Системи штучного інтелекту

Чернетка конспекту лекцій

16 вересня 2022 р.

Матеріал конспекту

16 вересня 2022 р.

Зміст

Частина I
Теорія

1 Мотивація та інтуїція

Як і в більшості навчальних завдань, вибір хороших атрибутів має першочегове значення. Навіть найкращий алгоритм не зможе компенсувати вибір поганих атрибутів для навчання. З іншого боку, вибір атрибутів часто є вкрай нетривіальним завданням. Наприклад, розглянемо кольорове зображення особи, в якому випадкові вихідні дані складаються із значень інтенсивності трьох кольорів: «Червоного», «Зеленого» та «Синього» (якщо ми використовуємо RGB-кодування). Тому, 1M піксельне зображення людини має 3M атрибутів. З іншого боку, вираз людини, ймовірно, можна охарактеризувати з < 56 ознак (на обличчі людини є ~ 56 м'язи). Отже, для ідентифікації конкретних виразів обличчя, дані великого розміру можуть бути представлені відповідними атрибумати атрибутами малого розміру.

Таким чином, ідеальний екстрактор ознак буде приймати у якості даних - зображення облич, а потім автоматично визначати їх ознаки для нас. Однак, на теперешній час, не існує ідеальних способів це зробити. У традиційних методах розпізнавання даних, розроблених з 50-х років, ці екстрактори ознак були жорстко закодовані на основі суб'єктивної інтуїції дослідників. Основна ідея, принесена світом епохи нейронних мереж, полягає в тому, що хороші властивості можна дізнатись з даних.

З іншого боку, базова модель не сильно розвинулася з часу її створення в 1950-х роках. Перша машина для навчання, названа персептроном, була лінійним класифікатором, побудованим поверх простого жорстко прописаного екстрактора ознак. Тепер практичні програми машинного навчання використовують деякі прославлені добре налаштовані лінійні класифікатори або відповідність шаблону.

Звичайно, якби поєднувались лише лінійні шари, сукупний ефект також був би лінійним і ми могли б згорнути всю архітектуру лише в один шар. Це пояснюється тим, що результат лінійного перетворення (матриці) залишається лінійним перетворенням. З іншого боку, введення нелінійності робить природним використання мереж з декількома шарами, оскільки їх неможливо лінійно поєднати.

Загалом, у цих примітках ми будемо розглядати машини контрольованого навчання на основі градієнта, де результат для вхідного $\boldsymbol{x}$ подається функцією:

$$\hat{\boldsymbol{y}} = F(\boldsymbol{W}, \boldsymbol{x}),$$

де, W - деякий вектор/матриця та F - деяка (можливо нелінійна) функція аргументів. Якість машин визначається на основі деякой функції втрати виду:

$$L(\boldsymbol{W}, \boldsymbol{y}, \boldsymbol{x}) = D(\boldsymbol{y}, F(\boldsymbol{W}, \boldsymbol{x})),$$

де, $D(\boldsymbol{y}, \hat{\boldsymbol{y}})$ є деякою мірою невідповідності між істинною величиною $\boldsymbol{y}$ та оціночною величиною $\hat{\boldsymbol{y}}$. Наша мета - знайти ваги, які мінімізують цю функцію втрат. Зробивши початкову здогадку, ми дивимося на дані та оновлюємо ваги в «найкращому напрямку», тобто обчислюємо градієнт:

$$\frac{\partial L(\boldsymbol{W}, \boldsymbol{y}, \boldsymbol{x})}{\partial W} = \frac{\partial D(\boldsymbol{y}, F(\boldsymbol{W}, \boldsymbol{x}))}{\partial \hat{\boldsymbol{y}}} \frac{\partial F(\boldsymbol{W}, \boldsymbol{x})}{\partial \boldsymbol{W}},$$

який є напрямом найкрутішого підйому, а потім робимо крок у зворотному напрямку:

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) - \eta(t)\frac{\partial D(\boldsymbol{y}, F(\boldsymbol{W}, \boldsymbol{x}))}{\partial \hat{\boldsymbol{y}}}\frac{\partial F(\boldsymbol{W}, \boldsymbol{x})}{\partial \boldsymbol{W}} \tag{1}$$

з деяким відповідно вибраним кроком $\eta(t)$. Залежно від того, скільки прикладів ми використовуємо для цього оновлення, ми отримуємо різні алгоритми (до цього ми повернемося пізніше).

З цього шаблону виникають три запитання: (1) Яку архітектуру $F(\boldsymbol{W}, \boldsymbol{x})$ ми повині використовувати?; (2) Яку функцію втрат $L(\boldsymbol{W}, \boldsymbol{y}, \boldsymbol{x})$ ми повинні використовувати?; (3) Чи варто використовувати простий крок градієнта (1) або якийсь більш досконалий алгоритм оптимізації?

1.1 Огляд Контрольованого Навчання

Термін контрольоване навчання походить від ідеї інструктора, який тренує машину для виконання якогось завдання, надаючи їй реальну відповідь, після того, як вона обчислює свою власну, щоб не допустити, щоб машина знову зробила помилку. Отже, ідеєю контрольованого навчання полягає в наступному: на основі прикладів вхідних даних та пов'язаних з ними результатів, потрібно вивчити правило прийняття рішень, яке дає найкращі показники відповідно до метрики, обраної для їх оцінки. Але що ми маємо на увазі під терміном «навчання»? Мітчелл (1997) наводить загальне його визначення в обчислювальній техніці: «Комп'ютер, як кажуть, вчиться на досвіді E стосовно певного класу завдань T та міри ефективності P, якщо продуктивність завдання в T, виміряна P, покращується з досвідом E». У нашому контексті завданням буде, наприклад, класифікація або регресія, показник ефективності буде посилатися на метрику, яку ми виберемо, а досвід відповідатиме набору точок даних, з яких нам доведеться вчитися.

За допомогою цієї ідеї було побудовано багато керованих алгоритмів навчання. Усі вони мають набір даних, що містить особливості, і для кожного з яких приклад x також пов'язаний зі змінною (або цільовою метою) y. Наприклад, MNIST - це такий набір даних, де зображення цифр пов'язані з цифрою, яку вони представляють.

Імовірнісне контрольоване навчання. Більшість алгоритмів засновані на оцінці розподілу ймовірностей $p(y|x)$. Найбільш поширеною стратегією для досягнення цього є спочатку параметризація проблеми (завдання), в результаті чого вона перетворюється на оцінку $p(y|x, \theta)$, а потім відбувається пошук оптимального параметра θ шляхом максимізації ймовірності (або часткової вірогідності для зручності) з використанням градієнтного спуску. Це стосується, наприклад, лінійної регресії та логістичної регресії. Обивда вони більш детально пояснюються в наступних розділах.

Метод опорних векторів. Оскільки це одна з найвпливовіших моделей контрольованого навчання, ми детально її пояснимо. Ця модель складається з пошуку найкращих розділювальних гіперплощин для наших даних. На відміну від раніше згаданих моделей, машина підтримки вектора не надає ймовірностей для оцінки ідентичності класу. Дійсно, ідентичність класу визначається лише там, де знаходиться точка даних щодо розділюючої гіперплощини. Математично це відповідає $w^\top x+b$, що може бути позитивним або негативним, де w - вектор ваг, пов'язаних з кожною ознакою, і b зміщення. Дійсно, w, будучи ортогональним розділювальній гіперплощині, точки, що належать до позитивної сторони гіперплощини - це точки з позитивним внутрішнім добутком з w, а з негативним внутрішнім добутком для інших. Тепер потужність векторних машин підтримки походить від **трюку ядра**. Щоб пояснити трюк, потрібно ввести теорему Представника:

$$\exists(\alpha_i)_{i=1..n},\ w = \sum_{i=1}^{n} \alpha_i x_i \text{ де, } x_i = \text{і-ий приклад}$$

Ми можемо побачити цю теорему в дії, коли виконуємо градієнтний спуск із залежністю втрат як цільовою функцією, оновлення відповідають або 0, або $-y_i x_i$ до конвергенції. Отже, врешті-решт, отримується w як лінійна комбінація $(x_i)_{i=1..n}$. Виходячи з цього ми отримуємо:

$$\exists(\alpha_i)_{i=1..n},\ w^\top x + b = b + \sum_{i=1}^{n} \alpha_i x^\top x_i$$

Тепер використання функцій ядра для отримання додаткових функцій може забезпечити певні переваги, оскільки дає додаткову інформацію. Припустимо, ми вибрали для цього функцію ядра ϕ, тоді нашими «новими» точками даних є $\phi(x)$. Якщо розмір $\phi(x_i)$ величезний, внутрішні результати в сумі попередніх обрахунків буде дуже дорого обчислювати. Трюк ядра дозволяє зробити ці обчислення набагато менш витратними: це відповідає тому факту, що для кожного вибору ядра ϕ завжди існує білінійна форма k така, що $k(x, x_i) = \phi(x)^\top \phi(x_i)$ і це відповідає функції, обчисленій у набагато менш розмірному просторі!

k-найближчі сусіди. Ця модель може бути використана для класифікації та регресії. Є непараметричною і працює в будь-яких ситуаціях, коли ми можемо визначити середнє значення за мітками. Модель визначає ідентичність класу кожної точки даних від її k-найближчих сусідів.

Дерево рішень. Ця модель полягає у розбитті вхідних даних на різні ділянки та визначенні мети кожної точки даних, розглядаючи, в яку область вона потрапляє. На кожному кроці побудови дерева, модель спочатку знаходить, за яким об'єктом слід зробити розбиття, а потім за цією вибраною ознакою визначає найкраще розбиття, використовуючи метрику «Прибуття інформації».

Вироджені види систем управління підкріпленням («вчителів»). Система підкріплення з керуванням по реакції (R — керована система) — характеризується тим, що інформаційний канал від зовнішнього середовища до системи підкріплення не функціонує. Дана система, незважаючи на наявність системи управління, відноситься до спонтанного навчання, оскільки випробовувана система навчається автономно, під дією лише своїх вихідних сигналів незалежно від їх «правильності». При такому методі навчання для управління зміною стану пам'яті не потрібно ніякої зовнішньої інформації;

Система підкріплення з керуванням по стимулах (S — керована система) — характеризується тим, що інформаційний канал від випробовуваної системи до системи підкріплення не функціонує. Незважаючи на не функціонування каналу від виходів випробовуваної системи, відноситься до навчання з учителем, оскільки в цьому випадку система підкріплення (вчитель) змушує випробувану систему виробляти реакції згідно певного правила, хоча й не береться до уваги наявність істинних реакцій випробовуваної системи.

Дана відмінність дозволяє глибше поглянути на відмінності між різними способами навчання, оскільки грань між навчанням з учителем і навчанням без вчителя тонша. Крім цього, таке розходження дозволило показати для штучних нейронних мереж певні обмеження для S та R — керованих систем (Теорема збіжності перцептрону).

1.2 Лінійні машини та їх обмеження

Щоб пояснити потребу в нелінійності та глибоких архітектурах, коротко нагадаємо класичні лінійні методи регресії та класифікації. Припускаємо, що кожна вибірка у вхідних даних

$$\mathcal{D} = (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})$$

містить k ознак та q залежних змінних.

1.2.1 Найменші квадрати лінійної регресії

Найбільш часто використовуваний статистичний метод - це лінійна регресія найменших квадратів. У цій обстановці наша мета - знайти ваги $\boldsymbol{W}$ такі, що

$$\boldsymbol{y}^{(i)} = \boldsymbol{W}\boldsymbol{x}^{(i)}.$$

якомога ближчі (ми приймаємо конвенцію, що терміни упередженості включаються, маючи ідентично одну особливість). Ця близькість на i-й вибірці вимірюється функцією квадратних втрат

$$L(\boldsymbol{W}, \boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)}) = \frac{1}{2}\left\|\boldsymbol{y}^{(i)} - \boldsymbol{W}\boldsymbol{x}^{(i)}\right\|_{l^2}^2.$$

Градієнт цієї функції втрат задається формулою:

$$\frac{\partial L(\boldsymbol{W}, \boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)})}{\partial W} = -(\boldsymbol{x}^{(i)})^T(\boldsymbol{y}^{(i)} - \boldsymbol{W}\boldsymbol{x}^{(i)})$$

Це означає, що, подивившись i-ту вибірку на t-му кроці спостереження, ми повинні змінити матрицю ваги відповідно до правила:

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) + \eta(t)(\boldsymbol{x}^{(i)})^T(\boldsymbol{y}^{(i)} - \boldsymbol{W}\boldsymbol{x}^{(i)}),$$

де, $\eta(t)$ - відповідним чином обраний розмір кроку (швидкість навчання). Якщо ми виберемо точку спостереження даних $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$, тоді цей алгоритм буде називатися *Стохастичний градієнтний спуск* (СГС) в дискретний час. Звичайно, якщо ми маємо обчислювальну здатність розглядати всі зразки перед оновленням, ми можемо врахувати загальну квадратну втрату:

$$L(\boldsymbol{W}, \mathcal{D}) = \sum_{i=1}^{m} L(\boldsymbol{W}, \boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)})$$

Якщо організуємо ввід даних у вигляді матриці

$$\boldsymbol{X} = \begin{bmatrix} \text{——} & \boldsymbol{x}^{(1)} & \text{——} \\ \text{——} & \boldsymbol{x}^{(2)} & \text{——} \\ & \vdots & \\ \text{——} & \boldsymbol{x}^{(m)} & \text{——} \end{bmatrix}_{m \times k}$$

і аналогічно організуємо прогнози як

$$\boldsymbol{Y} = \begin{bmatrix} \text{——} & \boldsymbol{y}^{(1)} & \text{——} \\ \text{——} & \boldsymbol{y}^{(2)} & \text{——} \\ & \vdots & \\ \text{——} & \boldsymbol{y}^{(m)} & \text{——} \end{bmatrix}_{m \times q}$$

тоді зможемо записате це, як

$$L(\boldsymbol{W}, \mathcal{D}) = \frac{1}{2}\left\|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{W}^T\right\|_F^2,$$

де, $\|A\|_F = \sqrt{TrA^TA}$ - норма Фробеніуса. Тоді градієент задається

$$\frac{\partial L(\boldsymbol{W}, \mathcal{D})}{\partial \boldsymbol{W}} = -\boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{W}^T). \tag{2}$$

та алгоритм, що є результатом відповідного оновленого правила

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) + \eta(t)\boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{W}^T)$$

називається *Пакетний градієнтний спуск* (ПГС) чи просто *Градієнтний спуск*. Звичайно, можна піти на компроміс між двома алгоритмами і використати частину навчальних ділянок для оновлення, і в цьому випадку алгоритм називається мініатюрний градієнтний спуск.

У випадку ПГС існує явне алгебраїчне рівняння, яке називається *нормальне рівняння* для оптимальних ваг (мінімізація L) встановивши праву сторону (2) в нуль:

$$\boldsymbol{W}\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{Y}^T\boldsymbol{X}.$$

Якщо $\boldsymbol{X}^T\boldsymbol{X}$ є оборотним, це рівняння має унікальне рішення

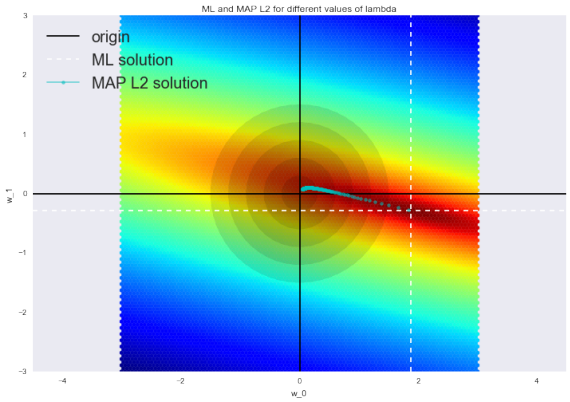
$$\boldsymbol{W} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{Y}^T\boldsymbol{X}.$$

1.2.2 Регуляризація лінійної регресії

Іноді бувають ситуації, коли ми навмисно збільшуємо зміщеність моделі заради її стабільності, тобто заради зменшення дисперсії моделі $\mathrm{Var}\left(\hat{f}\right)$. Однією з умов теореми Маркова-Гаусса є повний стовпчиковий ранг матриці X. В іншому випадку рішення $\vec{w} = \left(X^TX\right)^{-1}X^T\vec{y}$ не існує, тому що не існуватиме зворотна матриця $\left(X^TX\right)^{-1}$. Іншими словами, матриця X^T буде сингулярна, або виродження. Таке завдання називається некоректно поставленим. Завдання потрібно скорегувати, а саме, зробити матрицю X^TX невиродженою, або регулярною (саме тому цей процес називається регуляризацією). Найчастіше в даних ми можемо спостерігати так звану мультиколінеарность - коли дві або декілька ознак сильно корельовані, в матриці X це проявляється у вигляді «майже» лінійної залежності стовпців. Наприклад, в задачі прогнозування ціни квартири по її параметрам «майже» лінійна залежність буде у ознак «площа з урахуванням балкона» і «площа без урахування балкона». Формально для таких даних матриця X^T буде оборотна, але через мультиколлінеарность у матриці X^T деякі власні значення будуть близькі до нуля, а у зворотній матриці $\left(X^TX\right)^{-1}$ з'являється екстремально великі власні значення, тому що власні значення оберненої матриці - це $\frac{1}{\lambda_i}$. Підсумком такого хитання власних значень стане нестабільна оцінка параметрів моделі, тобто додавання нового спостереження в набір тренувальних даних призведе до зовсім іншого рішення. Одним із способів регуляризації є регуляризація Тихонова, яка в загальному вигляді виглядає як додавання нового члена до середньоквадратичної помилки:

$$\mathcal{L}\left(X, \vec{y}, \vec{w}\right) \quad = \quad \frac{1}{2n}\left\|\vec{y} - X\vec{w}\right\|_2^2 + \left\|\Gamma\vec{w}\right\|^2$$

Часто матриця Тихонова виражається як добуток деякого числа на одиничну матрицю: $\Gamma = \frac{\lambda}{2}$ E. У цьому випадку завдання мінімізації середньоквадратичної помилки стає завданням з обмеженням на L_2 норму. Якщо продиференціювати нову функцію вартості за параметрами



моделі, прирівняти отриману функцію до нуля і висловити $\vec{w}$, то ми отримаємо точне рішення задачі.

$$\vec{w} = \left(X^T X + \lambda E\right)^{-1} X^T \vec{y}$$

Така регресія називається гребеневою регресією (ridge regression). А гребенем є якраз діагональна матриця, яку ми додаємо до матриці X^T, в результаті виходить гарантовано регулярна матриця.

Таке рішення зменшує дисперсію, але стає зміщеним, тому що мінімізується також і норма вектора параметрів, що змушує рішення зрушуватися в бік нуля. На малюнку нижче на перетині білих пунктирних ліній знаходиться МНК-рішення. Блакитними точками позначені різні рішення гребеневої регресії. Видно, що при збільшенні параметра регуляризації λ рішення зсувається в бік нуля.

1.2.3 Перцептрон

Для того, щоб використовувати алгоритми регресії для багатокласової класифікації, нам потрібно розглядати їх результати як значення, присвоєні різним категоріям. Найпростіший спосіб перетворити ці значення на оцінки ймовірностей - це встановити їх пороговою величиною за допомогою якоїсь функції F, застосованої поступово: $F(\boldsymbol{W}\boldsymbol{x}^{(i)})$. Це означає, що ми оцінюємо ймовірність $x^{(i)}$, що належать до категорії q за q-ою координатою функції $F(\boldsymbol{W}\boldsymbol{x}^{(i)})$. Для цього завдання необхідно кодувати основне значення $\boldsymbol{y}^{(i)}$ як один вектор, що має значення:

$$(\boldsymbol{y}^{(i)})_j = \delta_{ij}$$

де, $\delta_{ij} = 1$ якщо $i = j$ та інші дорівнюють нулю. У цій обстановці наша мета - знайти такі ваги, що задовольняють

$$\boldsymbol{y}^{(i)} \approx F(\boldsymbol{W}\boldsymbol{x}^{(i)})$$

якомога точніше. На i-ій вибірці близькість у цьому випадку вимірюється за *втратами перцептрону*

$$L(\boldsymbol{W}, \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) = (F(\boldsymbol{W}\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)})^T \boldsymbol{W}\boldsymbol{x}^{(i)}$$

1.2.4 Логістична регресія

Для двійкової класифікації логістична регресія є популярною лінійною машиною. Ми кодуємо дві категорії як $\{0, 1\}$ й використовуємо сигмоїдну функцію

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

для порогового значення. Тобто, оцінка ймовірності, яку ми призначаємо вибірці, що відноситься до категорії 1 визначається

$$\mathrm{P}(y_i = 1) = \sigma(\boldsymbol{w}_i \boldsymbol{x}_i)$$

Функцією природних втрат, яку слід використовувати в цьому випадку, є негативна вірогідність вибірки

$$L(\boldsymbol{w}, \boldsymbol{x}^{(i)}, y_i) = -\log \sigma(y_i \boldsymbol{w} \boldsymbol{x}^i).$$

Атрибут $\boldsymbol{w}$ може навчитися градієнтним спуском.

1.2.5 Обмеження лінійних машин

Лінійні класифікатори дають хороші результати лише тоді, коли дані приблизно лінійно відокремлені (тобто межа між класами є гіперплощиною). На жаль, ймовірність цього для великої вибірки даних надзвичайно мала. Один із підходів полягає у використанні результатів попередньої обробки для побудови нових функцій, до яких застосовуються лінійні методи, такі як логістична регресія. Інновація глибокого навчання полягає в тому, що ці особливості також вивчаються з даних.

1.3 Латентна змінна модель

1.3.1 Вступ

Латентна модель змінної — це статистична модель, яка відносить набір спостережуваних змінних (так званих змінних маніфесту) до набору латентних змінних... Ці змінні можуть бути дихотоміальними, ординальними або номінальними змінними. Їх умовні розподіли вважаються біноміальними або багатонаціональними.

1.3.2 Що можуть представляти латентні змінні?

Змінні, які б полегшили завдання, якби вони були відомі: Розпізнавання облич: стать людини, орієнтація обличчя. Розпізнавання об'єктів: параметри пози об'єкта(розташування, орієнтація, масштаб), умови освітлення. Частини позначення мовлення: сегментація речення на синтаксичні одиниці, дерево аналізу. Розпізнавання мовлення: сегментація речення на фонеми або телефони. Розпізнавання рукописного тексту: сегментація рядка на символи. Розпізнавання об'єктів/Аналіз сцени: сегментація зображення на компоненти(об'єкти, частини,...)

1.3.3 Латентна змінна модель

Латентна змінна - це змінна, яка безпосередньо не спостерігається і передбачає вплив на змінні відповіді (змінні маніфесту) Латентні змінні, як правило, входять до економетричної/статистичної модель (латентна змінна модель) з різними цілями: Представлення ефекту незаслужених

коваріатів/факторів, а потім облік незаслуженої неоднорідності між суб'єктами (латентні змінні використовуються для представлення ефекту цих незаслужені фактори)
Облік помилок вимірювання (латентні змінні представляють «істинні» результати, а маніфестні змінні являють собою їх «порушені» версії)
Узагальнення різних вимірювань однакових (безпосередньо) незаслужених характеристик (наприклад, якості життя),так що зразки одиниць можуть бути легко замовлені / класифіковані на основі цих рис (представлені латентні змінні)
Ці моделі, як правило, класифікуються відповідно до:
характер змінних відповіді (дискретні або безперервні)
характер латентних змінних (дискретних або безперервних)
включення чи ні окремих коваріатів

1.3.4 Приклад латентної змінної моделі

- Модель аналізу факторів: фундаментального інструменту в багатоваріатної статистики узагальнити кілька (безперервних) вимірювань через кількість (безперервних) латентних рис; коваріати не включені
- Моделі теорії реагування на елементи: моделі для предметів (категоричні відповідей) вимірювання загальної латентною рисою, яка передбачається (або рідше дискретні) і, як правило, представляють здібності або психологічне ставлення; найважливіша модель IRT був запропонований Раш (1961); зазвичай коваріати не включені
- Узагальнені лінійні змішані моделі:(моделі випадкових ефектів): розширення класу узагальнених лінійних моделей (GLM) для безперервних або категоричні відповіді, які прираховують незаслужену неоднорідність, поза дією спостережуваних коваріатів
- Модель скінченної суміші: модель, яка використовується навіть для однієї змінної відповіді,в яких суб'єкти, як передбачається, походять від субпопуляцій, що мають різні розподіли змінних відповіді; зазвичай коваріати виключаються
- Латентна модель класу: модель для змінних категоричної відповіді на основі дискретної латентної змінної, рівні якої відповідають латентним класи в населенні; зазвичай коваріати виключаються
- Модель регресії скінченної суміші (модель латентної регресії): версія скінченної суміші (або моделі латентного класу), яка включає коваріати, що впливають на умовний розподіл відповіді змінні та/або розподіл латентних змінних

1.4 Штучний загальний інтелект

1.4.1 Історія AI та AGI

Назва "Штучний інтелект"була запропонована та прийнята на конференції в Дартмуті 1956 р. [McCorduck [2009]], організована Марвіном Мінським, Джоном Маккарті, Клодом Шенноном та Натаном Рочестером. З моменту свого народження в 1956 році, Штучний інтелект пережив золотий вік з 1956-1974 рр., "AI winter"з 1974-1980 рр., Бум у 1980-1987 рр. і другий "AI winter"з 1987-1993 рр.

У перші дні штучного інтелекту, найбільше зусиль, докладених у цій галузі, було створити машину, яка могла б подолати бар'єр між матерією та розумом. Порівняно з сучасними дослідженнями, які більше зосереджуються на конкретній галузі, перші дослідження більше зосереджуються на штучному загальному інтелекті (Artificial General Intelligence).

«Теоретик логіки», створений Алленом Ньюеллом і Гербертом А. Саймоном був націлений на досягнення рівня людських навичок вирішення проблем [McCorduck [2009]]. Ця програма успішно доводила деякі математичні теореми, деякі докази були ще більш елегантними, ніж людські. Його автори стверджують, що вони «вирішували проблему поважного розуму/тіла, пояснюючи, як система складається з матерії може мати властивість розуму.» [Crevier [1993]]

Тим не менш, деякі серйозні проблеми, такі як обмеження символічного мислення і відсутність обчислювальної потужності привели до провалу створення загального штучного інтелекту. Крім того, люди почали сумніватися, чи дійсно система штучного інтелекту є свідомістю. Філософ Джон Серла представив уявний експеримент «Китайська кімната» [Cole [2004]], який стверджував, що така система не має людської свідомісті. У цьому уявному експерименті, створено систему штучного інтелекту, яка приймає китайські символи в якості вхідних даних, а потім дає вивід введеного, опрацьований за допомогою деяких правил програми, і це означає, що ця система розуміє китайську мову?

Такі філософські питання не заважали дослідникам рухатися вперед, якщо система штучного інтелекту могла працювати так само, як людина, то чи є вона свідомою, не так важливо. Нарешті, у 1980-х роках Expert System [McCorduck [2009]], система штучного інтелекту, що фокусується на вирішенні проблем у певній галузі, досягла великих успіхів у бізнес-застосуванні. З цього моменту люди зрозуміли, що "Вузький штучний інтелект"може стати більш практичним об'єктом і почати докладати зусиль для розробки AI для конкретних доменів.

2 Набір даних ImageNet

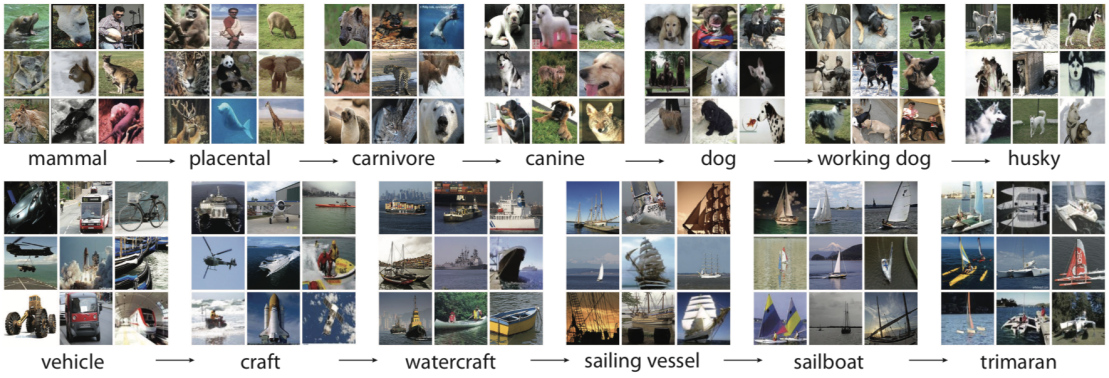
ImageNet відповідає проекту комп'ютерного зору, щодо створення великої візуальної бази даних, яка буде використана для розробки програмного забезпечення розпізнавання візуальних об'єктів. У цьому проекті ImageNet вручну позначили понад 14 мільйонів зображень, щоб вказати, які об'єкти присутні на зображенні, і принаймні в одному мільйоні цих зображень також передбачено обмежувальні рамки. ImageNet містить понад 20 тисяч неоднозначних категорій. З 2010 року проект ImageNet проводить щорічний конкурс програмного забезпечення - ImageNet Large Scale Visual Recognition Challenge (ILSVRC), де програми змагаються за правильну класифікацію та виявлення об'єктів та сцен.

Цей набір даних було вперше представлено Deng et al. [2009] в якості афіші на Конференції з комп'ютерного зору та розпізнавання образів (CVPR) 2009 року у Флориді .

Більш того, набір даних ImageNet та змагання ILSVRC зіграли надзвичайно важливу роль у недавній еволюції спільноти машинного навчання, коли у 2012 році переможець змагання досягнув рекордного на той час рівня похибки - 16% (типово для перемоги команди досягли рівня помилок близько 25%), використовуючи глибокі згорткові нейронні мережі. Ця видатна продуктивність розкриває потенціал архітектури глибоких нейронних мереж і часто розглядається, як відправна точка сильно зростаючого інтересу до машинного навчання загалом та глибокого навчання, зокрема, яке ми спостерігали протягом останніх кількох років.

