# Анотація

Ця стаття розглядає проблему масштабованості пошуку архітектури, формулюючи завдання диференційовано. На відміну від звичайних підходів застосування еволюції або навчання з підкріпленням над дискретним та недиференційованим пошуковим простором, наш метод базується на безперервному розслабленні архітектурного представлення, що дозволяє ефективно шукати архітектуру за допомогою градієнтного спуску. Широкі експерименти з CIFAR-10, ImageNet, Penn Treebank і WikiText-2 показують, що наш алгоритм перевершує відкриття високопродуктивних згорткових архітектур для класифікації зображень і повторюваних архітектур для моделювання мови, хоча на порядок швидше, ніж state-of-the-art недиференційованих методів.

# Вступ

Відкриття сучасних архітектур нейронних мереж вимагає значних зусиль людських фахівців. Останнім часом спостерігається зростаючий інтерес до розробки алгоритмічних рішень для автоматизації їхвласного процесу проектування архітектури. Автоматично шукані архітектури досягли високої конкурентоспроможності в таких завданнях, як класифікація зображень (Zoph and Le, 2016; Zoph et al., 2017; Liu et al., 2017 b, a; Real et al., 2018) і виявлення об'єктів ( Zoph et al., 2017).

Кращі алгоритми пошуку існуючої архітектури вимагають обчислень, незважаючи на їх чудову продуктивність. Наприклад, для отримання сучасної архітектури для CIFAR-10 і ImageNet було потрібно 1800 днів GPU навчання (RL) (Zoph et al., 2017) або 3150 GPU днів еволюції (Real et al., 2018) ). Запропоновано кілька підходів для прискорення, таких як запровадження конкретної структури простору пошуку (Liu et al., 2017b, a), вагові коефіцієнти або прогноз продуктивності для кожної окремої архітектури (Brock et al., 2017; Baker et al. , 2018) і розподіл ваги між декількома архітектурами (Pham et al., 2018b; Cai et al., 2018), але основний виклик масштабованості залишається. Невід'ємною причиною неефективності для домінуючих підходів, напр. на основі RL, еволюції, MCTS (Negrinho і Gordon, 2017), SMBO (Liu et al., 2017a) або байєсівська оптимізація (Kandasamy et al., 2018), є той факт, що пошук архітектури розглядається як чорна коробка Проблема над дискретним доменом, що призводить до великої кількості необхідних архітектурних оцінок.

У цій роботі ми підходимо до проблеми з іншого боку і пропонуємо метод ефективного пошуку архітектури під назвою DARTS (Диференційований пошук ARchiTecture). Замість того, щоб шукати дискретний набір архітектур-кандидатів, ми розслабляємо простір пошуку, щоб бути безперервним, так що архітектура може бути оптимізована по відношенню до продуктивності набору валідації за допомогою градієнтного спуска. Ефективність даних на основі оптимізації на основі градієнта, на відміну від неефективного пошуку в чорній коробці, дозволяє DARTS досягати конкурентних показників за допомогою сучасних технологій, використовуючи на порядки менше ресурсів обчислення. Це також перевершує інший ефективний метод пошуку архітектури ENAS (Pham et al., 2018b). Примітно, що DARTS є простішим, ніж багато існуючих підходів, оскільки він не включає жодних контролерів (Zoph and Le, 2016; Baker et al., 2016; Zoph et al., 2017; Pham et al., 2018b), гіпермережі (Brock et al., 2016). ., 2017) або прогностичні показники (Liu et al., 2017a), але воно є достатньо загальним для пошуку як згорткових, так і рекурентних архітектур.

Ідея пошуку архітектур в рамках безперервного домену не є новою (Saxena і Verbeek, 2016; Ahmed і Torresani, 2017; Shin et al., 2018), але є кілька основних відмінностей. Хоча попередні роботи прагнуть тонко налаштувати певний аспект архітектури, наприклад, форми фільтрів або структури розгалужень у згортковій мережі, DARTS здатний виявляти високопродуктивні архітектури зі складною топологією графів у просторі пошуку. Більш того, DARTS не обмежується жодною конкретною архітектурною сім'єю і здатний виявляти як згорткові, так і рекурентні мережі.

