УДК 517.9 *http://dx.doi.org/10.30970/vam.2024.33.12029*

НАВЧАННЯ ГЛИБОКИХ МОДЕЛЕЙ: АЛГОРИТМИ ОПТИМІЗАЦІЇ ЯК КЛЮЧ ДО ЕФЕКТИВНОСТІ

М. Стопець, Ю. Щербина

Львівський національний університет імені Івана Франка,
вул. Університетська 1, Львів, 79000, Україна
e-mail: myroslava.stopets@gmail.com, yuriy.shcherbyna@lnu.edu.ua

Через надзвичайно велику кількість даних, які ми опрацьовуємо процес, побудови та навчання моделі є дуже затратним і потребує значних обчислювальних ресурсів. Саме тому постає проблема як можна оптимізувати цей процес, від якого значною мірою залежить результат глибокого навчання. Ефективні алгоритми оптимізації можуть значно поліпшити здатність моделі до навчання, допомагаючи їй вловлювати складні закономірності, видобувати змістовні репрезентації та робити точні прогнози.

В межах цієї праці було досліджено проблему оптимізації в навчанні глибоких моделей[1]. Описано переваги та недоліки різних підходів до процесу оптимізації навчання. На прикладі стандартного набору даних MNIST було перевірено ефективність таких стратегій оптимізації: алгоритми оптимізації з адитивною швидкістю навчання, наближені методи другого порядку, базові алгоритми оптимізації. Як результат перевірки проведено висновки щодо того, як різні оптимізатори впливають на зменшення похибки передбачень, як обирати алгоритм оптимізації залежно від задачі, яку потрібно вирішити.

Ключові слова: алгоритми опимізації, глибокі моделі, навчання моделей, ефективність мережі, наближені методи другого порядку, адитивна швидкість навчання.

1. Вступ

Останніми роками глибоке навчання стало трансформаційною технологією, яка зробила революцію в різних галузях. Однак навчання глибоких моделей є дуже нетривіальним завданням, яке потребує різних методів оптимізації для подолання складнощів, притаманних глибоким нейронним мережам. Безпрецедентний успіх глибокого навчання призвів до проривів у багатьох галузях, враховуючи охорону здоров'я, автономне водіння, фінанси та багато інших. Тому розуміння і розробка ефективних методів оптимізації для навчання глибоких моделей має першорядне значення. Поліпшуючи ефективність, швидкість збіжності та можливості узагальнення глибоких моделей, методи оптимізації допомагають розкрити весь потенціал глибокого навчання і вирішувати все складніші та масштабніші проблеми.

Оптимізація для навчання глибоких моделей фокусується на пошуку оптимального набору параметрів, який мінімізує певну цільову функцію, наприклад, функцію втрат[2], шляхом ітеративного налаштування параметрів моделі за допомогою алгоритмів оптимізації на основі градієнта[3].

[1] Навчання моделі – фаза, на якій обирається найліпша комбінація ваг і зміщень для мінімізації функції втрат.

[2] Функція втрат є метрикою, яка вимірює різницю між прогнозованими значеннями моделі та справжніми значеннями даних.

[3] Градієнт – вектор часткових похідних функції втрат щодо параметрів моделі. Градієнт вказує напрям і швидкість найшвидшого зростання функції втрат і допомагає оновлювати параметри моделі, щоб зменшити втрати.

2. Особливості та проблеми алгоритмів оптимізації для глибоких моделей

Алгоритми оптимізації, які використовують для навчання глибоких моделей, відрізняються від традиційних алгоритмів оптимізації в багатьох аспектах. У випадку машинного навчання першочерговою метою є оптимізація показника ефективності P, який, зазвичай, визначається стосовно тестового набору і може бути складним для прямої оптимізації. Натомість ми оптимізуємо іншу функцію вартості (1) з розрахунком на те, що мінімізація J призведе до поліпшення P. Ця відмінність засвідчує непрямий характер оптимізації.

$$\mathrm{J} = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}_i, y_i), \tag{1}$$