Він був розроблений з використанням ієрархічної структури WordNet, де кожне значуще поняття (яке може бути описане кількома словами або словосполученнями) називається "набором синонімів"або "синтаксисом". Структура ImageNet переходить від піддерев (найбільш загальних) до синсетів (точніші), забезпечуючи в середньому 500–1000 зображень для ілюстрації кожного синсет. На малюнку 1, ми можемо побачити знімок двох гілок піддерев, що стосуються «ссавців» і «транспортних засобів».

Рис. 1: Знімок двох гілок від кореня до листків ImageNet: верхній рядок піддерево від ссавця; нижній ряд - піддерево автомобіля. Для кожного синсету представлено 9 зображень із довільною вибіркою. Малюнок з сайту Deng et al. [2009]



Властивості ImageNet

- Масштаб: Це найбільший набір даних чистих зображень, доступний для спільноти дослідників зору, з точки зору загальної кількості зображень, кількості зображень в категорії (у середньому понад 600), а також кількості категорій (5247)
- Ієрархія: Категорії організовані у семантичну ієрархію. Дійсно, категорії взаємопов'язані між собою семантичними відносинами, де найважливішим є "is a". Наприклад, існує 147 різних категорій собак.
- Точність: оскільки цей набір даних має бути надійним, він був розроблений таким чином, щоб досягти середнього рівня точності 99,7%.
- Різноманітність: ImageNet також був розроблений з метою, щоб об'єкти на зображеннях мали різне вигляд, положення, точки зору, пози, а також безлад і оклюзії у фоновому режимі.

Дизайн ImageNet

- Збір зображень кандидатів: Зображення кандидатів були зібрані з Інтернету шляхом запитів зображень із декількох пошукових систем. Оскільки пошукові системи, як правило, обмежують кількість зображень, які можна отримати, щоб отримати якомога більше зображень, було розширено набір запитів, додавши запити із словами з батьківського синсету. Щоб ще більше збільшити та урізноманітнити образи кандидатів, було також перекладено запити іншими мовами, зокрема китайською, іспанською, голландською та італійською.
- Очищення зображень кандидатів: Використовуючи Amazon Mechanical Turk, вони перевірили кожне зображення кандидата, зібране на попередньому кроці для даного синсету. Для кожного завдання маркування вони представили користувачам набір зображень-кандидатів та визначення цільового синтезу (включаючи посилання на Вікіпедію). Потім користувачів попросили перевірити, чи містить кожне зображення об'єкти синхронізації. Для того, щоб отримати високу точність на цих ярликах, у них було декілька користувачів, які власноруч позначали одне і те ж зображення, і зображення вважається позитивним, лише якщо воно отримало переконливу більшість голосів. Крім того, вони помітили, що різні категорії вимагають різних рівнів консенсусу серед користувачів. Наприклад, для отримання хорошого консенсусу щодо зображень "бірманської кішки" може знадобитися п'ять користувачів, для зображень "котів" потрібна набагато менша кількість.

Альтернативи ImageNet:

Хорошою альтернативою набору даних ImageNet є набір даних CIFAR-10. Перевага цього набору даних полягає в тому, що він містить зображення із значно меншою роздільною здатністю (кольорові зображення 32x32), розділені на 10 різних класів. Оскільки зображення мають меншу роздільну здатність, а набір даних - набагато менший (60 000 зображень, тобто 6000 зображень на клас), набагато швидше навчати мережі на них. Отже, гарною практикою є початкова підготовка нашої мережі на наборі даних CIFAR-10 для виправлення помилок або тестування різних ідей або архітектур реалізації. Крім того, можна також використовувати набір даних CIFAR-100 (60 000 зображень, розділених на 100 класів, тобто 600 зображень на

клас).

Рис. 2: CIFAR-10 приклад фото

Open Images - це набір даних із майже 9 мільйонів URL-адрес для зображень. Ці зображення були анотовані ярликами та обмежуючи рамки, що охоплюють тисячі класів. Набір даних містить навчальний набір з 9 011 219 зображень, набір перевірок з 41 260 зображень та тестовий набір із 125 436 зображень.

Рис. 3: Open Images приклад фото

Wikimedia Commons є дочірнім сайтом Вікіпедії і містить понад 40 мільйонів безкоштовних медіафайлів, - з великою часткою фотографій, які ви можете використовувати як альтернативу ImageNet. Медіа-файли у Вікісховищі, як правило, позначаються однією або кількома категоріями з ієрархічної системи категорій Вікімедіа.

3 Навчання шляхом зворотного поширення

Тут ми дізнаємося про те, як отримати "хороші можливості"за допомогою зворотного поширення, описане LeCun et al. [1998], для модульно-градієнтного навчання машини.

Зворотне поширення відіграє основну роль в алгоритмі навчання нейронних мереж. Це часто добро працює, але для того, щоб отримати гарні результати, потрібно налаштувати різні аспекти моделі, такі як кількість шарів і вузлів, швидкість навчання тощо. Хоча універсальних рецептів вибору немає, є деякі теорії та хитрощі, що полегшують прийняття рішень.

У автоматичному машинному навчанні методи градієнтного навчання є найбільш вдалим підходом. Дуже важливою у цих стратегіях є здатність до узагальнення, тобто робити правильні прогнози, використовуючи дані, яких система ще ніколи не бачила. Зазвичай дані зашумлені, оскільки можуть бути помилки вимірювання, що призведе до того, що мережі будуть відрізнятися одна від одної та від справжньої функції. Методи узагальнення намагаються виправити помилки з кращою швидкістю та якістю.

3.1 Стандартне зворотне поширення

Найпростіша форма градієнтного навчання передбачає набір модулів, кожен з яких реалізує певну функцію

$$X_n = F_n(W_n, X_{n-1})$$

, де X_n представляє вихідні дані модуля, W_n - регульовані параметри, і X_{n-1} - вхідні дані модуля. Часткові похідні E^p можна обчислити, використовуючи зворотню рекурсію

$$\frac{\partial E^p}{\partial W_n} = \frac{\partial F(W_n, X_{n-1})}{\partial W_n}\frac{\partial E^p}{\partial X_n}$$

$$\frac{\partial E^p}{\partial X_{n-1}} = \frac{\partial F(W_n, X_{n-1})}{\partial X_{n-1}}\frac{\partial E^p}{\partial X_n}$$

Коли рівняння застосовуються до модулів від рівня N до рівня 1, можна обчислити всі часткові похідні функції витрат на всі параметри. А процес обчислення градієнтів називається зворотнім поширенням. Класичними рівняннями зворотнього поширення є

$$\frac{\partial E^p}{\partial Y_n} = F'(Y_n)\frac{\partial E^p}{\partial X_n}$$

$$\frac{\partial E^p}{\partial W_n} = X_{n-1}\frac{\partial E^p}{\partial Y_n}$$

$$\frac{\partial E^p}{\partial X_{n-1}} = W_n^T\frac{\partial E^p}{\partial Y_n}$$

Таким чином, найпростіший алгоритм градієнтного спуску

$$W(t) = W(t-1) - \eta\frac{\partial E}{\partial W}$$

, де η - скалярна константа.

Зворотне поширення може бути дуже повільним для багатошарових мереж, і немає жодних гарантій того, що мережа буде добре сходитися або взагалі сходитися. Однак є деякі хитрощі, які допомагають знайти хороше рішення та одночасно пришвидшити конвергенцію.

3.2 Загальні класи модулів

Класи станів

Змінні стану: Кожен вузол в архітектурі глибокого навчання має клас стану, що містить дві змінні :

1. Поле стану: Зберігає результуючі дані функції, обчислені від входів до вузла, який прямо поширюється.
2. Похідна: похідна енергії, що зворотньо поширюється відносно стану, що зворотньо поширюється.

Обидві ці змінні можуть бути скалярними, векторними або матричними. Нижче наведено деякі загальновживані класи модулів разом із полем класу стану, який вони мають.

3.2.1 Лінійний модуль

Вони становлять суть будь-якої глибокої архітектури і використовуються для здійснення всіляких лінійних перетворень. Найпростішим та найбільш часто використовуваним лінійним модулем є $f(x) = Wx$, де вхідний множиться x вагою, яка навчається шляхом зворотного розповсюдження.

1. Стан : WX_{in} прямо поширена до X_{out}
2. Похідна :
   - $\frac{\partial E}{\partial X_{out}} W$ зворотньо поширена до X_{in}
   - $X_{out} \frac{\partial E}{\partial W}$ зворотньо поширена до W

3.2.2 Евклідова відстань

Це тип вимірювання відстані між двома векторами у просторі. Зазвичай використовується як функція втрат для обчислення квадратичної помилки. Його змінні:

1. Стан : $\frac{1}{2}\|X_{in} - Y\|^2$ прямо поширена до X_{out}
2. Похідна :
   - $X_{in} - Y$ зворотньо поширена до X_{in}
   - $Y - X_{in}$ зворотньо поширена до Y_{in}

3.2.3 Y-з'єднувач та доповнення

Модуль розгалуження Y-Connector, відомий як дублікат одного входу X_{in} у кілька виходів. Модуль Додавання або Суматор поєднує кілька входів і робить арифметичну суму на виході. Ці два модулі доповнюють один одного, похідним одного є поле стану іншого, і навпаки.

1. Y-з'єднувач клас стану :
   (а) Стан : $X_{out_k} = X_{in} \; \forall \; k \in [1 \dots K]$ прямо поширена до кожного X_{out_k}
   (б) Похідна : $\sum\limits_k \frac{\partial E}{\partial X_{out_k}}$ зворотньо поширена до X_{in}

2. Додане поле стану :

   (а) Стан : $X_{out} = \sum_k X_{in_k}$ прямо поширена до X_{out}.

   (б) Похідна : $\frac{\partial E}{\partial X_{out}}$ зворотньо поширена до кожного X_{in_k}.

3.2.4 Модуль комутатора

Він приймає k X_{in_k} входів Y виходів. Вхід Y діє як перемикач, який вибирає один з k входів на основі його значення.

1. Стан : $X_{out} = \sum_k \sigma(Y-k)X_k$ прямо поширена до X_{out}.

2. Похідна : $\sigma(Y-k)\frac{\partial E}{\partial X_{out}}$ зворотньо поширена до кожного X_{in_k}.

3.2.5 Soft(arg)max Модуль

Він використовується для перетворення оцінки в дискретний розподіл ймовірностей. Він приймає k входів і обчислює k виходів, де сума виходів складає 1 (і всі значення належать до інтервалу [0,1]), і, отже, це корисно для нормалізації. Використовується у багатокласовій класифікації, такі як багаточленна логістична регресія.

1. Стан : $X_{out_k} = \frac{\exp(\beta X_{in_j})}{\sum_i \exp(\beta X_{in_i})}$ вппрямо поширена до k X_{out_k} виходів.

2. Похідна :

   - Якщо $i = j$: $\beta X_{out_i}(1 - X_{out_i})$
   - Якщо $i \neq j$: $-\beta X_{out_j} X_{out_i}$

   І те, і інше поширюється до кожного k X_{in_k} виводу.

Очевидно, Softmax оновлює всі входи, незалежно від того, $i = j$ чи ні. Це β «коефіцієнт температури понад 1 кБ», запозичений безпосередньо з термодинаміки. Це просто константа і не впливає на архітектуру, окрім лише зміни значень кроку градієнта.

Softmax і Комбо перемикач

Перемикач має приємну властивість вибирати лише один вхід, тоді як softmax має приємну властивість "навчати"кожний вхід X_{in_k}.Зазвичай вони поєднуються в остаточному шарі, особливо в задачах класифікації, результати Softmax надсилаються на рівень перемикача.

4 Зворотне поширення та його хитрощі

4.1 Стохастичне навчання проти пакетного

На кожній ітерації існує два способи обчислення градієнта, стохастичне та пакетне навчання. Для стохастичного навчання в кожній ітерації вибирається *один приклад* навчального набору, тоді як для пакетного навчання передається весь набір даних. Обидва підходи мають переваги.

Переваги стохастичного навчання:

- Стохастичне навчання зазвичай набагато швидше.
- Часто призводить до кращих рішень.
- Його можна використовувати для відстеження змін.

Переваги пакетного навчання:

- Умови конвергенції добре зрозумілі.
- Воно може використовувати багато методів прискорення.
- Теоретичний аналіз простіший.

4.1.1 Перетасування прикладів

Навчальний набір для перетасовки допомагає мережам швидше вчитися, оскільки мережа стикається з новим і незнайомим зразком на кожній ітерації. Цей фокус стосується лише стохастичного навчання.

4.1.2 Нормалізація входів

Коли середнє значення кожної вхідної змінної наближається до нуля, збіжність досягається швидше. Зміщення середнього вводу від нуля часто зміщує оновлення в певному напрямку, що уповільнює швидкість навчання. Деякі загальні вказівки щодо перетворення вхідних даних:

- Середнє значення кожної змінної у навчальному наборі має бути близьким до нуля.
- Масштабуйте вхідні змінні до однакових коваріантностей.
- Вхідні змінні повинні бути некорельованими.

4.1.3 Сигмоїд

Нелінійні функції активації є центральними для нейронних мереж, оскільки вони дозволяють мережам вивчати нелінійні взаємозв'язки. Найбільш поширеною функцією активації є сигмоїдна, яка відображає значення [0,1]. Два найпоширеніші приклади:

$$f(x) = \frac{1}{1 + exp(-x)}$$

$$f(x) = tanh(x)$$

Деякі хитрощі щодо стигмоїди:

- $tanh(x)$ часто сходяться швидше, ніж стандартна логістична функція.
- Рекомендована сигмовидна: $f(x) = 1.7159tanh(\frac{2}{3}x)$.
- Іноді корисно додати лінійний термін до $tanh(x)$, щоб уникнути плоских плям. Наприклад $f(x) = tanh(x) + ax$.

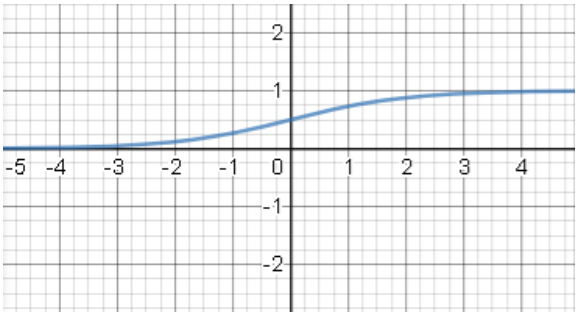


Рис. 4: Сигмоїд -1/1+exp(-x)

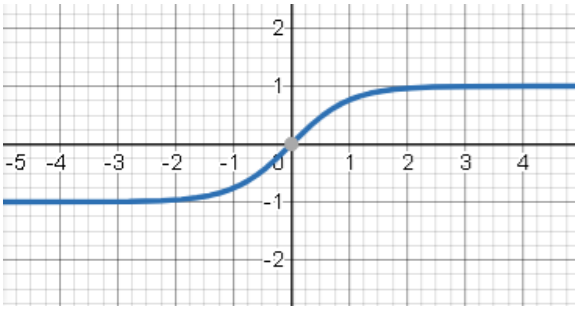


Рис. 5: Сигмоїд tanh(x)

4.2 Вибір цільових значень

Традиційно проблеми класифікації, як правило, визначають двійкові цільові значення, такі як $\{-1, 1\}$, але це має два основних недоліки.

По-перше, вага може перестати змінюватися, коли похідна сигмоподібної залози дорівнює приблизно нулю. Незважаючи на те, що тоді ваги присвоюються більшим значенням, їх відповідні градієнти все ще множаться на невелику похідну сигмовидної форми, що призводить до неоновлення ваги. По-друге, процес навчання неправильно класифікує вхідні дані, не вказуючи на низьку довіру через насичені результати та великі ваги.

Рішення: встановіть цільове значення в області сигмовидної функції. Особливо, ідеально оптимальним способом є присвоєння точки максимальної другої похідної на сигмоїді цільовим значенням.

4.3 Ініціалізація ваг

Ініціалізовані ваги переважно мають бути малими, оскільки їх наступні ваги можуть коливатися в лінійній області сигмоїди, що дозволяє навчальному процесу справді вивчати закономірності.

Потрібні 3 кроки:

- Нормалізуйте навчальні набори.
- Розгортання сигмоподібних функцій.

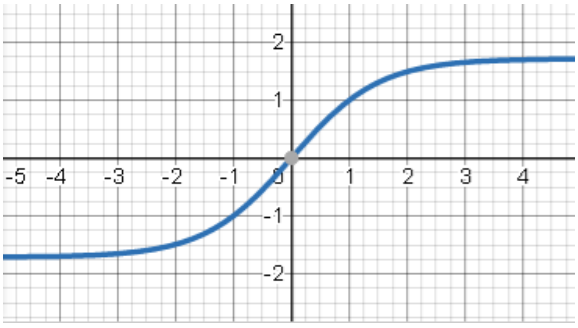


Рис. 6: Сигмоїд 1.7159*tanh(2*x/3)

- Призначаються ваги випадковим чином на основі розподілу із середнім нулем та стандартним відхиленням $\sigma_w = m^{(1/2)}$ (m - кількість входів в одиниці).

4.4 Вибір темпів навчання

Підбір персоналізованого відношення η_i до W_i може покращити конвергенцію. Для цього для обчислення потрібні другі похідні, оскільки для деяких ваг потрібна мала швидкість навчання, щоб збігатися, тоді як для інших потрібні великі швидкості навчання, щоб сходитися з належною швидкістю.

Потрібні 3 міркування:

- Призначено різні темпи навчання різній вазі.
- Забезпечені темпи навчання пропорційні квадратному кореню з числа введених даних.
- Передбачається, що ваги у вищих шарах будуть меншими, ніж у нижчих шарах.

Інші методи включають:
Імпульс:

$$\Delta w(t+1) = \eta \frac{\partial E_{t+1}}{\partial w} + \mu \Delta w(t)$$

Допомогає більше в пакетному градієнті, ніж у стохастичному.
Адаптивні темпи навчання:
Контролює швидкість конвергенції, регулюючи швидкість навчання на основі помилки.

4.5 Функції радіальної основи проти сигмоподібних одиниць

Типовою альтернативою сигмоїдної функції є мережа радіальних базисних функцій. Блок RBF охоплює лише невелику локальну область вхідного простору, яка пришвидшує процес навчання. Однак місцезнаходження може спричинити проблеми у розмірності простору.

4.6 Числення багатошарових систем

Числення - це розділ математики, який займається швидкістю зміни величин на відсотки. У цьому розділі ми розглянемо основи диференціального числення, матричного числення та застосуємо їх для обчислення градієнтів багатошарових систем. Ефективне обчислення цих градієнтів зверху вниз буде корисним в алгоритмі зворотного розповсюдження, як ми побачимо далі.

4.6.1 Похідні, часткові похідні, Якобіан

Похідна: Нехай $y = f(x)$ є дійсною функцією реальної змінної такою, що $x, y \in \mathbb{R}$, похідна f в точці a задається формулою

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h} \tag{3}$$

Вона позначається $\frac{dy}{dx}$ і вимірює швидкість зміни y щодо зміни x.

Часткова похідна: Припустимо, функція $y = f(\boldsymbol{x})$ приймає вектор $\boldsymbol{x} \in \mathbb{R}^d$ як вхід, а потім часткова похідна $f(\boldsymbol{x})$ відносно одного з елементів вектора $\boldsymbol{x}$ задається $\frac{\partial f}{\partial x_i}$. Вона вимірює швидкість зміни в y щодо невеликої зміни в x_i, тоді як інші елементи вектора залишаються незмінними.

Якобіан: Функція Якобіан $\boldsymbol{y} = f(\boldsymbol{x})$, де $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$, - матриця, що містить усі часткові похідні елементів y $\boldsymbol{y}$ з елементів y $\boldsymbol{x}$.
Припустимо, більш формально $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ беремо $\boldsymbol{x} \in \mathbb{R}^n$ на введення і даємо $\boldsymbol{y} \in \mathbb{R}^m$ як виведення, Якобіан $\mathbf{J}$ задається як:

$$\mathbf{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{4}$$

Якщо функція має дійсне значення, то матриця Якобіана містить лише один рядок, який є транспонуванням градієнта дійсної функції відносно вектора. Ми тут формально не визначатимемо «Градієнт», а використовуватимемо позначення Якобіана.

4.6.2 Правило ланцюга, правило ланцюга вищого розміру

Припустимо, у нас є композиція з двох диференційованих функцій f і g така, що $y = g(x)$ і $z = f(y)$ правило ланцюга допомагає нам обчислити похідну складеної функції $z = f \circ g$ щодо x. Це визначається як

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{5}$$

Це можна поширити у композицію функцій у вищих вимірах. Нехай $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ та $\mathbf{g} : \mathbb{R}^p \to \mathbb{R}^n$ такі, що $\boldsymbol{y} = g(\boldsymbol{x})$, $\boldsymbol{z} = f(\boldsymbol{y}), \boldsymbol{x} \in \mathbb{R}^p, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{z} \in \mathbb{R}^m$, тоді ми можемо написати

правило ланцюжка для складеної функції $\boldsymbol{z} = \mathbf{f} \circ \mathbf{g}$ наступним чином, використовуючи матриці Якобіана

$$J_{\mathbf{f}\circ\mathbf{g}}(\boldsymbol{x}) = J_{\mathbf{f}}(\boldsymbol{y}) J_{\mathbf{g}}(\boldsymbol{x}) \tag{6}$$

4.6.3 Переглянуто матричне числення та правило ланцюга

Матричне числення дає нам узагальнену структуру для роботи з функціями, які працюють з векторами або матрицями як вхідні та вихідні. Існує два позначення для представлення похідних функцій за допомогою матричного числення, заснованого на тому, чи слід розглядати вектори як вектори рядків або стовпців. Ми будемо використовувати позначення розмітки чисельника, де ми представляємо вектор як вектор стовпця. Ми будемо використовувати x для позначення скаляра, $\boldsymbol{x}$ для позначення вектора стовпця, що містить елементи $x_1, \cdots, x_n$, $\boldsymbol{X}$ для позначення матриця, що містить x_{ij} у i-му рядку та j-му стовпці. Похідні різних скалярних, векторних та матричних комбінацій наведені нижче.

Похідна	y	$\boldsymbol{y}$
x	$\frac{\partial y}{\partial x}$	$\frac{\partial \boldsymbol{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$
$\boldsymbol{x}$	$\frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$

$$\frac{\partial y}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix} \tag{7}$$

$$\frac{\partial \boldsymbol{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{12}}{\partial x} & \cdots & \frac{\partial y_{1n}}{\partial x} \\ \frac{\partial y_{21}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{2n}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{m1}}{\partial x} & \frac{\partial y_{m2}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix} \tag{8}$$

Відтепер ми можемо використовувати наведені вище позначення для представлення похідних як скаляр/вектор/матрицю відносно скаляру/вектору/матриці. Ми також перепишемо правило ланцюжка, використовуючи ці позначення. Припустимо, що X, Y, Z є скалярними, векторними або матричними, ми можемо записати правило ланцюжка для складеного відображення як

$$\frac{\partial Z}{\partial X} = \frac{\partial Z}{\partial Y} \frac{\partial Y}{\partial X} \tag{9}$$

Припустимо $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ і $\mathbf{g} : \mathbb{R} \to \mathbb{R}^n$ як $\boldsymbol{y} = g(x)$, $z = f(\boldsymbol{y}), x \in \mathbb{R}, \boldsymbol{y} \in \mathbb{R}^n, z \in \mathbb{R}$, ми можемо обчислити похідну z відносно x, використовуючи правило ланцюга матричного числення,

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial x} \tag{10}$$

$$= \begin{bmatrix} \frac{\partial z}{\partial y_1} \frac{\partial z}{\partial y_2} \cdots \frac{\partial z}{\partial y_n} \end{bmatrix} \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_n}{\partial x} \end{bmatrix} \tag{11}$$

$$= \sum_{i=0}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x} \tag{12}$$

Наведений вище результат можна також інтерпретувати так, ніби ми хитаємо x на невелике значення по усіх проміжних y_i^s, які залежать від зміни x, що призводить до зміни остаточного z. Отже, похідну такого складеного відображення z можна записати, застосувавши правило ланцюга щодо всіх незалежних проміжних змінних, що з'єднують z і x, і беручи над ними суму.

4.6.4 Обчислення градієнтів у багатошарових системах

Багатошарова або багатомодульна система - це мережа, що складається з декількох модулів, розташованих вертикально, де кожен модуль є функцією над своїми входами та виходами, як показано на Риснку 7. Давайте будемо дотримуватися випадку, коли вихід мережі є скалярною змінною, оскільки нейронні мережі можна розглядати як багаторівневі системи зі скалярними втратами як вихід. Застосуємо ланцюгове правило для обчислення похідних проміжних змінних щодо скалярного виходу таких систем.

Тепер розглянемо i -й модуль, і якщо ми знаємо похідну виходу модуля щодо енергії E, яку ми хочемо мінімізувати, ми можемо обчислити похідну E щодо вхідного модуля X та параметри модуля W, використовуючи правило ланцюжка.

$$X_i = F_i(X_{i-1}, W_i) \tag{13}$$

$$\frac{\partial E}{\partial W_i} = \frac{\partial E}{\partial X_i} \frac{\partial X_i}{\partial W_i} \tag{14}$$

$$= \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i} \tag{15}$$

$$\frac{\partial E}{\partial X_{i-1}} = \frac{\partial E}{\partial X_i} \frac{\partial X_i}{\partial X_{i-1}} \tag{16}$$

$$= \frac{\partial E}{\partial X_i} \frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}} \tag{17}$$

Розміри цих похідних можна легко обчислити, використовуючи позначення розмітки чисельника матричного числення. Зверніть увагу, що похідна скаляра по відношенню до вектора є вектором рядка в цьому позначенні, де як градієнт, який представляє те саме, позначається вектором стовпця.

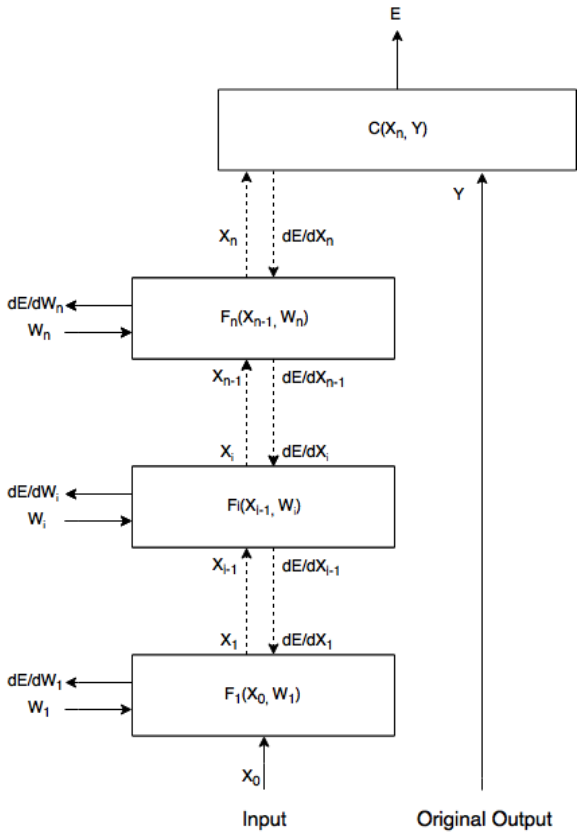


Рис. 7: Багатошарова мережа, що відображає потік входів, виходів і градієнтів

4.7 Оновлення ваги - Методи оптимізації

Алгоритми машинного навчання оновлюють параметри за допомогою ітераційного процесу. Процес передбачає мінімізацію функцій витрат.У цьому розділі ми представимо деякі методи оптимізації, які зазвичай використовуються в нейронних мережах.

4.7.1 Градієнтний спуск

Градієнтний спуск - це метод оптимізації, який намагається знайти мінімум опуклої функції витрат, беручи її похідну першого порядку. У той час як похідні в основному представлені скалярами, градієнти представлені векторами, компоненти яких є частковою похідною функції витрат за різними вимірами. Хоча використання градієнтного спуску та навчання з усім навчальним набором як однієї партії не є ідеальним у налаштуваннях нейронної мережі, це оригінальна ідея багатьох похідних методів оптимізації, таких як міні-пакетний градієнтний спуск, стохастичний градієнтний спуск (SGD) та стохастичний градієнтний спуск з імпульсом.

Припускаючи довільну функцію витрат (функція помилки) $J(\theta)$ з двома ознаками, є опуклою функцією параметра θ. Алгоритм градієнтного спуску розпочне процес із випадкової точки

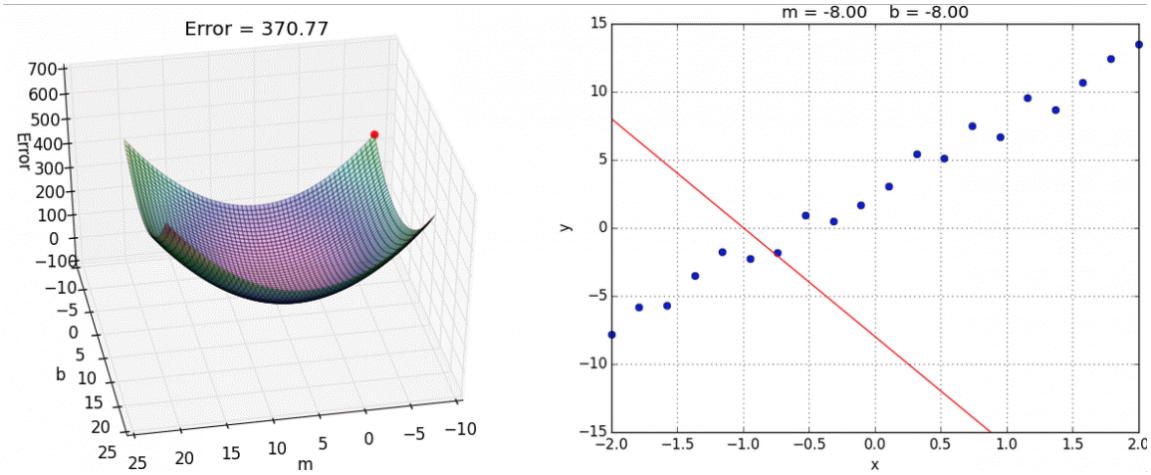


Рис. 8: Знімок анімованого графіка з блогу Аліхана Тежані, який чудово демонструє процес градієнтного спуску.

