УДК 004:005:51

НЕЙРОННІ МЕРЕЖІ ТА АЛГОРИТМ ОПТИМІЗАЦІЇ ADAM

М. С. Малюта, Ю. В. Поремський

Анотація. У статті розглядаються штучні нейронні мережі та алгоритм оптимізації Adam. Пояснюється, що таке мережа – сукупність штучних нейронів, зв'язаних між собою, причому кожен нейрон отримує на вхід дані, перемножує їх на ваги, сумує з урахуванням біаса і пропускає через активаційну функцію. Описано процес навчання мережі як мінімізації функції втрат (loss) за допомогою методів оптимізації – ітеративного оновлення ваг у напрямку градієнта помилки. Наведено принцип роботи алгоритму Adam, що поєднує ідеї методу RMCпроп та імпульсу, використовуючи ковзні середні першого і другого моментів градієнтів. У практичній частині демонструється код простої нейронної мережі на Python (класифікація MNIST) з використанням оптимізатора Adam та аналізуються результати. Стаття завершується висновками про ефективність Adam у процесі навчанні нейромереж.

Ключові слова: нейронна мережа, штучний нейрон, градієнтний спуск, алгоритм Adam, оптимізація.

Вступ. Нейронні мережі – основа сучасного машинного навчання, що довели свою ефективність у багатьох задачах (комп'ютерному зорі, обробці мови тощо) [5], [1]. У контексті дисципліни «Алгоритми та структури даних» важливо розуміти основи їх роботи та методи навчання. Архітектура мережі задається шарами штучних нейронів, а функція, якою мережа апроксимує співвідношення «вхід – вихід», зберігається у вагових коефіцієнтах зв'язків між нейронами [2], [3]. Процес навчання зводиться до оптимізації – пошуку таких ваг, які мінімізують помилку моделі на навчальних даних. Нижче наведено пояснення структури нейрона і мережі, а також принципи оптимізації навчання, зокрема алгоритм Adam.

Основна частина. Штучна нейронна мережа – це система, натхненна мозком, що складається з безлічі взаємопов'язаних штучних нейронів [2]. Кожен нейрон отримує вхідні сигнали, перемножує їх на відповідні ваги (коефіцієнти зв'язків), може додавати зміщення і обчислює зважену суму. Потім ця сума проходить через активаційну функцію (наприклад, ReLU, сигмоїду), яка надає нейрону нелінійність [3]. Отже, кожен нейрон видає вихідний сигнал, який передається на входи нейронів наступного шару.

Архітектура мережі складається з послідовних шарів: вхідний шар приймає дані (розмір шару відповідає кількості ознак, наприклад, пікселів зображення), далі можуть бути один або декілька прихованих шарів, а в останньому, вихідному шарі, формується остаточний результат (ймовірності класів, значення регресії тощо). Приклад повністю з'єднаної (dense) мережі: кожен нейрон одного шару пов'язаний з усіма нейронами наступного шару [2], [3]. Інакше кажучи, вихід будь-якого шару використовується як вхід для наступного. На кожному кроці шар обчислює лінійне перетворення і застосовує нелінійну функцію активації, після чого передає результат далі [3].

Отже, обчислення у мережі проходить у прямому напрямі: дані потрапляють у вхідний шар, далі за формулами виду (1):

$$a_i = \phi\left(\sum_j \omega_{ij}\, a_j + b_i\right), \quad (1)$$

де a_j – вихідні сигнали попереднього шару;

w_{ij} – вага зв'язку;

b_i – біас;

ϕ – активаційна функція (наприклад, $\phi(x) = \max(0, x)$ для ReLU чи $\phi(x) = 1/(1 + e^{-x})$ для сигмоїди) [3], потім надходять у вихідний шар.

Самі ваги w_{ij} визначають те, які функції мережа може апроксимувати, і їх потрібно навчити за допомогою спеціальних алгоритмів оптимізації.

Оптимізація начальної моделі. Під час навчання мережі необхідно знайти такі значення ваг, за яких відхилення передбачень мережі від правильних відповідей мінімальне. Це відхилення формалізується через функцію втрат – наприклад, середньоквадратичну помилку (MSE) для регресії чи категорійну крос-ентропію для класифікації. Модель намагається мінімізувати функцію втрат, підбираючи ваги. Така задача є звичайною оптимізаційною [1], [3].

У найпростішому випадку застосовується градієнтний спуск: на кожній ітерації обчислюється градієнт функції втрат щодо кожної ваги і ваги оновлюються у напрямі протилежному градієнту. Для випадку стокастичного навчання алгоритм називається стохастичним градієнтним спуском (SGD). Просте оновлення ваг в SGD має такий вигляд (формула 2):

$$w \leftarrow w - \eta \nabla_w L, \tag{2}$$

де η – швидкість навчання (learning rate);

$\nabla_w L$ – градієнт втрат за вагами.