де n – кількість прикладів у наборі даних; L – функція втрат; y_i – дійсний результат з набору даних; а $\hat{y}_i$ – прогноз моделі. У загальному випадку, процес знаходження оптимальних ваг на кожній епосі навчання можна описати рівнянням (2), у якому η – швидкість навчання

$$\mathrm{w} = w - \eta dL. \tag{2}$$

Зазвичай ми говоримо про оптимізатори як ті, що використовують або пакетні (обробка всіх навчальних прикладів одночасно), або стохастичні методи (обробка одного прикладу за раз). На практиці найчастіше доцільно використовувати так звані мініпакетні методи, які часто також називають стохастичними. Вони досягають балансу, використовуючи більше ніж один, але менше, ніж всі навчальні приклади. На вибір розміру мінімальної вибірки впливають різні чинники, зокрема баланс між точністю та обчисленнями, використання апаратних архітектур, вимоги до пам'яті й ефекти регуляризації. Алгоритми, які приймають тільки градієнт, зазвичай надійніші і можуть працювати з меншими розмірами пакетів, тоді як методи другого порядку, що охоплюють матрицю Гессіана, потребують більших розмірів пакетів, щоб мінімізувати коливання в оцінках.

3. Види оптимізаторів

3.1. Базові алгоритми

3.1.1. Стохастичний градієнтний спуск (SGD)

Стохастичний градієнтний спуск – це поширений алгоритм оптимізації, який є вдосконаленням традиційного алгоритму градієнтного спуску. Ідея градієнтного спуску згідно з навчанням глибоких моделей полягає в знаходжені мінумуму функції, тобто мінімуму на площині помилок.

Найважливішою властивістю SGD та пов'язаної з ним мініпакетної або стохастичної оптимізації на основі градієнта є те, що час оновлення обчислень не зростає зі збільшенням кількості навчальних прикладів. Це дає змогу збігатися навіть тоді, коли кількість навчальних прикладів стає дуже великою. Це можливо завдяки тому, що SGD оновлює параметри моделі шляхом обчислення та застосування градієнтів до випадково вибраних підмножин навчальних прикладів, а не до всього набору даних. Проте в цьому і недолік такого методу, він чутливий до нерепрезентативних прикладів.

3.1.2. Метод імпульсу (Momentum)

Для прискорення навчання, особливо в умовах високої кривизни, малих, але постійних градієнтів або зашумлених градієнтів доречним є використання методу імпульсу. Алгоритм накопичує експоненціально спадаючу змінну – середнє минулих градієнтів – і продовжує рухатися у їхньому напрямі

$$\mathrm{w} = w - \eta v \tag{3}$$

$$v = (1 - \beta)dJ + \beta v_{t-1}. \tag{4}$$

У стохастичному градієнтному спуску розмір кроку є просто нормою градієнта, помноженою на швидкість навчання. У цьому випадку розмір кроку залежить від того, наскільки велика і наскільки вирівняна послідовність градієнтів. Алгоритм вводить змінну v, яка відіграє роль швидкості – це напрям і швидкість, з якою параметри рухаються в просторі параметрів. Швидкість усталюється як експоненціально спадаюче середнє значення від'ємного градієнта. Розмір кроку є найбільшим, коли багато послідовних градієнтів спрямовані в одному напрямі. З цієї причини метод імпульсу може бути особливо корисним, коли важливо уникати локальних мінімумів і сідлових точок.

3.2. Алгоритми з адивною швидкістю навчання

Гіперпараметри – це змінні, яким розробники надають точні значення, щоб керувати тим, як модель навчається та які показники ефективності досягає. Швидкість навчання є одним із чисельної кількості гіперапараметрів. Вона регулює ваги нейронної мережі стосовно градієнта втрат і демонструє, як часто нейронна мережа оновлює вивчені дані. Усталення швидкості навчання є складним завданням, оскільки вона суттєво впливає на продуктивність моделі. У ході досліджень було вивчено альтернативні підходи до усталення швидкості навчання, зокрема шляхом її автоматичної адаптації в процесі навчання, враховуючи чутливість окремих параметрів. Однак важливо зазначити, що це правило може бути застосоване лише в контексті повної пакетної оптимізації. Хоча конкретні деталі цих алгоритмів можуть відрізнятися, їх спільною метою є підвищення ефективності та результативності процесу навчання шляхом адаптації швидкості навчання відповідно до поточного стану моделі та даних, що трапляються.

3.2.1. AdaGrad

Ключова ідея оптимізатора полягає в тому, щоб масштабувати швидкість навчання обернено пропорційно до квадратного кореня з суми квадратів градієнтів, що спостерігалися до цього моменту. Масштабуючи швидкість навчання так, AdaGrad дає більше оновлень для параметрів, які мають великі часткові похідні функції втрат, що призводить до швидкого зменшення швидкості навчання. З іншого боку, параметри з малими частковими похідними отримують менше оновлень і мають порівняно менше зниження швидкості навчання. Такий підхід допомагає досягти більшого прогресу в пологих напрямах простору параметрів.