на графіку функції помилки та обчислить градієнт у цій точці. Далі алгоритм зробить крок у напрямку, протилежному градієнту. Цей процес можна записати як $\theta = \theta - \eta\Delta J(\theta)$, де Δ - це градієнт, а η - розмір кроку або швидкість навчання. Оскільки негативний градієнт - це місце, де графіку функції найшвидше зменшується, алгоритм буде продовжувати цей ітераційний процес, поки не буде знайдений загальний (або локальний) мінімум. Звідси походить назва «градієнтний спуск». Аліхан Тежані [1] створив анімований графік, який точно ілюструє цей процес.

Псевдокод реалізації:

```
for i in num_of_iterations:
        gradient = calculate_gradient(data, loss)
        theta = theta - learning_rate * gradient
```

4.7.2 Стохастичний градієнтний спуск (SGD)

Градієнтний спуск іноді також називають пакетним градієнтним спуском, що означає, що алгоритм використовує весь навчальний набір для обчислення градієнта на кожному кроці. Іноді цей процес може бути неможливим. Впровадження SGD дозволяє розраховувати градієнт, використовуючи лише одну точку даних x_i, y_i за раз, поки не буде вичерпаний весь набір даних, а вибір точки даних на кожному кроці є випадковим. Цей процес можна записати як $\theta = \theta - \eta\Delta J(\theta; x_i, y_i)$

Псевдокод впровадження SGD:

```
for i in num_of_iterations:
  random_shuffle(data)
  for example in data:
```

[1] https://alykhantejani.github.io/a-brief-introduction-to-gradient-descent/

```
gradient_sgd = calculate_gradient(example, loss)
theta = theta - learning_rate * gradient_sgd
```

SGD економить обчислювальну потужність, використовуючи один приклад за раз. Однак він втрачає перевагу векторизації, що сприяє прискоренню тренувального процесу. Коли ми реалізуємо нейронну мережу, кількість прикладів, що використовуються для обчислення градієнта, знаходиться десь між 1 (SGD) і загальною кількістю прикладів (пакетний GD). Це називається міні-пакетним градієнтним спуском. Він поєднує в собі найкраще з обох світів: процес навчання можна пришвидшити за допомогою векторизації, і він менш інтенсивний на обчисленнях на кожному кроці. Загальна кількість прикладів у кожній партії або розмірі партії становить 32, 64, 128, 256 або число, що має ступінь 2.

4.7.3 RMSprop

RMSprop, що розшифровується як "середньоквадратична опора - ще один метод адаптивної оптимізації, запропонований Джеффом Хінтоном. Ідея RMSprop полягає в згладжуванні градієнта шляхом ділення його значення на експоненціально зважене середнє квадратичне градієнтів. Порівняно з SGD, це зменшує шум у крокових напрямках. Це дуже схоже на SGD з імпульсом, але з квадратичними градієнтами. Раніше J використовувалось для представлення функції витрат. Ці позначення популярні в умовах машинного навчання. Зокрема, до глибокого навчання, тепер давайте трохи змінимо позначення, використовуючи E для представлення функції помилки (вартості). З RMSprop функція оновлення параметрів тепер стає

$$E(\theta)_t = \beta E(\theta)_{t-1} - (1-\beta)[\Delta E(\theta_t)]^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{[E(\theta_t)]^2}\Delta\theta_t$$

Щоб краще зрозуміти концепцію, може бути корисно подивитись блог Себастьяна[2] на “Adadelta”. Coursera[3] має 7-хвилинне відео, спеціально присвячене цьому методу оптимізації.

4.7.4 Імпульс

Алгоритм SGD страждає від нестабільності. Імпульс Qian [1999] - це проста спроба зменшити цю нестабільність, використовуючи зміщений градієнт, а не лише використовуючи оцінювач градієнта для поточного кроку. Це зменшує коливання градієнта і може прискорити процес оптимізації. Далі псевдокод Алгоритму.

4.7.5 Адам

Алгоритм оптимізації Адама може бути найпопулярнішим алгоритмом оптимізації на сьогодні. Адам означає «Адаптивна оцінка моменту» і вперше був представлений у 2014 році?. Оптимізатор Адама зазвичай вимагає незначного налаштування своїх гіперпараметрів і добре працює на практиці. Алгоритм вносить певну плавність градієнтного спуску, зменшуючи ефект стохастичності градієнта та зменшуючи розмір кроку, коли всередині міні-партії кроку є велика дисперсія. Оптимізатор Адама дуже близький до RMSprop, різниця полягає в тому, що Адам використовує

[2] http://ruder.io/optimizing-gradient-descent/index.html#batchgradientdescent
[3] https://www.coursera.org/learn/deep-neural-network/lecture/BhJlm/rmsprop

Algorithm 1 Momentum

$\alpha = learning\ rate\ \gamma = decay\ rate$

procedure
  θ_0
  $m_0 \leftarrow 0$
  while convergence condition **do**
    $g_t \leftarrow grad(f_t(\theta_{t-1}))$
    $m_t \leftarrow \gamma m_{t-1} + \alpha g_t$
    $\theta_t \leftarrow \theta_{t-1} - m_t$
  return θ_t

зміщений градієнт (використовуючи швидкість затухання для імпульсу першого порядку) замість неупередженого градієнта.

Алгоритм можна описати за кілька кроків

- обчислити градієнт партії
- обчислити упереджену оцінку градієнта (перший момент). Як і в SGD з Імпульсом, він використовує градієнт попереднього кроку зі швидкістю затухання β_1.
- обчислити упереджену оцінку дисперсії (момент другого порядку) зі швидкістю затухання β_2.
- правильний перший та другий момент (щоб мати глобальні спадні норми)
- оновити градієнт першим моментом, поділеним на квадратний корінь другого моменту.

Далі псевдокод Алгоритму.

Algorithm 2 Adam

Запропонованими гіперпараметрами за замовчуванням є $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ і $\epsilon = 10^{-8}$.

procedure
  θ_0
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $t \leftarrow 0$
  while convergence condition **do**
    $g_t \leftarrow grad(f_t(\theta_{t-1}))$
    $t \leftarrow t + 1$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-2} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow m_t (1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t (1 - \beta_2^t)$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
  return θ_t

4.7.6 Еластичне усереднення SGD

- **Вступ**

  Еластичне усереднення стохастичного градієнтного спуску (EASGD) Zhang et al. [2015a] - це метод оптимізації, який значно пришвидшує паралелізацію обчислень для навчання великих моделей глибокого навчання при роботі на декількох графічних картах. Розгляньте карту GPU як місцевого працівника, який має власний вектор локальних параметрів. Комунікація та координація роботи місцевих робітників спирається на силу пружності, яка пов'язує вектори параметрів, які вони обчислюють, із центральною змінною, що зберігається майстром. Середнє значення часу та простору над векторами параметрів, обчислене місцевими робітниками, розглядається як кориговане середнє для оновлення центральної змінної (глобальний середній параметр). Основна ідея цього алгоритму полягає у тому, щоб дозволити кожному працівникові підтримувати власну якісну роботу, одночасно зменшуючи накладні витрати на комунікацію між майстром та місцевими робітниками. Крім того, цей алгоритм пропонує швидку конвергентну мінімізацію. EASGD може застосовуватися до налаштувань глибокого навчання, таких як паралельне навчання згорткових нейронних мереж.

  Розглянемо $F(x)$ як функцію в паралельному обчислювальному середовищі з $p \in \mathbf{N}$ працівниками та майстром. Стохастична задача оптимізації ілюструється наступним чином

  $$min_x F(x) := \mathbf{E}[f(x, \xi)]$$

  де x - параметр моделі, який слід оцінити, а ξ - випадкове збурення. Наведене вище рівняння можна переформулювати у вигляді розподіленої форми нижче

  $$min_{x^i, \tilde{x}} \sum_{i=1}^{p} \mathbf{E}[f(x^i, \xi^i)] + \frac{\rho}{2}||x^i - \tilde{x}||^2 \qquad s.t. x^i = \tilde{x}, i = 1, 2, ...p,$$

  Ми позначаємо x^i як місцеві змінні, а $\tilde{x}$ як центральну змінну. Квадратичний термін покарання гарантує, що місцеві працівники не потраплять в різні локальні оптимуми, які знаходяться далеко від центральної змінної. Крім того, EASGD передбачає, що кожен працівник має доступ до всього набору даних.

- **EASGD**

  Як згадувалось у вступі, стаття зосереджена на стохастичному налаштуванні. Запропоновані в роботі алгоритми мають два типи: синхронне та асинхронне еластичне усереднення SGD. Для обох типів EASGD ми маємо позначення

  $$\Phi(x^i, \tilde{x}) := \mathbf{E}[f(x^i, \xi^i)] + \frac{p}{2}||x^i - \tilde{x}||^2.$$

  Шляхом лінеаризації $\Phi(x^i, \tilde{x})$, ми отримуємо

  $$\mathbf{L}_t^\rho(x^i, \tilde{x}) = \left[\left\langle g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t), x^i - x_t^i \right\rangle + \frac{1}{2\eta}||x^i - x_t^i||^2\right] + \left[\rho\left\langle \tilde{x}_t - x_t^i, \tilde{x} - \tilde{x}_t \right\rangle + \frac{1}{2\eta}||\tilde{x} - \tilde{x}_t||^2\right]$$

  щоб рівняння можна було природним чином розкласти, щоб оновити x_{t+1}^i і $\tilde{x}_{t+1}$. Перший компонент, зафіксований у перших квадратних дужках, походить від лінеаризації $\Phi(x^i, \tilde{x})$

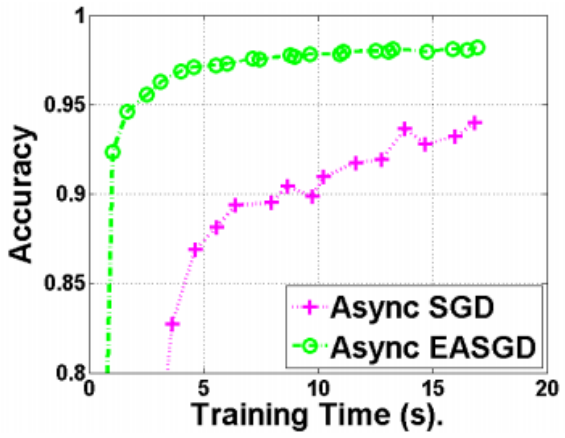


Рис. 9: Порівняння SGD та EASGD при 4 GPU на наборі даних MNIST.

щодо x^i, а другий компонент - від лінеаризації $\Phi(x^i, \tilde{x})$ щодо $\tilde{x}_t$. t позначає індекс ітерації, $g_t^i(x_t^i)$ позначає стохастичний градієнт f відносно x^i обчислений на ітерації t, і η швидкість навчання. Тоді ми можемо виконати оновлення таким чином, що

$$x_{t+1}^i = x_t^i - \eta(g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t)),$$

$$\tilde{x}_{t+1} = \tilde{x}_t + \eta \sum_{i=1}^{p} \rho(x_t^i - \tilde{x}_t).$$

Отримавши правило оновлення, їх можна легко реалізувати в паралельному обчислювальному середовищі з одним ведучим та p працівниками, де місцеві працівники виконують оновлення x^i незалежно від основного оновлення на $\tilde{x}$. Зверніть увагу, що в цьому налаштуванні ми оновлюємо x^i і $\tilde{x}$ для кожної ітерації t. Кожен працівник читає поточне значення центральної змінної $\tilde{x}$ і використовує її для оновлення локальної змінної x^i. Потім майстер чекає оновлення x^i від усіх працівників p перед оновленням центральної змінної. Це синхронна версія алгоритму EASGD.

Натомість асинхронний варіант алгоритму дозволяє кожному працівникові мати власний годинник t^i для оновлення x^i, а майстер потім оновлює $\tilde{x}$ щоразу, коли місцеві працівники закінчують τ кроків їх градієнтних оновлень. Ми називаємо τ як *період спілкування*. Повний алгоритм показаний в Алгоритмі 1.

- **Імпульс EASGD**

  Імпульс EASGD є варіантом Алгоритму 1.

  Алгоритм 2 для EAMSGD.

Algorithm 3 Asynchronous EASGD

$\alpha = moving\ rate$, $\eta = learning\ rate$, $\tau = communication\ period$, $\tilde{x}$ *is initialised randomly*

$t^i = 0$
$x^i = \tilde{x}$
Repeat
  $x \leftarrow x^i$
  $if\ (\tau\ divides\ t^i)\ then$
    $a) x^i \leftarrow x^i - \alpha * (x - \tilde{x})$
    $b) \tilde{x} \leftarrow \tilde{x} + \alpha * (x - \tilde{x})$
  end
  $x^i \leftarrow x^i - \eta * g^i_{t^i}(x)$
  $t^i \leftarrow t^i + 1$
$Until forever$

Algorithm 4 Asynchronous EAMSGD

$\alpha = moving\ rate$, $\eta = learning\ rate$, $\tau = communication\ period$, $\delta = momentum\ term$, $\tilde{x}$ *is initialised randomly*

$t^i = 0$
$v^i = 0$
$x^i = \tilde{x}$
Repeat
  $x \leftarrow x^i$
  $if\ (\tau\ divides\ t^i)\ then$
    $a) x^i \leftarrow x^i - \alpha * (x - \tilde{x})$
    $b) \tilde{x} \leftarrow \tilde{x} + \alpha * (x - \tilde{x})$
  end
  $v^i \leftarrow \delta * v^i - \eta * g^i_{t^i}(x + \delta * v^i)$
  $x^i \leftarrow x^i + v^i$
  $t^i \leftarrow t^i + 1$
$Until forever$

Він базується на схемі імпульсу Нестерова.
Імпульс Нестерова - це дещо інша версія методу оновлення імпульсу, яка стала популярною останнім часом. Pathy [2018] Це можна визначити наступним чином:

$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W - \beta V_{t-1}, X, y)$$
$$W = W - V_t$$

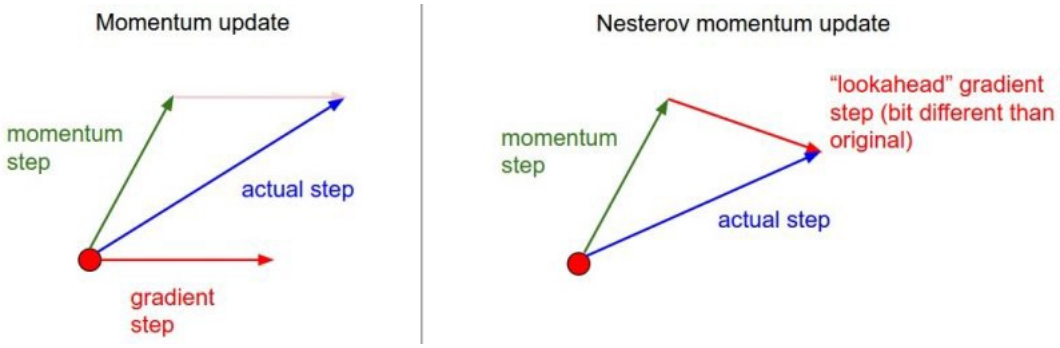


Рис. 10: Оновлення імпульсу Нестерова. Джерело: слайди класу Стенфорд CS231n Pathy [2018]

У цьому випадку оновлення місцевого працівника буде змінено наступним чином:

$$v_{t+1}^i = \delta v_t^i - \eta g_t^i (x_t^i + \delta v_t^i)$$

$$x_{t+1}^i = x_t^i + v_{t+1}^i - \eta\rho(x_t^i - \tilde{x}_t)$$

Де δ - імпульсний термін. Це оригінальний алгоритм EASGD коли ми встановлюємо $\delta = 0$. Повний алгоритм наведений в Алгоритм 2.

4.8 Ініціалізація ваги

Ми вивчили загальні архітектури того, як побудувати нейронну мережу, зокрема, згорткові нейронні мережі. Однак у глибоких архітектурах НМ, щоб корисно та ефективно навчати мережу, нам доведеться спочатку ініціалізувати її параметри належним чином. Вибір початкових ваг насправді може мати глибокий вплив як на швидкість конвергенції, так і на кінцеву якість мережі.

Щоб зрозуміти, чому стандартний градієнтний спуск так сильно відрізняється від глибоких нейронних мереж з різною вагою ініціалізації, давайте зробимо простий розрахунок. Правило ланцюга стверджує, що для обчислення швидкості зміни функції L щодо однієї зі змінних $\mathbf{W}_{jk}^{(i)}$ (часткова похідна від L щодо $\mathbf{W}_{jk}^{(i)}$, ми можемо обчислити похідну відносно змінної вищого рівня $vectx_k^{(i+1)}$, що залежить від $\mathbf{W}_{jk}^{(i)}$ спочатку, а потім помножте його на похідну від $\boldsymbol{x}_k^{(i+1)}$ щодо $\mathbf{W}_{jk}^{(i)}$. Математично для приватного випадку нейронної мережі, як багатошаровий

персептрон, це можна записати як:

$$\frac{\partial \mathrm{L}}{\partial \mathbf{W}_{jk}^{(i)}} = \frac{\partial \mathrm{L}}{\partial \boldsymbol{x}_k^{(i+1)}} \frac{\partial \boldsymbol{x}_k^{(i+1)}}{\partial \mathbf{W}_{jk}^{(i)}}$$

Перший доданок справа може бути обчислений рекурсивно, як ми побачимо нижче. Другий член праворуч - це єдине місце, яке безпосередньо стосується ваги $\mathbf{W}_{jk}^{(i)}$ і може бути розбито на:

$$\begin{aligned}\frac{\partial \boldsymbol{x}_k^{(i+1)}}{\partial \mathbf{W}_{jk}^{(i)}} &= \frac{\partial \mathrm{f}(\boldsymbol{s}_j^{(i)})}{\partial \boldsymbol{s}_j^{(i)}} \frac{\partial \boldsymbol{s}_j^{(i)}}{\partial \mathbf{W}_{jk}^{(i)}} \\ &= \mathrm{f}'(\boldsymbol{s}_j^{(i)})\, \boldsymbol{x}_j^{(i)}.\end{aligned}$$

З цього ви можете бачити, що якщо вихідні дані мають тенденцію до нуля, градієнти також роблять це, що призводить до того, що ваги припиняють оновлення. З іншого боку, якщо $\boldsymbol{s}_j^{(i)}$ виявиться занадто екстремальним, активації f можуть мати нульові похідні і знову заважати вагам оновлюватись, коли ви з використанням нелінійності сигмоїдної або гіперлінійної нелінійності тангенса

Далі ми коротко обговоримо загальні способи ініціалізації ваги на практиці, а також їх мінуси та плюси.

Ініціалізація всіх нулів. У класичному алгоритмі машинного навчання нормально та інтуїтивно зрозуміло, що ми можемо встановити всі ваги $w_1, w_2, \ldots, w_n$ рівними нулю, оскільки доцільно вважати, що приблизно половина ваг буде позитивною, а половина - негативною після відповідної попередньої обробки та нормалізації даних. Однак нульова ініціалізація була б проблематичною в архітектурі НМ, оскільки, якщо кожен нейрон мережі видає однакові результати, тоді всі вони обчислюють однакові градієнти під час зворотного розповсюдження і зазнають однакових оновлень параметрів. Зрештою, кожен нейрон вивчить однакові ваги.

Ініціалізація випадкового числа. Досить прямо, замість ініціалізації всіх нулів, ми все ще хочемо встановити всі ваги нейронів, близьких до нуля. Тож ми могли випадковим чином призначити невелике число кожній вазі. Ідея полягає в тому, що замість 0, усі нейрони на початку випадково присвоєні та унікальні, тому вони будуть обчислювати різні градієнти під час зворотного розповсюдження та отримувати чіткі оновлення. Типова реалізація на практиці для однієї вагової матриці може виглядати приблизно так: $\mathbf{W} = 0,01 * \texttt{np.random.randn}(D, M)$, де в Numpy, функція **randn** з нульового середнього, до одиничного стандартного відхилення від гауссового розподілу. Зверніть увагу, що, як правило, не обов'язково, що менші цифри працюватимуть краще. Наприклад, шар НМ, що має дуже малі ваги, під час зворотного розповсюдження обчислюватиме дуже малі градієнти для своїх даних (оскільки цей градієнт пропорційний значенню ваг). Це може різко прибрати " градієнтний сигнал ", що тече назад через мережу, і може стати проблемою для глибоких мереж.

З цієї причини ця стратегія працює на практиці для малих мереж, але спричинить проблеми з дедалі глибшими мережами. Інтуїція полягає в тому, що уявіть, що ми ініціалізуємо всі ваги до невеликої кількості, і при глибокій мережі під час пересилання, коли ми множимо маленькі $\mathbf{W}$ на кожному шарі, це швидко стискається і згортається всі ці значення, коли ми множимо ваги знову і знову. Врешті-решт, ми отримуємо всі активації нульовими, а це не те, що ми

хочемо.

Ініціалізація з калібруванням дисперсії. Однією з проблем, наведених вище, є що розподіл виходів з випадково ініціалізованого нейрона має дисперсію, яка буде зростати із збільшенням кількості входів. Ініціалізація Ксав'єра полягає в тому, щоб полегшити цю проблему, нормалізуючи дисперсію вихідного сигналу кожного нейрона до 1, масштабуючи його вектор ваги за квадратним коренем його In-вимірності (тобто за кількістю входів). Більш конкретно, рекомендація полягає у ініціалізації вектора ваги кожного нейрона як $\boldsymbol{w} = \texttt{np.random.randn}(n)/\texttt{np.sqrt}(n)$, де n - кількість його входів. Ескіз виведення такий: Розглянемо внутрішній добуток $s = \sum_i^n \boldsymbol{w}_i \boldsymbol{x}_i$, між вагами $\boldsymbol{w}$ та введенням $\boldsymbol{x}$, що дає вихідну активацію нейрона перед нелінійністю. Ми можемо вивчити дисперсію s:

$$
\begin{aligned}
Var(s) &= Var(\sum_i^n \boldsymbol{w}_i \boldsymbol{x}_i) \\
&= \sum_i^n Var(\boldsymbol{w}_i \boldsymbol{x}_i) \\
&= \sum_i^n [E(\boldsymbol{w}_i)]^2 Var(\boldsymbol{w}_i) + E[(\boldsymbol{x}_i)]^2 Var(\boldsymbol{w}_i) + Var(\boldsymbol{x}_i) Var(\boldsymbol{w}_i) \\
&= \sum_i^n Var(\boldsymbol{x}_i) Var(\boldsymbol{w}_i) \\
&= (nVar(\boldsymbol{w})) Var(\boldsymbol{x})
\end{aligned}
$$

На третьому кроці ми припускаємо, що дані нормалізувались із нульовим середнім значенням входів та ваг, тому $E[\boldsymbol{x}_i] = E[\boldsymbol{w}_i] = 0$. З наведеного вище рівняння це означає, що нам слід взяти з одиниці Гаусса, а потім масштабувати її на $a = \sqrt{1/n}$, щоб зробити її дисперсію $1/n$. Це дає ініціалізацію $\boldsymbol{w} = \texttt{np.random.randn}(n)/\texttt{np.sqrt}(n)$.

Ініціалізація Ксав'є. Ініціалізація Ксав'є є ефективним методом ініціалізації ваг, які, як було доведено, мають ефективні результати на практиці, що пропонується Ксав'є Глоро та Йошуа Бенджо в Glorot and Bengio [2010]. У статті автор стверджував, що коефіцієнт нормалізації може бути важливим при ініціалізації глибинних мереж через мультиплікативний ефект через шари, і вони пропонують процедуру ініціалізації, щоб приблизно задовольнити цілі збереження дисперсій активації та дисперсії градієнтів, що розповсюджуються назад при русі вгору або вниз по мережі, що полягає у підтримці сигналу в розумному діапазоні значень через багато шарів. Вага ініціалізується розподілом Гауса:

$$
\mathbf{W} \sim N\big[0, \frac{2}{n_{in} + n_{out}}\big]
$$

що дає вагу з нульовим середнім значенням та дисперсією $\frac{2}{n_{in}+n_{out}}$. Більше того, у статті автор також представив нормалізовану версію ініціалізації з рівномірного розподілу наступним чином:

$$\mathbf{W} \sim U\big[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\big]$$

Зверніть увагу, що оскільки ініціалізація Xavier спрямована на вирішення проблем зникнення або вибуху градієнта, якщо ви використовуєте ReLU як функцію активації (що не призводить до зникнення чи вибуху градієнтів), зазвичай ініціалізація Xavier не використовується з функцією активації ReLU.

Ще одна ініціалізація на практиці. Недавня стаття на цю тему, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification від He et al. [2015], отримує ініціалізацію спеціально для нейронів ReLU, дійшовши висновку, що дисперсія нейронів у мережі повинна становити $2,0/n$. Це дає ініціалізацію $\boldsymbol{w} = \texttt{np.random.randn}(n)*\texttt{np.sqrt}(2.0/n)$, і є поточною рекомендацією для використання на практиці в конкретному випадоку нейронних мереж з нейронами ReLU. На практиці метод продемонстрував свою надійність та ефективність для підрозділів ReLU. Можна уважно поглянути на папір, щоб докопатися до деталей.

Ініціалізація зсувів. Ініціалізувати зсуви, що рівні нулю, можливо і поширено на практиці, оскільки порушення асиметрії забезпечується малими випадковими числами у вагах. Що стосується нелінійності ReLU, деякі люди люблять використовувати невелике постійне значення, таке як 0,01, для всіх зсувів, оскільки це гарантує, що всі одиниці ReLU спрацьовують на початку і, отже, отримують і поширюють певний градієнт. Однак незрозуміло, чи це забезпечує постійне вдосконалення (насправді деякі результати, здається, вказують на те, що це працює гірше), і частіше просто використовувати ініціалізацію зсуву 0.

4.9 Функції нелінійної активації

Для введення нелінійності в нейронні мережі використовуються різні функції активації. Це часто необхідно, оскільки без нелінійності в багатошаровій мережі підсумовування всіх лінійних функцій на кожному рівні просто дасть іншу лінійну функцію, яка змушує мережу поводитися так само, як одношарова мережа. Це також має значення з біологічної точки зору. Активаційні функції - це абстракції, що представляють швидкість потенційної роботи в клітинах, іншими словами, накільки довго ми повинні передавати сигнал наступному нейрону.

Щоб додати нелінійність, функції активації беруть виходи з попереднього рівня і виконують певну нелінійну математичну операцію над ним. Ось деякі загальновживані функції активації.

4.9.1 Сигмоїдальна

Сигмоїдальна функція, також відома як функція логістичної активації. Функція приймає на вхід справжнє значення та перетворює його на значення від 0 до 1.

$$\sigma(x) = \frac{1}{\big(1+\exp(-x)\big)} = \frac{\exp(x)}{\big(1+\exp(x)\big)}$$

Зокрема, вона перетворює великі від'ємні числа в 0, а великі додатні числа в 1. Історично вважалося, що сигмоїдальна функція має хорошу інтерпретацію, оскільки її перетворення має

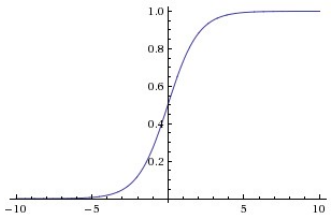


Рис. 11: Ділянка сигмоїдальної функції

діапазон від 0 до 1, що добре відповідає показникам. Це також корисно на вихідному рівні, якщо метою є передбачення ймовірності.

However, the function does have some drawbacks in practice. The output is not zero-centered, which lead to inefficient gradient update. It will encounter vanishing gradients for values near 1 and 0. It is also computationally expensive by taking exponentials.

Zero-centering

Zero-centering is the process of subtracting the mean from the data along all dimensions. Thus the new mean becomes zero. This ensures that there are both positive and negative values.

$$
\begin{aligned}
L &= \text{Loss function} \\
\boldsymbol{x} &= \text{Output of sigmoid unit} \\
\boldsymbol{y} &= \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{b} \\
\frac{\partial L}{\partial \boldsymbol{w}} &= \frac{\partial L}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}} = \frac{\partial L}{\partial \boldsymbol{y}} x \qquad\qquad (x > 0)
\end{aligned}
$$

In sigmoid, the output is always positive and hence the gradient on the weights w will during backpropagation become either all be positive, or all negative depending on the sign of the gradient of the loss w.r.t y. This will have undesirable zig-zag path during optimization as all weights must be updated in the same direction and cannot be manipulated independently. The algorithm will take a lot more steps to converge.

Vanishing Gradients Problem

Sigmoid suffers from the problem of vanishing gradients, i.e., it has close to zero gradients almost everywhere. Since it squashes the entire number line into the interval [0,1], it starts from increasing from zero but then starts decreasing to zero again shortly, and so the gradients are quite small and would be even smaller after the stacking of multiple layers. The result of the gradients saturating is that our weights stop getting updated, and hence our network stops learning. This is one of the main reasons why "deep"architectures were only popular "in theory"in the early days – it was proposed that they could solve any problem *in theory*, but in practice it was very difficult to achieve this.

4.9.2 Hyperbolic Tangent (tanh)

The tanh function is very similar to Sigmoid except it takes a real-valued number but convert it into a range between -1 and 1. Or it can also be seen as a combination of 2 sigmoid functions. After the conversion, the negative inputs will still be negative, zero inputs are mapped near zero, and the positive inputs are still positive.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{\exp(2x) - 1}{\exp(2x) + 1} = 2\sigma(2x) - 1$$

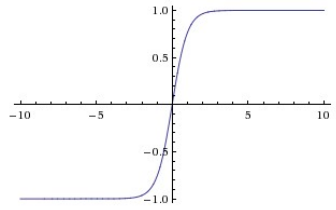


Рис. 12: Plot of the Hyperbolic Tangent Function

Unlike Sigmoid, the outputs of tanh are zero-centered, which makes it a more preferable choice in practice. But it still suffers from the same vanishing gradient problem described earlier as in Sigmoid; the following activations aim at getting rid of this issue.