Традиційний SGD часто потребує ретельного підбору η, може «стрибати» за умови нерівномірних градієнтів або уповільнюватися у разі коливань помилок.

Для покращення збіжності вводять методи імпульсу та адаптивні алгоритми, які самі регулюють крок оновлення для кожної ваги. Метою оптимізації є швидке та стабільне досягнення мінімуму втрат для отримання моделі з високою точністю як на тренувальних, так і на невідомих даних [1], [3].

Алгоритм оптимізації Adam. Алгоритм Adam (Adaptive Moment Estimation) поєднує ідеї двох підходів – RMSProp та імпульсу [6]. У Adam підтримуються два «рухомі середніх» для кожної ваги: m_t – оцінка першого моменту (середнього значення градієнтів) та v_t – оцінка другого моменту (середнього квадрата градієнтів). На ітерації t ці значення оновлюються як ковзні середні (формула 3):

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla L, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla L)^2, \end{aligned} \tag{3}$$

де ∇L – поточний градієнт втрат по відповідній вазі,

$\beta_1, \beta_2 \in [0,1)$ – коефіцієнти затухання ($\beta_1 = 0.9$, $\beta_2 = 0.999$).

Через ініціалізацію нулями, відразу після початку з'являється зміщення оцінок, тому застосовують поправки, а саме: корекцію зміщення моментів (формула 4) та корекції першого та другого моментів (формула 5).

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{4}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{5}$$

Оновлення ваг відбувається за формулою:

$$w \leftarrow w - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}, \tag{6}$$

де ε – невелике число (наприклад, 10^{-8}), для уникнення ділення на нуль.

Завдяки таким оновленням, Adam використовує адаптивний крок навчання для кожної ваги: він зменшує крок за напрямом, де градієнт постійно великий (через v_t), і збільшує там, де градієнт малий. У підсумку Adam має швидшу збіжність та меншу необхідність тонкого підбору η, порівняно з простим SGD [3].

Також експерименти показують, що адаптивні оптимізатори (наприклад, AdamW – варіант Adam з урахуванням регуляризації) можуть давати кращу точність та стабільність на тесті, ніж звичайний SGD [4]. З іншого боку, деякі дослідження зазначають, що чистий SGD зі схиленою швидкістю навчання іноді узагальнюється краще, але загалом Adam є зручним і потужним методом для початкового навчання мереж через його адаптивність.

Розглянемо простий приклад реалізації нейронної мережі на Python із використанням бібліотеки PyTorch. Завдання – класифікація рукописних цифр із набору MNIST. Модель – послідовна (Sequential) з одним схованим шаром. Для оптимізації використаємо Adam (рис. 1).

```
      import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms

# 1) Підготовка даних MNIST
transform = transforms.ToTensor()
train_data = datasets.MNIST(root='data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='data', train=False, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=1000, shuffle=False)

# 2) Визначення простої мережі
class SimpleNet(nn.Module):
  def __init__(self):
    super(SimpleNet, self).__init__()
    self.flatten = nn.Flatten()
    self.fc1 = nn.Linear(28*28, 128) # прихований шар з 128 нейронами
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(128, 10)   # вихідний шар на 10 класів
  def forward(self, x):
    x = self.flatten(x)
    x = self.relu(self.fc1(x))
    x = self.fc2(x)
    return x

model = SimpleNet()
criterion = nn.CrossEntropyLoss() # функція втрат для класифікації
optimizer = optim.Adam(model.parameters(), lr=0.001) # використання Adam

# 3) Навчання моделі
for epoch in range(3):
  model.train()
  for data, target in train_loader:
    optimizer.zero_grad()
    output = model(data)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()

# 4) Оцінка точності на тесті
model.eval()
correct = 0
total = 0
with torch.no_grad():
  for data, target in test_loader:
    output = model(data)
    pred = output.argmax(dim=1)
    correct += (pred == target).sum().item()
    total += target.size(0)
print(f"Test Accuracy: {100*correct/total:.2f}%")
```

Рис. 1. Лістинг коду нейронної мережі

У цьому коді наведені кроки: завантаження даних, визначення моделі з одним прихованим шаром (вхідний шар має 784 нейрони, прихований – 128, вихід – 10), вибір функції втрат та оптимізатора Adam, навчання моделі протягом 3 епох, а потім перевірка точності на тестовому наборі.

