Програма використовує метод Сімпсона для чисельного інтегрування, апроксимуючи функцію параболами на підінтервалах і сумуючи ці площі для обчислення інтегралу. Він визначає функцію $f(x)$, запитує у користувача межі інтеграції та кількість підінтервалів, обчислює значення функції у вузлових точках і використовує формулу методу Сімпсона для отримання результату.

Підсумовуючи це дослідження, зазначимо, що було проведено порівняльний аналіз методу Гаусса та методу Сімпсона, які застосовуються для розв'язання систем лінійних рівнянь та чисельного інтегрування відповідно. Метод Гауса показав високу універсальність та стійкість до помилок округлення, але може виявитися менш ефективним у разі великих систем або наявності особливостей у матриці. Метод Сімпсона натомість забезпечує високу точність інтегрування та простоту реалізації, але може бути обчислювально складним для складних функцій або великих інтервалів. Практичні приклади, реалізовані на Python, демонструють специфічні переваги та недоліки кожного методу, що підкреслює важливість вибору відповідного методу залежно від конкретних умов задачі.

Список використаних джерел

1. Верещак Р. Метод Гауса для розв'язання систем лінійних рівнянь: повний огляд. *Методи розв'язування систем лінійних алгебраїчних рівнянь*. 29.11.2023. URL: https://www.mathros.net.ua/metod-gaussa-rozvjazok-systemy-linijnyh-rivnjan-metodom-gaussa.html (дата звернення: 17.05.2024).
2. Метод Сімпсона: Основи та практичне застосування. *mathros.net.ua*. Mar. 3, 2024. URL: https://medium.com/@MathrosNetUa/метод-сімпсона-основи-та-практичне-застосування-e5803a0e9cc1 (дата звернення: 17.05.2024).
3. Gauss elimination method python program. *Codesansar*. URL: https://www.codesansar.com/numerical-methods/gauss-elimination-method-python-program.htm (дата звернення: 17.05.2024).
4. Simpson's Rule. *Mathematical Python*. URL: https://patrickwalls.github.io/mathematical-python/integration/simpsons-rule/ (дата звернення: 17.05.2024).

УДК 004.6

Кизь О. А., *здобувач 2 курсу спеціальності 122 Комп'ютерні науки,*
науковий керівник:
Якубич К. О., *асистент кафедри інформаційних технологій*

МЕТОД ГРАДІЄНТНОГО СПУСКУ

Донецький національний університет імені Василя Стуса, м. Вінниця

Методи оптимізації широко використовуються в обчислювальних задачах для пошуку найкращих рішень чи найоптимальніших параметрів. Вони допомагають знаходити мінімуми або максимуми функцій, розв'язувати задачі лінійного та нелінійного програмування, задачі розподілу ресурсів та багато інших. Для вирішення різних завдань у дослідженнях використовуються різні методи. Є алгоритми, які завершують свою роботу після певної кількості кроків, та інші, які пристосовуються до знаходження рішення на певних типах задач, і також є еврис-

тичні підходи, які можуть дати наближені розв'язки для деяких завдань, хоча не завжди точні. Ці методи мають різні алгоритми, один із яких – методи градієнтного спуску, який ми більш детально розглянемо.

Градієнтний спуск – це алгоритм оптимізації. Він використовується для покращення продуктивності нейронної мережі шляхом налаштування параметрів мережі так, щоб різниця між прогнозами мережі та фактичними / очікуваними значеннями мережі (що називаються втратою) була якомога меншою. Градієнтний спуск приймає початкові значення параметрів і використовує операції, засновані на численні, щоб налаштувати їх значення до значень, які зроблять мережу максимально точною [2]. Інакше кажучи, це один із методів оптимізації, який дає змогу нейронній мережі вчитися, а кінцева мета оптимізації – знайти такі значення параметрів, за яких функція помилки досягне свого мінімуму (рис. 1).

Розглянемо суть методу детальніше. Градієнт вказує напрям найшвидшого зменшення помилки. У контексті нейронних мереж він допомагає визначити, як зміна параметрів моделі впливає на функцію помилки. Метод використовує градієнт для пошуку шляху з найшвидшим зменшенням помилки, оновлюючи параметри моделі до досягнення мінімуму помилки. Етапами алгоритму градієнтного спуску є:

- Ініціалізація параметрів: початкові значення параметрів моделі обираються у випадковий спосіб або за певними правилами.
- Обчислення градієнта: обчислюється градієнт функції в поточній точці, вказуючи напрям найшвидшого зростання:

$$\nabla f(x_1^0, x_2^0, x_n^0) = \left(\frac{df}{dx_1}, \frac{df}{dx_2}, \frac{\partial f}{dx_n}\right),$$

де $\frac{df}{dx_i}$ – часткова похідна функції f за змінною x_i, обчислена у точці (x_1^0, x_2^0, x_n^0).

- Оновлення параметрів: параметри моделі оновлюються у напрямі, протилежному градієнту, за допомогою кроку (швидкості навчання), щоб зменшити значення функції в цій точці.
- Перевірка зупинки: проводиться перевірка критерію зупинки, який може бути досягненням максимальної кількості ітерацій, досягненням необхідної точності або іншими умовами зупинки.
- Повторення ітерацій: якщо критерій зупинки не виконано, процес повторюється, починаючи з обчислення градієнта в новій точці.