4.9.3 ReLU

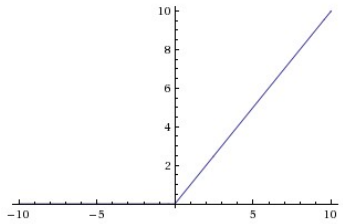


Рис. 13: Plot of ReLU Function

ReLU function (**Re**ctified **L**inear **U**nit), takes a simple expression. It is just thresholding at 0. When the input is negative, the output is 0, and when input is positive, the output is just the input itself.

$$f(x) = \max(0, x)$$

ReLU is very computationally efficient as it is just thresholding without calculating exponentials, it also solves the vanishing gradient issue when inputs are positive. However, ReLU outputs are not zero-centered, and the gradient vanishes when input is negative. The thresholding also makes from a biological perspective, wherein neurons fire only when the input signal is above a certain threshold. In this case of a positive input, we simply propagate it as-is, and hence it doesn't suffer from the vanishing gradient problem.

4.9.4 Leaky ReLU

This is used to fix the vanishing gradient problem of ReLU for negative numbers. Instead of bringing the negative value to zero, we allow a small, non-zero gradient when the unit is not active.

$$f(x) = max(x, \alpha x), \text{where } \alpha \text{ is a constant, say } 0.01$$

In parameterized ReLU, we can also replace the constant gradient with a variable gradient which can be learnt along with other parameters of the network.

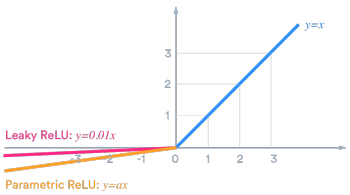


Рис. 14: Plot of Leaky ReLU and PrReLU Function

4.9.5 ELU

ELU (**E**xponential **L**inear **U**nit) is a new attempt to fix the vanishing gradient and not zero-centering issues of ReLU. It is the same as ReLU when the inputs are positive and it takes exponential when inputs are negative. By doing this, the mean will be pushed toward zero and also it will have a small gradient when inputs are negative. Note that ELU has an extra α constant which should be a positive number in practice.

$$f(x) = \begin{cases} x & \text{for } x \geq 0 \\ \alpha(e^x - 1) & \text{for } x < 0. \end{cases}$$

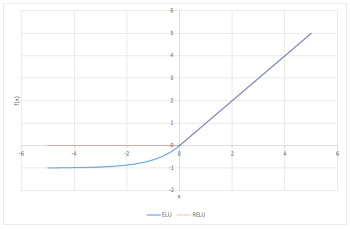


Рис. 15: Plot of ELU and ReLU Function

4.10 Функції втрат

4.10.1 Евклідова функція втрат

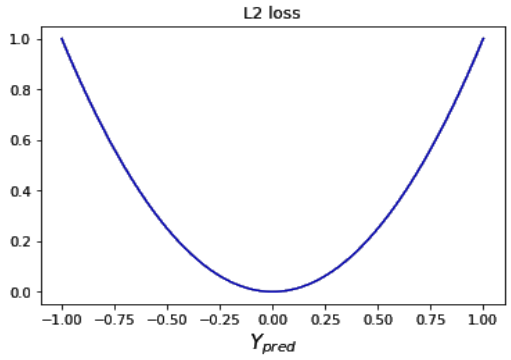


Рис. 16: Графік функції втрат L2

Спершу поговоримо про звичайні функції втрат для вирішення проблем регресії, що безпосередньо прогнозують цільові значеня замість прогнозування цільових міток. Для цих цілей найчастіше застосовуються Евклідова функція втрат і функція втрат L2. Що є квадратичними різницями між розрахунковими значеннями та цільовими значеннями, або $l2$-нормами помилки:

$$\mathcal{L} = \sum_i (\hat{y}_i - y_i)^2 = \|\hat{y} - y\|_2^2$$

Слід зазначити, що квадратична складова функція втрат L2 робить її особливо вразливою до неординарностей. Щоб уникнути цієї чутливості, часто використовується середньоквадратичне відхилення, яке є квадратним коренем стандартної функція втрат L2:

$$\mathcal{L} = \sqrt{\sum_i (\hat{y}_i - y_i)^2} = \|\hat{y} - y\|_2$$

Іншим поширеним варіантом є використання середньоквадратичної логарифмічної функції втрати, яка використовується, щоб уникнути чутливості до неординарностей за умови, що і справжні цільові значення, і оціночні значення є великими числами. Іще однією особлиістю є вище покарання за заниження оцінки, ніж за завищення оцінки, у порівнянні з функцією втрат L2.

$$\mathcal{L} = \sum_i (\log(\hat{y}_i + 1) - \log(y_i + 1))^2$$

4.10.2 Функція втрат L1

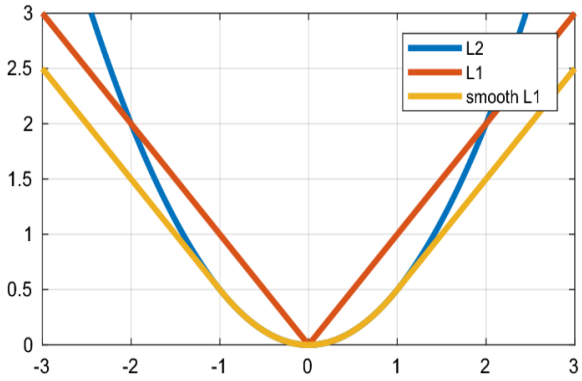


Рис. 17: Функція втрат L1, L2 та згладжена L1

Функція втрат L1 - це абсолютна різниця між розрахунковими значеннями та існуючими цільовими значеннями. Іншими словами, це $l1$-норма помилки. Підсумовуючи кожне цільове значення y_i та відповідну оцінку $\hat{y}_i$, функція втрат L1 виражається як

$$\mathcal{L} = \sum_i |\hat{y}_i - y_i| = \|\hat{y} - y\|_1$$

Функції втрат L1 іноді надають перевагу у порівнянні з функцією втрат L2, особливо коли ознака має великі розміри. Це пов'язано з тим, що функції втрат L1 є більш стійкою до неординарностей і дає більш розріджене рішення, ніж функції втрат L2, що дозволяє спростити вибір ознак і пришвидшити обчислення. Однак зверніть увагу, що похідна функції втрат L1 не існує при 0, тому часто використовується її згладжена версія:

$$\mathcal{L} = \sum_i \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & if|\hat{y}_i - y_i| < 1 \\ |\hat{y}_i - y_i| - \frac{1}{2}, & otherwise \end{cases}$$

Ця версія функції втрат L1 також усуває чутливість до вибухових градієнтів (див. Girshick 2015).

Варто варто запамятати, що функції втрат L1 та L2 при застосуванні для оцінки ймовірностей при використанні функції активації sigmoid (або softmax) матимуть немонотонні часткові похідні відносно виходу останього шару (див. підтвердження у Janocha & Czarnecki, 2017). А у випадку сильно помикової класифікації прикладів, навчання буде значно сповільнено через плоский градієнт на кінцях сигмоїдної функції. Ось чому функції втрат L1 та L2 зазвичай не використовуються разом із функцією активації sigmoid або softmax, а частіше використовують з ReLU.

4.10.3 Розходження Кульбака — Лейблера

В той час як відстань $l2$ вимірює природну відстань між двома точками в евклідовому просторі, розходження Кульбака — Лейблера вимірює різницю між двома розподілами ймовірностей

$p(x), q(x)$:

$$KL(p||q) = \sum_i p(x_i) \log \frac{p(x_i)}{q(x_i)}$$

Зверніть увагу, що розходження Кульбака — Лейблера є невідємною, але асиметричною мірою відстані, тобто $KL(q||p) \neq KL(p||q)$. Розбіжність Кульбака — Лейблера випливає з теорії інформації та поняття ентропії. Ентропія розподілу ймовірностей,

$$H = -\sum_i p(x_i) \log p(x_i)$$

є оцінкою кількості інформації, що міститься в даних. Використовуючи поняття ентропії, ми можемо визначити кількість інформації, що буде втрачено, якщо замінити наш початковий розподіл параметризованим наближенням, просто подивившись різницю значень журналу для кожного розподілу, вимірюючи тим самим, наскільки згадане наближення відрізняється від очікуваного розподіл ймовірностей.

4.10.4 Функція втрат на основі перехресної ентропії - Двійкова

Розглянемо випадок коли ми маємо набір даних для навчання з q(x), який є дискретним розподілом ймовірності спостережуваних міток $y \in \{0, 1\}$. Якщо наша модель вибере p(x) як розподіл прогнозу, тобто, $p(y = 1|x) = \hat{y}(x)$, ми можемо розрахувати розходження Кульбака — Лейблера між p(x) і q(x) як

$$\begin{aligned} KL(q||p) &= \sum q(y \mid x) \log \frac{q(y \mid x)}{p(y \mid x)} && (18) \\ &= -\sum q(y \mid x) \log p(y \mid x) + \sum q(y \mid x) \log q(y \mid x) && (19) \\ &= -q(y = 1 \mid x) \log p(y = 1 \mid x) - q(y = 0 \mid x) \log p(y = 0 \mid x) && (20) \\ &= -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) && (21) \end{aligned}$$

Це також є перехресною ентропією. Тому значення втрати з використанням перехресної ентропії визначають відстань між емпіричним розподілом даних та прогнозованим розподілом за моделлю. На практиці, виконуючи двійкові класифікації з використанням нейронних мереж, ми зазвичай додаємо сигмоїдний або логістичний шар в кінці, щоб отримати оцінку ймовірності для кожного класу як p і $1-p$, перед тим, як розрахувати втрату на основі перехресної ентропії.

4.10.5 Функція втрат на основі перехресної ентропії — Багатокласова

Функція втрат на основі перехресної ентропії можна легко узагальнити до багатокласових класифікаційних задач, де серед усіх можливих класів існує лише один справжній клас. Коли є мітки класу k, для кожного класу c цільові значення все ще виражаються е двійковому вигляді:

$$y_i = \begin{cases} 1, & \text{if } i = c \\ 0, & \text{otherwise} \end{cases}$$

Нехай p_i є ймовірністю кожного класу, тоді $\sum p_i = 1$. Для отримання таких імовірнісних оцінок для багатокласної задачі замість сигмопоїдного шару в попередньому випадку використовується

шар softmax:

$$p(y = c|x) = \frac{\exp z_c}{\sum_j \exp z_j}$$

Отже, багатокласова функція втрат на основі перехресної ентропії має наступний вигляд

$$\mathcal{L} = -\sum_i^k y_i \log \hat{y}_i$$

4.10.6 Багатоміткова функція втрат на основі перехресної ентропії

Подібним чином існує також варіант функції втрат на основі перехресної ентропії, який можна застосувати до типу багатокласових класифікацій, де кілька міток можуть мати значення 1 одночасно, і в цьому випадку функція softmax більше не застосовується. Типовою практикою є повернення до використання сигмоїдної функції активації на вихідному шарі мережі, так що оцінювані ймовірності кожного класу не залежать від ймовірностей інших класів. У цьому випадку наша функції втрат на основі перехресної ентропії приймає вигляд:

$$\mathcal{L} = -\sum_i^k y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

4.10.7 Завісні втрати та їх варіанти

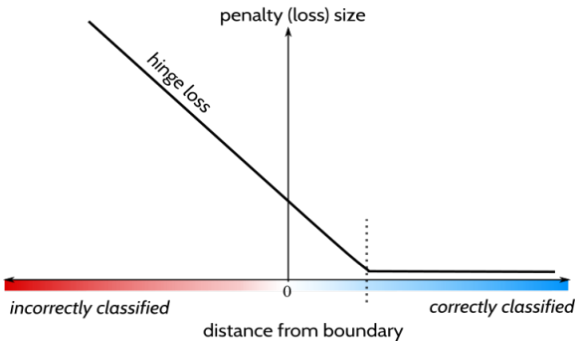


Рис. 18: Графік завісних втрат

Завісна втрата використовуються для розділення позитивних та негативних прикладів, найчастіше у методі опорних векторів. Для кожного прикладу x_i з двійковою міткою $y_i \in \{-1, 1\}$ завісні втрата передбачення $\hat{y}_i$ визначається як

$$\mathcal{L} = \max(0, 1 - y * \hat{y})$$

У цьому формулюванні, щоб мінімізувати втрати, в усіх позитивних прикладах має виконуватись співвідношення $\hat{y} > 1$, а у всих негативних $\hat{y} < -1$.

Зверніть увагу, що завісна втрата не усюди диференційована, тому її часто згладжують шляхом піджесення до квадрату:

$$\mathcal{L} = (\max(0, 1 - y * \hat{y}))^2$$

Для багатокласових задач з k-класами доступний інший варіант завісної втрати:

$$\mathcal{L} = \sum_{c}^{k} \max(0, 1 - \mathbb{1}_{y,c} * \hat{y})$$

де $\mathbb{1}_{y,c} = \begin{cases} 1, & \text{if } y = c \\ -1, & \text{otherwise} \end{cases}$

4.10.8 Об'єднані включеня

Для вирішення багатьох завдань у галузі глибокого навчання дані часто надходять з багатьох джерел. При використанні даних з двох різних джерел (наприклад, тексту та зображення) для спільного прогнозування, функції втрат, як правило, формулюються так щоб враховувати включення (багатовимірний простір ознак) з обох джерел даних, а також міри подібності для тренування двох включень одночасно.

Зокрема, функція f відображає x_1, дані з джерела 1 на включення z_1 у багатовимірний простір, з вагами w_1; функція g відображає x_2 на включення z_2 з вагами w_2. Функція подібності відображає z_1, z_2 до $z_1^\top w_3 z_2$, з іншим набором ваг w_3. Тоді оптимізація спільного вкладення еквівалентна максимізації (log) ймовірності

$$p_{w_1,w_2,w_3}(s_{1,2}|x_1, x_2) = \int p_{w_1}(z_1|x_1) p_{w_2}(z_2|x_2) p_{w_3}(s_{1,2}|z_1, z_2) dz_1 dz_2 \tag{22}$$

$$\geq \max_{z_1,z_2} p_{w_1}(z_1|x_1) p_{w_2}(z_2|x_2) p_{w_3}(s_{1,2}|z_1, z_2) \tag{23}$$

де $s_{1,2} = [y_1 == y_2]$, ступінь подібності.

Звідси втрата від спільного вбудовування приймає вигляд

$$\mathcal{L} = \max_{z_1,z_2} \log p_{w_1}(z_1|x_1) + p_{w_2}(z_2|x_2) + p_{w_3}(s_{1,2}|z_1, z_2) \tag{24}$$

$$\tag{25}$$

4.11 Геометрія функції втрат

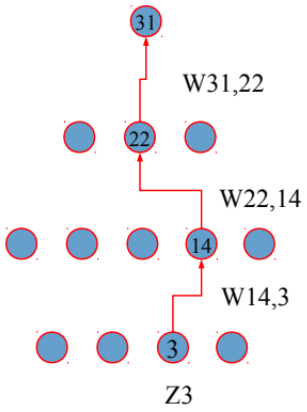


Рис. 19: Глибока мережа з ReLU

Питання, на яке ми намагаємося відповісти, полягає в тому, чи можемо ми охарактеризувати геометрію функції втрат у глибоких нейронних мережах. Візьмемо повнозв'язну нейронну мережу з функцією активації ReLU. Внесок окремого шляху, P, мережі у вихідний результат - це добуток входу, $\boldsymbol{W}_{ij}$ та ваг вздовж шляху, $\boldsymbol{W}_{ij}$ і двійкової індикаторної змінної $\delta_P(\boldsymbol{W}, \boldsymbol{X})$, що приймає значення True, якщо ReLU активовано у всіх шарах, або False в інших випадках.

$$\hat{\boldsymbol{Y}} = \sum_P \delta_P(\boldsymbol{W}, \boldsymbol{X}) \prod_{ij \in P} \boldsymbol{W}_{ij} \boldsymbol{X}_{P_{start}}$$

Ми використаємо завісні втрати, щоб зробити двійкову змінну залежною від бажаного результату. Таким чином, коефіцієнт, C_P, буде включати вхід $\boldsymbol{X}$, вихід $\boldsymbol{Y}$ і вагу $\boldsymbol{W}$.

$$L(\boldsymbol{W}) = \sum_P C_P(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}) \prod_{ij \in P} \boldsymbol{W}_{ij}$$

Будь-яка зміна в $\boldsymbol{W}$ вирішить, активний чи неактивний шлях, і, отже, вплине на C_P. Функція втрат - це поліном у $\boldsymbol{W}$, а градус - це кількість шарів. Даний поліном є поштучно безперервним з "перемиканням"(тобто, він може бути увімкнутим або вимкнутим залежно від значення $\boldsymbol{W}$) та має частково випадкові коефіцієнти.

Згідно з *Теорією випадкових матриць*, для нормованого $\boldsymbol{W}$ існує велика кількість мінімумів, які майже повністю еквівалентні, тобто значення функції втрат буде подібним. Крім того, кількість місцевих мінімумів, що мають високу енергію, дуже мала.

Щоб краще це зрозуміти, давайте припустимо, що ми починаємо випадково в просторі пошуку. Тоді немає шансів, що ми потрапимо в місцеві мінімуми з високою енергією, оскільки їх дуже важко знайти. Однак ми також не досягнемо глобальних мінімумів, оскільки застрягмо на попередньому рівні через величезну кількість таких точок у вузькому діапазоні. Експериментально це вірно, якщо ми тренуємо дуже велику нейронну мережу з правильним налаштуванням гіперпараметрів, ми досягаємо тієї ж помилки в межах щільного розмаху.

4.12 Заповнення

Під час процесу згортки те, що ми робимо, фактично є зменшенням розміру вхідної матриці. Розмір вихідної матриці після одного шару згортки визначається кроком та розміром встановленого вами фільтра. Крок - це гіперпараметр, за допомогою якого ми зсуваємо фільтр. Коли крок рівний 1, ми переміщуємо фільтри по одному пікселю за раз. Але коли крок будь-яке число більше за 1, результат буде меншим. Тому нам потрібно "збільшити"нашу вхідну матрицю перед розрахунком. Процес називається "заповненням".

В деяких ситуаціях зручно заповнювати вхід нулем навколо межі, що ми називаємо "нульовим заповненням". Розмір "нульового заповнення це гіперпараметр. Заповнення нулем дозволяє нам легко контролювати розмір нашої вихідної матриці.

Наприклад на вході ми маємо матрицю розіром 7×7, і фільтр 3×3. Припустимо, наш крок становить 1, після одного рівня згортки вихідна матриця має розмір 5×5. Коли крок дорівнює 2, розмір вихідної матриці становить 3×3. Таким чином, ми можемо отримати формулу з допомогою індукції:

Коли $stride = 1$,

$$\frac{(7-3)}{1} + 1 = 5$$

Коли $stride = 2$,

$$\frac{(7-3)}{2} + 1 = 3$$

Коли $stride = 3$ (не працює у цьому випадку),

$$\frac{(7-3)}{3} + 1 = 2.33$$

Нарешті ми маємо,

$$\frac{N-F}{stride} + 1 = output\ size.$$

Виходячи з наведеної вище формули, ми можемо зробити припущення, скільки нулів нам потрібно заповнити.

Припустимо, у нас все ще є вхідна матриця 7×7 та фільтр 3×3, якщо ми заповнимо 0 на 1 пікселів (двосторонні) межу, вихідний розмір буде знову 7×7.

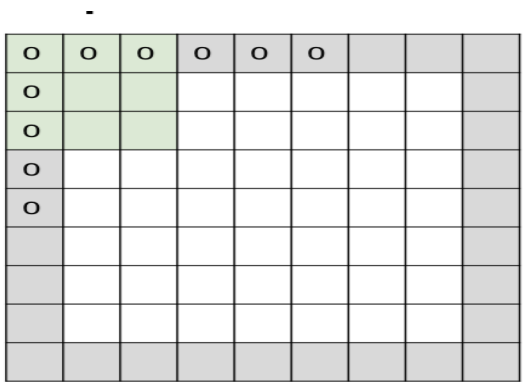


Рис. 20: Заповнення

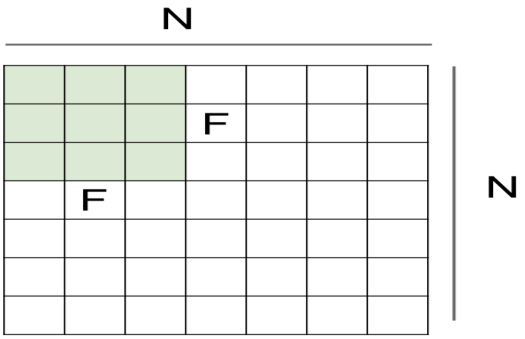


Рис. 21: параметри

В загальному випадку, якщо ми хочемо зберегти розмір вхідної матриці, формула має мати вигляд

$$W = \frac{W - F + 2P}{S} + 1,$$

Тут розмір фільтра - F, W - розмір входу, P - розмір заповнення, S - крок.

$$P = \frac{(W-1) * S - (W - F)}{2}$$

В $Tensorflow$ існує два методи заповнення: $SAME$ і $VALID$. $SAME$ означає, що вихідна карта ознак має такі ж просторові розміри, що і вхідна карта ознак. Нульове заповнення вводиться для того, щоб форма збігалася з необхідною, однаковою з усіх боків вхідної карти. $VALID$ означає відсутність заповнення.

У *Pytorch* існує три методи заповнення: *Reflection pad*, *Zero pad*, подібний до методу заповнення *SAME* в *Tensorflow*, *Constant pad*. *Reflection pad* - це метод, який накладає вхідний тензор, використовуючи відображення вхідної межі. *Constant pad* встановлює границі вхідного тензора з постійним значенням.

5 Цільове розповсюдження

5.1 Введення

Останнім часом глибокі нейронні мережі досягли великих успіхів у складних задачіх штучного інтелекту[4],в основному покладаючись на зворотне поширення в якості основного режиму для кожного шару з глибокими рівнями. Пов'язані з цим різні набори параметрів виконують розподіл кредитів.

Однак, коли ми розглядаємо більш глибокі мережі, наприклад, останні кращі конкуренти ImageNet увійшли на 19й або 22й рівень, покладаючись на довгострокову або сильнішу нелінійність, склад багатьох нелінійних операцій стає більш нелінійним. Конкретизуюємо, розглянувши склад багатьох гіперболічних тангенціальних одиниць. В цілому, це означає, що похідні, отримані в результаті зворотної торгівлі, стають дуже малими або надто великими.

Це обмеження співпраці з точними похідними і гладкими мережами є основною мотивацією машинного навчання для вивчення розподілу кредитів в глибоких мережах. Основна ідея тут полягає в обчисленні цілей, а не градієнтів на кожному шарі, які можуть бути ефективно застосовані, коли одиниці обмінюються стохастичними бітами.

5.2 Реалізація

Основна ідея розповсюдження цілі полягає в тому, щоб замість втрати градієнта пов'язати значення активації кожної одиниці перенаправлення з цільовим значенням. Цільове значення означає близьке до активації значення і яке наносить менше шкоди.

5.2.1 Формулювання цілей

Розглянувши процес навчання звичайного глибокої нейронної мережі, ми могли б визначити структуру як[5]:

$$h_i = f_i(h_{i-1}) = s_i(W_i h_{i-1}), i = 1, ..., M$$

в якому h_i - прихований шар, у той час як h_M - останній шар, тобто вивід мережі, і h_0 - це те, що вводимо, x, s_i - функція активації, W_i - вага i-го шару, а f_i є відображенням прямого

[4]Bengio, 2009; Hinton et al., 2012; Krizhevsky et al., 2012; Sutskever et al., 2014

[5]Lee, Dong-Hyun, et al. "Difference Target Propagation." Difference Target Propagation, ArXiv, 25 Nov. 2015, arxiv.org/abs/1412.7525.

подання зазначеного шару.

Навчання мережі із зворотним розповсюдженням відповідає поширенню сигналів помилок по мережі. У дуже глибоких мережах із сильною нелінійністю поширення помилок може стати марним у нижчих шарах через труднощі, пов'язані з сильною нелінійністю. Одним із можливих способів вирішення цих помилок може бути призначення "цільового"$\hat{h}$, який міг би задовольнити умові:

$$loss(h_M(\hat{h}_i; \theta_W^{i,M}), y) < loss(h_M(h_i(x; \theta_W^{0,i}); \theta_W^{i,M}), y)$$

З втратою MSE ми маємо:

$$\text{L}_i(\hat{h}_i, h_i) = ||\hat{h}_i - h_i(x; \theta_W^{0,i})||_2^2$$

Розглянемо функцію втрат, ми можемо отримати правило оновлення для W_i з градієнтом, розглядаючи $\hat{h}$ як константу:

$$W_i^{(t+1)} = W_i^{(t)} - \eta_{f_i} \frac{\partial L_i(\hat{h}_i, h_i)}{\partial W_i} = W_i^{(t)} - \eta_{f_i} \frac{\partial L_i(\hat{h}_i, h_i)}{\partial h_i} \frac{\partial h_i(x; \theta_W^{0,i})}{\partial W_i}$$

За допомогою цього виразу ми можемо використовувати похідні в якомусь локальному шарі, оскільки більшість обчислювальних робіт виконуються в нейронах. Оскільки при застосуванні зворотного розповсюдження через багато шарів завжди виникає нелінійність, це вирішить більшість цих проблем. Для розповсюдження цілі нам потрібно знайти $\hat{h}_{i-1}$ з задачі вищого рівня $\hat{h}_i$ та з h_i. Тож, швидше за все, буде задоволено слабше припущення:

$$L(\hat{h}_i, f_i(\hat{h}_{i-1})) < L_i(\hat{h}_i, f_i(h_{i-1}))$$

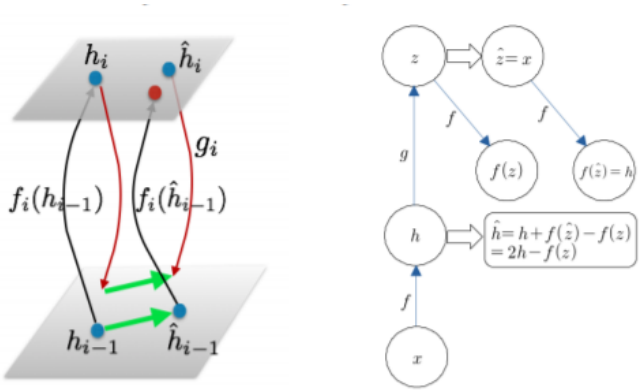


Рис. 22: (ліворуч) Як обчислити ціль у нижньому шарі за допомогою поширення цілі різниці. $f_i(h_{i-1})$ має бути ближче до h_i, ніж $f_i(h_{i-1})$. (праворуч) Діаграма зворотного розмноження автокодер за допомогою розповсюдження цільової різниці

5.2.2 Правильне призначення цілей кожному шару

Питання про присвоєння кредиту, яке ми повинні розглянути, полягає в тому, як призначити належну ціль кожному шару, щоб забезпечити збільшення ймовірності глобального зменшення збитків.
Згадаймо зворотне розповсюдження, яке дозволяє обчислювати градієнт втрат w.r.t на виході кожного шару, використовуючи правило ланцюга. Обчислюваний нами градієнт можна розглядати як сигнал помилки, який поширюється рекурсивно від вихідного шару до вхідного шару. Виразок обчислення такий:

$$\delta \mathbf{h}_{i-1} = \frac{\partial L}{\partial \mathbf{h}_{i-1}} = \frac{\partial L}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \delta \mathbf{h}_i \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}$$

коли в налаштуваннях цільової підтримки різниця між $\hat{h}$ і h визначає напрямок оновлень. Таким чином, ми можемо замінити $\delta \mathbf{h}_{i-1}$ і $\delta \mathbf{h}_i$ попереднього рівняння різницею наступним чином:

$$\hat{\mathbf{h}}_{i-1} - \mathbf{h}_{i-1} = (\hat{\mathbf{h}}_i - \mathbf{h}_i) \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \frac{1}{2} \frac{\partial ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2}{\partial \mathbf{h}_{i-1}}$$

Оновлення параметрів на кожному шарі в налаштуваннях цільової підтримки можна отримати за допомогою кроку стохастичного градієнтного спуску та записати наступним чином:

$$W_i^{(t+1)} = W_i^{(t)} - \eta \frac{\partial ||\hat{\mathbf{h}}_i - \mathbf{h}_i||_2^2}{\partial W_i}$$

Відповідно до рівняння, ми можемо переписати як

$$L_i(\hat{\mathbf{h}}_i, f_i(\hat{\mathbf{h}}_{i-1}) < L_i(\hat{\mathbf{h}}_i, f_i(\mathbf{h}_{i-1})$$

З наведеного вище виразу розповсюдження цілі можна розглядати, оскільки ціль нижнього шару обчислюється з цілі верхнього шару так, ніби градієнти зниження застосовані до шару. шарів, існує інший метод, розроблений Бенджо [6] щоб використати «приблизний зворот». Наприклад, припустимо, що ми маємо функцію g_i яка

$$f_i(g_i(\hat{\mathbf{h}}_i)) \approx \hat{\mathbf{h}}_i$$

Заміна $\hat{\mathbf{h}}_{i-1} = g_i(\hat{\mathbf{h}}_i)$ який є ваніллю поширення цілі, представлене Бенджо. Одне, на що варто звернути увагу, це те, що g_i інвертує f_i лише поблизу цілей. Якщо ми маємо $g_i = f_i^{-1}$, це означає, що відображення зворотного зв'язку було якраз оберненим до прямої передачі, то ми отримуємо ідеальне розповсюдження цілі наступним чином:

$$L_i(\hat{\mathbf{h}}_i), f_i(\hat{\mathbf{h}}_{i-1})) = L_i(\hat{\mathbf{h}}_i, g_i(\hat{\mathbf{h}}_i)) = L_i(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_i) = 0$$

Як тільки ми знаємо, як призначити правильні цілі для кожного шару, тоді єдине, що нам потрібно зробити, це мінімізувати локальні цільові втрати на рівні для мінімізації глобальних втрат.

[6] Bengio, Y., Thibodeau-Laufer, E., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In ICML'2014.

6 Чи завжди “глибше” краще?

Якщо коротко, ні. Наприклад, є три різні випадки, коли неглибокі мережі працюють краще, тому що їх можна навчити краще завдяки їхній низькій обчислювальній вартості.