У наших експериментах (розділ 3) ми показуємо, що DARTS здатний сконструювати згортковий елемент, який досягає 2,83 ± 0,06% випробувальної похибки на CIFAR-10 для класифікації зображень, яка є конкурентоспроможною за результатами дослідження шляхом регуляризації еволюції (Real et al., 2018) отримані з використанням трьох порядків величини великих обчислювальних ресурсів. Така ж згорткова клітина також досягає 26,9% помилки top-1, яка переходить до ImageNet (мобільна установка), яка порівнянна з найкращим методом RL (Zoph et al., 2017). На задачі мовного моделювання DARTS виявляє повторювану осередок, яка досягає 56,1 процвітання в Penn Treebank (PTB) в один графічний день, перевершуючи як налаштований LSTM (Meliset al., 2017), так і всі існуючі автоматично шукані клітини на основі NAS (Zoph). і Le, 2016) іENAS (Pham et al., 2018b).

Наші внески можна підсумувати таким чином:

* Ми вводимо новий алгоритм для пошуку диференційованої мережевої архітектури, який застосовується як до згорткових, так і до рекурентних архітектур.
* Через широкі експерименти з класифікації зображень та завдань мовного моделювання ми показуємо, що пошук на основі градієнтної архітектури досягає висококонкурентних результатів на CIFAR-10 і перевершує сучасний рівень PTB. Це дуже цікавий результат, вважаючи, що до цих пір найкращі архітектурні методи пошуку використовуються недиференційованими методами пошуку, наприклад. на основі RL (Zoph et al., 2017) або еволюція (Real et al., 2018; Liu et al., 2017b).
* Ми досягаємо чудової ефективності пошуку архітектури (з 4 графічними процесорами: похибка 2.83% на CIFAR10 за 1 день; 56.1 розгубленість на PTB за 6 годин), яку ми пов'язуємо з використанням оптимізації на основі градієнта, на відміну від недиференційованих методів пошуку.
* Ми показуємо, що архітектури, засвоєні DARTS на CIFAR-10 і PTB, можуть бути передані до ImageNet і WikiText-2 відповідно.

# Пошук диференційованої архітектури

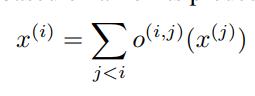
Ми описуємо наш пошуковий простір у загальному вигляді в секції. 2.1, де процедура обчислення для анархітектури (або клітинки в ній) представлена у вигляді спрямованого ациклічного графа. Потім ми запровадимо просту безперервну схему релаксації для нашого пошукового простору, що призводить до диференційованої мети навчання для спільної оптимізації архітектури та її ваг (розділ 2.2). Нарешті, ми пропонуємо метод апроксимації, щоб зробити алгоритм обчислювально здійсненним і ефективним (Сек. 2.3).

## Пошуковий простір

Також як Zoph et al. (2017); Real et al. (2018); Liu et al. (2017a, b), шукаємо обчислювальну клітину як будівельний блок кінцевої архітектури. Вивчену клітинку можна або складати, щоб сформувати згорткову мережу, або рекурсивно з'єднати, щоб сформувати рекурентну мережу.

Кожен вузол x (i) є латентним представленням (наприклад, картою ознак у згорткових мережах), і кожен спрямований край (i, j) пов'язаний з деякою операцією o (i, j), яка перетворює x (i). Будемо вважати, що комірка має два вхідних вузла і один вихідний вузол. Для згорткових осередків вхідні вузли визначаються як вихідні клітинки у попередніх двох шарах (Zoph et al., 2017). Для повторюваних комірок вони визначаються як вхідні дані на поточному кроці, а стан, що переноситься з попереднього кроку. Вихід комірки отримують, застосовуючи операцію відновлення (наприклад, конкатенацію) до всіх проміжних вузлів.

