра θ(i) на кожному кроці часу t, ми спочатку оновлюємо параметр Адаграда, який ми потім векторизуємо. Для стислості ми використовуємо g(t) для позначення градієнта на етапі часу t. g(t,i) є частковою похідною цільової функції до параметра θ на кроці часу t:

$$g_{i,j} = \nabla_\theta J(\theta_{t,i}) \quad (6)$$

Оновлення SGD для кожного параметра θ на кожному кроці часу t:

$$\theta_{t+1,i} = \theta_{t,i} - \eta * g_{t,i} \quad (7)$$

У своєму правилі оновлення Adagrad змінює загальну швидкість навчання η на кожному кроці часу t для кожного параметра θ(i) на основі минулих градієнтів, обчислених для θ(i):

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} + g_{t,i} \quad (8)$$

де, Gt∈Rd×d – діагональна матриця, де кожен діагональний елемент;
i, i – сума квадратів градієнтів θ до часу t;
ε – термін згладжування, що дозволяє уникнути поділу на нуль.

Оскільки G(t) містить суму квадратів минулих градієнтів до всіх параметрів θ вздовж своєї діагоналі ми можемо векторизувати нашу реалізацію, виконуючи матрично-векторний добуток ⊙ між ними G(t) і g(t):

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} \odot g_{t,i} \quad (9)$$

Однією з головних переваг Adagard є те, що вона виключає необхідність ручного налаштування швидкості навчання. Більшість реалізацій використовують значення за замовчуванням 0,01 і залишають його на цьому. Основним недоліком Adagard є його накопичення у знаменнику квадратичних градієнтів: оскільки кожен доданий термін є позитивним, накопичена сума продовжує зростати під час навчання. Це, в свою чергу, призводить до того, що швидкість навчання скорочується і в кінцевому підсумку стає нескінченно малою, і тоді алгоритм більше не може отримати додаткові знання.

RMSProp – модифікований метод AdaGrad, в якому накопичення градієнта замінене на експоненціально зважене ковзне середнє, тобто ігнорує значення після певного числа ітеріцій (10), (11), (12):

$$g = \frac{1}{m}\nabla_\theta \sum L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right) \tag{10}$$

$$s = decay_{rate} * s(1 - decay_{rate}) g^T g \tag{11}$$

$$\theta = \theta - \frac{\epsilon_k * g}{\sqrt{s + eps}} \tag{12}$$

AdaDelta – розширений метод AdaGrad, який дозволяє усунути проблеми розпаду швидкості навчання. Замість того, щоб накопичувати всі попередні градієнти, Adadelta обмежує простір накопичених градієнтів деякими фіксованими розмірами w. Замість неефективного зберігання попередніх градієнтів у квадраті, сума градієнтів рекурсивно визначається як середнє затухання всіх минулих квадратних градієнтів. Середній показник E [g²] (t) на етапі часу t потім залежить тільки від поточного та попереднього середнього значення градієнта:

$$\Delta\theta_t = -\eta * g_{t,i} \tag{13}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{14}$$

Тоді правило зміни коефіцієнтів має наступний вигляд:

$$W^{N+1} = W^N - \frac{\eta}{\sqrt{E[g^2]_N + \varepsilon}} \tag{15}$$

Особливостями даного методу є:

1.не потрібно вказувати початкову швидкість навчання.

2.для кожного параметра обчислюються різні курси навчання.

3.запобігання затримці (розпаду) курсів навчання.

Оскільки ми обчислюємо індивідуальні швидкості навчання для кожного параметра, тому буде доцільно обчислити окремі зміни імпульсу для кожного параметра і зберегти їх окремо. Саме на основі такого твердження було створено новий модифікований метод оптимізації – *Adam(Adaptive Moment Estimation).*