“Прості моделі можуть бути дуже потужними, оскільки вони набагато швидші, і їх можна навчити на набагато більшій кількості даних”, роблячи модель менш складною та змушуючи її навчатися на більшій кількості даних, часто можна отримати кращі результати.

6.1 Вбудовування слів

Що таке моделювання мови? В основному, враховуючи оточуючі слова, передбачення слова. Отже, в основному вам необхідний розподіл. Традиційно ми використовуємо n-грамні моделі. Функція втрат: виконує функцію Softmax (нормована експоненційна функція) над усіма словами означає, що потрібно підсумувати всі слова, що дійсно дорого.

6.2 `word2vec`

Mikolov et al. [2013b] представив моделі Continuous Bag of Words (CBOW) та Skip-gram як обчислювально ефективні методи для розробки високоякісного векторного представлення слів та словосполучень із неструктурованого тексту. Обидві моделі використовують гіпотезу розподілу - ідею в лінгвістиці, згідно з якою слова, що трапляються в подібному контексті, матимуть схоже значення.[7] У контексті вбудовування слів це означає, що гарне векторне представлення слів повинно мати слова, які з’являються в подібному контексті “близько” один до одного у просторі вбудовування. І модель CBOW, і Skip-gram використовують архітектуру нейронної мережі з єдиним прихованим шаром, щоб відповідати вбудованим словам відповідно до розподільчої гіпотези. Однак цільові функції моделей різні.

CBOW навчається передбачати цільове слово, враховуючи контекстне вікно перед словом та

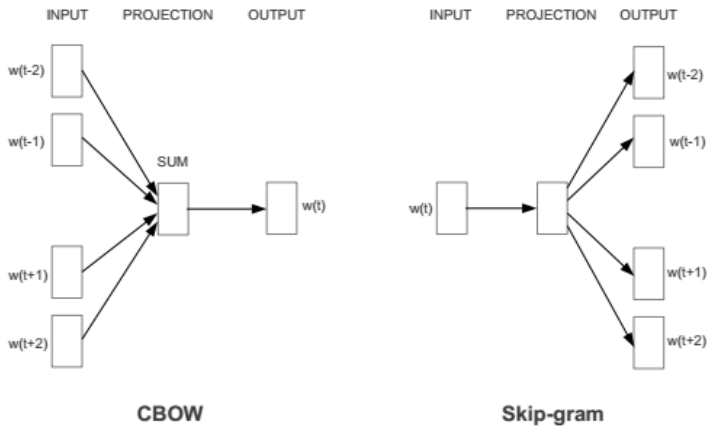


Рис. 23: CBOW та Skip-Gram моделі

[7] Дивись Distributional approaches to words meanings

контекстне вікно, що слідує за словом. Вхідними даними для моделі є набір навчальних речень, де кожне слово представлено у вигляді унітарного вектора. Нехай $\mathcal{V} \in \mathbb{R}^{d \times |V|}$ буде матрицею контекстних слів і $\mathcal{U} \in \mathbb{R}^{|V| \times d}$ буде матрицею цільових слів де $|V|$ - розмір словникового запасу та d - розмірність вбудованого векторного простору. Нехай $\boldsymbol{v}_k = \mathcal{V}\boldsymbol{w}^{(k)}$ де $\boldsymbol{w}^{(k)}$ є унітарним вектором. $\boldsymbol{v}_k$ - вектор, який є поданням $\boldsymbol{w}^{(k)}$ в контекстній матриці. Нехай $\boldsymbol{u}_k = \mathcal{U}^T\boldsymbol{w}^{(k)}$. $\boldsymbol{u}_k$ - вектор, який є поданням $\boldsymbol{w}^{(k)}$ у матриці цільового слова. Нехай $\boldsymbol{w}^{(i)}$ буде унітарним вектором для поточного цільового слова та $\boldsymbol{w}^{(i-c)}, \boldsymbol{w}^{(i-(c-1))}, \ldots, \boldsymbol{w}^{(c+1)}, \boldsymbol{w}^{(c+n)}$ будуть унітарними векторами для контекстних слів для прикладу для моделі CBOW із контекстним вікном розміру c. Нехай $\boldsymbol{w}^{\mathcal{C}_i}$ позначає набір контекстних слів для $\boldsymbol{w}^{(i)}$. Оцінка слова, даного набору контекстних слів, є скалярним добутком між середнім значенням векторів контексту та даним словом:

$$\text{score}(\boldsymbol{w}^{(k)}, \boldsymbol{w}^{\mathcal{C}_i}) = \boldsymbol{u}_k^\top \left(\frac{1}{2c} \sum_{j \in \mathcal{C}_i} \boldsymbol{v}_j \right).$$

Функція softmax використовується для прогнозування ймовірності слова з набором контекстних слів:

$$\text{softmax}(\boldsymbol{w}^{\mathcal{C}_i})_k = \frac{\exp\left(\text{score}(\boldsymbol{w}^{(k)}, \boldsymbol{w}^{\mathcal{C}_i})\right)}{\sum_{k=1}^{|V|} \exp\left(\text{score}(\boldsymbol{w}^{(k)}, \boldsymbol{w}^{\mathcal{C}_i})\right)}.$$

Модель CBOW тренується з використанням перехресних ентропійних втрат, що вказує на відстань між передбачуваним розподілом ймовірностей $\text{softmax}(\boldsymbol{w}^{\mathcal{C}_i})$ і справжнім розподілом, що заданий унітарним вектором відповідно до цільового слова. Повна функція для цільового слова $\boldsymbol{w}^{(i)}$ та контекстні слова $\boldsymbol{w}^{\mathcal{C}_i}$ is

$$\text{J}(\mathcal{U}, \mathcal{V}) = -\text{score}(\boldsymbol{w}^{(i)}, \boldsymbol{w}^{\mathcal{C}_i}) + \log \sum_{k=1}^{|V|} \exp\left(\text{score}(\boldsymbol{w}^{(k)}, \boldsymbol{w}^{\mathcal{C}_i})\right).$$

Останніми вбудованими словами є рядки навченої матриці $\mathcal{U}$.

На відміну від CBOW, **Skip-gram** навчається передбачати попередні та майбутні слова з урахуванням "поточного" слова. Тепер цільові слова - це слова у вікні, що оточує поточне слово, і поточне слово тепер є контекстним. Нехай $\mathcal{V}, \mathcal{U}, \boldsymbol{v}_k$, та $\boldsymbol{u}_k$ будуть матричними та векторними представленнями для контекстного слова та цільових слів, як визначено вище для CBOW. Нехай $\boldsymbol{w}^{\mathcal{T}_i}$ позначимо набір цільових слів для контекстного слова $\boldsymbol{w}^{(i)}$. Тоді оцінка даного слова з урахуванням контекстного слова є скалярним добутком відповідних векторних подань:

$$\text{score}(\boldsymbol{w}^{(j)}, \boldsymbol{w}^{(k)}) = \boldsymbol{u}_k^\top \boldsymbol{v}_j.$$

Зауважте, що $2c$ бали потрібно обчислювати для кожного контекстного слова. Знову ж таки, функція softmax використовується для перетворення оцінок у ймовірності:

$$\text{softmax}(\boldsymbol{w}^{(j)})_k = \frac{\exp\left(\text{score}(\boldsymbol{w}^{(j)}, \boldsymbol{w}^{(k)})\right)}{\sum_{k=1}^{|V|} \exp\left(\text{score}(\boldsymbol{w}^{(j)}, \boldsymbol{w}^{(k)})\right)}.$$

Тепер $2c$ - розраховані вектори ймовірності. Потрібно зробити умовне припущення про незалежність, щоб розкласти на фактори спільну ймовірність цільових слів як добуток їх умовних ймовірностей з урахуванням контекстного слова. Отже, передбачається умовна незалежність цільових слів,

враховуючи контекстні слова. Використовуючи перехресну ентропію, цільову функцію для моделі Skip-gram для контекстного слова $\boldsymbol{w}^{(j)}$

$$\mathrm{J}(\mathcal{U}, \mathcal{V}) = -\sum_{k=0, k\neq j}^{2c} \mathrm{score}(\boldsymbol{w}^{(j)}, \boldsymbol{w}^{(j-c+k)}) + 2c \log \sum_{l=1}^{|V|} \exp\left(\mathrm{score}(\boldsymbol{w}^{(j)}, \boldsymbol{w}^{(k)})\right).$$

Останніми вбудованими словами є рядки навченої матриці $\mathcal{U}$.

Зверніть увагу, що друге підсумовування в цілі Skip-gram надзвичайно дороге з точки зору обчислень. У наступній статті, Mikolov et al. [2013a] впроваджує вдосконалення цієї моделі, щоб зробити її більш обчислювально ефективною. Міколов запропонував два методи для апроксимації другого підсумовування за всіма словами в словниковому запасі. **Hierarchical Softmax** - це обчислювально ефективний метод апроксимації повного softmax за допомогою двійкового дерева.

Більш популярне альтернативне наближення здійснюється за допомогою **Negative Sampling**. Ідея, яка стоїть за негативною вибіркою (Negative Sampling), полягає в тому, що подібні слова повинні бути близькими один до одного у просторі вбудовування та далекими від слів, що відрізняються. Перший крок цільової функції Skip-gram змушує слова, які знаходяться близько один до одного, також бути близькими у просторі вбудовування. Другий крок змушує слова, які не з'являються в одному контексті, бути вбудованими далеко одне від одного у просторі вбудовування. Без цього другого кроку ціль можна було б звести до мінімуму, зіставивши всі слова в один і той же вбудований вектор. Знаючи це, другий крок можна піддати апроксимації, витягуючи слова з розподілу шуму, де цільові слова виключаються. Отже, цільова функція зведена до мінімуму, коли контекстне слово вбудовано близько до цільових слів, але далеко від слів, отриманих із розподілу шуму. Цільовою функцією при skip-gram негативній вибірці є

$$\mathrm{J}(\mathcal{U}, \mathcal{V}) = -\sum_{k=0, k\neq j}^{2c} \log \sigma\left(\mathrm{score}(\boldsymbol{w}^{(j)}, \boldsymbol{w}^{(j-c+k)})\right) + \sum_{k=1}^{K} \log \sigma\left(\mathrm{score}\left(\boldsymbol{w}^{(j)}, \tilde{\boldsymbol{w}}^{(k)}\right)\right)$$

де кожен $\tilde{\boldsymbol{w}}^{(k)}$ випливає з розподілу шуму слів, які не є цільовими словами поточного контекстного слова.

6.2.1 doc2vec

doc2vec(Quoc Le [2014]) далі розширює концепцію word2vec для вбудовування рівня документа/абзацу. Версія розподіленої пам'яті моделі абзацу вектора (PV-DM) - це невелике розширення до моделі CBOW у word2vec. Замість навчання лише векторів слів, додається та навчається вектор унікальної функції документа. У моделі конкатенація або середнє значення цього вектора документа з контекстом із трьох слів використовується для прогнозування четвертого слова. Вектор абзацу представляє відсутню інформацію з поточного контексту і може діяти як пам'ять про тему абзацу.

doc2vec можна застосувати до класифікації тексту на рівні документа. Є також приклади застосувань у тематичному моделюванні та додаванні тегів до документів.

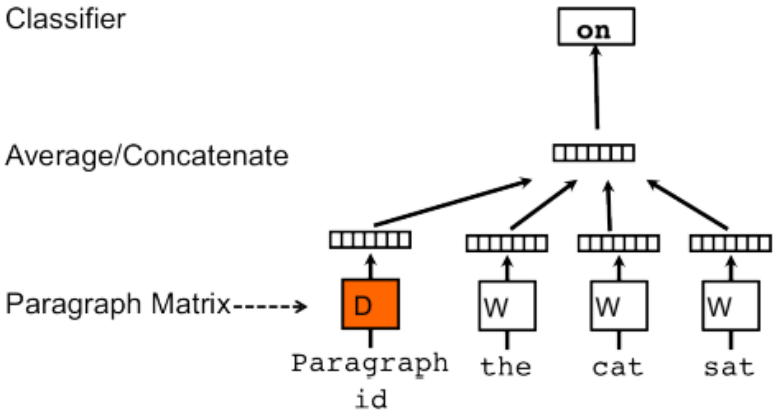


Рис. 24: Фреймворк для вивчення абзацу вектора.(Джерело:Quoc Le [2014])

6.3 GloVe

Обмеження word2vec полягає в тому, що модель використовує лише локальну лінійну контекстну інформацію. GloVe (Pennington et al. [2014]) є ще одним підходом до вирішення проблеми, що поєднує в собі глобальну матрицю факторизації та методи віконного локального контексту. Модель ефктивно використовує статистичну інформацію шляхом навчання лише на ненульовиих елементах у матриці співіснування слово-слово, а не на всій розрідженій матриці або на окремих контекстних вікнах у великому корпусі.

Завдання - мінімізувати витрати

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^\top w_j' + b_i + b_j' - logX_{ij})^2$$

де X_{ij} - матриця співіснування для кожної пари слів у корпусі, w_i є вектором слова для слова i та b_i - відповідний термін зміщення.

6.3.1 Матриця співіснування

Матриця співіснування слово-слово враховує X, записи яких X_{ij} позначає кількість разів слова j, що трапляються в контексті слова i. Наприклад, записи на діагоналі матриці спільних випадків представляють кількість співучасностей тих самих слів у контекстному вікні. Одне речення "Допоможи мені, допоможи тобі"мало б матрицю (3, 3), а діагональ була б кількістю співучасностей (допомога, допомога), (мені, мені) та (ти, ти). У цьому випадку першим підрахунком буде 2 з урахуванням розміру вікна 4.

Потім ми визначаємо $X_i = \Sigma_k X_{ik}$ - кількість випадків, коли якесь слово з'являється в контексті слова i. І тоді ми маємо $P_{ij} = P(j|i) = X_{ij}/X_i$ як ймовірність того, що слово j з'являється в контексті слова i.

Наведений нижче приклад із статті про походження показує, як певні аспекти значення можуть бути вилучені безпосередньо з цих ймовірностей співіснування.

Table 1: Co-occurrence probabilities for target words *ice* and *steam* with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like *water* and *fashion* cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.

Probability and Ratio	k = *solid*	k = *gas*	k = *water*	k = *fashion*
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

На закінчення, приклад показує, що відповідною відправною точкою для вивчення векторів слів має бути співвідношення ймовірностей співіснування. Тому ми хотіли б, щоб наші вектори наближались

$$w_i^\top w'_j + b_i + b'_j - logX_{ij} \tag{26}$$

6.3.2 Функція зважування

Додавання функції ваги, визначеної вище, може допомогти масштабувати кількість необроблених випадків повторення, з неперевантаженням.

Починаючи з початкової статті, продуктивність моделі слабко залежить від відсікання, яке вони фіксують $x_{max} = 100$. Для значення α, що виявляється, що $\alpha = 3/4$ дає скромне вдосконалення в порівнянні з лінійною версією $\alpha = 1$.

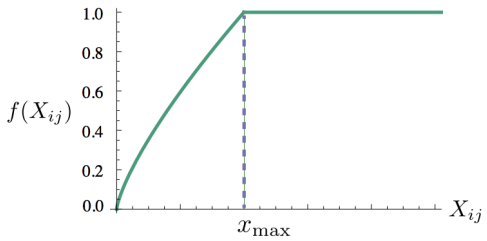


Figure 1: Weighting function f with $\alpha = 3/4$.

6.3.3 Складність моделі

Оскільки Glove - це модель, що базується на підрахунку, легше перевірити складність моделі. Складність просто пов'язана з кількістю невід'ємних записів у матриці співіснування, $\|X\|$. І це доводить $\|X\| \geq O(C)$, де C - розмір корпусу, що означає, що модель Glove не споживає занадто багато обчислювальних ресурсів.

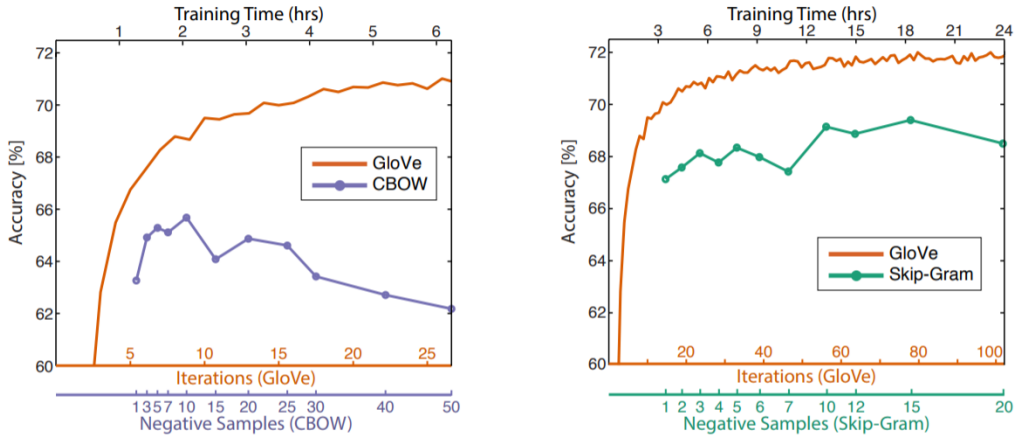
6.3.4 Продуктивність моделі

Модель Glove тестували за трьома завданнями: аналогія слів, подібність слів та розпізнавання іменованих сутностей. Завдання аналогії слів вимагають від алгоритму передбачити слово, якому відповідне дане слово та іншу пару, наприклад: «Понеділок до вівторок - це як субота до _?". Завдання подібності слів використовує кореляцію рангу Спірмена для оцінки подібності, наданої різними алгоритмами. Завдання «Розпізнавання іменованої сутності» вимагає від алгоритму ідентифікувати особу, місцезнаходження, організацію та різне в тексті. Модель Glove була протестована на інших моделях, включаючи SVD, CBOW та Skip-gram на багатьох наборах даних, і вона перевершує інші моделі у всіх завданнях. Дослідники також виявили, що модель Glove працює ефективніше з меншим розміром вікна та більшим розміром для задач аналогії синтаксичних слів.

6.3.5 Порівняння з word2vec

Порівняння між Glove та моделлю Word2Vec у завданні аналогії слова. Дослідники порівняли точність прогнозування двох моделей протягом однакового часу тренувань, вони збільшили час тренувань Glove, додавши ітерації, і збільшили час тренування CBOW/Skip-gram за допомогою негативної вибірки. Знову ж таки, модель рукавички перевершила CBOW та Skip-gram. Зі збільшенням часу тренувань точність рукавичок постійно зростає, але точність моделі Word2Vec постійно коливається, навіть зменшується. Однак у тесті дослідники використовували параметри за замовчуванням для CBOW та Skip-gram, але оптимальне налаштування для моделі Glove.

Рис. 25: Порівняння Glove та word2vec Pennington et al. [2014]



(a) GloVe vs CBOW

(b) GloVe vs Skip-Gram

6.3.6 Висновок

Як алгоритм вбудовування, заснований на підрахунку, Glove має свою перевагу в економії обчислювальних ресурсів і можливості обчислювати паралельно. Однак це не означає, що це

абсолютно кращий алгоритм вбудовування в будь-яких сценаріях. Алгоритми прогнозування, такі як CBOW та Skip-gram, також мають свої власні сценарії, такі як завдання прогнозування. Люди повинні ретельно продумати, вибираючи алгоритми вбудовування.

6.4 Класифікація тексту

Класифікація тексту широко використовується для розуміння відгуків користувачів (так званий сентимент-аналіз або аналіз тональності тексту), категоризації документів та виявлення спаму. Існує кілька способів підходу до цієї задачі класифікації, наприклад, використання лінійного класифікатора: SVM(машини опорних векторів), імовірнісний класифікатор: наївний баєсівський та глибока нейронна мережа: CNN. У цьому розділі ми зосередимося на знайомстві із такими поняттями як наївний баєсів класифікатор та символьно-рівневі нейронні мережі.

6.4.1 Наївний баєсів класифікатор

Перш за все, нам потрібно описати текст у числовому виді, які можна розглядати як функції вводу для алгоритмів машинного навчання. Існує добре відомий прийом, який називається «мішок слів», який полягає у обчисленні частоти кожного слова в тексті як гістограми. Однак у цій техніці слід мати на увазі, що ми лише використовуємо лише частоту слів в тексті й ігноруємо їх позиції в документі.

Наївний баєсів класифікатор належить до ймовірностних класифікаторів. Для деякого документу d, класифікатор повертає передбачений клас $\hat{c}$ який дає максимальну апостеріорну ймовірність.

$$\hat{c} = \underset{c \in C}{\operatorname{argmin}} \mathbb{P}(c|d) \tag{27}$$

де, C - всі можливі класи в документі d. Суть наївного баєсівського класифікатора полягає у використанні теореми Байєса для перетворення виразу (27) до такого виду,

$$\hat{c} = \underset{c \in C}{\operatorname{argmin}} \frac{\mathbb{P}(d|c)\,\mathbb{P}(c)}{\mathbb{P}(d)} \approx \mathbb{P}(d|c)\,\mathbb{P}(c) \tag{28}$$

де ми вважаємо $\mathbb{P}(d|c)$ і $\mathbb{P}(c)$ як апріорні ймовірності. Після цього, ми розглядаємо документ d як множину ознак "мішків слів"$f_1, f_2, ..., f_n$,

$$\hat{c} = \underset{c \in C}{\operatorname{argmin}} \mathbb{P}(f_1, f_2, ..., f_n|c)\,\mathbb{P}(c) \tag{29}$$

Як було сказано раніше, ми ігноруємо позицію слів в тексті. Крім цього, ми вважаємо умовно-незалежними кожен з $\mathbb{P}(f_i|c)$. Звідси,

$$\mathbb{P}(f_1, f_2, ..., f_n|c) = \mathbb{P}(f_1|c)\,\mathbb{P}(f_2|c)...\,\mathbb{P}(f_n|c) \tag{30}$$

Отже, ми отримуємо наївний баєсів класифікатор,

$$Classifier = \underset{c \in C}{\operatorname{argmin}} \mathbb{P}(c) \prod_{f} \mathbb{P}(f_i|c) \tag{31}$$

Проте, щоб використати наївний баєсів класифікатор до текстового документу, нам потрібно знати позицію слів в тексті. Тому ми просто додаємо індекс кожного слова по порядку.

$$Classifier = \underset{c \in C}{\operatorname{argmin}} \mathbb{P}(c) \sum_{i \in position} \mathbb{P}(w_i|c) \tag{32}$$

Для обчислення $\mathbb{P}(w_i|c)$, нам потрібно обчислити частоту слова w_i серед всіх текстових документів з певним класом c,

$$\hat{\mathbb{P}}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)} \tag{33}$$

де V це словник, що містить сі слова, що коли-небудь з'являлися в навчальних документах. Після, ми тренуємо класифікатор, максимізуючи ймовірність кожного слова, заданого класом теми, описаного вище.

6.4.2 Символьно-рівневі згорткові нейронні мережі

З самого початку використання згорткових нейронних мережей, вони чудово себе зарекомендували у задачах комп'ютерного зору, тому люди почали пробувати використати їх у обробці природної мови. Натхненні успішностю навчання згорткових мереж на піксельному рівні зображень, Чжан та ЛеКун та ін. досліджували, як застосувати згоркові нейронні мережі до текстів на рівні символів для завдання класифікації тексту.Було виявлено, що ці нейромережі не потребують знань щодо синтаксичних або семантичних структур речень, фраз чи навіть слів [Zhang et al., 2015b].

Модель приймає послідовність кодованих символів як вхідні дані, і кожен символ базується на одному із гарячих кодуваннь. Вхідні функції подаються у дві згорткові нейромережі з різними прихованими шарами. Обидві згорткові нейромережі мають глибину в 9 шарів - 6 згорткових шарів та 3 повністю зв'язаних шари, як показано на малюнку 26. В докладі [Zhang et al., 2015b], Чжан та ЛеКун та ін. показують як вхідні дані з 1014 ознак зменшуються до 34 після 6 шарів згортки та операцій 'max-pool'. Зокрема, на модельному рівні 3, 4 та 5 існують лише операції згортки без операцій 'max-pool'.Головний компонент цієї моделі вичисляє одновимірну згортку.Для більш детальної інформації, припустимо що є функція введення $g(x)$, функція ядра $f(x)$ та функція згортки $h(y)$ які визначені як зображено нижче,

$$g(x) \in [1, l] \to \mathbb{R} \tag{34}$$

$$f(x) \in [1, k] \to \mathbb{R} \tag{35}$$

$$h(y) \in [1, \left\lfloor \frac{l-k}{d} \right\rfloor + 1] \to \mathbb{R} \tag{36}$$

згортка $h(y)$ між $f(x)$ і $g(x)$ з вікном d задається наступним чином,

$$h(y) = \sum_{x=1}^{k} f(x) g(yd - x + c) \tag{37}$$

операція 'max-pool' $h(y)$ функції g(x) задається наступним чином,

$$h(y) = \max_{x=1}^{k} g(yd - x + c) \tag{38}$$

де $c = k - d + 1$ є константою зміщення.

Порівнюючи з 'мішком' з n-грамаммми моделлю, яка дозволяє використовувати часткову інформацію про текст та рекурентні нейронні мережі, такі як довга короткочасна пам'ять

(LSTM), яка може фіксувати залежність слів у довгому реченні, згорткові мережі на рівні символів може успішно працювати над завданням класифікації тексту без слів. Застосовуючи згорткові мережі на рівні символів до різних текстових даних, автори також роблять висновок, що їх модель краще працює на випадкових даних, створених користувачами, таких як огляд Amazon, оскільки, схоже, згорткові мережі добре виявляють орфографічні помилки та смайлики.

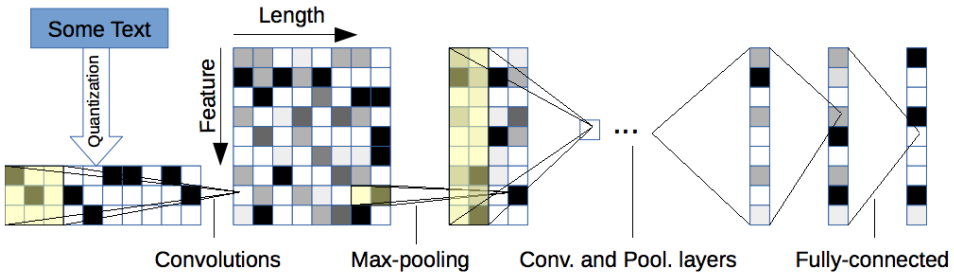


Рис. 26: Згорткова мережа на рівні сиволів, прийнята [Zhang et al., 2015b]

6.5 FastText

Ідея FastText близька до безперервного 'мішка' слів (CBOW), а основна відмінність FastText і CBOW полягає в наступному: FastText використовує слова даного тексту, щоб передбачити мітку цього тексту, а мета CBOW передбачає ймовірність середнього слова, поданого словами у сусідів. Структури двох моделей досить схожі. FastText знову можна вважати неглибокою нейронною мережею: він має один прихований шар і один вихідний рівень. Прихований шар є текстовим поданням і генерується шляхом взяття середнього значення всіх векторів слів у тексті, а потім нелінійний шар активації (SoftMax) використовується для обчислення ймовірності іншого класу, до якого даний текст повинен бути позначений як . FastText - це модель навчання з вчителем, а CBOW - це навчання без вчителя. Для FastText також слід розглянути обчислювальну складність, подібну до CBOW: Якщо занадто багато різних класів, нелінійному шару активації знадобиться багато часу, щоб обчислити ймовірність належності тексту до різних класів. Тож ієрархія softmax знову використовується у FastText для вирішення складної проблеми, і складність зменшилася з O (hlog2 (k)) до O (log (T)) за допомогою SoftMax, де h - розмірність подання тексту, а k - кількість класів, а T - найвищі цільові показники T.[Joulin et al., 2016]

6.5.1 Цільова функція the FastText

Цільова функція FastText показана нижче[Joulin et al., 2016], і це є від'ємний логарифм функції втрат. Стандартною функцією втрат є мультикласова логістична втрата.[Maaten, 2018] Об'єктна функція має менші втрати, якщо ймовірність даного тексту, позначеного як коректний клас, висока, і має більші втрати, якщо ймовірність даного тексту, позначеного як коректний клас, низька.Відповідно до [Joulin et al., 2016], стохастичний градієнтнийс спуск і асинхронна паралель використовуються для оптимізації функції втрат. Про це буде розказуватись

пізніше.

$$-\frac{1}{N}\sum_{n=1}^{N} y_b log(f(BAX_n)) \tag{39}$$

$$l(x,y) = log(\frac{\exp(y^T x)}{\sum_{k=1}^{K}\exp(x_k)}) \tag{40}$$

6.5.2 N-грам для FastText

Оскільки FastText бере тільки середнє значення векторів усіх слів у тексті, може бути велика ймовірність втратити синтаксис речення (граматика) або структуру речення. Причиною цього є те, що просте додавання всіх векторів слів втратить порядок слів у реченні. Наприклад, "Її собака переслідує мою собаку"не має різниці від "Моя собака переслідує її собаку якщо просто підсумувати всі 1 грам, і в результаті вийде той самий текстовий вектор, який представлятиме два тексти, використані в попередніх прикладах . Це не має сенсу, оскільки два речення у попередніх прикладах, очевидно, мають протилежне значення. Це розглядається як слабкість FastText, і тому одним із можливих рішень є введення поняття n грам у FastText. Знову ж таки, з моїми попередніми прикладами, якщо використовується бі-грам, "переслідує мою"у першому прикладі та "переслідує її"у наступному прикладі все одно збереже значення речення. Це одна з основних причин, чому n-грам корисний для FastText. У [Joulin et al., 2016] показано,що FastText може мати найкращу продуктивність використовуючи до 5 n-грам на тестових даних.