Кожен проміжний вузол обчислюється на основі всіх його попередників:



Спеціальна нульова операція також включена, щоб вказати на відсутність зв'язку між двома вузлами. Таким чином, завдання вивчення комірки зводиться до вивчення операцій по її краях.

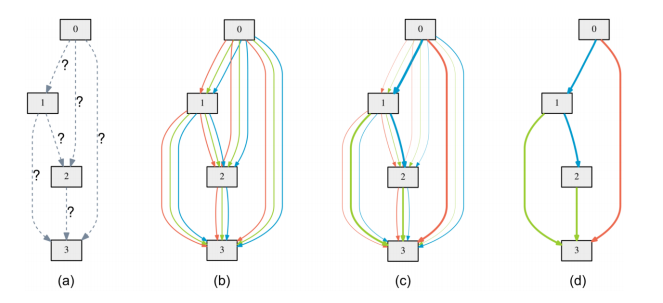
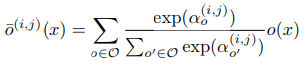


Рисунок 1 - Огляд DARTS: (a) Операції на краях спочатку невідомі. (b) Безперервна релаксація простору пошуку шляхом розміщення на кожному краю суміші операцій-кандидатів. (c) Спільнеоптимізація ймовірностей змішування і вагових коефіцієнтів мережі шляхом вирішення двовимірної задачі оптимізації. (d) Індукування остаточної архітектури з вивчених ймовірностей змішування.

## Continuous Relaxation and Optimization

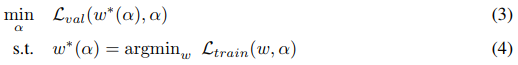
Нехай O - набір операцій-кандидатів (наприклад, згортка, максимальне об'єднання, нуль), де кожна операція являє собою деяку функцію o (·), яка буде застосована до x(i). Для того, щоб простір пошуку був безперервним, ми розслабляємо категоричний вибір конкретної операції як софтмакс над усіма можливими операціями:



де операції змішування ваг для пари вузлів (i, j) параметризовані за допомогою вектора α (i, j) розмірності | O |. Після релаксації завдання пошуку архітектури зводиться до вивчення безлічі безперервних змінних α α (i, j), як показано на малюнку 1. Після закінчення пошуку дискретна архітектура виходить шляхом заміни кожної змішаної операції o¯ (i j) з найбільш вірогідною операцією, тобто o (i, j) = argmaxo∈O α (i, j) o. У подальшому ми посилаємося на α як (кодування) архітектури.

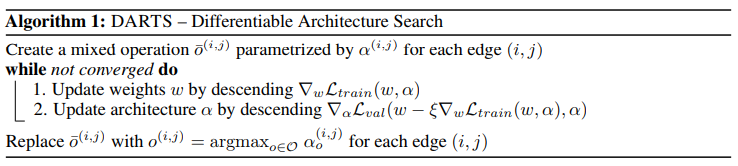
Після релаксації наша мета полягає в спільному вивченні архітектури α і ваг w у всіх змішаних операціях (наприклад, ваги фільтрів згортки). Аналогічний архітектурному пошуку за допомогою RL (Zoph and Le, 2016; Zoph et al., 2017; Pham et al., 2018b) або еволюція (Liu et al., 2017b; Real et al., 2018), де ефективність набору перевірок трактується як винагорода або придатність, DARTS прагне оптимізувати втрату валідації, але з використанням градієнтного спуску.

