# Аннотація

Ми пропонуємо пошук ефективної нейронної архітектури (ENAS), швидкий і недорогий підхід до автоматичного моделювання. В ENAS контролер виявляє архітектури нейронних мереж шляхом пошуку оптимального підграфа в межах великого обчислювального графа. Контролер навчається з алгоритмом градієнта для вибору підграфа, який максимізує очікувану винагороду на перевірочних даних. Тим часом модель, що відповідає обраному підграфі, навчається з метою мінімізації канонічної поперечної втрати ентропії. Обмін параметрами між дитячими моделями дозволяють ENAS доставляти сильні емпіричні показники, використовуючи набагато менше GPU, ніж існуючі підходи до автоматичного моделювання, і, зокрема, у 1000 разів дешевше, ніж стандартний пошук нейронної архітектури. На базі даних Penn Treebank, ENAS виявляє нову архітектуру, яка досягає випробувального розгубленості 55.8, встановлюючи нове, сучасне серед всіх методів без обробки після навчання. На наборі даних CIFAR-10, ENAS знаходить нову архітектуру, яка досягає 2.89% похибки випробування, що відповідає 2.65% випробувальної похибки NASNet (Zoph et al., 2018).

# Вступ

Незважаючи на вражаючу емпіричну продуктивність, пошук архітектури є обчислювально дорогим і трудомістким, наприклад. Zoph et al. (2018) використовують 450 графічних процесорів протягом 3-4 днів (тобто 32 400-43 200 GPU годин). Між тим, використання менших ресурсів має тенденцію до отримання менш переконливих результатів (Negrinho & Gordon, 2017; Baker et al., 2017a). Ми спостерігаємо, що обчислювальне вузьке місце NAS є підготовкою кожної дитини до збіжності, а лише для вимірювання її точності при відкиданні всіх тренованих ваг. Основний внесок цієї роботи полягає в підвищенні ефективності NAS, змушуючи всі дочірні моделі обмінюватися вагами, уникаючи підготовки кожної моделі дитини з нуля до збіжності. Ідея має очевидні ускладнення, оскільки різні моделі дітей можуть використовувати свої ваги по-різному, але була заохочена попередньою роботою з transfer learning та multitask learning, яка встановила, що параметри, отримані для конкретної моделі, можуть бути використані для інших моделей на інших завдань, з невеликими або відсутніми модифікаціями (Razavian et al., 2014; Zoph et al., 2016; Luong et al., 2016). Ми емпірично показуємо, що не тільки обмін параметрами між дитячими моделями можливий, але він також встановлює дуже сильні показники. Зокрема, на CIFAR-10, наш метод досягає помилки випробування 2,89%, порівняно з 2,65% по NAS. На Penn Treebank, наш метод досягає тестового розгубленості 55.8, що значно перевершує тестовий розгубленість NAS від 62.4 (Zoph & Le, 2017) і є новим сучасним серед підходів Penn Treebank, які не використовують пост-обробку навчання. Важливо відзначити, що у всіх наших експериментах, для яких ми використовуємо один графічний процесор Nvidia GTX 1080Ti, пошук архітектури займає менше 16 годин. У порівнянні з NAS це скорочення GPU-годин більш ніж у 1000 разів. Завдяки своїй ефективності, ми називаємо наш метод Ефективний пошук нейронної архітектури (ENAS).

# Методи

Центральним елементом ідеї ENAS є спостереження, що всі графіки закінчують ітеруватись, можна розглядати як підграфи більшого графіка. Іншими словами, ми можемо представляти пошуковий простір NAS за допомогою одного спрямованого ациклічного графа (DAG). Рисунок 2 ілюструє загальний приклад DAG, де архітектуру можна реалізувати, взявши підграф DAG. Інтуїтивно, DAG ENAS є суперпозицією всіх можливих дочірніх моделей у просторі пошуку NAS, де вузли являють собою локальні обчислення, а ребра представляють потік інформації. Локальні обчислення на кожному вузлі мають свої власні параметри, які використовуються тільки тоді, коли активується конкретне обчислення. Таким чином, конструкція ENAS дозволяє розподіляти параметри між усіма дочірніми моделями, тобто архітектурами, у просторі пошуку. Нижче ми обговоримо ENAS з прикладом, який ілюструє, як створити осередок для повторюваних нейронних мереж з певної DAG і контролера (розділ 2.1). Потім ми пояснимо, як навчати ENAS і як отримувати архітектури від контролера ENAS (розділ 2.2). Нарешті, ми пояснимо наш пошуковий простір для проектування згорткових архітектур (розділи 2.3 і 2.4).