У випадку опуклої оптимізації[1] AdaGrad підтверджує очікувані теоретичні влас-

[1]Оптимізація, в якій цільова функція є опуклою функцією, а допустимою множиною є опукла множина. Однією з найважливіших особливостей є те, що задачу можна звести до задачі знаходження локального мінімуму.

тивості. Алгоритм адаптує швидкість навчання індивідуально для кожного параметра на основі накопиченого квадратичного градієнта, що дає змогу йому швидко збігатися для опуклих функцій. Однак, коли справа доходить до навчання моделей глибоких мереж, емпіричні спостереження виявили, що оптимізатор може мати деякі обмеження, бо швидкість навчання може стати занадто малою до досягнення локально опуклої структури, що стає на заваді подальшому прогресу.

3.3. RMSprop (Root Mean Square Propagation)

RMSProp модифікує AdaGrad, вводячи експоненціально зважене рухоме середнє квадратів градієнтів замість того, щоб накопичувати їх з часом. Відкидаючи крайні значення минулої історії за допомогою рухомого середнього, RMSProp може швидко збігатися після знаходження локально опуклої структури, так ніби він був ініціалізований в цій структурі

$$\mathrm{w} = w - \frac{\eta}{\sqrt{v+e}} dJ. \tag{5}$$

$$v = (1-\beta)(dJ)^2 + \beta v_{t-1} \tag{6}$$

3.3.1. Adam (adaptive moment estimation)

Такий алгоритм поліпшує свого попередника, поєднуючи його ідею з методом імпульсу. Він використовує квадрати градієнтів для масштабування швидкості навчання, як RMSprop, і використовує змінне середнє значення градієнта, а не сам градієнт, як це робить SGD з імпульсом. Перевага цього оптимізатора – здатність працювати з зашумленими та розрідженими наборами даних, які трапляються в реальних задачах. Він також може вирішувати задачі, де оптимальне рішення лежить в широкому діапазоні значень параметрів. Емпірично, Adam демонструє, що він часто може перевершити інші методи оптимізації, такі як стохастичний градієнтний спуск і його модифікації, у швидкості збіжності та узагальненні для нових даних

$$\mathrm{w} = w - \frac{\eta}{\sqrt{\widetilde{s}+e}} \widetilde{v} \tag{7}$$

$$v = (1-\beta_1) dJ + \beta_1 v_{t-1}; \widetilde{v} = \frac{v}{1-(\beta_1)^t} \tag{8}$$

$$s = (1-\beta_2)(dJ)^2 + \beta_2 s_{t-1}; \widetilde{s} = \frac{s}{1-(\beta_2)^t}. \tag{9}$$

Проте, як і кожен алгоритм, він має свої слабкі місця, такі як: чутливість до гіперпараметрів (а їх у цьому оптимізаторі 4), використання пам'яті та відсутність гарантії збіжності. Тому оптимальний вибір оптимізатора залежить від конкретного набору даних і задачі, що розв'язується.

3.4. Наближені методи другого порядку

Методи похідних другого порядку забезпечують кращу траєкторію навчання по локальній кривизні поверхні помилок за допомогою додаткової інформації з Гессіана[1]. Використання матриць Гессе полегшує точне налаштування гіперпара-

[1]Гессаін – визначник матриці Гессе, тобто матриці других похідних функції втрат щодо параметрів моделі.

метра шляхом адаптивної зміни розміру кроку (тобто швидкості навчання) відповідно до різних етапів навчання.

3.4.1. Метод Ньютона

Метод Ньютона є основним методом похідних другого порядку. Обмеженням методу Ньютона є те, що він потребує обчислення оберненої матриці Гессіана, як наслідок він має меншу популярність через важкі обчислення та використання пам'яті. Тому в останні роки було запропоновано кілька альтернативних методів похідних другого порядку для вирішення проблем, пов'язаних з цим методом.

3.5. BFGS (Broyden-Fletcher-Goldfarb-Shanno algorithm)

Цей алгоритм належить до класу алгоритмів, які називаються квазіньютонівськими[2] методами, що апроксимують другу похідну (так званий гессіан) для оптимізаційних задач, в яких другу похідну неможливо обчислити.

BFGS зазвичай використовують для традиційних моделей машинного навчання, таких як лінійна регресія, логістична регресія, які мають меншу розмірність простору параметрів порівняно з глибокими нейронними мережами. Адже глибокі мережі зазвичай мають величезну кількість параметрів. Оптимізація BFGS зі збільшенням розмірності простору параметрів стає надзвичайно складною, бо вона передбачає обернення великої матриці Гессіана. Ця операція інверсії є дорогою стосовно пам'яті, що робить її недоцільною для глибоких моделей.

