УДК 004.85-048.34 **DOI:** 10.31673/2412-9070.2024.021822

Л. Т. АЛЕКСІНА, ст. викладач;
А. П. БОНДАРЧУК, доктор техн. наук, професор,
Державний університет інформаційно-комунікаційних технологій, Київ

ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ ДЛЯ МАШИННОГО НАВЧАННЯ

Не існує єдиного найкращого алгоритму оптимізації гіперпараметрів. Різні алгоритми оптимізації відповідають різноманітним завданням оптимізації гіперпараметрів із відмінними обмеженнями. Для прискорення оптимізації гіперпараметрів потрібно розпаралелити навчальні виконання різних випробувань, запровадити розподілене навчання, достроково зупиняючи безперспективні випробування. Рекомендується використовувати бібліотечний підхід для підтримання послуги оптимізації гіперпараметрів. Серед бібліотек оптимізації гіперпараметрів із відкритим вихідним кодом поки що найкращою є Ray Tune.

Оптимізація гіперпараметрів (ОГП) — це процес виявлення набору гіперпараметрів, який дає оптимальну модель. Йдеться про оптимальну модель, яка мінімізує заздалегідь визначену функцію втрат на заданому наборі даних. Це є повторюваним процесом навчання моделі, за винятком того, що нейронна мережа щоразу тренується з різним набором гіперпараметрів та виявленням оптимального набору гіперпараметрів.

Під час пошуку за сіткою користувачі вказують обмежений набір значень для кожного гіперпараметра, а потім вибирають пробні гіперпараметри з декартового добутку цих значень. Після того, як сітку побудовано, розпочинаються випробування ОГП зі значеннями сітки. Пошук за сіткою буває невдалим, коли кількість гіперпараметрів або простір пошуку параметра збільшуються, оскільки в цьому разі необхідна кількість оцінок зростатиме в геометричній прогресії. Ще однією проблемою пошуку за сіткою є його неефективність. Оскільки такий пошук однаково трактує кожен набір кандидатів гіперпараметрів, він буде споживати багато обчислювальних ресурсів у неоптимальному просторі конфігурації, не витрачаючи при цьому достатньо обчислювальної потужності на оптимальний простір.

Ключові слова: гіперпараметр; оптимізація гіперпараметрів; безмодельний метод; байєсівський метод; метод множинності; бібліотека оптимізації гіперпараметрів.

Вступ

Гіперпараметри — це параметри, значення яких потрібно задати до початку процесу навчання моделі. Швидкість навчання, розмір партії та кількість прихованих шарів є прикладами гіперпараметрів. На відміну від значень параметрів моделі, наприклад ваг і зміщення, гіперпараметри не можуть бути засвоєні в процесі навчання.

Дослідження показують, що вибране значення гіперпараметрів може впливати як на якість навчання моделі, так і на часові та ресурсні вимоги алгоритму навчання. Тому гіперпараметри мають бути налаштовані в такий спосіб, щоб бути оптимальними для навчання моделі. У наш час оптимізація гіперпараметрів є стандартним інструментом у процесі розроблення моделей глибокого навчання.

Аналіз дослідження. Оптимізація гіперпараметрів (ОГП) — це процес виявлення набору гіперпараметрів, який дає оптимальну модель. Під оптимальною розуміють модель, яка мінімізує заздалегідь визначену функцію втрат на заданому наборі даних. Це є повторюваним процесом навчання моделі, за винятком того, що нейронна мережа щоразу тренується з різним набором гіперпараметрів. При цьому буде виявлено оптимальний набір гіперпараметрів. Зазвичай кожен запуск тренування моделі називається випробуванням. Весь експеримент — це пробний цикл, в якому здійснюється одне випробування за одним доти, доки не буде досягнуто кінцевих критеріїв. Переважно, щоб мати справедливу оцінку, для кожного дослідження ОГП використовується один і той самий набір даних.

Кожне випробування має чотири етапи.

Крок 1. Навчання нейронної мережі набором значень гіперпараметрів.

Крок 2. Оцінювання результату навчання (моделі).

Крок 3. Процес ОГП перевіряє, чи було виконано кінцеві критерії, наприклад, чи закінчився бюджет на випробування або чи досягла модель, вироблена в цьому випробуванні, цільового оцінювання ефективності.