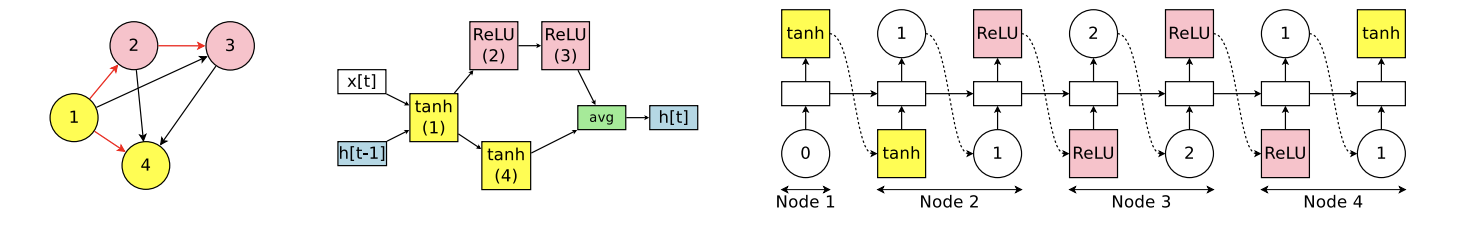


Рисунок 1- Приклад рекурентної клітки в нашому пошуковому просторі з 4 обчислювальними вузлами. Ліворуч: обчислювальна DAG, що відповідає повторюваної комірки. Червоні ребра являють собою потік інформації на графіку. Середній: рецидивна клітина. Праворуч: Виходи контролера RNN, які призводять до осередку в середині і DAG ліворуч. Зауважимо, що вузли 3 і 4 ніколи не є вибірковими за допомогою RNN, тому їх результати усереднюються і розглядаються як вихід клітини.

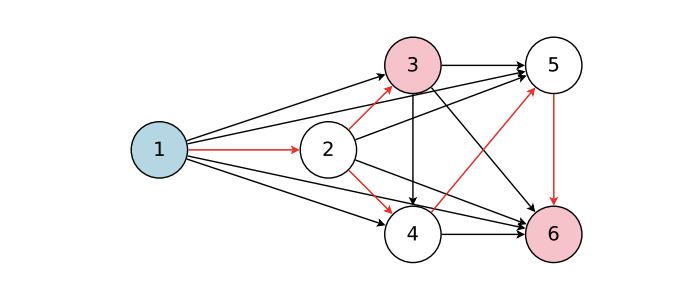


Рисунок 2 - Графік відображає весь простір пошуку, червоні стрілки визначають модель в просторі пошуку, яку вирішує контролер. Тут вузол1 є входом в модель, тоді як вузли 3 і 6 є виходами моделі

# Проектування рекурентної клітини

Для проектування повторюваних клітин ми використовуємо DAG з N вузлами, де вузли являють собою локальні обчислення, а ребра представляють потік інформації між N вузлами. Контролер ENAS є RNN, який вирішує: 1) які ребра активовані і 2) які обчислення виконуються на кожному вузлі DAG. Цей дизайн нашого простору пошуку для осередків RNN відрізняється від простору пошуку для осередків RNN у Zoph & Le (2017), де автори фіксують топологію своїх архітектур як бінарне дерево і тільки вивчають операції в кожному вузлі дерева. На відміну від цього, наш пошуковий простір дозволяє ENAS проектувати як топологію, так і операції в осередках RNN, а отже, і більш гнучку. Для створення повторюваної комірки контролер RNN відбирає N блоків рішень. Тут ми ілюструємо механізм ENAS через простий приклад рекурентної комірки з N = 4 обчислювальними вузлами (візуалізується на малюнку 1). Нехай xt є вхідним сигналом для повторюваної комірки (наприклад, вбудовування слова), а ht − 1 є виходом з попереднього етапу часу. Ми пробуємо наступним чином.