Може бути зрозуміліше ,якщо подивитися на документ https://arxiv.org/pdf/1607.01759.pdf

6.5.3 HOGWILD! Паралелізація стохастичного градієнтного спуску

HOGWILD! є реалізацією стохастичного градієнтного спуску(SGD) що паралелізує оновлення цільової функції. Оновлення SGD має послідовний характер, оскільки потрібні інкрементальні ітерації, щоб мінімізувати помилку в цільовій функції.Було сказано, що мотивація використання HOGWILD! особливо сильна, коли набори даних надзвичайно великі,так як це має місце в багатьох випадках проблемної класифікації тексту. Найбільша перешкода у впровадженні HOGWILD! це те , що раніше паралельні обчислення мали можливість блокувати (запобігати появі одного обчислення, що перезаписує інше) під час обробки. HOGWILD! - це стратегія, яка виключає необхідність блокування, щоб можна було проводити паралельні обчислення.

У HOGWILD! ділиться всіма системними процесорами, які повторюються через обчислення та записують у пам'ять без блокування. Твердження полягає в тому, що коли дані мають розріджений характер (як у випадку з текстовими даними), ітерації через SGD оновлюють параметр цільової функції настільки мізерно, що перезапис пам'яті надзвичайно рідкісний. Крім того, якщо відбудеться перезапис, друга проблема полягає в тому, що помилка, внесена перезаписом, є незначною.

HOGWILD! має на меті мінімізувати наступну функцію$f : X \subseteq \mathbb{R}^n \to \mathbb{R}$:

$$f(x) = \sum_{e\in E} f_e(x_e) \tag{41}$$

Тут, e це мала підмножина $\{1,\ldots,n\}$ і x_e позначає значення вектора x (наш цільовий вектор параметрів) на координатах, індексованих e. (Примітка: процедура передбачає "гаряче кодування".)

Для HOGWILD! алгоритму, загальнодоступна модель має p процесорів. Змінна рішенння x зберігається у спільній пам'яті, щоб кожен процесор міг читати та оновлювати x. b_v позначає один із стандартних базових елементів у $\mathbb{R}^n$, з v яке знаходиться у межі $1, \ldots, n$. Тобто, b_v дорівнює від 1 д v-го компоненту і 0 в іншом випадку. γ фіксована константа кроку. $G_e(x)$ позначає градієнт або субградієнт функції f_e помножений на $|E|$ так, що f_eрозширюється від координат e до всіх із $\mathbb{R}^n$. Це по суті справедливо, враховуючи, що $G_e(x) = 0$ на будь-яких координатах, не в e.

Потім кожен процесор дотримується процедури: вибірково додає терм $e \in E$ рівномірно обчислює градієнт f_e на x_e, обирає $v \in e$ рівномірно, потім онволює x_v і запису в пам'ять. Згодом оновлення здійснюються лише до змінних, проіндексованих e (а всі інші змінні залишаються недоторканими в e).

Algorithm 5 HOGWILD!

1: **loop**
2: Обираємо разок e рівномірно-випадково з E
3: Читаємо нинішній стан x_e і обчислюємо $G_e(x)$
4: **for** $v \in e$ **do** $x_v \leftarrow x_v - \gamma b_v^T G_e(x)$
5: **end loop**

6.6 Вбудовування всіх речей - StarSpace

6.6.1 Введення

Вбудовування слів (embedening words) привернуло велику увагу суспільства, але те, що варто зауважити для тих, хто щойно почав знайомитись з цією областю, ідея полягає в тому, що вбудовування слів не обмежується лише світом слів, тобто це не лише техніка представлення слів. Хорошим прикладом є модель TagSpace, вироблена в 2014 році [8]

По суті, вбудовування слів - це векторні подання слів або документів. Використовуючи такі моделі, як `word2vec`, який є одним із методів нейронної мережі для генерації подання слів. Для більш загальних цілей Facebook запропонував модель під назвою StarSpace [9] [10], який відображає різні сутності в одному векторному просторі вбудовування. Порівнюючи подібність у векторному просторі, ми можемо вирішити такі проблеми, як передбачення найкращої мітки для введення та найбільш можливої відповіді на запит у пошуковій системі, яка насправді є завданням пошуку подібного вводу пара-мітка/речення-речення. Він досягає досить хорошої продуктивності (яка подається в різних розділах) для різних завдань, але це загальність, тобто застосовується для порівняння різних об'єктів, що робить її унікальною.

6.6.2 Позначення

Використовуючи опис "вбудованого слова простір об'єктів (словник) D позначається як $F^{D \times d}$ матриця, де i^{th} рядок F_i - вектор розмірності d представлення для функції i^{th}. Тоді вбудовуванням

[8] `http://emnlp2014.org/papers/pdf/EMNLP2014194.pdf`
[9] `https://arxiv.org/pdf/1709.03856.pdf`
[10] Це також проект з відкритим кодом на `https://github.com/facebookresearch/Starspace`

для сутності α є $\sum_{i \in \alpha} F_i$

6.6.3 Тренування

Ідея моделі полягає в тому, що проблему, яку ми хочемо вирішити, можна розглядати як "завдання подібності". Отже, навчальний процес полягає у порівнянні сутностей: ми хочемо, щоб векторне представлення для "подібної"сутності відображалося якомога ближче у векторному просторі d розмірності, відкидуючи вектори, що представляють різні сутності. Отже, формат функції збитків тут досить подібний до того, що ми бачили в `word2vec`. Протягом епохи він намагається звести до мінімуму:

$$\sum_{(a,b) \in E^+, b^- \in E^-} L^{batch}(sim(a,b), sim(a,b_1^-), \ddots, sim(a,b_k^-)$$

Правильна пара походить від E^+, а від'ємні пари - від E^-. Компоненти обох наборів залежать від завдань. Як і модель пропуску грамів, автор прийняв k-негативну стратегію вибірки. Наприклад, для завдань маркування тексту α - це документи, b^+ - правильна мітка, b^- - зразки з набору міток (крім правої).

Функція подібності та форма специфікації функції втрат - це гіперпараметри. У поточному проекті з відкритим кодом для функції подібності реалізовано як подібність косинусів, так і внутрішній продукт; для функції збитків - граничні рангові втрати $(max(0, \mu - sim(a,b))$, де μ - параметр маржі та негативна втрата журналу softmax $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log p_{ij}$ є реалізованою.

6.6.4 Тестування

Під час тестування, залежно від завдання, яке ви намагаєтеся вирішити, ви можете або використовувати представлення для випадків, що знаходяться нижче, або для прогнозування ви використовуєте функцію подібності $sim(\cdot, \cdot)$ для ранжирування всіх можливих міток для заданого α.

6.6.5 Ефективність

Автори демонструють ефективність моделі у шести завданнях,

- Завдання класифікації/маркування тексту, такі як аналіз настроїв: ефективність перевіряється на наборах даних AG, DBpedia, Yelp. StarSpace перевершує низку методів і працює аналогічно fastText.
- Проблема рейтингу, наприклад ранжування веб-документів за запитом: StarSpace значно перевершує TFIDF та fastText, це тому, що StarSpace може виконувати вбудовування рівня речення та документа, тоді як інші не можуть.
- Завдання рекомендації, включаючи рекомендації на основі спільної фільтрації та рекомендації на основі вмісту, де вміст визначається окремими функціями, такими як слова документів: для рекомендацій на основі вмісту для рекомендацій до статті StarSpace перевершує низку методів і працює аналогічно fastText.

- Вбудовування графіків: для TransE та StarSpace з однаковим розміром вбудовування надають однакову продуктивність.
- Вивчення вкладених слів, речень або документів: протестовано на SentVal і показало хорошу ефективність, особливо у завданні STS (семантична текстова подібність).

6.7 Згорткові Нейроні Мережі для текстових даних

Згорткові Нейроні Мережі (Convolutional Neural Networks, ConvNet) допомагають отримати вражаючі результати при обробці зображень та розпізнавання голосу. Операція згортки допомагає комп'ютеру знайти приховану інформацію серед пікселів та звукових сигналів. Однак, ми також хочемо побачити чи здатна ConvNet виконувати Обробку Природної Мови (Natural Language Processing, NLP).

Деякі нейроні мережі вже здатні давати вражаючі результати з обробки мовних даних. Рекурентна Нейрона Мережа (Recurrent Neural Network, RNN) доволі гарно підходить для обробки текстових даних. У додаток до цього, Довга Короткочасна Пам'ять (Long Short-Term Memory, LSTM), яка є модифікацією RNN, широко використовується для обробки природних мов. TreeRNN є одним із варіантів Рекурсивної Нейроної Мережі (Recursive Neural Network) і може генерувати дерева розбору, які легче інтерпретувати. ConvNet може використовуватися як напряму для отримання результатів NLP аналізу, так і для обробки даних перед їх використанням у інших моделях нейроних мереж для отримання кращих результатів.

За допомогою ConvNet з налаштуванням невеликої кількості параметрів можна отримати чудові результати для деяких задач класифікації речень Kim [2014]. Першим кроком при обробці речень є створення вектору для кожного слова за допомогою технології вкладання слів (word embedding), наприклад word2vec. Після чого речення перетворюється у матрицю, де кожний рядок - k-мірний вектор слова. Так само, як кольорові зображення мають три RGB-канали, ця модель може також використовувати різні методи вкладання слів для отримання декількох каналів. На рис.27 можна побачити як автор використовує два канали при згортці: статичний та динамічний. Обидва канали отримуються за допомогою word2vec, але тільки динамічний змінюється під час навчання моделі методом зворотного поширення помилки (backpropagation), тоді як статичний залишається незмінним. Операція згортки використовує фільтри (їх також називають ядрами), які застосовують h слів для створення нових ознак. Після чого застосовується операція об'єднання по максимуму за весь час (max-over-time pooling) до отриманих ознак, де під "часом" мається на увазі позиція у реченні Collobert et al. [2011]. Ця операція виконується для того, щоб виділити найбільш важливі ознаки - ті, які мають найбільше значення.

Дана модель використовує декілька фільтрів (з різними розмірами вікон) для отримання декількох ознак. Ці ознаки утворюють передостанній шар і передаються повністю зв'язаним шарам softmax, вихідними даними яких є розподіл вірогідності між мітками. Також застосовується фільтрація для того, щоб запобігти появі ознак, між якими є кореляція Srivastava et al. [2014]. На деяких добре відомих наборах даних, таких як MR, SST-2, CR, MPQA, вона показує кращі результати у порівнянні з іншими алгоритмами машинного навчання, такими як RAE, MV-RNN, Tree-CRF. Недавнє дослідження, яке порівнює ConvNet з GRU та LSTM, показує, що на деяких наборах даних ConvNet може досягти кращих результатів для деяких завдань NLP Yin et al. [2017]. Дослідники також пропонують комбінований метод з ConvNet та LSTM Wang et al. [2016] для аналізу тональності тексту. На рис.28 показано як цей метод використовує шари згортки для обробки окремих речень, після чого слідує шар об'єднання по максимуму

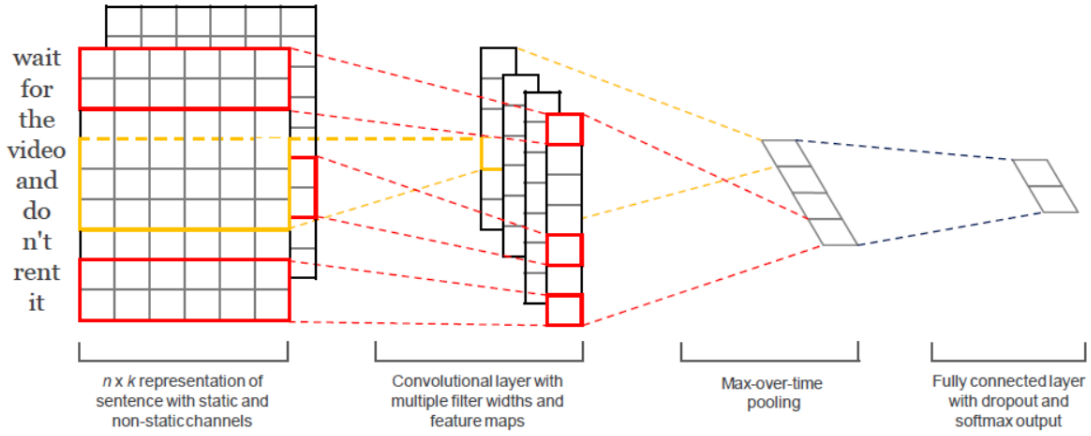


Рис. 27: ConvNet для класифікації речень Kim [2014]

для фільтрації найважливіших ознак з операції згортки. Далі виходи шарів об'єднання подаються на послідовний шар LSTM, результат якого подається на лінійний декодер для отримання фінального результату. Експерименти на деяких англійських та китайських наборах даних показали, що дана модель працює краще, ніж ConvNet, RNN або LSTM.

6.8 Згорткові Нейроні Мережі для Комп'ютерного Зору

6.8.1 Основні поняття про зображення

Що таке зображення?
Зображення представлено в комп'ютері в трьох окремих матрицях, які відповідають червоному, зеленому та синьому каналам. Піксель - це одна з багатьох маленьких зон світла на екрані дисплея, з яких складається зображення. Припустимо, що зображення має 64 на 64 пікселі, це означає, що воно має три матриці 64 на 64, що відповідають значенням інтенсивності червоного, зеленого та синього пікселів. Значення варіюються від 0 до 255 Pritchard et al. [1994]. Математично кажучи, зображення представлене матрицею значень інтенсивності, тобто зображення можна розглядати як дискретну функцію f від $R^2 \to R$. Ця 2D-функція виводить значення інтенсивності пікселя для кожної координати (x, y). Тобто зображення може бути трансформоване за допомогою зміни пікселів на відповідних координатах.

Для чого потрібна обробка зображень?
Є дві основні причини чому люди виконують трансформацію зображень. По-перше, вона використовується для вдосконалення/зменшення інформації на зображенні для людських намірів та інтерпретації. По-друге, вона використовується для автономного сприйняття комп'ютером. Наприклад: для ідентифікації людського обличчя, покращення рентгенівських знімків, виявлення номерного знаку, тощо.

Трансформація зображень
При обробці зображень ядро, яке також називають маскою, є матрицею згортки. Воно використовується

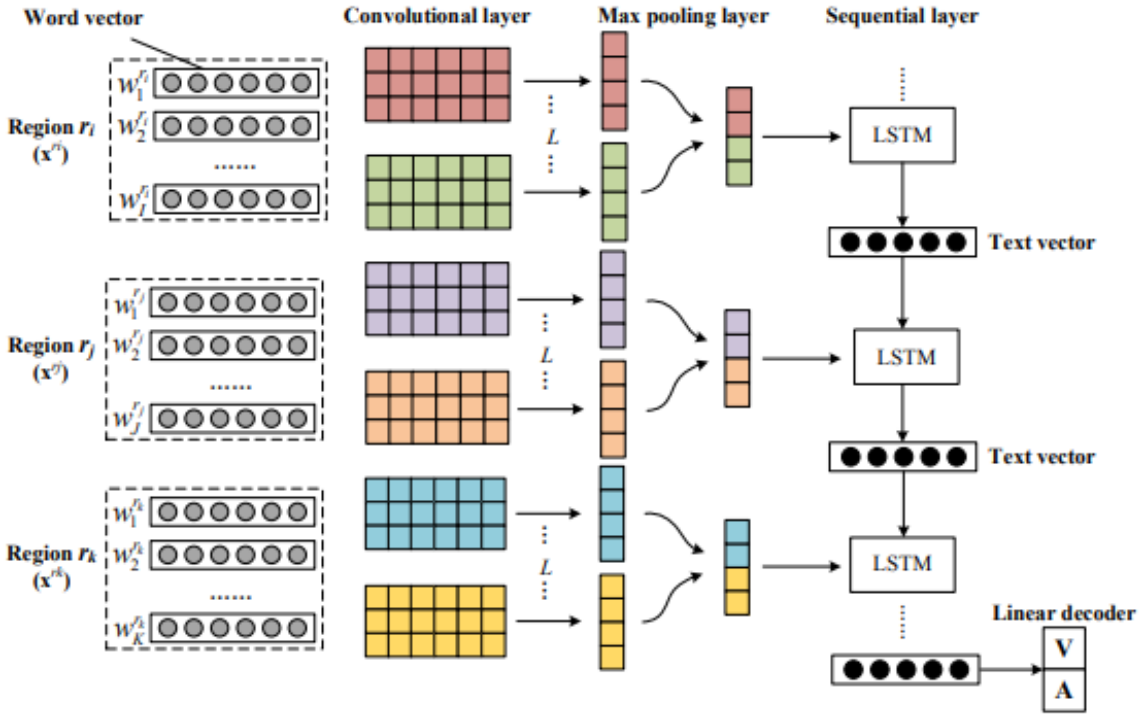


Рис. 28: Архітектура запропонованої CNN-LSTM моделі Wang et al. [2016]

для копіювання, розмиття, загострення та виявлення границь зображень.

Для реалізації перетворення зображення використовується метод фільтрації, який називають згорткою між зображенням і фільтром ядра як показано на рис. 29.

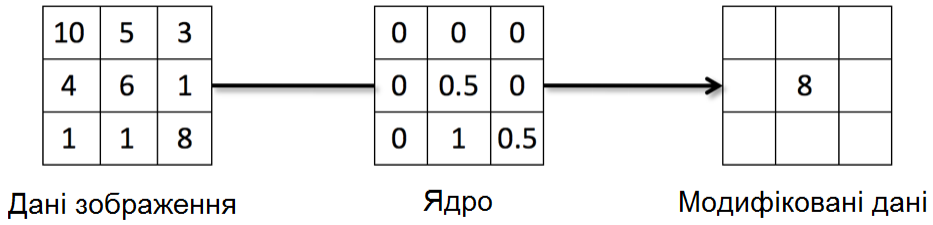


Рис. 29: Операція трансформації зображення

Як показано на рис. 30, беремо фільтр 5×5×3, проходимо ним по всьому зображенню та обчислюємо скалярний добуток кожного положення фільтру. Позначимо параметри фільтру w. Позначимо захоплену 5×5×3 частину зображення як x. Вихідні дані для цієї частини зображення можна обчислити, взявши скалярний добуток між фільтром та даною частиною

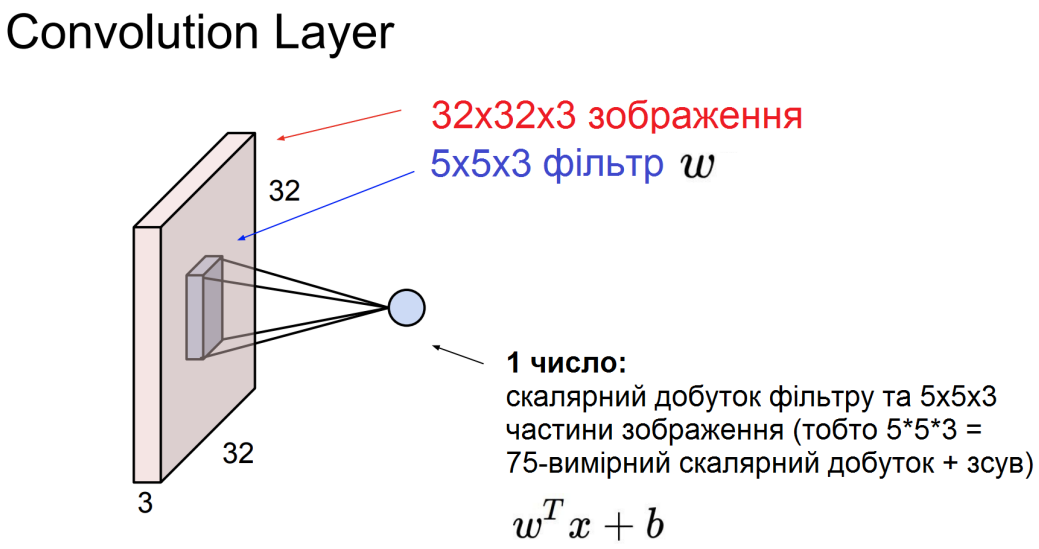


Рис. 30: Приклад фільтру

зображення.

$$y = w^\top x + b$$

Використовуємо вихідні значення скалярних добутків кожної частини зображення для створення вихідної карти активації. Кожен фільтр відповідає одній вихідній карті активації. Вихідна карта активації є вхідними даними для наступного шару мережі.

6.8.2 Лінійна фільтрація

Згортка та перехресна кореляція

Існує два поширених методи лінійної фільтрації: перший - це перехресна кореляція, а другий - згортка (ми не розглядаємо нелінійну фільтрацію в цьому розділі). Загальна ідея полягає в заміні кожного пікселя лінійними комбінаціями сусідніх пікселів. Структура лінійних комбінацій називається ядром. Перехресна кореляція досить схожа на згортку, єдина відмінність полягає в тому, що ядро згортки - це горизонтально та вертикально "перевернута" версія ядра перехресної кореляції. Математично, згортку можна описати так:

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(i-u, j-v) H(u,v) \tag{42}$$

або

$$G = H * F \tag{43}$$

перехресну кореляцію можна описати як:

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(i+u, j+v) H(u,v) \tag{44}$$

або

$$G = H \bigotimes F \tag{45}$$

F - матриця вхідного зображення, H - матриця ядра, G - вихідне зображення. Згортка має комутативні та асоціативні властивості. Причина, чому згортці надають перевагу порівняно з кореляцією, полягає в тому, що вона має кращі математичні властивості. Зокрема, згортка є асоціативною, тоді як кореляція, в загальному випадку, - ні. Отже, згортку можна обчислювати за допомогою перетворення Фур'є. Зокрема, перетворення Фур'є згортки $f*g$ двох функцій f і g дорівнює добутку перетворень Фур'є двох функцій. Використовуючі математичні позначення:

$$F(f * g) = kF(f)F(g) \tag{46}$$

Типи лінійної фільтрації
В основному виділяють чотири типи лінійної фільтрації: тотожність (identity), виявлення країв (edge detection), загострення (sharpen) та розмиття (blur) MatlabTricks, які відображено нижче. Згортка може бути реалізована за допомогою безлічі різних лінійних ядер для досягнення різних намірів людини.

$$Identity: \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad Edge\ Detection: \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} or \begin{bmatrix} 0 & 1 & 0 \\ 1 & -10 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{47}$$

$$Sharpen: \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad Gaussian\ blur: \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad Box\ blur: \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{48}$$

6.8.3 Гіперпараметри для фільтрації

Три гіперпараметри регулюють розмір вихідних даних: глибина, крок і нульове заповнення.

1. Глибина відповідає кількості фільтрів, які ми хотіли б використовувати. Кожен фільтр навчається шукати різні властивості у вхідних даних.

2. Крок фільтра визначає на яку відстань переміщується фільтр. Коли крок дорівнює 1, фільтри переміщуються по одному пікселю за раз. Коли крок дорівнює 2, фільтри стрибають через піксель кожний раз, коли вони переміщаються. Тоді ми отримуємо менший розмір вихідних даних.

3. Як показано у статті, іноді корисно використовувати нульове заповнення для управління розміром вихідних даних. Розмір нульового заповнення - це ще один гіперпараметр. Найбільш часто він використовується для того, щоб вихідне зображення мало такий самий розмір, як і вхідне зображеня Pathy [2018].

Розмір вихідних даних $W_2 \times H_2 \times D_2$ можна обчислити, використовуючи початковий розмір (W та H), розмір фільтрів (F), кількість фільтрів (K), крок (S), а також розмір нульового заповнення (P). Формула вихідного розміру наступна:

$$W_2 = (W - F + 2P)/S + 1 \tag{49}$$

$$H_2 = (H - F + 2P)/S + 1 \tag{50}$$

$$D_2 = K \tag{51}$$

На рис. 31 можна побачити приклад використання двох фільтрів розміром 3×3 із кроком 2 та розміром нульового заповнення 1. Вхідний розмір дорівнює 5×5×3. При обчислені за формулами вище, отримуємо вихідний розмір 3×3×2.

$$W_2 = (5 - 3 + 2 * 1)/2 + 1 = 3 \tag{52}$$

$$H_2 = (5 - 3 + 2 * 1)/2 + 1 = 3 \tag{53}$$

$$D_2 = 2 \tag{54}$$

Сині квадрати показують одну із частин, де застосовується перший фільтр. Вихід для цього конкретного місця обчислюється через скалярний добуток фільтра та частини вхідних даних в синьому квадраті, а потім додається зсув. Кожен елемент вихідних даних обчислюється таким чином. Pathy [2018]

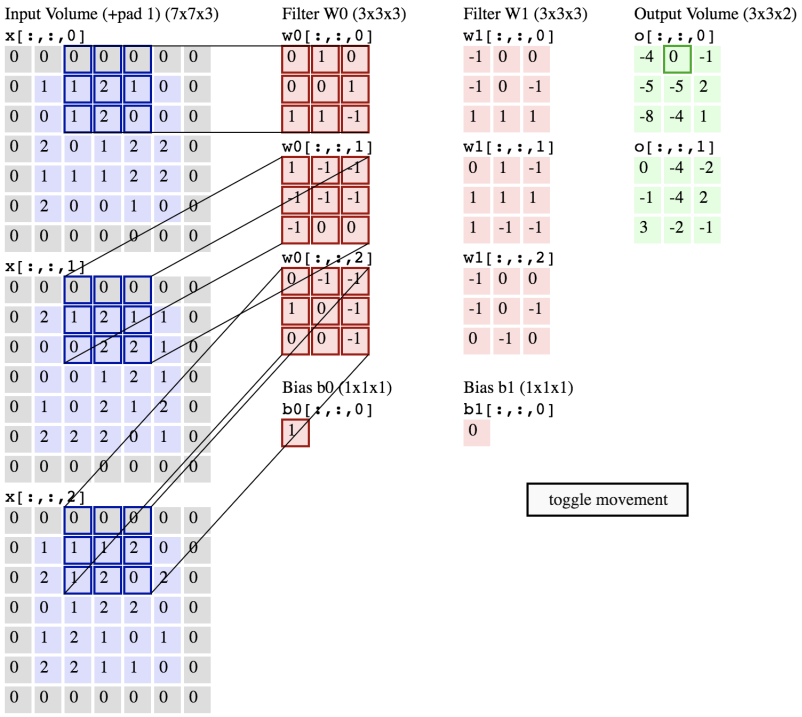


Рис. 31: Приклад шару згортки

6.9 Візуалізація даних

У даному розділі ми побачимо, як дослідити багатовимірний набір даних із сотнями ознак чи змінних, отримаємо невелике представлення про домен даних, і побачимо як знайти приховані шаблони в наборі даних, для кращого розуміння датасету шляхом зменшення розмірності набору даних з довільного числа до двох або трьох, щоб можна було візуалізувати їх на екрані. Перше, що спадає нам на думку, коли запитують про зменшення розмірності - Метод Головних Компонент (Principle Component Analysis, PCA; Hotelling [1933]). На жаль, PCA - це лінійний алгоритм, і він не зможе інтерпретувати складний поліноміальний зв'язок між ознаками. Основною проблемою алгоритмів лінійного зменшення розмірності є те, що вони зосереджені на розміщенні несхожих точок якомого далі один від одної у просторі з меншою розмірністю. Але для того, щоб представити дані високої розмірності на низькорозмірному нелінійному многовиді, важливо, щоб подібні точки даних були представлені близько один до одного, що зазвичай неможливо за допомогою алгоритмів лінійного зменшення розмірності. З іншого боку, t-розподілене вкладення стохастичної близькості (t-Distributed Stochastic Neighbour Embedding, t-SNE; van der Maaten and Hinton [2008]) - це алгоритм нелінійного зменшення розмірності, який відображає кожну точку багатовимірного простору в двох- або трьох-вимірну точку евклідового простору таким чином, що подібні об'єкти розташовуються поруч, а несхожі об'єкти відповідають віддаленим точкам з високою ймовірністю. Цей алгоритм надзвичайно

добре працює на багатьох реальних наборах даних.

6.9.1 t-SNE алгоритм:

Вкладення стохастичної близькості (Stochastic Neighbour Embedding, SNE; Hinton and Roweis [2002]) перетворює багатовимірні евклідові відстані між точками даних в умовні ймовірності, які відображають подібність. Подібність точки даних x_i точці x_j -це умовна ймовірність $p_{j|i}$, що x_i вибрав би x_j як свого сусіда, якби сусіди були обрані пропорційно їх гаусовій густині ймовірності з центром в x_i. Тобто алгоритм перетворює найкоротшу відстань між точками у ймовірність подібності точок. Математичне рівняння умовної ймовірності $p_{j|i}$ подано наступним чином.

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

$p_{j|i}$ є відносно високою для сусідніх точок даних і буде близькою до нуля для далеко віддалених точок даних для резонних значень дисперсії (σ_i) гаусівського розподілу. Нехай y_i і y_j є низьковимірними аналогами високовимірних даних x_i та x_j відповідно, і можливо обчислити аналогічну умовну ймовірність в цьому низьковимірному просторі з обмеженням, яке ми задаємо дисперсією Гауса, яка використовується при обчисленні умовних ймовірностей $q_{i|i}$ рівною $1/\sqrt{2}$. Оскільки нас цікавить тільки моделювання попарних подібностей, ми встановлюємо значення $p_{i|i}$ і $q_{i|i}$ рівними нулю. Математичне рівняння умовної ймовірності $q_{j|i}$ задається наступним чином.