Якщо результат випробування відповідає кінцевій умові, пробний цикл розривається, і експеримент завершується. Оптимальними гіперпараметрами вважаються такі значення гіперпараметрів, котрі дали найкращий результат оцінювання моделі. Якщо кінцева умова не виконується, процес переходить до кроку 4.

Крок 4. Процес ОГП видає новий набір значень гіперпараметрів і починає нове випробування (тренувальний запуск моделі).

Значення гіперпараметрів, які використовуються в кожному дослідженні, генеруються вручну або автоматично за допомогою алгоритму ОГП.

Метою дослідження є порівняння бібліотек оптимізації гіперпараметрів із відкритим вихідним кодом для навчання моделей на локальних машинах або на серверах, до яких є прямий доступ.

Основна частина

Більшість алгоритмів ОГП можна розділити на три сегменти: безмодельна оптимізація, баєсова оптимізація та оптимізація множинності.

У ***безмодельних методах*** не робиться жодних припущень щодо навчальної моделі, а кореляція між випробуваннями ОГП ігнорується. Пошук за сіткою і випадковий пошук є найчастіше застосовними методами.

Під час пошуку за сіткою користувачі вказують обмежений набір значень для кожного гіперпараметра, а потім вибирають пробні гіперпараметри з декартового добутку цих значень. Після того, як сітку побудовано, розпочинаються випробування ОГП зі значеннями сітки. Пошук за сіткою зазнає невдачі, коли кількість гіперпараметрів або простір пошуку параметра збільшуються, оскільки в цьому разі необхідна кількість оцінок зростатиме в геометричній прогресії. Ще однією проблемою пошуку за сіткою є його неефективність. Оскільки пошук за сіткою однаково трактує кожен набір кандидатів гіперпараметрів, він буде марнувати багато обчислювальних ресурсів в неоптимальному просторі конфігурації, не витрачаючи при цьому достатньо обчислювальної потужності на оптимальний простір.

Випадковий пошук працює через випадкову вибірку простору конфігурації гіперпараметра доти, доки не закінчиться певний бюджет на пошук. Такий підхід має дві переваги перед пошуком за сіткою. По-перше, випадковий пошук може оцінити більше значень для кожного гіперпараметра, що збільшує шанс знайти оптимальний набір гіперпараметрів. По-друге, випадковий пошук має простіші вимоги до розпаралелювання. Недоліком випадкового пошуку є невизначеність. Немає гарантії, що оптимальний набір гіперпараметрів може бути знайдений у межах лімітованого обчислювального бюджету. Теоретично, якщо допускається достатня кількість ресурсів, випадковий пошук може додати належну кількість випадкових точок у пошуку, тому він, як і очікувалося, визначить оптимальний набір гіперпараметрів. На практиці як базовий використовується випадковий пошук.

Баєсова оптимізація [1] — це найсучасніша оптимізаційна система для глобальної оптимізації функцій «чорної скриньки». Вона широко використовується для різних проблемних налаштувань, зокрема класифікації зображень, розпізнавання мови та нейронного мовного моделювання. Баєсові методи оптимізації можуть застосовувати різні вибірки, такі як регресія процесів Гаусса [2] та деревоподібний підхід оцінювача Парзена (ДПОП) [3], для обчислення кандидатів на гіперпараметри у просторі пошуку. Баєсові методи оптимізації використовують статистичні методи для обчислення нових пропозицій щодо значень гіперпараметрів на основі значень, застосованих у минулих випробуваннях, та результатів їх оцінювання. Тому баєсовий пошук із більшою ймовірністю визначає оптимальні гіперпараметри в даному просторі пошуку та в разі обмеженого бюджету виконання вибрані значення гіперпараметрів стають все ближчими і ближчими до оптимального значення після кожного випробування в процесі пошуку.

Налаштування гіперпараметрів на великих наборах даних може тривати кілька годин, а іноді навіть днів. Для прискорення ОГП було розроблено ***методи множинності*** [4]. У цьому підході мінімізується функція втрат за допомогою так званих низькоточних наближень функції фактичних втрат. Це метод оцінювання того, наскільки добре алгоритм навчання моделює набір даних. Якщо результат моделі далекий від очікуваних прогнозів, функція втрат має виводити більше число. В іншому разі вона повинна вивести менше число.