3.5.1. L-BFGS (Limited memory BFGS)

L-BFGS – це розширення алгоритму BFGS, яке вирішує проблему, яка пов'язана з великою кількістю параметрів у наборі даних. Алгоритм не потребує зберігання повного наближення оберненої матриці, а приймає спрощення оберненого гессіана на попередній ітерації алгоритму.

4. Підготовка даних та архітектура моделі

Для подальших досліджень поведінки різних алгоритмів оптимізації обрано набір даних MNIST. Для того, щоб значення пікселів між зображеннями були пропорційними, нормалізовую дані так, щоб вони перебували у діапазоні [0; 1]. Найяскравіший піксель матиме значення 1, а найтемніший – 0. Також для кращої масштабованості та ефективності дані поділяю на мініпакети, в кожному з яких є 32 приклади.

Обрана архітектура для моделі – нейронна мережа прямого поширення. Модель побудованої мережі охоплює такі шари:

- шар входу. Першим шаром є лінійний шар з 784 входами, що відповідають розміру зображення (28x28 пікселів);
- приховані шари. Мережа містить три прихованих шари з кількістю нейронів 128, 64 та 32, відповідно. Кожен з них застосовує функцію активації ReLU *(rectified linear unit)*[1];

[2]Квазіньютонівські алгоритми – це алгоритми оптимізації другого порядку, які апроксимують обернену матрицю Гессе за допомогою градієнта. Тобто не потрібно знати або точно обчислювати матрицю Гессе та її обернену на кожному кроці алгоритму.

[1]ReLU – це функція активації, яка повертає 0, коли отримує негативне значення на вході і для

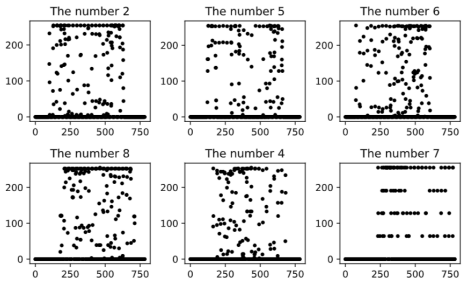

Рис. 1. Дані до нормалізації
Data before normalization

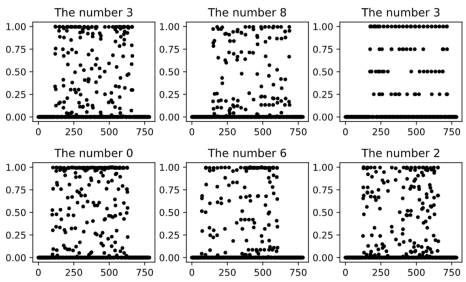

Рис. 2. Дані після нормалізації
Data after normalization

– шар виходу. Останній шар має 10 виходів, які відповідають можливим класам (цифрам від 0 до 9).

Для навчання моделі як функції втрат використовують перехресну ентропію. Швидкість навчання для усіх алгоритмів задана однаковою – 0.001.

5. Результати

Якщо говорити про навчання глибоких моделей, вибір алгоритму оптимізації може суттєво вплинути на ефективність. У наведених результатах можна побачити різну поведінку для чотирьох алгоритмів (рис. 4). В результаті подана остаточна ефективність для кожного з оптимізаторів, які були включені у порівняння, виглядає так:

```
Final Test Performance (SGD): 94.10%
Final Test Performance (Adagrad): 92.45%
Final Test Performance (RMSprop): 96.40%
Final Test Performance (Adam): 97.05%
```

Рис. 3. Остаточна ефективність отримана з використанням того чи іншого алгоритму оптимізації
The final efficiency is obtained using one or another optimization algorithm

Стохастичний градієнтний спуск навчається порівняно повільно є менш стабільним і може застрягнути в точках локальних мінімумів. З оптимізатором RMSprop ми отримуємо постійну збіжність і хорошу стабільність і ліпшим показник ефективності моделі. Алгоритм оптимізації Adam можна вважати "золотим стандартом", адже у цьому випадку і зазвичай він збігається найшвидше і часто досягає найвищої точності тесту. Однак він може потребувати більше обчислювальних ресурсів через додаткові обчислення. AdaGrad ефективний для сценаріїв з розрідженими даними. Загалом у виборі правильного оптимізатора треба враховувати конкретні вимоги та обмеження задачі машинного навчання. Точне налаштування гіперпараметрів також може вплинути на продуктивність цих алгоритмів.