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq i}\exp(-\|y_i - y_k\|^2)}$$

Те, що ми порахували до цих пір, - це ймовірність подібності точок у високовимірному просторі і ймовірність подібності точок у відповідному просторі низької розмірності. Тому тут SNE намагається мінімізувати різницю між цими двома умовними ймовірностями. Щоб обчислити мінімізацію суми різниць умовних ймовірностей SNE мінімізує суму розбіжностей Кульбака-Лейблера (КЛ) серед усіх точок даних за допомогою методу градієнтного спуску. А оскільки розбіжності КЛ асиметричні за своєю природою і дуже неефективні з обчислювальної точки зору, для оптимізації функції витрат t-SNE намагається мінімізувати суму різниці умовних ймовірностей. Це робиться шляхом симетризації функції витрат SNE. Крім того, t-SNE використовує розподіл із повільно спадним хвостом у низьковимірному просторі, щоб полегшити як проблеми скупченості, так і проблеми оптимізації SNE. Симетризація SNE здійснюється шляхом мінімізації розбіжності КЛ між спільним розподілом ймовірностей P у високовимірому просторі і спільним розподілом ймовірностей Q в просторі низької розмірності в порівнянні з умовним розподілом, яке робить SNE.

$$C = KL(P||Q) = \sum_i \sum_j p_{ij}\log\frac{p_{ij}}{q_{ij}}$$

де, $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$ та

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

Ми можемо інтерпретувати функцію втрат наступним чином:
Велика p_{ij}, змодельована малою q_{ij}, матиме великий штраф, а мала p_{ij}, змодельована великою q_{ij}, не призведе до такого великого штрафу, оскільки ми більше дбаємо про точність результату відносно сильно пов'язаних вузлів у просторі з великою розмірністю.
Градієнт для цієї функції витрат задається наступним чином:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$

6.9.2 Фізична інтерпретація t-SNE:

Мінімізація функції витрат здійснюється за допомогою градієнтного спуску. І фізично градієнт можна інтерпретувати як результуючу силу, створювану набором пружин між конкретною точкою відображення та всіма іншими точками цього відображення. Всі пружини прикладають силу в напрямку y_i і y_j. Пружина між двома точками відштовхує або притягує ці дві точки залежно від того, чи відстань між ними у відображенні занадто мала або занадто велика для представлення подібності між двома точками у високовимірних даних. Сила, яку пружина застосовує, пропорційна її довжині та жорсткості, яка визначається різницею $(p_{ij} - q_{ij})$ між схожістю точок даних та точок відображення. Оптимізація t-SNE в даній інтерпретації еквівалентна пошуку положення точок, в якому буде спостерігатися рівновага сил.

6.9.3 Оптимізації t-SNE:

В t-SNE використовуються дві основні оптимізації:
Перша оптимізація називається "раннє стиснення". У даній оптимізації на ранніх ітераціях оптимізації до цільової функції додається L_2-штраф на відстані в просторі низької розмірності, що тягне за собою стиснення всіх точок в нулі. У зв'язку з цим, кластерам буде легше переходити один через одного, щоб правильно розташуватися в просторі.
Друга оптимізація називається "раннє перебільшення". У даній оптимізації на ранніх ітераціях p_{ij} множаться на деяке позитивне число, наприклад на 4. Так як q_{ij} залишаються тими ж самими, вони занадто маленькі, щоб моделювати відповідні p_{ij}. Як наслідок, утворюються дуже щільні кластери, які широко розкидані в просторі низької розмірності. Це створює багато пустого простору, який використовується кластерами, щоб легко міняти і знаходити найкраще взаємне розташування.

6.9.4 Корисні посилання з методами візуалізації даних:

Ось посилання на код, якщо ви хочете випробувати цей алгоритм Laurens Van Der Maaten: t-SNE. Ось хороша стаття про те, як ефективно використовувати t-SNE, пояснена на кількох чудових прикладах, опублікована за адресою distill.pub.

6.9.5 Порівняння з іншими методами

Візуалізація даних важлива для машинного навчання, оскільки вона може розкрити структуру даних і може бути використана для отримання інформації безпосередньо або в якості дослідницького аналізу, який використовується в подальшому моделюванні. У багатьох випадках розмірність

даних висока, і їх структура не може бути легко візуалізована або зрозуміла. Було розроблено кілька підходів до візуалізації даних з високими розмірами за рахунок зменшення розмірності, включаючи проекцію Саммона (Sammon Mapping, Sammon) [Sammon, 1969], метод криволінійних компонент (curvilinear components analysis, CCA) [Demartines and Herault, 1997], вкладення стохастичної близькості (Stochastic Neighbor Embedding, SNE) [Hinton and Roweis, 2002], Isomap [Tenenbaum and Langford, 2000], лінійне вкладання (Linear Embedding, LLE) [Roweis and Saul, 2000], та Laplacian Eigenmaps [Belkin and Niyogi, 2002].

Ці методи генерують нелінійне відображення даних високої розмірності у представлення меншої розмірності, і тому відрізняються від класичних методів лінійного зменшення розмірностей, таких як Метод говних компонент (Principle Component Analysis, PCA) [Hotelling, 1933] та Багатовимірне шкалювання (classical multidimensional scaling, MDS) [Torgerson, 1952]. При застосуванні до реальних даних більшість описаних вище методів не здатні представити як локальну, так і глобальну структуру вихідного набору на відображені меншої розмірності, що є недоліком, який усувається в t-SNE [van der Maaten and Hinton, 2008].

Проблема скупчення і її вирішення в t-SNE

"Проблема скупчення" є однією з причин того, що вже відомі методи, такі як SNE та Sammon Mapping, не можуть точно представляти як локальну, так і глобальну структуру даних великих розмірів у низьковимірному поданні [van der Maaten and Hinton, 2008]. Щоб обгрунтувати цю проблему, припустимо, що справжній вимір наших даних - це десятивимірний простір, а дані, до яких ми маємо доступ, у набагато вищому вимірі. Наша мета - створити двовимірне представлення групи точок, приблизно рівномірно розподілених на десятивимірній множині та зосереджених навколо точки i. Об'єм опуклої оболонки цих точок може бути апроксимований гіперсферою і масштабується як r^m, де r - радіус сфери, а m - розмірність. Як результат, якщо ми хочемо точно представити відстань між точками близькими до i на нашому двовимірному відображенні, то більшість помірно віддалених точок потрібно буде розмістити дуже далеко від i. Крім того, якщо ми хочемо точно представити відстань між точками, помірно близькими до i, більшість точок з невеликою відстанню до i будуть розміщені занадто близько до i на двовимірному відображенні. Ця проблема відома як "проблема скупчення" [van der Maaten and Hinton, 2008]. Градієнт розбіжності Кульбака-Лейблера між P і спільним розподілом ймовірностей Q на основі t-розподілу Стьюдента, який є цільовою функцією, що використовується при оптимізації t-SNE, має властивості, які долають проблему скупчення і дозволяють t-SNE представляти як локальну так і глобальну структуру високовимірних даних на низьковимірному відображенні.

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{55}$$

Зокрема, градієнт t-SNE діє так, щоб розсовувати точки даних, які мають велику попарну відстань в багатовимірному просторі, але моделюються малими попарними відстанями в низьковимірному представленні. Градієнт t-SNE також зближує точки даних, які мають невелику попарну відстань в просторі високої розмірності, але велику попарну відстань у низьковимірному представленні. Нарешті, на ті точки, які потрапляють між цими двома випадками, не впливає градієнт t-SNE, оскільки він приймає значення нуля, коли точки мають «подібну» попарну відстань як у низьких, так і у великих розмірностях. Це, наприклад, відрізняється від градієнта

SNE, який лише створює тяжіння між точками даних з низькою попарною відстанню у великому розмірному просторі та великою попарною відстанню при низьковимірному відображенні, але нічого не робить в іншому випадку [van der Maaten and Hinton, 2008].

Експерименти з іншими методами:

[van der Maaten and Hinton, 2008] провів кілька експериментів на різних наборах даних, порівнюючи якість візуалізації між t-SNE і Sammon mapping, Isomap, LLE, CCA, SNE, MVU і Laplacian Eigenmaps. Прекрасним прикладом збереження t-SNE як локальної, так і глобальної структури вихідних даних є набір даних MNIST. Представлення t-SNE цих даних утворює десять чітких кластерів, що відповідають цифрам 0-9 причому такі цифри, як 3 і 8 візуалізуються відносно близько один до одного, але також відносно далеко від 7 і 4. Це демонструє відображення t-SNE глобальної структури набору даних MNIST. Крім того, вивчення даних всередині кожного кластера показує, що t-SNE також фіксує локальну структуру, таку як орієнтація рукописних цифр. На відміну від цього, Sammon mapping створює те, що виглядає як однорідна сфера точок даних із перекриттям відображень класів цифр. Лише шість із десяти класів добре видно. Isomap та LLE також створюють відображення, в яких класи мають значне перекриття [van der Maaten and Hinton, 2008].

Ці методи були застосовані до інших наборів даних з дещо подібними результатами. Чітка відмінність між t-SNE та іншими методами нелінійного зменшення розмірності заключається в можливості t-SNE створювати кластери з подібними точками, та ефективно відокремлювати різнорідні кластери один від одного. Результатом інших методів, як правило, є локальні кластери, які суттєво перетинаються зі своїми сусідами у низьковимірному представленні.

Рекомендації щодо використання:

На відміну від традиційних методів зменшення лінійної розмірності (PCA, MDS), t-SNE використовує навчання для представлення даних велокої розмірності у меншу розмірність. Тому конкретне відображення, яке буле створено за допомогою t-SNE, залежить від декількох гіперпараметрів, а саме від складності та швидкості навчання. Більше того, вибір значень гіперпараметрів іноді може призвести до відображень, які не чітко або не правильно представляють справжню базову структуру даних. Тому інтерпретація результатів, отриманих за допомогою t-SNE, є більш складною у порівнянні з PCA або MDS і вимагає більшої кількості експериментів та уважності. Наприклад, відстань між кластерами сильно залежить від вибору перплексії, і деякі варіанти можуть не дати правильної глобальної структури [Wattenberg et al., 2017]. Вибір перплексії також може суттєво вплинути на форми кластерів, породжених відображенням t-SNE, які знову ж таки не завжди точно відображають вихідні дані [Wattenberg et al., 2017]. Тому важливо отримати інтуїтивне уявлення про те, як перплексія і швидкість навчання можуть впливати на відображення, породжені t-SNE для різних базових даних, щоб ефективно використовувати t-SNE на практиці.

7 Шаблони природних сигналів

Нейронні мережі можна використовувати для моделювання аудіо, зображень, тексту чи інших сигналів. Сигнали представляються у вигляді послідовності скалярів. Аудіо часто представляється як хвильові висоти, а зображення як значення пікселів, а текст часто представляється у

вигляді векторів унітарного коду.

Природні сигнали (не штучні чи синтетичні), як правило, демонструють дві важливі закономірності, які обидві мають вирішальне значення для згорткових нейронних мереж.
1. Стаціонарність - Шаблони всередині сигналу повторюються в багатьох місцях протягом сигналу. Тип функцій, що зустрічаються в сигналі, не залежить від місця розташування всередині сигналу. Статистика однієї частини сигналу така ж, як і будь-якої іншої частини. Деякі джерела також зазначають, що стаціонарність передбачає, що "функції, корисні в одній ділянці, також можуть бути корисними для інших ділянок."[UFLDL]

2. Локальність - Кореляція між близькими точками даних є високою, проте зменшуться чим далі знаходяться точки даних між собою.

Приклад для аудіо

Стаціонарність - Частота хвилі звукових сигналів утворює синусоїдальний малюнок із подібними підсегментами піків та долин, що неодноразово трапляються протягом сигналу.

Локальність - Кореляція висока між двома піками звукової хвилі в сусідні моменти часу, але кореляція низька між віддаленими піками. Звуки мають " локальні "властивості, такі як тимчасовіші або плавніші в частотно-часовому інтервалі. [Espi et al.]. If an audio signal were shuffled, so that the index of the data no longer represented a position in time, the audio signal would no longer adhere to the stationarity and locality assumptions.

Приклад для зображення

Стаціонарність - Значення пікселів частини зображення можуть повторюватися в декількох місцях зображення. Наприклад, об'єкти на зображенні можуть утворювати тінь із сірих пікселів. Ділянки тіні можуть виникати багато разів у декількох місцях на зображенні.

Локальність - - Пікселі, які дуже близькі один до одного на зображенні, можуть мати однакові або подібні кольорові значення, але при порівнянні пікселів, що знаходяться все далі та далі, стає менш імовірним, що значення пікселів співвідносяться між собою.

7.1 Як щодо довгого вхідного сигналу?

Розглянемо повністю підключену лінійну мережу. Коли дані передаються через лінійний шар, кожен сигнал множиться на матрицю ваг $\boldsymbol{W}$ щоб отримати наступний прихований шар ($h = \boldsymbol{W}\vec{x}$). Розміри матриці вагів повинні відповідати довжині вектора сигналу. За дуже довгих векторів сигналу розмір цієї матриці вагів та кількість ваг, які можна засвоїти шляхом зворотного розповсюдження, можуть стати неможливими. Це викликає потребу в операторі згортки. Нейронні мережі із згортковими шарами є обчислювально ефективними.

Для побудови оператора згортки ми почнемо з розгляду множення матриці на вектор. Помноживши матрицю на вектор, вийде ненормована відстань з косинусом між рядками матриці та вектором, або вирівнювання кожного рядка з вектором. Натомість ми можемо використовувати властивості стаціонарності та локальності даних для створення обчислювально ефективного обчислення, яке зберігає найважливішу інформацію.

7.2 Побудова оператора згортки

Нехай в нас є матриця $\boldsymbol{A}$ і вектор $\boldsymbol{x}$ (наш вхідний сигнал) представлені як:

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,k} \end{bmatrix} = \begin{bmatrix} \text{——} & \boldsymbol{a}^{(1)} & \text{——} \\ \text{——} & \boldsymbol{a}^{(2)} & \text{——} \\ & \vdots & \\ \text{——} & \boldsymbol{a}^{(m)} & \text{——} \end{bmatrix}_{m \times k}, \ \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}_{n \times 1}$$

так що $n \gg k$. Очевидно, що вони не піддаються безпосередньому множенню. Тому ми використаємо припущення *локальності*, і розширимо рядки $\boldsymbol{A}$ щоб охопити n стовпці, заповнивши $n-2$ стовпці нулями. Заповнення здійснюється нулями, оскільки ми припускаємо, що елементи в сигналі, які знаходяться далеко один від одного, не впливають один на одного. Таким чином наша модифікована матриця $\hat{\boldsymbol{A}}$ має вигляд:

$$\hat{\boldsymbol{A}} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & 0 & 0 & \ldots & 0 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 & 0 & \ddots & 0 \\ a_{m,1} & a_{m,2} & \cdots & a_{m,k} & 0 & 0 & \ldots & 0 \end{bmatrix}_{m \times n}$$

Тому,

$$\hat{\boldsymbol{A}}\boldsymbol{x} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & 0 & 0 & \ldots & 0 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 & 0 & \ddots & 0 \\ a_{m,1} & a_{m,2} & \cdots & a_{m,k} & 0 & 0 & \ldots & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} =$$

$$= \begin{bmatrix} \text{——} & \hat{\boldsymbol{a}}^{(1)} & \text{——} \\ \text{——} & \hat{\boldsymbol{a}}^{(2)} & \text{——} \\ & \vdots & \\ \text{——} & \hat{\boldsymbol{a}}^{(m)} & \text{——} \end{bmatrix} \begin{pmatrix} | \\ \boldsymbol{x} \\ | \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{a}}^{(1)}\boldsymbol{x} \\ \hat{\boldsymbol{a}}^{(2)}\boldsymbol{x} \\ \vdots \\ \hat{\boldsymbol{a}}^{(m)}\boldsymbol{x} \end{pmatrix}_{m \times 1}$$

Далі ми хочемо створити функцію будування згортки. Іншими словами ми хочемо використати припущення *стаціонарності* і побачити ефект (проекцію) деякого заданого вектора $\boldsymbol{a^{(i)}}$ на підвектори $\boldsymbol{x}$ довжини k, рухаючись по одному (або більше) кроку за раз по вектору $\boldsymbol{x}$. Отже ми хочемо, щоб j-им елементом i-ї карти особливості був $\boldsymbol{a^{(i)}}\boldsymbol{x}_{j:j+k-1}$, де $\boldsymbol{x}_{j:j+k-1}$ підвектор від x_j до x_{j+k-1} (включно).

Очевидно, таких функціональних карт буде m ; побудуємо один для $\boldsymbol{a^{(1)}}$. Для цього побудуємо $(n-k+1) \times n$ матрицю, взявши k елементів $\boldsymbol{a}^{(1)}$, повторюючи їх по головній діагоналі, і доповнивши решту нулями, наступним чином:

$$\boldsymbol{T}^{(1)} = \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,k} & 0 & 0 & \ldots & 0 \\ 0 & a_{1,1} & a_{1,2} & \ldots & a_{1,k} & 0 & \ldots & 0 \\ 0 & 0 & a_{1,1} & a_{1,2} & \ldots & a_{1,k} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & a_{1,1} & a_{1,2} & \ldots & a_{1,k} \end{bmatrix}_{(n-k+1) \times n}$$

Така матриця називається **матриця Тепліца**. Подібним чином можна створити m таких матриць Тепліца для кожного $\boldsymbol{a}^{(i)}$ where $i \in \{1, 2, \cdots, m\}$, які відповідають m функціональним картам об'єктів. Нижще зображено для $k = 3$:

$$\boldsymbol{T}^{(1)}\boldsymbol{x} = \begin{pmatrix} \boldsymbol{a}^{(1)}\boldsymbol{x}_{1:1+k-1} \\ \boldsymbol{a}^{(1)}\boldsymbol{x}_{2:2+k-1} \\ \vdots \\ \boldsymbol{a}^{(1)}\boldsymbol{x}_{n-k+1:n} \end{pmatrix}_{(n-k+1)\times 1} = \begin{pmatrix} \boldsymbol{a}^{(1)}\boldsymbol{x}_{1:3} \\ \boldsymbol{a}^{(1)}\boldsymbol{x}_{2:4} \\ \vdots \\ \boldsymbol{a}^{(1)}\boldsymbol{x}_{n-2:n} \end{pmatrix}_{(n-2)\times 1}$$

Матриця побудована таким чином, щоб ми могли отримати такий тип згортки, *i.e.* це дозволяє нам розглядати лише k суміжних елементів одночасно, тим самим використовуючи припущення *локальності* про відсутність залежності між елементами розташованими далеко (в цьому випадку, далі ніж $k-1$ елементів) один від одного.

Тепер, коли ми розуміємо, як виводяться згортки, важливо пов'язати їх із традиційним поняттям згорткових нейронних мереж. Проводячи аналогії, з тим, що ми бачили вище, відповідає ConvNet, який:

- є одновимірним: і $\boldsymbol{x}$ і $\boldsymbol{a}^{(1)}$ вектори а не матриці
- має ядро розміром k ($= 3$ в прикладі вище). Ми використали припущення локальності, обмежившись $k-1$-ми сусідами певної комірки. Відповідно, ядра мають $\overleftarrow{\boldsymbol{a}^{(i)}}$, враховуючи, що коли ми виконуємо згортку, ми її перевертаємо.
- має крок $s = 1$: Ми переміщуємось лише на один елемент у $\boldsymbol{x}$ на кожному кроці (*e.g.* $\boldsymbol{x}_{1:3} \to \boldsymbol{x}_{1+s:3+s} = \boldsymbol{x}_{2:4}$).

Для подальшого читання читачеві рекомендується звернутися до щоденника Крістофера Олаха про згортки, Dumoulin and Visin [2016] та їх відповідність GitHub репозиторій про згорткову арифметику, і слайди Андрія Карпаті по ConvNets (cs231n).

8 Згортки

Світ композиційний. У згортковій мережі об'єкти високого рівня - це композиції об'єктів низького рівня. Ось чому ми, як правило, бачимо кілька згорнутих шарів.

8.1 Згортки еквіваріантні змінам

Тут *еквіваріант до зміни* означає, що зміна функцій введення призводить до еквівалентної зміни результатів. Отже якщо вхід 1,2,3,0,0 приводить до 1,0,0 в результаті згортки, то вхід 0,0,1,2,3, де вхідні значення зміщені на 2 вправо, може привести до 0,0,1 де попередні вхідні значення зміщені на 2 вправо. Ця характеристика має сенс, оскільки згортковий шар по суті є детектором ознак. Наприклад, на двох зображеннях різних граней згортковий фільтр, який виявляє око, виявлятиме їх у відповідних місцях, як показано на figure 32.

Навпаки, *інваріантність перетворень*означає, що результат буде абсолютно однаковим для перетворень вхідних функцій. Отже, якщо вхід 1,2,3,0,0 перетворюється в 1,0,0 в результаті операції, яка є інваріантною до трансляції, тоді вхід 0,0,1,2,3, де вхідні значення зміщені на 2 в бік вправо, призведе до точно таких же результатів 1,0,0, як і раніше. Наприклад, якщо на зображенні є собака, то модель, яка класифікує тварин, передбачатиме собаку незалежно від

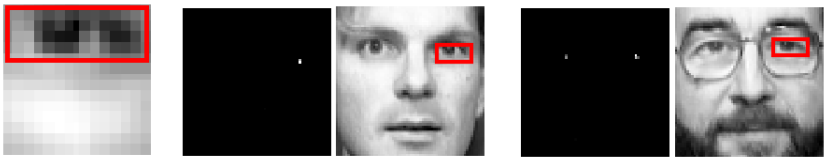


Рис. 32: Зліва на право:зображення лівого ока, який виявляє згортковий фільтр, і 2 пари (його вихід та відповідне виявлення ока). Створено тренуванням згорткового автокодера переможець забирає все, на даних Olivetti Face Data. (Рисунки з Park and Ni [2017])

того, де вона знаходиться на зображенні. Класифікатор CNN приблизно досягає незмінності трансляції за допомогою шарів об'єднання. Об'єднуючий шар повертає лише максимальне значення у своєму полі введення, тому вихід мережі стає толерантним до невеликих, неважливих збурень. Оскільки розмір подання зменшується до 1x1, мережа стає толерантною до більших перекладів.

Інваріантність перекладу CNN може бути проблемою, якщо CNN будується спеціально, коли кінцевий згортковий шар із розміром репрезентації 1x1 навчається виявляти дрібні частини об'єктів, таких як колеса, фари та двері. В такому випадку, навіть якщо частини вхідного зображення будуть розміщені на випадкових неправильних позиціях, мережа все ще може класифікувати це як зображення автомобіля, оскільки просторова інформація про те, де особливості з'являються у вхідному зображенні, втрачається. У такому випадку більш глибока архітектура, яка виявляє особливості вищого рівня на остаточному згортковому шарі, повинна ігнорувати особливості низького рівня, щоб мати ненавмисні прогнози, хоча вона все одно не буде розрізняти невеликі розбіжності. Більше того, для CNN важко бути толерантним до інших перетворень, таких як обертання та розтягування, через його конструкцію, якщо ми не будуємо дуже глибоку та широку мережу та / або не використовуємо експоненційно більше маркованих даних [Canziani, 2017].

Капсульна мережа [Sabour et al., 2017] намагається вирішити цю проблему будучи *інваріантною до точки зору*: це приводить до вихідної сутності, для якої вхідними даними є проекції і трансформації. Більше того, вона виявляє трансформації інформації вхідних данних, такі якобертання, розтягування та зсув від початкової сутності, роблячи мережу *еквіваріантною до трансляції* в той же час.

8.2 Одновимірна згортка

Традиційно

$$Y_i = \sum_{j \in [0,k-1]} W_j X_{i+j}$$

Градієнти:

$$\frac{\partial C}{\partial x_k} = \sum_{j \in [0,k-1]} W_{k-j} y_k$$

$$\frac{\partial C}{\partial W_k} = \sum_i \frac{\partial C}{\partial y_i} x_{i+k}$$

8.3 Одновимірна згортка векторів

$$\boldsymbol{x} = (x_1 x_2 \dots x_n)^\top$$
$$\boldsymbol{w} = (w_1 w_2 \dots w_k)^\top$$

Тоді,

$$(\boldsymbol{x} * \boldsymbol{w})_i = \sum_{\max(1,i-k+1)}^{\min(i,n)} x_j w_{i-j+1}$$

8.4 Похідна одновимірної згортки

$$\boldsymbol{y} = \boldsymbol{x} \star \boldsymbol{w} = \boldsymbol{W}\boldsymbol{x}$$

де $\boldsymbol{x}$ і $\boldsymbol{y}$ стовпці, і $\boldsymbol{W}$ - матриця Тепліца, пов'язана з $\boldsymbol{w}$

$$\frac{\partial C}{\partial \boldsymbol{x}} = \frac{\partial C}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial C}{\partial \boldsymbol{y}} \boldsymbol{W}$$

де $\frac{\partial C}{\partial \boldsymbol{x}}$ і $\frac{\partial C}{\partial \boldsymbol{y}}$ - рядки

$$\nabla_{\boldsymbol{x}} C \doteq \left(\frac{\partial C}{\partial \boldsymbol{x}}\right)^\top = \left(\frac{\partial C}{\partial \boldsymbol{y}} \boldsymbol{W}\right)^\top = \boldsymbol{W}^\top \nabla_{\boldsymbol{y}} C = \nabla_{\boldsymbol{y}} C \star \overleftarrow{\boldsymbol{w}}$$

Зверніть увагу:

1. Якщо $\boldsymbol{x} = \big(x_1 x_2 \dots x_n\big)^\top$, $(\nabla_{\boldsymbol{x}} C)^\top \doteq J_{\boldsymbol{x}} C \doteq \frac{\partial C}{\partial \boldsymbol{x}}$ як

   $$\nabla_{\boldsymbol{x}} C \doteq \begin{pmatrix} \frac{\partial C}{\partial x_1} \\ \frac{\partial C}{\partial x_2} \\ \vdots \\ \frac{\partial C}{\partial x_n} \end{pmatrix}_{n \times 1}$$

   і

   $$J_{\boldsymbol{x}} C \doteq \big(\tfrac{\partial C}{\partial x_1} \tfrac{\partial C}{\partial x_2} \cdots \tfrac{\partial C}{\partial x_n}\big)_{1 \times n}$$

2. Щоб краще проілюструвати чому виконується остання еквівалентність, розглянемо випадок, коли $k = 3$

   $$\boldsymbol{w} = (w_1 w_2 w_3)^\top \tag{56}$$

   $$\boldsymbol{W} = \begin{bmatrix} w_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ w_2 & w_1 & 0 & 0 & \dots & 0 & 0 \\ w_3 & w_2 & w_1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & w_3 \end{bmatrix}$$

$$\boldsymbol{W}^\top = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & \dots & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & \dots & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}$$

Оскільки $\overleftarrow{\boldsymbol{w}} = (w_3 w_2 w_1)^\top$, ми можемо побачити, що, насправді, ми виконуємо згортку $\overleftarrow{\boldsymbol{w}}$оскільки порядок ненульових елементів у кожному рядку перевертається, незалежно від зміщення.

Чисельний приклад такий. Враховуючи що

$$\nabla_{\boldsymbol{y}} C = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

і

$$\boldsymbol{w} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Тоді

$$\nabla_{\boldsymbol{y}} C \star \overleftarrow{\boldsymbol{w}} = \begin{pmatrix} 4 \cdot 3 \\ 4 \cdot 2 + 5 \cdot 3 \\ 4 \cdot 1 + 5 \cdot 2 + 6 \cdot 3 \\ \vdots \end{pmatrix}$$

$$\boldsymbol{W}^\top \nabla_{\boldsymbol{y}} C = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

Ви можете помітити, що третій елемент ідентичний.

8.5 Двовимірна згортка

Двовимірна згортка — це розширення попередньої одновимірної згортки шляхом згортання горизонтального і вертикального напрямків в двовимірній просторовій області, зазвичай застосовується для таких завдань обробки зображень, як підвищення різкості, розмиття, виявлення країв.

Наступний приклад показує як працює операція двовимірної згортки. Спочатку, переверніть ядро по горизонталі і вертикалі. Потім помістіть перший елемент ядра в кожен піксель зображення (елемент матриці зображення).Тоді кожен елемент ядра буде стояти поверх елемента матриці зображення. Математичне вираження наведено в рівнянні 42

8.5.1 Різні варіанти

- Варіант «такий само» ("same") виводить масив того ж розміру, що і було отримано на введення.

- Варіант «повний» ("full") повертає весь результат.
- Варіант «дійсний» ("valid") повертає тільки ті елементи, які повністю покриті ядром.

9 Методи регуляризації та нормалізації

Завжди малюйте нейронні мережі від низу до верху, знизу - ваш введення, зверху - висновок.

Пакетна нормалізація творить дива. «Я навіть більше не використовую відсів».

По суті, від кожного пакета ви віднімаєте пакетне середнє, а потім ділите на пакет стандарт.

Як ви оцінюєте ці моделі? Точність не єдине. Також потрібно враховувати інші фактори, такі як використання пам'яті і енергоспоживання.

Більший пакет робить повнозв'язний шар швидше при тому ж обсязі обчислень, оскільки повнозв'язні шари використовують матричне множення.

Залишкове (що пропускається) з'єднання.
Коли ви вводите обхідні з'єднання, все стає справді добре.
Воно перетворює ваш графік втрат в більш опуклий.

1x1 перетворити в 1x1: дуже хороший трюк, який використовується InceptionNet.

9.1 Загальні методи регуляризації

Регуляризація - одна з найважливіших тем у глибокому навчанні (та машинному навчанні). Було виявлено та застосовано багато технік, що дозволяють алгоритмам краще працювати на нових вхідних даних, покращуючи узагальнюючу здібність. Існує декілька форм регуляризації, але ми зосередимося на двох основних техніках, широко вживаних у глибокому навчанні:

9.1.1 Пакетна нормалізація

Виходячи з назви можна зрозуміти, що ця техніка використовується для нормалізації даних у діапазоні від 0 до 1. Зазвичай дана процедура виконується під час передачі вхідних даних для того, щоб уникнути фактору масштабності, що базується на тренувальних даних. Однак розповсюдження цієї процедури на приховані шари допомагає нам підвищити продуктивність, подібно до того, як це роблять інші техніки регуляризації [Ioffe and Szegedy, 2015a]

Як працює пакетна нормалізація? Пакетна нормалізація нормалізує входи кожного шару таким чином, що вони мають середнє значення вихідної активації нуль та стандартне відхилення одиницю. Це аналогічно тому, як стандартизовані входи в мережі. Основна причина цього - зробити навчання наступного рівня більш ефективним. Однак у суспільстві глибокого навчання лишається тема для дискусії - слід нормалізовувати значення до чи після застосування функції активації? Наразі на практиці, як правило, краще працює та ширше застосовується нормалізація значень перед застосуванням активації.