Функція втрат [5] є ключовим компонентом розроблення алгоритму машинного навчання. Конструкція функції втрат безпосередньо впливає на точність моделі. Незважаючи на те, що апроксимація вводить компроміс між продуктивністю оптимізації та часом виконання, на практиці прискорення здебільшого переважує помилки апроксимації.

У разі сервісного підходу процес ОГП відбувається у віддаленому обчислювальному кластері, керованому службою ОГП сервісу. Треба лише надати службі навчальний код та вибрану конфігурацію алгоритму ОГП та запустити завдання ОГП. Сервіс керує як розподілом обчислювальних ресурсів, так і виконанням робочого процесу ОГП. Він відстежує результат кожного випробування і повертає остаточні оптимальні гіперпараметри, коли всі випробування завершено.

Сервісний підхід забезпечує справжні переваги в роботі з «чорною скринькою». Не потрібно турбуватися стосовно керування власними серверами, налаштування пробних працівників і вивчення того, як модифікувати навчальний код для роботи з різними алгоритмами ОГП. Усі ці завдання бере на себе

служба ОГП сервісу. Як користувач сервісу ОГП просто потрібно передати параметри в сервіс, а потім сервіс автоматично запускає ОГП і повертає оптимальні гіперпараметри в кінці. Сервіс також піклується про автоматичне масштабування та відновлення після невдалих пробних завдань. Завдяки цим перевагам сервісний підхід зараз є найпоширенішим підходом ОГП у виробничому середовищі глибокого навчання.

Hyperopt — це легка та проста у використанні бібліотека Python для послідовного та паралельного ОГП. Такі алгоритми ОГП, як випадковий пошук, ДПОП та адаптивний ДПОП, реалізовано в Hyperopt. Є можливість у три кроки використати Hyperopt [6], щоб дістати відповідь, саме які класифікатори найкраще відповідають глибокому навчанню. Для цього потрібно:

- по-перше, створити цільову функцію, яка, по суті, є функцією-обгорткою фактичного навчального коду, але зчитує значення гіперпараметрів зі змінної args;
- по-друге, визначити простір пошуку для вибраного гіперпараметра;
- по-третє, вибрати алгоритм ОГП, який використовує значення гіперпараметрів із простору пошуку і передає їх цільовій функції для запуску процесу оптимізації.

Hyperopt є хорошим варіантом для невеликих або ранніх стадій модельних навчальних проєктів. Він простий у застосуванні і дружній до модифікації. До того ж Hyperopt може використовувати обчислення Spark [7] для паралельного запуску ОГП.

Optuna також є бібліотекою Python, призначеною для автоматизації пошуку гіперпараметрів. Вона підтримує пошук у великому просторі та раннє спрощення для неперспективних випробувань, а також розпаралелювання на кількох потоках або процесах без зміни коду. Optuna є успішною версією Hyperopt і її можливості візуалізації набагато кращі. Досліджуючи взаємодію між параметрами на графіку, візуалізація в пошуку гіперпараметрів дає багато інформації, тому легко визначити, які параметри ефективніші за інші. Візуалізація Optuna корисна та інтерактивна, вона має добре підтримуваний вихідний код. У порівнянні з Hyperopt, Optuna вимагає, щоб більша частина логіки ОГП була визначена в цільовій функції.

Загальну схему коду можна подати так. Спочатку визначаємо простір пошуку та генеруємо значення гіперпараметрів за trial.suggest_xxx функцією. Далі запускаємо навчання моделі з вибіркових значень гіперпараметрів, а потім — метод оцінювання, щоб обчислити продуктивність моделі та повернути цільове значення.

За допомогою Optuna є змога запустити розподілений ОГП на одній машині або в кластері машин. Налаштування розподіленого виконання досить просте і може бути виконано в три кроки. По-перше, потрібно запустити сервер реляційних баз даних (MySQL), по-друге, створити дослідження з аргументом зберігання, і, по-третє, розподілити дослідження між кількома вузлами та процесами.