Adam базується на основі адаптивної оцінку моменту. Адаптивна оцінка моменту (Adam) – новий метод, який обчислює адаптивні швидкості навчання для кожного параметра. На додаток до зберігання експоненціально затухаючого середнього числа минулих квадратних градієнтів, таких як AdaDelta, Adam також зберігає експоненціально затухаюче середнє значення минулих градієнтів M(t), подібне до Momentum. Правило поновлення ваг для Adam визначається на основі використання оцінок двох різних моментів (формули (13) і (14)), у першому з яких використовуються обчислені раніше значення часткових похідних (як в методі Momentum), а в другому їх квадрати. Метод Adam вважається досить стійким до вибору значень гіперпараметров 1 β, 2 β, і тому часто пропонується в якості методу оптимізації:

$$m_N = \beta_1 m_{N-1} + (1-\beta_1) g_n \quad (16)$$

$$v_N = \beta_2 v_{N-1} + 91 - \beta_2 g^2{}_N \quad (17)$$

Перерахунок ваг відбувається за формулою (15):

$$w^{N+1} = w^N - \frac{\eta}{\sqrt{\hat{v_n}+\varepsilon}} \quad (18)$$

Оцінка продуктивності розглянутих алгоритмів.

Оптимальне рішення (набір вагових коефіцієнтів), отримане в ході навчання нейронної мережі, безпосередньо впливає на її здатність до узагальнення, тобто на точність та якість, з яким мережа видає вірні відповіді на наборі прикладів тестової множини, на яких не проводилось навчання. Однак в даний час точні висновки про здатність нейронних мереж до узагальнення знаходяться на стадії розробки та вивчення. Також багато сучасних досліджень доводять, що кількість локальних мінімумів функції помилки навчання зростає з числом незалежних параметрів, що в свою чергу, негативно позначається на здатності мережі до узагальнення. Для багатьох моделей з великим числом параметрів, адаптивні методи можуть сходитися до рішень, які кардинально відрізняються від рішень, отриманих в результаті роботи SGD. В даний час адаптивні алгоритми часто є методами за замовчуванням в багатьох модулях для машинного навчання (наприклад, Scikitlearn, Keras). Однак існують дослідження [8], які підтверджують низьку продуктивність даних методів (Adam, RMSProp, Adagrad) на наборах даних з великим кількістю параметрів , в порівнянні з методом стохастичного градієнтного спуску. Даний недолік пояснюється тим, що алгоритми з адаптивною швидкістю навчання більш схильні до перенавчання, внаслідок чого їх похибка на тестовій вибірці може істотно підвищуватися.

Порівняння практичних результатів оптимізації з використанням різних алгоритмів оптимізації.

Розглянемо результати навчання нейромережі для класифікації зображень рукописних цифр з набору даних MNIST з використанням алгоритмів оптимізації Momentum, Adam, Adadelta, Adagrad, SGD, реалізованих в бібліотеці Tensorflow. Набір MNIST включає 60000 навчальних і 10000 тестових прикладів і часто використовується для тестування і налагодження різних нейромережевих алгоритмів.

Для даного експерименту було створено просту згорткову нейронну мережу, яка складається з трьох згорткових блоків з активацією ReLU. В якості функції втрат було використано cross entropy loss (крос ентропія). На рис. 3 показано значення функції помилки для описаної вище мережі.