Позначимо через Ltrain і Lval тренування і втрату валідації відповідно. Обидві втрати визначаються не тільки архітектурою α, але і вагою w в мережі. Мета архітектурного пошуку полягає в тому, щоб знайти α∗, що мінімізує втрати валідації Lval (w ∗, α ∗), де ваги w∗, пов'язані з архітектурою, отримуються шляхом мінімізації втрат на навчання w∗ = argmin w Ltrain (w, α∗). Це має на увазі задачу оптимізації на двох рівнях (Anandalingam and Friesz, 1992; Colson et al., 2007) з α як змінну верхнього рівня і w як змінну нижнього рівня:



Вкладена формулювання також виникає в оптимізації гіперпараметрів на основі градієнта (Maclaurin et al., 2015; Pedregosa, 2016), що пов'язано в тому сенсі, що безперервну архітектуру α можна розглядати як особливий тип гіперпараметра, хоча його розмірність істотно вище, ніж скалярні гіперпараметри (наприклад, швидкість навчання), і її важче оптимізувати.

## Наближення



Вирішення двовимірної оптимізації точно є непосильним, оскільки вимагатиме перерахунку w ∗ (α) шляхом вирішення внутрішньої задачі (4), коли відбувається будь-яка зміна в α. Таким чином, ми пропонуємо приблизну процедуру оптимізації, де w і α оптимізуються шляхом чергування кроків градієнтного спуску у вагових і архітектурних просторах відповідно (Alg. 1). На етапі k з урахуванням струмової архітектури α\_(k – 1) отримуємо w\_k, переміщуючи w\_(k – 1) у напрямку мінімізації тренувальної витрати Ltrain (wk − 1, αk − 1). Потім, зберігаючи фіксовані ваги wk, ми оновлюємо архітектуру таким чином, щоб зменшити втрату валідації після одного етапу сходження градієнта w.r.t. ваги:

де ξ - швидкість навчання для цього віртуального кроку градієнта. Мотивація (5) полягає в тому, що ми хотіли б знайти архітектуру, яка має низьку втрату валідації, коли її ваги оптимізовані за допомогою спуску градієнтів, де однокрокові розгортані ваги служать сурогатом для w ∗ (α). Для мета-навчання для трансферу моделей був використаний подібний підхід (Finn et al., 2017). Примітно, що динаміка нашого ітеративного алгоритму визначає гру Stackelberg (Von Stackelberg, 1934) між оптимізатором (лідером) і оптимізатором w (послідовник), який зазвичай вимагає від лідера передбачити наступний крок наступного кроку для досягнення рівноваги. Хоча в даний час ми не знаємо про гарантії конвергенції нашого алгоритму оптимізації, на практиці вона здатна сходитися з прийнятним вибором ξ. Ми також зауважимо, що коли імпульс для оптимізації ваги включений, єдина навчальна мета (5) змінюється відповідно, і весь наш аналіз все ще застосовується.

Архітектурний градієнт дається шляхом диференціації (5) w.r.t. α (пропускаємо індекс кроку k для стислості):

де позначає вагові коефіцієнти для однокрокової моделі вперед. Градієнт (6) містить матрично-векторний продукт у своєму другому члені, який є дорогим для обчислення. На щастя, складність може бути значно зменшена за допомогою апроксимації кінцевої різниці.

Оцінювання кінцевої різниці вимагає лише двох прямих пропусків для ваг і двох зворотних проходів для α, а складність зменшується від O (| α || w |) до O (| α | + | w|). Коли ξ = 0, похідна другого порядку в (6) зникне. У цьому випадку градієнт архітектури задається LαLval (w, α), що відповідає простої евристиці оптимізації втрати валідації за умови, що α і w незалежні один від одного. Це призводить до деякого прискорення, але емпірично гірших показників, згідно з нашими експериментальними результатами в табл. 1 та табл.2. 0 як наближення другого порядку.

## Виведення дискретних архітектур

Після отримання безперервної архітектури, що кодує α, дискретну архітектуру отримують

1. Зберігаючи k найсильніших попередників для кожного проміжного вузла, де сила ребра визначається як. Щоб зробити нашу похідну архітектуру порівнянною з такою в існуючих роботах, ми використовуємо k = 2 для згорткових клітин (Zoph et al., 2017; Real et al., 2018) і k = 1 для рекурентних клітин (Pham et al., 2018b).
2. Замінюючи кожну змішану операцію як найбільш ймовірну операцію, беручи argmax.