Ray Tune [8] — це бібліотека Python для ОГП у будь-якому масштабі, побудована поверх Ray [9], який надає простий універсальний API для створення розподілених застосунків. Бібліотека Ray Tune підтримує практично будь-які framework машинного навчання, включно з PyTorch, XGBoost, MXNet і Keras. Вона також підтримує найсучасніші алгоритми ОГП, зокрема Population Based Training (PBT), BayesOptSearch і HyperBand/ASHA. Крім того, Ray Tune надає механізм для інтеграції алгоритмів ОГП з інших бібліотек ОГП, наприклад підтримується інтеграція з Hyperopt.

Розподілене виконання є найбільшою перевагою Ray Tune порівняно з Optuna та Hyperopt. Ray Tune дає змогу прозоро розпаралелювати роботи між кількома графічними процесорами та кількома вузлами. На противагу Optuna і Hyperopt, не потрібно вручну налаштовувати розподілене середовище і виконувати робочі програми на кожному сервері. Ray Tune виконує ці кроки автоматично. Для цього слід спочатку налаштувати Ray-кластер командою «*ray up tune-cluster.yaml*» (саме в *tune-cluster.yaml* описано конфігурацію кластера, яка оголошує обчислювальні ресурси кластера). Потім виконуємо команду, яка відправить код ОГП із локальної машини в головний вузол кластера: «*ray submit tune-cluster.yaml tune_script.py --start -- --ray-address={server_address}*». Далі Tune [10] призначає ресурси, копіює код ОГП на сервери та запускає розподілене виконання.

Окрім розподіленого виконання ОГП Ray Tune також підтримує запуск розподіленого навчання для однопробного автоматичного керування контрольними точками та ведення журналу TensorBoard. Ці функції додають великої цінності Ray Tune завдяки високій відмовостійкості та простому усуненню помилок.

Ray Tune забезпечує інтеграцію між базовим навчальним ramework (наприклад, TensorFlow і PyTorch) і передовими алгоритмами ОГП (баєсовим пошуком або ДПОП), а також ранню зупинку (ASHA). Це уможливлює виконання розподіленого пошуку ОГП простим і надійним способом.

Обговорення результатів проведеного дослідження. З аналізу результатів дослідження випливає, що Ray Tune проста у використанні та відповідає вимогам ОГП майже кожного модельного навчального

проєкту. Ця бібліотека має чудову документацію, підтримує передові алгоритми ОГП, а також використовує ефективне та просте керування розподіленим виконанням робіт.

Застосування Ray Tune для реалізації завдання ОГП дуже просте.

По-перше, визначається цільова функція. У функції зчитуємо значення гіперпараметрів зі змінної конфігурації, запускаємо навчання моделі та повертаємо рейтинг оцінювання.

По-друге, встановлюються гіперпараметри та простір пошуку їх значень.

По-третє, запускається виконання ОГП, щоб зв'язати цільову функцію та простір пошуку разом.

```
# Step 1: define objective_function
def objective_function(config):
 model = ConvNet( )
 model.to(device)
  optimizer = optim.SGD(model.parameters( ), lr=config["lr"], momentum=config["momentum"])
 for i in range(10):
 train(model, optimizer, train_loader)
 acc = test(model, test_loader)
 tune.report(mean_accuracy=acc)
# Step 2: define search space for each hyperparameter
search_space = {
 "lr": tune.sample_from(lambda spec: 10**(-10 * np.random.rand( ))),
"momentum": tune.uniform(0.1, 0.9)
}
# Uncomment this to enable distributed execution
# `ray.init(address="auto")`
# Step 3: start the HPO execution
analysis = tune.run( objective_function, num_samples=20,
 scheduler=ASHAScheduler(metric="mean_accuracy", mode="max"), config=search_space)
# check HPO progress and result
# obtain a trial dataframe from all run trials
# of this `tune.run` call.
dfs = analysis.trial_dataframes
```

Об'єкт планувальника, ASHAScheduler, передається у функцію tune.run на кроці 3. ASHA [11] — це масштабований алгоритм для принципової ранньої зупинки [12]. На високому рівні ASHA припиняє випробування, які є менш перспективними, і виокремлює час і ресурси на більш перспективні випробування. Правильне налаштування параметра *num_samples* може зробити пошук набагато ефективнішим і тоді є можливість підтримувати більший простір пошуку.