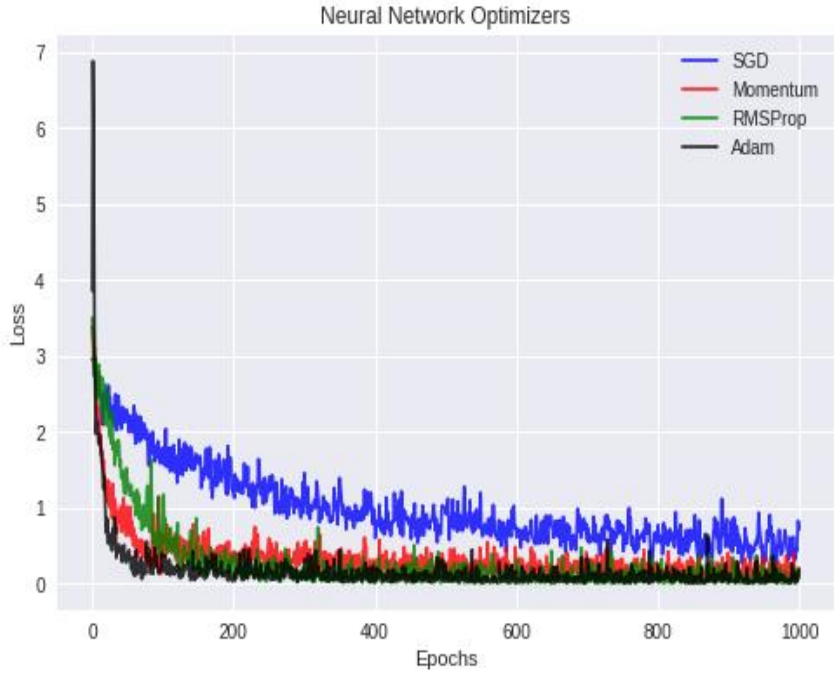


Рис.3. Зміна фунції помилки в залежності від методу оптимізації

У табл. 1 наведені значення точності класифікації мережею, навченою з використанням розглянутих методів, на тестовій вибірці датасету MNIST.

Таблиця 1

SGD	Momentum	RMSProp	Adam
90.84%	96.62%	97.93%	98.4%

Висновки. В ході дослідження були розглянуті основні методи оптимізації, що застосовуються в сучасних методах навчання нейронних мереж. У процесі дослідження було проведено аналіз властивостей і особливостей розглянутих методів, а також сформульовані умови і обґрунтування їх найбільш оптимального з точки зору продуктивності і точності на навчальній і тестовій вибірках. Дані аналізу супроводжуються практичними результатами, які підтверджують сформульовані рекомендації по використанню розглянутих методів класичного і стохастичного градієнтного спуску, імпульсних, адаптивних алгоритмів оптимізації в задачах навчання нейронних мереж.

Результати експерименту показали, що метод RMSProp і Adam показали найкращі результати на даному наборі даних. Momentum та SGD є стабільно ефективними, проте поступаються в точності і продуктивності іншим методам. Точність результатів, досягнутих в ході роботи SGD і методу моментів пояснюється тим, що вихідна вибірка є збалансованою та позитивно позначається на продуктивності цих методів. Таким чином, виходячи з даних спостережень, можна зробити висновок , що результати проведених теоретичних досліджень підтверджуються даними емпіричних розрахунків і можуть бути використані в якості рекомендацій до застосування розглянутих оптимізаційних методах в завданнях машинного навчання.

Список використаних джерел:

1. Kelley, Henry J. (1960). Gradient theory of optimal flight paths. Ars Journal 30(10): 947–954. doi:10.2514/8.5282. (англ.)

2. Arthur E. Bryson [en] (1961, April). A gradient method for optimizing multi-stage allocation processes. In Proceedings of the Harvard Univ. Symposium on digital computers and their applications. (англ.)

3. Dreyfus, Stuart (1962). The numerical solution of variational problems. Journal of Mathematical Analysis and Applications 5 (1): 30–45. doi:10.1016/0022-247x(62)90004-5. (англ.)

4. Dreyfus, Stuart (1973). The computational solution of optimal control problems with time lag. IEEE Transactions on Automatic Control 18 (4): 383–385. doi:10.1109/tac.1973.1100330. (англ.)

5. Schmidhuber, Jürgen (2015). Deep Learning. Scholarpedia 10 (11): 32832. Bibcode:2015SchpJ..1032832S. doi:10.4249/scholarpedia.32832.(англ.)

6. Ruder, S. An overview of gradient descent optimization algorithms / S. Ruder // Cornell University Library. – 2016. – URL: https://arxiv. org/abs/1609.04747

7. Jordan, J. Intro to optimization in deep learning: Gradient Descent/ J. Jordan // Paperspace. Series: Optimization. – 2018. – URL: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

8. Seppo Linnainmaa[en] (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki, 6-7. (англ.)

9. Anish Singh Walia Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent – URL: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

10. Fletcher, R. Practical methods of optimization / R. Fletcher. – Wiley, 2000. – 450 p.