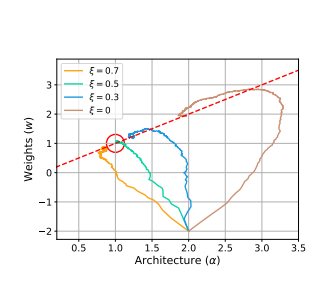


Рисунок 2: Динаміка навчання нашого ітераційного алгоритму при Lval (w, α) = αw - 2α + 1 і Ltrain (w, α) = w2 - 2αw + α2, починаючи з (α (0), w (0)) = ( 2, −2). Аналітичним рішенням для відповідної задачі двох-рівневої оптимізації є (α∗,w∗)=(1, 1), що висвічується в червоному колі. Пунктирна червона лінія вказує на можливий набір, де обмеження (4) виконується точно (а саме ваги в w є оптимальними для даної архітектури α). Приклад показує, що відповідний вибір ξ допомагає сходитися до кращого локального оптимуму.

# Пошук архітектури

## Пошук свердловинних осередків на CIFAR-10

Ми включаємо наступні операції в O: 3 × 3 і 5 × 5 роздільних згорток, 3 × 3 і 5 × 5 розріджених відокремлюваних згорток, 3 × 3 максимального об'єднання, 3 × 3 середнього об'єднання, ідентичності і нуля. Усі операції мають один крок (якщо застосовується) і згорнуті карти властивостей доповнюються для збереження їх просторового дозволу. Ми використовуємо порядок ReLU-Conv-BN для згорткових операцій, і кожна зрізана згортка завжди застосовується двічі (Zoph et al., 2017; Real et al., 2018; Liu et al., 2017a).

Наша згорткова клітина складається з N = 7 вузлів, серед яких вихідний вузол визначається як поглиблена конкатенація всіх проміжних вузлів (вхідні вузли виключені). Іншу частину налаштувань слідує Zoph et al. (2017); Liu et al. (2017a); Real et al. (2018), де мережа потім формується шляхом складання декількох осередків разом. Перший і другий вузли осередку k задаються рівними виходам клітини k − 2 і осередку k − 1 відповідно, а в разі необхідності вставляються згортки 1 × 1. Клітини, розташовані на 1/3 і 2/3 від загальної глибини мережі, є редукційними осередками, в яких всі операції, розташовані поруч з вхідними вузлами, мають два кроки. Архітектура кодування, таким чином, є (α normal, α-reduce), де α-normal поділяється всіма нормальними клітинами, а α-reduce розділяється всіма клітинами відновлення.

Оскільки архітектура буде змінюватися протягом усього процесу пошуку, ми завжди використовуємо специфіку пакета для нормалізації пакетів ніж глобальна ковзна середня. Досліджувані афінні параметри в усіх нормалізаціях пакетів вимикаються під час процесу пошуку, щоб уникнути масштабування виходів операцій-кандидатів.

Щоб виконати пошук архітектури, ми зберігаємо половину даних тренувань CIFAR-10 як набори перевірок. Невелика мережа, що складається з 8 осередків, навчається з використанням DARTS на 50 епох, з розміром партії 64 (як для навчальних, так і для валідаційних наборів) і початковим числом каналів 16. Цифри, що були обрані для того, щоб мережа могла вписатися в один GPU. Ми використовуємо імпульс SGD для оптимізації ваг w, причому початкова швидкість навчання ηw = 0,025 (віджигається до нуля за косинусним графіком), імпульс 0,9 і розпад ваги 3 × 10−4. Ми використовуємо Адама як оптимізатор для архітектурнихзмінних (α в обох нормальних і редукційних клітинах), з початковою швидкістю навчання ηα = 3 × 10−4, імпульсом β = (0.5, 0.999) і ваговим спадом 10−3. Пошук займає один день з одним тільки GPU.