Висновки

Ray Tune та інші бібліотеки ОГП досягають своєї межі, коли потрібно підтримувати різні команди та різні проєкти глибокого навчання в одній спільній системі ОГП. У Ray Tune відсутня ізоляція обчислень, що призводить до двох великих проблем: по-перше, пакетні версії різних навчальних кодів можуть викликати конфлікти під час виконання розподіленого ОГП в Ray Tune, і, по-друге, Ray Tune не застосовує сегрегацію користувачів. Дуже складно побудувати віртуальну межу в Ray Tune для різних команд, щоб обмежити використання обчислювальних ресурсів.

Проте рекомендуємо використовувати Ray Tune замість інших бібліотек ОГП із таких причин:

- проста у використанні;
- чудова документація та багато прикладів;
- розподілене виконання є автоматичним і програмним;
- підтримується розподілене навчання для одного випробування;
- має функцію планувальника, наприклад ASHAScheduler, яка може значно знизити обчислювальні витрати, припинивши неперспективні випробування раніше.

Список використаної літератури

*1. **Wundervald B.** Bayesian Linear Regression. June 2019. URL: https://www.researchgate.net/publication/333917874_Bayesian_Linear_Regression*

*2. **Wang J.** An Intuitive Tutorial to Gaussian Processes Regression. 22 September 2020. URL: https://arxiv.org/abs/2009.10862*

3. ***Watanabe S.*** *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance. 21 April 2023. URL:*
https://arxiv.org/abs/2304.11127
4. ***Feurer M., Hutter F.*** *Hyperparameter Optimization. 2019. URL:*
www.automl.org/wp-content/uploads/2019/05/AutoML_Book_Chapter1.pdf
5. ***Introduction*** *to Loss Functions by DataRobot. 30 April 2018. URL:*
https://www.datarobot.com/blog/introduction-to-loss-functions/
6. ***Agrawal T.*** *On Using Hyperopt: Advanced Machine Learning. 20 June 2018. URL:*
http://mng.bz/PxwR
7. ***Hyperopt*** *Documentation. URL:*
http://hyperopt.github.io/hyperopt/scaleout/spark/
8. ***Ray Tune:*** *Hyperparameter Tuning. URL:*
https://docs.ray.io/en/latest/tune/index.html
9. ***Ray*** *Documentation. URL:*
https://docs.ray.io/en/latest/index.html
10. ***Running*** *Distributed Experiments with Ray Tune. URL:*
http://mng.bz/71QQ
11. ***ASHA*** *(tune.schedulers.ASHAScheduler). URL:*
http://mng.bz/JlwZ
12. ***Liam Li.*** *Massively Parallel Hyperparameter Optimization. 12 December 2018. URL:*
http://mng.bz/wPZ5

L. Aleksina, A. Bondarchuk

HYPERPARAMETERS OPTIMIZATION FOR THE MACHINE LEARNING

There is no single best Hyperparameter Optimization algorithm. Different optimization algorithms meet different Hyperparameter Optimization tasks with different constraints. To accelerate the Hyperparameter Optimization, it is necessary to parallelize training executions of various tests, introduce distributed training and prematurely stop unpromising tests. It is recommended to use a library approach to support the hyperparameter optimization service. Ray Tune is the best by far among the open source Hyperparameter Optimization.

Hyperparameter optimisation (HPO) is the process of identifying a set of hyperparameters that yields an optimal model. By optimal, we mean the model that minimises a predefined loss function on a given data set. This is a repeated model training process, except that the neural network is trained with a different set of hyperparameters each time. In this process, the optimal set of hyperparameters will be identified.

During the grid search, users specify a limited set of values for each hyperparameter and then select trial hyperparameters from the Cartesian product of these values. Once the grid is constructed, the GPGs are tested with the grid values. The grid search suffers when the number of hyperparameters becomes larger or the parameter search space becomes larger, as in this case the required number of estimates will grow exponentially. Another problem with grid search is its inefficiency. Since grid search treats each set of hyperparameter candidates equally, it will spend a lot of computational resources in the suboptimal configuration space without spending enough computational power on the optimal space.

Keywords: hyperparameter; hyperparameter optimization; model-free method; Bayesian method; multiplicity method; hyperparameter optimization library.