позитивного значення на вході, повертає його ж.

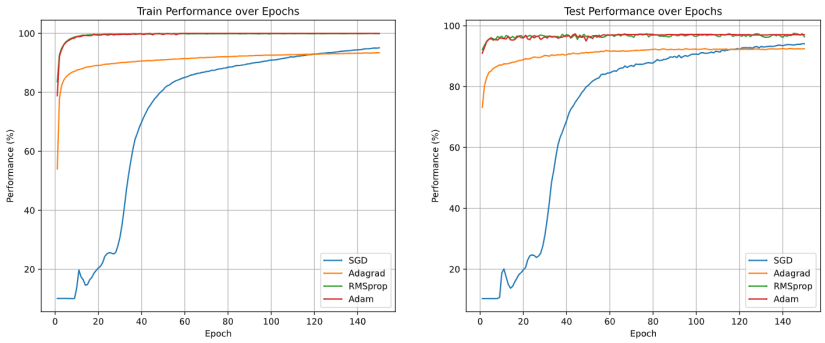

Рис. 4. Зміна ефективності прогнозів зроблених моделлю на навчальній і тестовій вибірках для різних оптимізаторів стосовно епохи навчання
Change in the efficiency of predictions made by the model on the training and test samples for different optimizers with respect to the training epoch

6. Висновок

Програмна реалізація, проведена в рамках цієї статті, підтверджує, що вибір відповідного оптимізатора має вирішальне значення для отримання оптимальної ефективності моделі. Основні аспекти та проблеми оптимізації при навчанні глибоких нейронних мереж були проаналізовані в ході роботи. Було розглянуто ключові методи оптимізації. Проте, хоч у цій статті детально досліджено вплив різних оптимізаторів на ефективність навчання глибоких нейронних мереж, є ще багато тем і методів, які не ввійшли у цей аналіз, але які також заслуговують на увагу в контексті оптимізації глибокого навчання.

Список використаної літератури

1. Goodfellow I. Deep Learning / Ian Goodfellow, Aaron Courville, Yoshua Bengio. – 2015.
2. Kochenderfer Mykel J. Wheeler. Algorithms for Optimization / Mykel J. Kochenderfer, A. Tim. – Massachusetts Institute of Technology. – 2019
3. Quoc V. On optimization methods for deep learning / V. Quoc, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, Y. Andrew. – Computer Science Department. Stanford University. Stanford. CA 94305. USA, 2011.
4. Goodfellow I.J. Qualitatively characterizing neural network optimization problems / I.J. Goodfellow, O. Vinyals, A.M. Saxe. – In International Conference on Learning Representations. – 2015.
5. Lili Jiang A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSProp, Adam) [Електронний ресурс] // Jiang Lili – Jun, 2020. – Режим доступу: https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c
6. Hong Hui Tan Review of second-order optimization techniques in artificial neural networks backpropagation / Hong Hui Tan, King Hann Lim. – Department of Electrical and Computer Engineering – Curtin University Malaysia. – IOP Conf. Series: Materials Science and Engineering, 2019.

Стаття: надійшла до редколегії 21.11.2023
доопрацьована 25.01.2024
прийнята до друку 10.04.2024

TRAINING DEEP MODELS: OPTIMIZATION ALGORITHMS AS A KEY TO EFFICIENCY

M. Stopets, Yu. Shcherbyna

Ivan Franko National University of Lviv,
1, Universytetska str., 79000, Lviv, Ukraine
e-mail: myroslava.stopets@gmail.com, yuriy.shcherbyna@lnu.edu.ua

Due to the extremely large amount of data that we work with, the process of building and training a model is very expensive and requires large computing resources. That is why it becomes a difficult problem to optimize this process, which is crucial for the performance of deep learning. Efficient optimization algorithms can significantly improve the model's learning capability, allowing it to capture complex patterns, extract meaningful representations, and make accurate predictions.

In this work, the problem of optimization in deep model training was investigated. The advantages and disadvantages of different approaches to the learning optimization process are described. The effectiveness of the following algorithms was tested on the standard MNIST dataset: optimization algorithms with additive learning rate, second-order approximation methods, and basic optimization algorithms. As a result of the test, conclusions were drawn on how different optimizers affect the reduction of prediction error and how to choose an optimization algorithm depending on a given task.

Key words: optimization algorithms, deep models, model training, network efficiency, second-order approximation methods, additive learning rate.