1. У вузлі 1: Контролер спочатку вибирає функцію активації. У нашому прикладі контролер вибирає функцію активації tanh, тобто вузол 1 повторюваної комірки повинен обчислити

h1 = tanh (xt · W (x) + ht − 1 · W (h) 1).

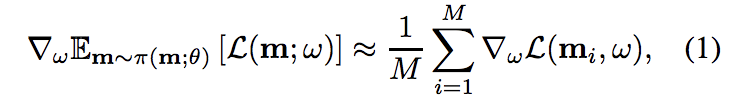
1. У вузлі 2: Контролер потім відбирає попередній індекс і функцію активації. У нашому прикладі він вибирає попередній індекс 1 і функцію активації ReLU. Таким чином, вузол 2 комірки обчислює h2 = ReLU (h1 · W (h) 2,1)
2. У вузлі 3: Контролер знову вибирає попередній індекс і функцію активації. У нашому прикладі він вибирає попередній індекс 2 і функцію активації ReLU. Отже, h3 = ReLU (h2 · W (h) 3,2).
3. У вузлі 4: Контролер знову вибирає попередній індекс і функцію активації. У нашому прикладі він вибирає попередній індекс 1 і функцію активації tanh, що призводить до h4 = tanh (h1 · W (h) 4,1).
4. Для виводу ми просто усереднюємо всі вільні кінці, тобто вузли, які не вибрані як входи для будь-яких інших вузлів. У нашому прикладі, оскільки індекси 3 і 4 ніколи не були вибірковими для входу для будь-якого вузла, повторювана клітина використовує своє середнє значення (h3 + h4) / 2 як вихідний. Іншими словами, ht = (h3 + h4) / 2.

У наведеному вище прикладі відзначимо, що для кожної пари вузлів j <ℓ існує незалежна матриця параметрів W (h) ℓ, j. Як показано в прикладі, вибираючи попередні індекси, контролер також вирішує, які матриці параметрів використовуються. Тому в ENAS всі повторювані клітинки в просторі пошуку поділяють один і той же набір параметрів. Наш пошуковий простір включає експоненційне число конфігурацій. Зокрема, якщо рекурентна клітина має N вузлів і ми допускаємо 4 функції активації (а саме tanh, ReLU, ідентичність і сигмоїда), то простір пошуку має 4 N × N! конфігурацій. У наших експериментах N = 12, що означає, що в нашому пошуковому просторі є приблизно 1015 моделей.

# Навчання ENAS і Deriving Architectures

Наша мережа контролерів - це LSTM зі 100 прихованими одиницями (Hochreiter & Schmidhuber, 1997). Це LSTM зразки рішень через softmax класифікаторів, в авторегресії мода: рішення на попередньому кроці подається в якості вхідного вбудовування в наступному кроці. На першому кроці мережа контролера отримує порожнє вбудовування в якості вхідних даних. В ENAS існує два набори ознак, які можна дізнатися: параметри контролера LSTM, позначені θ, і спільні параметри дочірніх моделей, позначені ω. Процедура навчання ENAS складається з двох фаз перемежування. Перша фаза тренується ω, спільні параметри дочірніх моделей, в цілому проходять через набір навчальних даних. Для наших експериментів Penn Treebank, ω тренується близько 400 кроків, кожен на мініатюрі з 64 прикладів, де градієнт isω обчислюється з використанням зворотного поширення через час, усіченого на 35 тимчасових кроках. Між тим, для CIFAR-10, ω навчається на 45 000 тренувальних зображеннях, розділених на мініпатчі розміром 128, де ∇ω обчислюється з використанням стандартного зворотного поширення. Друга фаза поїздів θ, параметри контролера LSTM, для фіксованого числа кроків, як правило, встановлюється в 2000 році в наших експериментах. Ці дві фази чергуються під час навчання ENAS. Більш детальна інформація наступна.