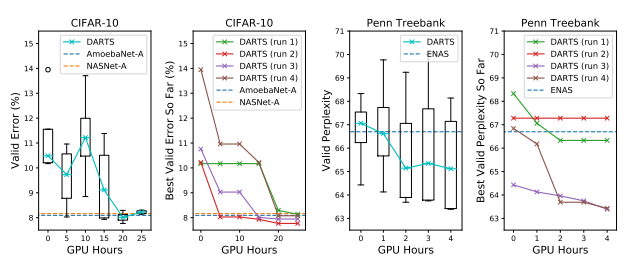


Figure 1 Пошук прогресу ДАРТС для згорткових клітин на CIFAR-10 і рекурентних клітинах на Penn Treebank. Ми відстежуємо останні архітектури з часом. Кожен знімок архітектури повторно тренується з нуля за допомогою навчального набору (для 100 епох на CIFAR-10 і для 300 епох на PTB) і потім оцінюється на наборі перевірок. Для кожного завдання ми повторюємо експерименти 4 рази з різними випадковими насінням, і повідомляємо про медіану і найкращу (за прогон) перевірку продуктивності архітектур з плином часу. В якості посилань ми також повідомляємо про результати (при тій же самій системі оцінювання; зі порівнянною кількістю параметрів) кращих існуючих осередків, виявлених за допомогою RL або еволюції, включаючи NASNet-A (Zoph et al., 2017) (1800 GPU днів), AmoebaNet-A (3150 ГПу днів) (Real et al., 2018) і ENAS (0.5 ДПУ в день) (Pham et al., 2018b).

## Пошук повторюваних осередків на Penn Treebank

Наш набір доступних операцій включає в себе особливу нульову операцію, а також лінійні перетворення, за якими слідують tanh, relu, sigmoid і ідентифікаційне відображення, відповідно. Вибір цих операцій-кандидатів здійснюється за Zoph and Le (2016); Pham et al. (2018b). Наша рекурентна клітина складається з N = 12 вузлів. Самий перший проміжний вузол отримують шляхом лінійного перетворення двох вхідних вузлів, підсумовування результатів і потім проходження через функцію активації tanh, як це зроблено в клітині ENAS (Pham et al., 2018b). Решту клітини вивчають. Інші налаштування подібні до ENAS, де кожна операція посилюється обхідним шосе (Zilly et al., 2016), а вихід клітини визначається як середнє значення всіх проміжних вузлів. Як і в ENAS, ми дозволяємо нормалізації партії в кожному вузлі, щоб запобігти вибуху градієнта під час пошуку архітектури, і відключити його під час оцінки архітектури. Вивчені афінні параметри в пакетній нормалізації відключені, як ми зробили для згорткових клітин. Наша повторювана мережа складається тільки з однієї клітини. А саме, ми не беремо на себе жодних повторюваних шаблонів в архітектурі, вертикально укладаючи клітини. Для пошуку архітектури, як вбудовування, так і приховані розміри встановлюються рівним 300. Параметри лінійних перетворень для всіх операцій-кандидатів на одному і тому ж краю є спільними (їхні форми мають розмір 300 × 300). Це дозволяє нам відповідати безперервній архітектурі в межах одного GPU. Потім мережу навчають на 50 епох, використовуючи SGD без імпульсу, зі швидкістю навчання ηw = 20, розміром партії 256, довжиною BPTT 35 і вагою 5 × 10−7. Ми застосовуємо варіаційне відсівання (Gal та Ghahramani, 2016) від 0,2 до вкладених слів, 0,75 до входу клітини і 0,25 до всіх прихованих вузлів. Відсіч 0,75 також застосовується до вихідного шару. Інші налаштування навчання ідентичні тим, що були в Merity et al. (2017); Yang et al. (2017). Подібно до сверточних архітектур, ми використовуємо Адама для оптимізації α, з початковою швидкістю навчання ηα = 3 × 10−3, імпульсом β = (0.9, 0.999) і ваговим розпадом 10−3. Пошук займає 6 годин на одному GPU.