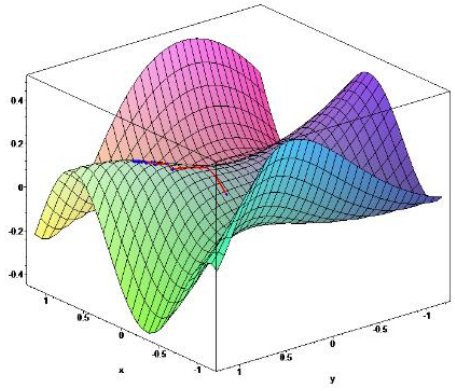

Рис. 1. Градієнтний спуск

Реалізація методу градієнтного спуску в MATLAB:

Згенеруємо випадкові дані для навчального набору:

```
X = randn(100, 2); % 100 прикладів з двома ознаками
y = randi([0, 1], 100, 1); % бінарна цільова змінна
```

Ініціалізуємо ваги нейронної мережі випадковими значеннями:

```
W = randn(2, 1);
```

Швидкість навчання:

```
learning_rate = 0.01;
```

Кількість ітерацій:

```
num_iterations = 1000;
```

Зберігаємо значення функції помилки на кожній ітерації:

```
error_history = zeros(num_iterations, 1);
```

Алгоритм градієнтного спуску для корекції ваг:

```
for i = 1:num_iterations
```

Обчислюємо прогнози моделі:

```
predictions = X * W;
```

Обчислюємо помилку (загальний квадрат помилки):

```
error = mean((predictions - y).^2);
error_history(i) = error;
```

Обчислюємо градієнт функції помилки:

```
gradient = 2 * mean(X .* repmat(predictions - y, 1, size(X, 2)), 1)';
```

Оновлюємо ваги:

```
W = W - learning_rate * gradient;
end
```

Виводимо остаточні ваги:

```
disp('Остаточні ваги:');
disp(W);
```

Виводимо графік зміни значення функції помилки на кожній ітерації:

```
plot(1:num_iterations, error_history, '-o');
xlabel('Ітерація');
ylabel('Значення функції помилки');
title('Зміна значення функції помилки');
```

Методи оптимізації в обчислювальних задачах є важливим інструментом для пошуку оптимальних рішень. Вони використовуються для знаходження мінімумів або максимумів функцій, розв'язання задач лінійного та нелінійного програмування, а також для інших завдань. Одним із методів оптимізації обчислювальних задач є метод градієнтного спуску, який дає змогу навчати нейронні мережі та знаходити оптимальні параметри. Він використовує градієнт функції помилки для оновлення параметрів моделі у напрямку найшвидшого зменшення помилки, що збільшує точність моделі та зменшує обчислювальні втрати.

Список використаних джерел

1. Градієнтний спуск: алгоритм та приклад на Python. *Robotdreams.* URL: https://robot-dreams.cc/uk/blog/331-gradiyentniy-spusk-algoritm-ta-priklad-na-python (дата звернення 28.04.2024).
2. Нельсон Д. Що таке градієнтний спуск? *UniteAI.* Серпень 23, 2020. URL: https://www.unite.ai/uk/what-is-gradient-descent/ (дата звернення 28.04.2024).
3. Лекції з дисципліни «Обчислювальна математика та програмування» для студентів денної та заочної форм навчання напряму підготовки 6.051301 «Хімічна технологія» / укл. Г. М. Дмитрієнко. Східноукраїнський національний університет імені Володимира Даля, 2010. 88 с.

УДК 004.9

Ілик В. В., *здобувач 2 курсу спеціальності 122 Комп'ютерні науки,*
науковий керівник:
Якубич К. О., *асистент кафедри інформаційних технологій*

ТЕОРІЇ АЛГОРИТМІВ У ЗАДАЧАХ ШТУЧНОГО ІНТЕЛЕКТУ

Донецький національний університет імені Василя Стуса, м. Вінниця

Вступ. Штучний інтелект (ШІ) є однією з найбільш динамічних галузей сучасної науки та техніки, яка швидко розвивається. В основі багатьох досягнень ШІ лежать теорії алгоритмів, які забезпечують ефективне розв'язання складних задач, аналіз великих обсягів даних та прийняття рішень. Розвиток алгоритмів для ШІ є критичним для прогресу у сферах обробки природної мови, розпізнавання образів, автоматизації та робототехніки.

Розробка ефективних алгоритмів для ШІ має велике значення для багатьох галузей економіки та суспільства. Зокрема, вони дають змогу автоматизувати рутинні процеси, підвищувати продуктивність, створювати нові можливості для аналізу даних та прогнозування. В умовах стрімкого зростання обсягів інформації, що потребує обробки, та складності задач, які потрібно розв'язувати, вдосконалення алгоритмів стає ключовим фактором успішного розвитку ШІ [1].

Аналіз останніх досліджень і публікацій. Останні дослідження в галузі теорії алгоритмів для ШІ зосереджені на таких напрямах:

1. Машинне навчання та глибоке навчання (алгоритми навчання на основі великих наборів даних, як-от нейронні мережі та методи підсилення).
2. Алгоритми оптимізації (методи оптимізації, включно з градієнтними методами, еволюційними алгоритмами та стохастичною оптимізацією, використовуються для покращення продуктивності моделей ШІ).
3. Алгоритми обробки природної мови (NLP) (розробка моделей для розуміння та генерації людської мови, зокрема трансформерів і моделей на основі BERT).
4. Алгоритми для автономних систем (включають плани дій і системи прийняття рішень для автономних транспортних засобів та роботів).

Метою цієї роботи є аналіз сучасних теорій алгоритмів, що використовуються в задачах штучного інтелекту, зокрема розгляд методів машинного навчан-