Навчання спільним параметрам ω дитячих моделей. На цьому кроці ми фіксуємо політику контролера π (m; θ) і виконуємо стохастичний спуск градієнта (SGD) на ω, щоб мінімізувати функцію очікуваних втрат Em∼π [L (m; ω)]. Тут L (m; ω) - стандартна втрата крос-ентропії, розрахована на мініатюрі навчальних даних, з моделлю m вибірки з π (m; θ). Градієнт обчислюється з використанням оцінки Монте-Карло

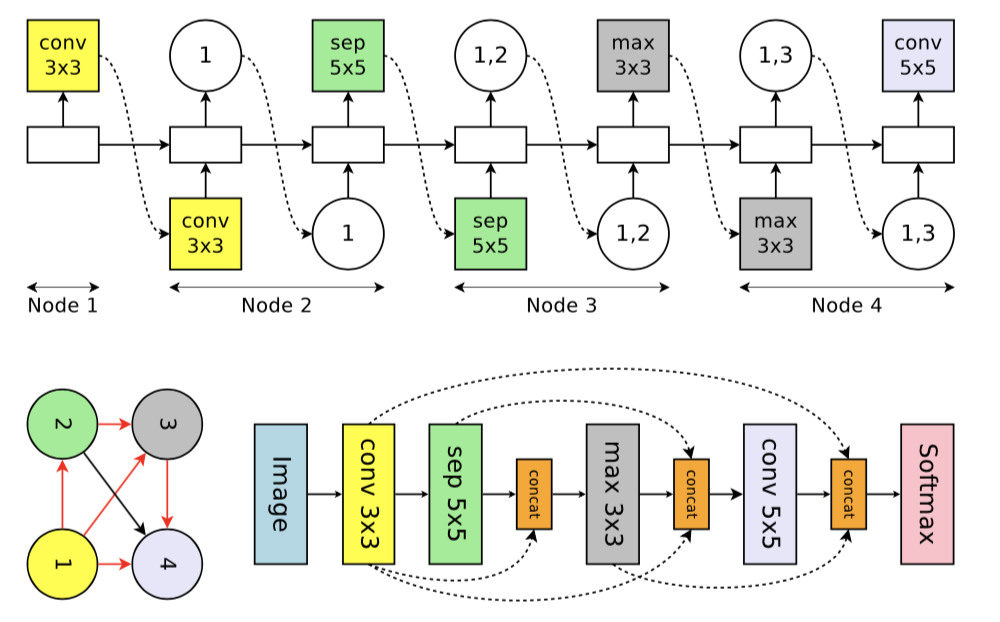


де mi's відбираються з π (m; θ), як описано вище. Рівняння 1 дає незміщену оцінку градієнта EωEm∼π (m; θ) [L (m; ω)]. Однак ця оцінка має більш високу дисперсію, ніж стандартний градієнт SGD, де m є фіксованим. Проте - і це, мабуть, дивно - ми виявили, що M = 1 працює просто чудово, тобто ми можемо оновити ω, використовуючи градієнт від будь-якої окремої моделі m, вибірки з π (m; θ). Як вже згадувалося, ми тренуємо ω протягом всього проходу через тренувальні дані.

Навчання параметрів контролера θ. На цьому кроці фіксуємо ω і оновлюємо параметри політики θ, прагнучи максимізувати очікувану винагороду Emπ (m; θ) [R (m, ω)]. Ми використовуємо оптимізатора Адама (Kingma & Ba, 2015), для якого градієнт обчислюється за допомогою REINFORCE (Williams, 1992), з базовою лінією ковзних середніх для зменшення дисперсії. Винагорода R (m, ω) обчислюється на наборі валідації, а не на навчальному наборі, щоб заохотити ENAS до вибору моделей, які добре узагальнюють, а не моделей, які переповнюють навчальний набір. У нашому експерименті мовної моделі функція винагороди є c / valid ppl, де розрахунки обчислюються на мініатюрі даних перевірки. У наших експериментах з класифікації зображень функцією винагороди є точність мініатюри зображень перевірки.