Результати для 3 епох виглядають так – див. рис. 2. Він демонструє, що навіть дуже проста мережа може ефективно класифікувати MNIST за умови використання Adam. Переваги Adam у цьому прикладі виявляються у швидшій збіжності: за ті ж епохи модель досягає ви-

щої точності порівняно з SGD з фіксованим кроком, а також Adam не вимагає дуже ретельного налаштування learning rate (β-параметри вбудовані за замовчуванням) [3]. Однак важливо перевіряти модель на окремому тестовому наборі, щоб уникнути перенавчання.

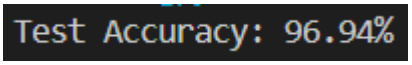

Рис. 2. Результат роботи нейронної мережі

Висновки. Нейронні мережі є універсальним підходом для побудови моделей, що апроксимують довільні функції. Вони складаються з послідовних шарів штучних нейронів: кожен нейрон обчислює зважену суму входів і пропускає її через нелінійну активацію [3]. Навчання мережі зводиться до оптимізації функції втрат за допомогою оновлення ваг. Алгоритм Adam є ефективним методом оптимізації: він адаптивно змінює швидкість навчання для кожної ваги на основі статистики градієнтів (першого та другого моментів) [3]. Завдяки цьому Adam часто забезпечує швидку збіжність і високу точність, зокрема у разі великих моделей і складних даних. Практична реалізація показала, що під час класифікації MNIST із використанням Adam прості мережі досягають високої точності (~98 %) відносно невеликою кількістю ітерацій. Отже, тема нейронних мереж і оптимізації (зокрема Adam) є актуальною для сучасних алгоритмів і наочним прикладом складних обчислювальних підходів в інформатиці.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Altinel M. A. Development of Deep Learning Optimizers: Approaches, Concepts, and Update Rules. *Researchgate.* 2025. URL: https://surl.li/zzzonp

2. De Miranda G. C., De Lima G. G., Farias T. An Introduction to Neural Networks for Physicists. *Researchgate.* 2025. URL: https://surl.li/tbosax

3. Cerra F., Callegari F., Querzoni L. A Short Introduction to Neural Networks and Their Application to Earth and Materials Science. *Researchgate.* 2024. URL: https://surl.li/zoglok

4. Selvakumari S., Durairaj M. A Comparative Study of Optimization Techniques in Deep Learning Using the MNIST Dataset. *Indian Journal of Science and Technology.* 2025. Vol. 18, № 10. P. 803–810. URL: https://indjst.org/articles/a-comparative-study-of-optimization-techniques-in-deep-learning-using-the-mnist-dataset

5. Large-Scale Deep Learning Optimizations: A Comprehensive Survey / X. He, F. Xue, X. Ren, Y. You. *Researchgate.* 2021. URL: https://surl.li/xhsbyd

6. Stochastic Gradient Descent. *Wikipedia.* URL: https://en.wikipedia.org/wiki/Stochastic_gradient_descent

УДК 517.9

ЗАСТОСУВАННЯ ТЕОРЕМ ПРО НЕРУХОМІ ТОЧКИ В ПРОБЛЕМАХ РОЗВ'ЯЗНОСТІ КРАЙОВИХ ЗАДАЧ ДЛЯ НЕЛІНІЙНИХ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ ДРУГОГО ПОРЯДКУ ЗІ ЗМІННИМИ КОЕФІЦІЄНТАМИ

А. І. Матвієва, Д. В. Шевченко, К. О. Бурячeнко

Анотація. У роботі продемонстровано застосування теореми Шаудера про нерухомі точки до доведення існування розв'язків нелінійних крайових задач для диференціальних рівнянь другого порядку зі змінними коефіцієнтами. Розглянуто приклади використання компактних операторів у функціональних просторах і показано вплив умов зростання нелінійності на існування розв'язку.

Ключові слова: нерухома точка; теорема Брауера; теорема Шаудера; нелінійні диференціальні рівняння; функціональний аналіз; крайова задача; компактний оператор.

Вступ. Теореми про нерухомі точки займають особливе місце в нелінійному аналізі, оскільки вони забезпечують один із найефективніших методів доведення існування розв'язків нелінійних рівнянь. Перші фундаментальні результати у цьому напрямі були отримані Л. Брауером на початку XX століття для неперервних скінченновимірних відображень. Подальший розвиток теорії відбувався завдяки працям Шаудера, який поширив ідеї Брауера на нескінченновимірні простори та сформулював теорему про нерухому точку для компактних операторів у нескінченновимірних банахових просторах. Ці результати стали ключовими для сучасного функціонального аналізу та дослідження нелінійних операторних рівнянь.