Виведення архітектур. Ми обговорюємо, як вивести нові архітектури з підготовленої моделі ENAS. Спочатку проаналізуємо кілька моделей з навченої політики π (m, θ). Для кожної вибіркової моделі ми обчислюємо її винагороду за однією мініатюрою, взятою з набору перевірок. Потім ми приймаємо тільки модель з найвищою нагородою, щоб перенавчатись з нуля. Можна поліпшити наші експериментальні результати, тренуючи всі зразкові моделі з нуля і вибравши модель з найвищою продуктивністю на окремому наборі перевірок, як це роблять інші роботи (Zoph & Le, 2017; Zoph et al., 2018; Liu et al., 2017; 2018). Однак наш метод дає подібні показники, хоча є набагато більш економічним.

# Проектування згорткових нейронних мереж



3. Приклад виконання повторюваної комірки в нашому пошуковому просторі з 4 обчислювальними вузлами, які представляють 4 шари в об'ємній мережі. Зверху: Вихід контролера RNN. Вліво ліворуч: обчислювальний DAG, що відповідає архітектурі мережі. Червоні стрілки позначають активні обчислювальні шляхи. Внизу справа: повна мережа. Пунктирними стрілками позначені пропуски з'єднань.

Розглянемо тепер пошуковий простір для згорткових архітектур. Нагадаємо, що в просторі пошуку повторюваної комірки контролер RNN відбирає два рішення на кожному блоці прийняття рішення: 1) який попередній вузол підключити до і 2) яку функцію активації використовувати. У просторі пошуку для згорткових моделей контролер RNN також відбирає два набори рішень у кожному блоці прийняття рішень: 1) до яких попередніх вузлів підключаються і 2) які операції обчислення використовувати. Ці рішення будують шар у згортковій моделі. Рішення про те, які попередні вузли підключатимуться, дозволяє моделі формувати пропущені з'єднання (He et al., 2016a; Zoph & Le, 2017). Зокрема, на шарі k, до k − 1 взаємно відмінні попередні індекси, що призводить до 2k − 1 можливих рішень на рівні k. Наведемо ілюстративний приклад вибірки згорткової мережі на малюнку 3. У цьому прикладі, при шарі k = 4, контролер відбирає попередні індекси {1, 3}, так що виходи шарів 1 і 3 об'єднуються уздовж. Їх розмірність глибини і передається на рівень 4. Тим часом, рішення про те, яку операцію обчислення слід використовувати, встановлює певний шар у згортку або середнє пул або макс. Шість операцій, доступних для контролера: звивини з розмірами фільтрів 3 × 3 і 5 × 5, глибинно-розділові звивини з розмірами фільтрів 3 × 3 і 5 × 5 (Chollet, 2017), а також максимальне об'єднання та середнє об'єднання розмір ядра 3 × 3. Що стосується рекурентних клітин, то кожна операція на кожному шарі нашої згорткової мережі ENAS має окремий набір параметрів. Роблячи описаний набір рішень на загальну кількість разів L, ми можемо проаналізувати мережу L-шарів. Оскільки всі рішення є незалежними, у просторі пошуку існують мережі 6L × 2L (L-1) / 2. У наших експериментах L = 12, в результаті чого в 1,6 × 10^29 можливі мережі.

## Проектування сверточних клітин

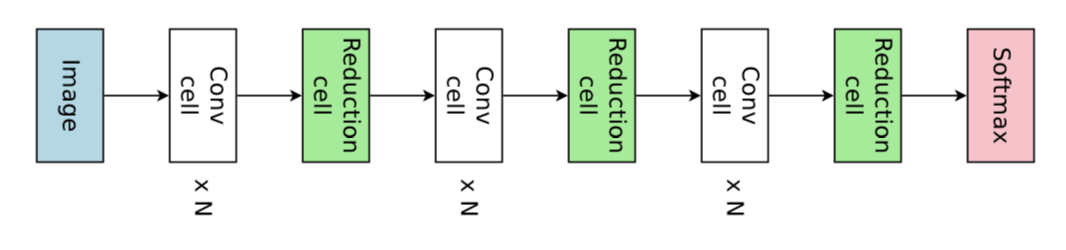


Рисунок 3 Підключення 3 блоків, кожен з N клітин згортки і 1 відновлювальної комірки, для створення кінцевої мережі.

Ми використовуємо обчислювальну DAG ENAS з B вузлами для представлення обчислень, які відбуваються локально в комірці. У цьому DAG вузол 1 і вузол 2 розглядаються як входи комірки, які є виходами двох попередніх осередків в кінцевій мережі (див. Фіг.4). Для кожного з решти B - 2 вузлів ми просимо контролера RNN зробити два набори рішень: 1) два попередніх вузла, які будуть використовуватися як вхідні дані для поточного вузла, і 2) дві операції, які будуть застосовані до двох вибірок. 5 доступних операцій: ідентичність, відокремлювана згортка з розміром ядра 3 × 3 і 5 × 5, і середнє об'єднання і максимальне об'єднання з розміром ядра 3 × 3. На кожному вузлі, після попередніх вузлів і їх відповідних операцій, відбираються операції застосовуються до попередніх вузлів, і їх результати додаються.

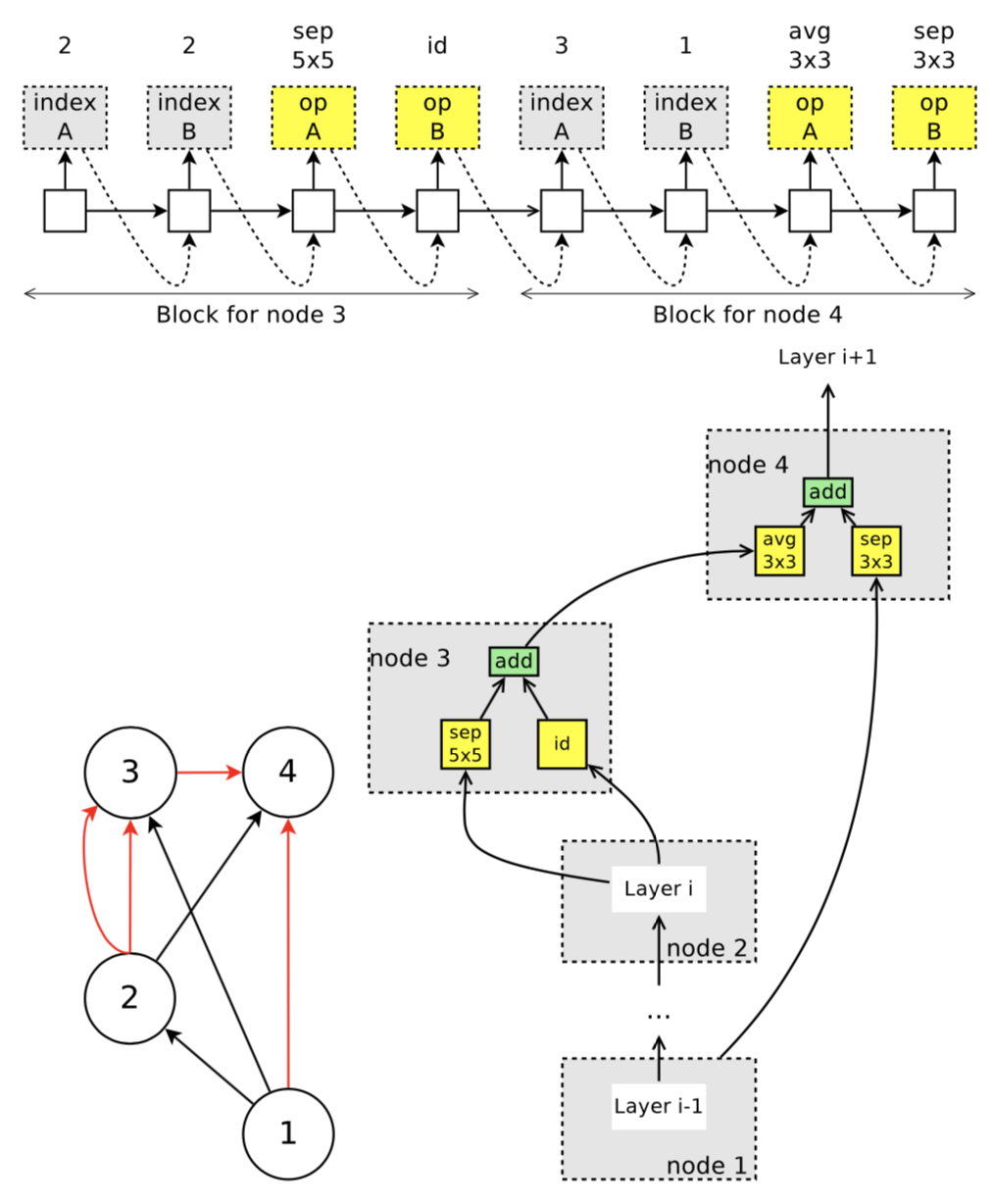


Рисунок 5 Приклад виконання контролера для нашого простору пошуку над згортковими осередками. Зверху: виходи контролера. У нашому пошуковому просторі для згорткових комірок, вузол 1 і вузол 2 є входами комірки, тому контролер має тільки розробити вузол 3 і вузол 4. Лівий нижній: відповідний DAG, де червоні ребра представляють активовані з'єднання. Внизу справа: згорткова комірка відповідно до вибірки контролера.

Як і раніше, ми ілюструємо механізм нашого пошукового простору на прикладі, тут з B = 4 вузлами (див. Рис. 5). Деталі наведені нижче.

1. Вузли 1, 2 є вхідними вузлами, тому для них не потрібні рішення. Нехай h1, h2 - виходи цих вузлів.

2. У вузлі 3: контролер зразків двох попередніх вузлів і двох операцій. На малюнку 5 зверху ліворуч - зразки вузла 2, вузла 2, відокремлюваного перекладу 5x5 і ідентичності. Це означає, що h3 = sep conv 5x5 (h2) + id (h2). 3. У вузлі 4: контролер зразків вузла 3, вузла 1, ср пул 3х3 і сеп-конв 3х3. Це означає, що h4 = avg pool 3x3 (h3) + sep conv 3x3 (h1).

4. Оскільки всі вузли, крім h4, використовувалися як вхідні дані для принаймні іншого вузла, єдиний вільний кінець, h4, розглядається як вихід клітини. Якщо є декілька вільних кінців, вони будуть зв'язані уздовж розмірності глибини для формування виходу клітини.

Редукційна комірка також може бути реалізована з простору пошуку, який ми обговорювали, просто шляхом: 1) вибірки обчислювального графа з простору пошуку, і 2) застосування всіх операцій зі ступенем 2. його вхідних даних у 2 рази. (2018), ми відібрали редукційну клітину, сконструйовану на згортковій комірці, і, таким чином, контролер RNN працював на загальну кількість блоків 2 (B - 2). Нарешті, ми оцінюємо складність цього простору пошуку. У вузлі i (3 ≤ i ≤ B) контролер може вибрати будь-які два вузли з i - 1 попередніх вузлів і будь-які дві операції з 5 операцій. Оскільки всі рішення є незалежними, існують (5 × (B - 2)!) 2 можливі клітини. Оскільки ми незалежно вибираємо для згорткової клітинки та редукційної комірки, кінцевим розміром простору пошуку є (5 × (B - 2)!) 4. При B = 7, як і в наших експериментах, пошуковий простір може реалізувати 1,3 × 1011 кінцевих мереж, що робить його значно меншим, ніж простір пошуку для цілих згорткових мереж (розділ 2.3).