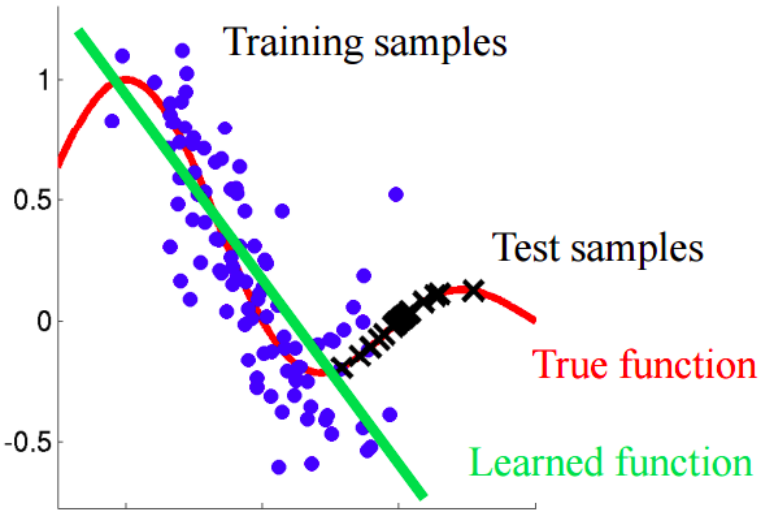


Рис. 33: Приклад коваріантного зсуву

Практичні переваги пакетної нормалізації:

Усунення коваріантного зсуву: Коваріантний зсув обумовлений розподілом навчальних даних. Для розуміння розгляннемо приклад: тренувальні дані складаються із зображень чорних котів, тоді як тестові дані також містять зображення котів різного кольору. Нейронна мережа, що навчалася на цьому незбалансованому наборі даних не зможе правильно спрогнозувати результат. Ця невідповідність між тренувальними та тестовими даними має назву "коваріантний зсув". Пакетна нормалізація ефективно усуває цей недолік, нормалізуючи всі значення на тренувальному і прихованих шарах та рівномірно розподіляє значення і ваги по усіх шарах нейронної архітектури. Коротку дискусію на цю тему можна переглянути на даному відео.

Зберагіє функції активацій: Розглянемо ситуацію, коли лише деякі з початкових вагів мають вищі значення та більшою мірою сприяють класифікації чи прогнозуванню результату. Цей дисбаланс призвів би до того, що деякі з активацій містили б вагомі значення, що каскадно передавалися б до інших шарів мережі. Це призводить до незбалансованих значень градієнта під час зворотнього розповсюдження, що спричиняє затухання градієнту. Пакетна нормалізація усуває це шляхом нормалізації кожного значення активації та результує потік рівномірно розподілених значень.

Робить навчання швидшим: Це одна з найвагоміших переваг використання пакетної нормалізації. Незважаючи на додаткові нормалізаційні обчислення, мережа навчається швидше за рахунок швидшої конвергенції.

Виконує певну регуляризацію: Пакетна нормалізація додає невеликий шум до мережі і, в деяких випадках, (наприклад, початкові модулі) було показано, що вона працює так само, як і відсівання.

9.1.2 Відсівання

Ви можете спитати, яке це відношення має до нейронної мережі? Давайте розглянемо приклад.

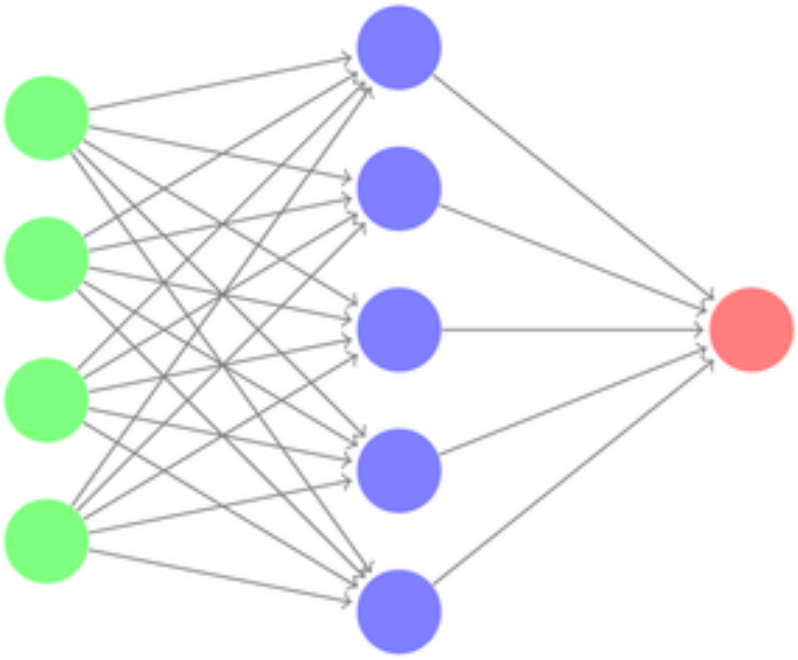


Рис. 34: Приклад простої нейронної мережі

Уявімо, що вхідний рівень (*зелений*) - це певне запитання від лектора, кожен нейрон у прихованому шарі (*синій*) - одна особа з аудиторії, а вихідний шар (*червоний*) - відповідь на задане запитання. Якщо вихідний нейрон виявить, що конкретний нейрон з прихованого шару завжди дає найкращу відповідь, він може знехтувати іншими і надати найбільшу вагу цьому нейрону. Припустимо, що ми вирішили заборонити відповідати одним нейронам і дали шанс іншим. Таким чином ми досягнемо рівноваги і змусимо вчитися усі нейрони з прихованого шару. Це точно описує концепцію відсівання, технічно вона працює наступним чином:

Визначається коефіцієнт відключення нейронів, який являє собою певну частку усіх нейронів (наприклад, 0.2). На кожному етапі відключаються випадково обрані нейрони відповідно до заздалегідь визначеної частки. Кінцевий результат обчислюється відповідно до комбінації результатів решти нейронів. З цією технікою усі нейрони матимуть шанс внести свій вклад у відповідь і будуть зобов'язані відповідати правильно, щоб зменшити функцію втрат моделі.

Отже, відсівання відноситься до ігнорування певного випадково обраного набору нейронів протягом тренувальної фази. В загалому, дана процедура відключає деякі нейрони та допомагає зменшити перенавчання. [Srivastava et al., 2014] Чому це важливо? Причина полягає в тому, що повністю зв'язаний шар мережі займає більшість параметрів, тому нейрони розвивають взаємозалежність між собою, що, в свою чергу, зменшує індивідуальну важливість кожного з нейронів та може призвести до надмірного пристосування до тренувальних даних. Значення

можливості вимкнення нейронів зазвичай встановлюється 0.5, хоча можна використовувати і значення 0.3 або 1.0. Коли значення проріджування встановлено в 1.0, ми просто не відключаємо жоден з нейронів. Але спільнота глибокого навчання стверджує, що 0.5 зазвичай є найкращим вибором.
Після закінчення навчання сважливо відключити фазу відсіву на етапах розробки та тестування. В іншому випадку прогнози цієї моделі не будуть стабільними, так як відсів додає певну частку невизначеності. Для наглядності нижче наведена схема нейронної мережі з відсівом.

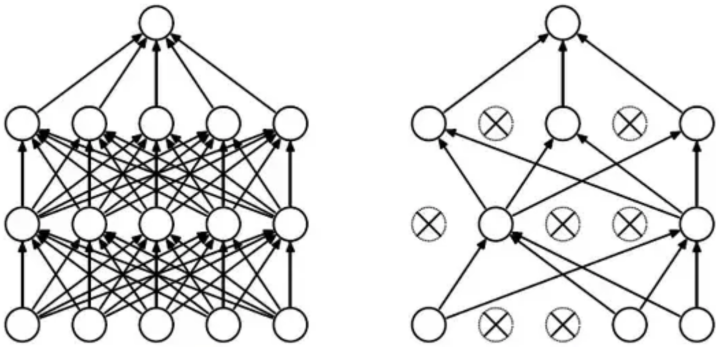


Рис. 35: Приклад нейронної мережі до і після застосування відсіювання

9.1.3 Порівняння відсівання та пакетної нормалізації

Відсівання - здебільшого спосіб регуляризації нейронної мережі, тоді як пакетна нормалізація забезпечує загальну стабільнсть мережі. Іншими словами, можна зрозуміти, що пакетна нормалізація - техніка для поліпшення оптимізації. Коли маємо великий набір даних, більш важливим фактором є оптимізація, ніж регуляризація, тому пакетна нормалізація корисніша для великих даних. Звісно, є можливість одночасного використання техніки відсіювання та пакетної нормалізації, але виявляється, що в більшості випадків це контрпродуктивно - може виявитись, що більшу частину часу працює тільки пакетна нормалізація.

Ключем до розуміння несумісності відсіювання та пакетної нормалізації є непослідовність поведінки нейронних дисперсій під час перемикання стану мережі. Враховуючи одну реакцію нейрону, коли стан змінюється з тренувального на тестовий, проріджування масштабуватиме відповідь враховуючи власний коефіцієнт відсіву, що фактично змінює дисперсію, в той час як пакетна нормалізаці все ще зберігає свою статистичну дисперсію. Ця невідповідність дисперсій може призвести до чисельної нестабільності. По мірі того, як заглиблюється сигнал, числове відхиленння кінцевих прогнозів може посилюватися, що знижує продуктивність системи. Ця концепція відома як "зсув дисперсії"для простоти. Натомість без проріджування реальні нейронні дисперсії результату виявлялися б дуже близькими до рухомих, накопичених пакетною нормалізацією, що також відповідає вищій точності при тестуванні. Стаття "Understanding the disharmony between batch normalization and dropout"від Li et al. [2018] допоможе заглибитись у подробиці.

9.1.4 L^1 регуляризація

Доки L^2 регуляризація є найбільш широко вживаною, L^1 також може бути використано для штрафування параметрів моделі. Порівняно з L^2 регуляриацією ,L^1 регуляризація призводить до розрідженого розв'язку, внаслідок чого підмножина ваг стає нульовою. Завдяки цій властивості L^1 широко використовується як інструмент для вибору ознак.

За допомогою L^1 регуляризації, проблему оптимізації квадратичної помилки можна сформувати як:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{1}{N}(Xw-y)^T(Xw-y) \\ \text{subject to} \quad & |w| \leq C \end{aligned}$$

Як описано в розділі 7.1.1, проблема стає:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{1}{N}(Xw-y)^T(Xw-y) + \frac{\lambda}{N}|w| \\ \underset{w}{\text{minimize}} \quad & E(w) + \frac{\lambda}{N}|w| \end{aligned}$$

9.1.5 Розширення даних

Одним із найефективніших способів покращити узагальнюючу здібність моделі є тренування на більшій кількості даних. У більшості випадків ми обмежені наявним обсягом даних, але для деяких задач можливо виконати розширення. Цей метод особливо ефективний для задачі розпізнавання об'єктів, де дані мають великі розмірності з великою варіативністю ознак. Ця варіативність може бути просто змодельована, наприклад, виконуючи переклад зображення зі зсувом, або навіть просте обертання чи масштабування є потужними методами для покращення узагальнюючої здібності моделі навіть після використання таких методів, як згортка та об'єднання. Розпізнавання мовлення - ще одне поле, де розширення даних часто використовується; у наслідок пертурбації можна виділити такі ознаки як висота тону або швидкість.

Додавання шуму до сигналу також можна вважати як тип техніки розширення даних. Навчання з додаванням випадкових шумів підвищує стійкість моделі до різного роду шуму. Окрім того, інжекційний шум у цільових змінних допомагає узагальненню навчання у випадках, коли мітки не дуже точні.

Де ми можемо розширювати дані у конвеєрі машинного навчання? Відповідь може здатися цілком очевидною; ми проводимо збільшення, перш ніж подавати дані в модель, чи не так? Так, але тут маємо два варіанти. Один із варіантів - виконати всі необхідні перетворення заздалегідь, істотно збільшивши розмір вашого набору даних. Інший варіант - виконати ці перетворення у міні-пакеті, безпосередньо перед тим, як подати його до моделі машинного навчання.

Перший варіант відомий як автономне збільшення. Цей метод є кращим для порівняно менших наборів даних, оскільки в реультаті ви збільшите розмір набору даних в таку кількість раз, що рівна кількості виконаних перетворень (наприклад, використавши усі зображення, розмір даних збільшиться у 2 рази).

Другий варіант відомий як онлайн-розширення або "розширення на льоту". Цей метод є кращим для більших наборів даних, оскільки ми не можемо дозволити суттєве збільшення розміру. Натомість виконується перетворення у міні-пакетах, які подаються на моделі. Деякі

фреймворки машинного навчання мають підтримку онлайн-розширення, яке можна прискорити на графічному процесорі.

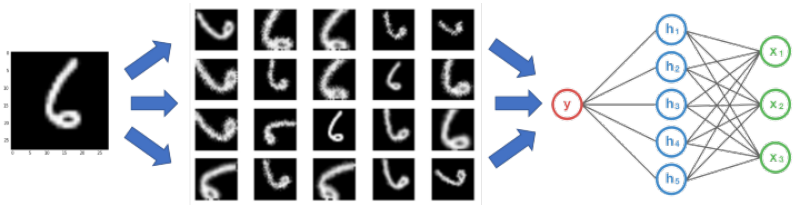


Рис. 36: Приклад розширення даних на наборі даних MNIST

9.1.6 Рання зупинка

При тренуванні нейронної мережі з великими наборами даних зміщення зменшувалося, дисперсія зростала, а помилка навчання зменшувалася на кожній ітерації навчання.Однак після певного моменту ми починаємо спостерігати збільшення помилки перевірки, тобто збільшення помилки узагальнення (див. малюнок 37 нижче для прикладу такої поведінки).

Рис. 37: Криві тренування, що показують відношення валідаційної і тренувальної втрат, де існує U-подібна крива ефективності валідаційного набору (source)

Знаючи цю особливість, ми можемо вибрати ті параметри, при яких помилка на валідаційних даних була мінімальною. На кожній ітерації, якщо помилка на валідаційному наборі покращується, ми зберігаємо параметри моделі доти, доки значення помилки не починають зростати. Цей метод відомий як "Рання Зупинка"**Early Stopping** і це один із найбільш вживаних методів через свою простоту та ефективність. Це також може бути розглянуто як алгоритм налаштування гіперпараметрів, де ми контролюємо проблему перенавчання.

9.1.7 Зниження ваги (L^2 регуляризація параметрів)

Внаслідок втрат квадратичної помилки ми маємо необмежену проблему оптимізації:

$$E(w) = \frac{1}{N}(Xw - y)^T(Xw - y)$$

де X - вхідні дані, y - вихідні та w ваговий параметр, який ми хочемо знайти за допомогою рівняння $w^* = \arg\min_w E(w)$. Без додаткових доказів ми знаємо, що вищезазначена цільова функція має наступне рішення замкнутої форми: $w^* = (X^TX)^{-1}X^Ty$. Апроксимацію легко вирішити, але конфігурація, що повертається, може призвети до проблеми перестановки, коли кількість вхідних даних у моделі велика, а сама модель - складна. Одним із підходів до вирішення цієї проблеми є регулювання можеді шляхом обмеження w^Tw через ваговий простір. Тоді задача оптимізації може бути сформульована наступним чином:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{1}{N}(Xw - y)^T(Xw - y) \\ \text{subject to} \quad & w^Tw \leq C \end{aligned}$$

де C - певна константа. Позначимо обмежену вагу як w_{reg}. Базуючись на основах лінійної алгебри, ми знаємо, що градієнт від $E(w_{reg})$ є ортогональним до $E(w)$ і спрямований у зворотньому напрямку від w_{reg} з метою мінімізації $E(w)$. Тоді градієнт від $E(w_{reg})$ пропорційний до $-w_{reg}$ який можна записати як $-2\frac{\lambda}{N}w_{reg}$ для майбутньої зручності використання. Це означає, що:

$$\begin{aligned} \nabla E(w_{reg}) &\propto -w_{weg} \\ &= -2\frac{\lambda}{N}w_{reg} \\ \nabla E(w_{reg}) + 2\frac{\lambda}{N}w_{reg} &= 0 \end{aligned}$$

де λ є певною константою, що залежить від C. Більша C вказує на меншу λ, і навпаки. Помітимо, що ліва частина цього рівняння є диференціацією $\frac{1}{N}(Xw - y)^T(Xw - y) + \frac{\lambda}{N}w^Tw$. Тоді обмежена задача оптимізації може бути сформульована як:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{1}{N}(Xw - y)^T(Xw - y) + \frac{\lambda}{N}w^Tw \\ \Rightarrow\underset{w}{\text{minimize}} \quad & E(w) + \frac{\lambda}{N}w^Tw \end{aligned}$$

що набагато простіше вирішити. Тепер для оновлення параметра ваги з урахуванням наведеного вище формулювання ми можемо використовувати алгоритм градієнтного спуску, як зазвичай, за допомогою:

$$\begin{aligned} w(t+1) &= w(t) - \eta\nabla E(w(t)) - 2\eta\frac{\lambda}{N}w(t) \\ &= w(t)(1 - 2\eta\frac{\lambda}{N}) - \eta\nabla E(w(t)) \end{aligned}$$

де η - швидкість навчання. Наведене вище правило оновлення називається зниженням ваги (або L^2 регуляризацією параметрів $-2\eta\frac{\lambda}{N}$, що змушує вагу експоненційно прямувати до нуля, якщо планується жодних оновлень. На практиці цей спосіб штрафує великі ваги і ефективно обмежує гнучкість нашої моделі.

9.1.8 Проріджування

Іншим популярним способом регуляризації є проріджування, запропонованее G.E.Hinton у 2012. Це прямий алгортм: на кожному навчальному кроці кожен нейрон (включаючи вхідні нейрони, але за вийнятком вхідних нейронів) має ймовірність p тимчасово "ігноруватись але він може бути активним наступного кроку. Гіперпараметр p називаєтьсякоефіцієнт проріджування, $(1-p)$ називається ймовірністю збереження. Нейрони більше не падають після навчання.

Відповідно до Géron [2017], одним з способів зрозуміти потужність концепції проріджування є той факт, що на кожному етапі навчання генерується унікальна нейронна мережа. Припустимо, що мережа має N нейронів, які допустимо проігнорувати, тоді в загалому існує 2^N можливих мереж, оскільки кожен з нейронів може бути або проігнорованим, або ні. Це настільки велика мережа, що для однієї і тієї ж нейронної мережі стає практично неможливо взяти одну й ту саму вибірку двічі, а отриману мережу можна розглядати як усереднений ансамбль усіх менших нейронних (під)мереж.

9.2 Пакетна нормалізація та нормалізація шару

9.2.1 Мотивація

При навчанні глибоких нейронних мереж необхідні методи **нормалізації**. Ідея нормалізації полягає в тому, щоб нормалізувати кожну скалярну ознаку окремо, досягнувши нульового середнього значення та одиничної дисперсії. Оскільки розподіл вхідних даних кожного шару змінюється під час навчання, так як і параметри попередніх шарів, це уповільнює навчання. Цей ефект посилюється із збільшенням глибини мережі.
Таке явище називається **Внутрішній Коваріантний Зсув**: зміна розподілу активацій мережі через зміну параметрів мережі під час навчання. Мотивація пакетної нормалізації **(Batch Normalization)** полягає в згладжуванні цього проблемного зсуву у вхідних розподілах.

9.2.2 Основна ідея та алгоритм BN перетворення

Основна ідея полягає в нормалізації за допомогою міні-пакетної статистики. Входи, що подаються в кожен наступний шар, будуть нормалізовані, використовуючи середню дисперсію активацій для всього міні-пакету. Показано як операція нижче.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathbb{V}[x^{(k)}] + \epsilon}}$$

(де ϵ це невеликий випадковий шум для забезпечення стабільності)
Просте використання цієї операції змінює здатність представлення шару, оскільки нормалізація спотворює результати навчання шару. Некоректно обмежувати вхідні дані, щоб дотримуватися нормального розподілу для кожного шару, оскільки, в результаті здатність мереж до навчання також буде обмежена. Тому для виправлення цієї ситуації вводяться два параметри, що піддаються навчанню. Вони застосовуються до кожного елемента і обробляються разом із вихідними параметрами моделі.

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Встановлюючи $\gamma^{(k)} = \sqrt{\mathbb{V}[x^{(k)}]}$ і $\beta^{(k)} = \mathbb{E}[x^{(k)}]$, вихідні активації можна відновити, а отже, і потужність представлення мережі може бути відновлена.

Ідеальна ситуацією буде використання середнього значення та дисперсії всього набору даних, але це нереально. Таким чином, середнє значення та дисперсія партії використовується як оцінка простоти.

Алгоритм пакетної нормалізації (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

9.2.3 Навчання та умовивід

BN перетворення - це диференційоване перетворення, яке вносить нормалізовані активації в мережу. Це гарантує, що, коли модель навчається, рівні можуть продовжувати навчання на розподілах вхідних даних, які демонструють менший внутрішній коваріантний зсув, тим самим прискорюючи навчання. Крім того, вивчене афінне перетворення, застосоване до цих нормалізованих активацій, дозволяє BN-перетворенню представляти перетворення ідентичності та зберігає пропускну здатність мережі.

Модель, що використовує пакетну нормалізацію, можна навчити, використовуючи пакетний градієнтний спуск, або SGD з розміром міні-партії $m > 1$. Далі підсумовується процедура навчання BN мереж.

Щоб вихідні дані детерміновано залежали лише від вхідних даних, необхідно, щоб як тільки мережа пройшла навчання,застосовувалася нормалізація з використанням сукупності, а не міні-пакетної статистики.

$$\widehat{x} = \frac{x - \mathbb{E}[x]}{\sqrt{\mathbb{V}[x] + \epsilon}}$$

Ми використовуємо неупереджену оцінку дисперсії $\mathbb{V}[x] = m/(m-1) * \mathbb{E}_\beta[\sigma_\beta^2]$, де очікування є перенавченим міні-пакетом розміру m, а σ_β^2 – дисперсія. Оскільки середнє значення та дисперсія фіксовані під час умовиводу, нормалізація - це просто лінійне перетворення, що застосовується до кожної активації. Алгоритм 2 узагальнює процедуру навчання пакетно-нормованих мереж.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

9.2.4 Нормалізовані пакетні згорткові мережі

Пакетна нормалізація може застосовуватися до будь-якого набору активацій у мережі. Наприклад, ми маємо перетворення, яке складається з афінного перетворення, за яким йде елементна нелінійність: $z = g(\boldsymbol{W}\boldsymbol{u} + \boldsymbol{b})$, де $\boldsymbol{W}$ та $\boldsymbol{b}$ досліджувані параметри моделі. Ми додаємо BN перетворення безпосередньо перед нелінійністю, нормуючи $x = \boldsymbol{W}\boldsymbol{u} + \boldsymbol{b}$. Отримуємо $z = g(BN(\boldsymbol{W}\boldsymbol{u}))$, де BN трансформація застосовується незалежно до кожного виміру $x = \boldsymbol{W}\boldsymbol{u}$. Для згорткових шарів ми хочемо, щоб різні елементи однієї і тієї ж карти об'єктів у різних місцях нормалізувались однаково. Для цього ми спільно нормалізуємо всі активації в міні-партії у всіх місцях.

Нормалізація пакету запобігає посиленню незначних змін параметра до більших та неоптимальних змін активацій у градієнтах. Це також робить навчання більш стійкими до масштабу параметрів. Наприклад, при пакетній нормілізації зворотне поширення через шар не залежить від масштабу його параметрів. Більше того, більші ваги призводять до менших градієнтів, і пакетна нормалізація стабілізує зростання параметрів.

9.2.5 Порівняння з нормою шару в RNN

Переваги
Зараз стохастичний градієнтний спуск (SGD) виявився основним підходом до глибокого навчання. Дане рішення є ефективним, але все одно вимагає багато часу та багато зусиль для налаштування таких гіперпараметрів, як швидкість навчання, початкові значення, відсівання, параметр регуляризації тощо. Але з BN життя стає набагато простішим. BN дозволяє підвищити швидкість навчання без побічних ефектів, щоб прискорити тренування, а також прискорити спад швидкості навчання. Це усуває відсівання та зменшує регуляризацію L2. Він замінює нормалізацію локальної відповіді (наприклад, використовується в мережі Alexnet). Це дозволяє ретелініше перемішувати навчальні дані, щоб запобігти постійному вибору одного і того ж прикладу в міні-пакеті, а також покращити дані перевірки на 1%. Це також зменшує фотометричні спотворення.

Недоліки порівняння з нормою шару в RNN
Нормалізація пакетів в значній мірі базується на розмірі міні-партії, в результаті чого її важко застосувати до повторюваних архітектур. Для подолання обмежень нормалізації пакету, було введено **нормалізацію шару**.
Замість обчислення середнього значення та дисперсії пакету в пакетній нормі, тепер використовується середнє значення та дисперсія всіх підсумованих входів до нейронів у шарі для кожної ітерації нормалізації шару. Таким чином, норма шару не залежить від розміру міні-пакету і може використовуватися з розміром 1. Окрім цього, вона також може використовувати ту саму операцію нормалізації як для навчання, так і для набору даних висновків.
Ось операція статистики рівня нормалізації.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

(де H — кількість прихованих одиниць у шарі)

9.3 1x1 Згортування

На відміну від загальної згортки з розміром ядра більше 1, яка використовується для інтеграції інформації локальної області, згортка 1x1 множить той самий скаляр до однієї карти функцій. Отже математично згортка 1x1 є по суті лінійною комбінацією карт об'єктів.

9.3.1 OverFeat

У багатьох архітектурах згорткових мереж, є один або кілька повністю зв'язаних шарів. Цей підхід підходить для таких завдань, як класифікація зображень. Але в цій роботі метою є виявлення об'єкта, і тому, застосовується розсувне вікно, яке генерує плями зображень із перекриттями. У цьому випадку обробка латок самостійно - це свого роду марнотратство.

З цього аспекту, згортка 1x1 представлена в OverFeat за допомогою Sermanet et al. [2013] як альтернатива повністю зв'язаному шару. В оригінальній роботі це також описується як "просторовий розмір вводу". На відміну від повністю зв'язаного шару, згортування по суті ефективні для цього підходу з розсувними вікнами. Замінюючи повністю зв'язані шари згортковими шарами 1x1, модель OverFeat може обробляти вхідне зображення більших розмірів. Рисунок 38 чудово це ілюструє.

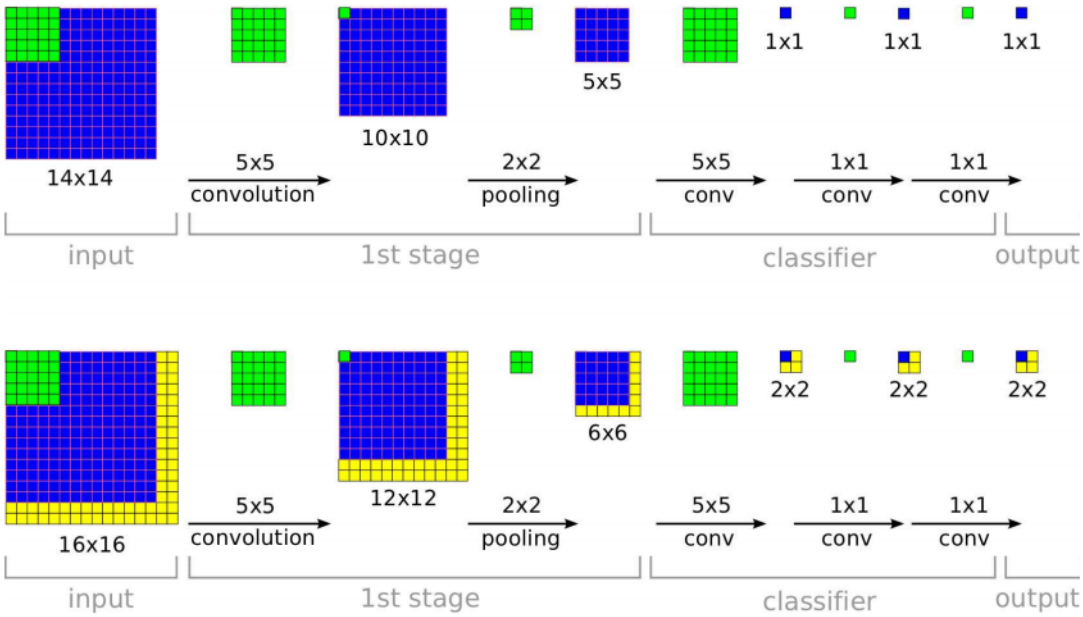
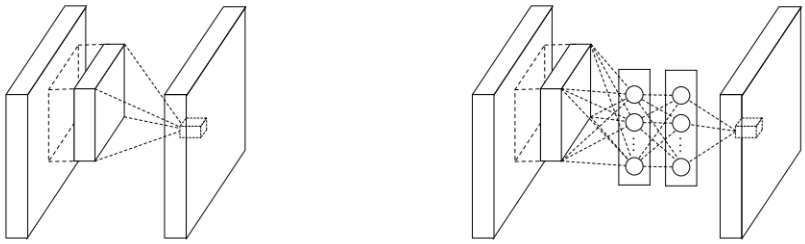


Рис. 38: Ефективність ConvNets для виявлення. (Джерело: Sermanet et al. [2013])

9.3.2 Network In Network

Тим часом, Lin et al. [2013] також представляє собою багатошаровий згорнутий шар персептрона (mlpconv) (Рисунок 39) у своїй моделі Network In Network (NIN), яка може застосувати додаткову згортку 1x1 після загальної згортки. (Є кілька більш складних способів використання шару mlpconv. Хоча, окрім теми згортки 1x1, оригінальний документ все ж варто прочитати.) Додаючи шари всередині себе, шар mlpconv збільшує репрезентативну силу архітектури NIN.

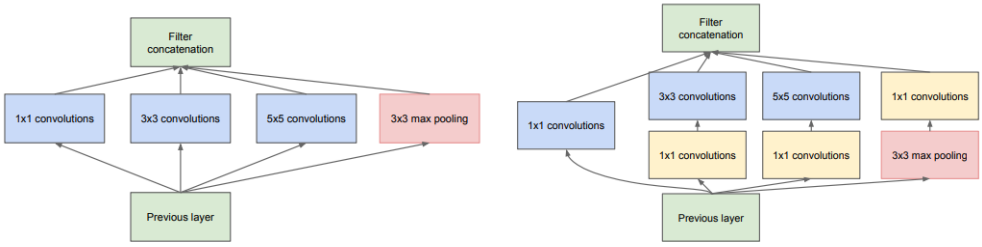


(a) Linear convolution layer (b) Mlpconv layer

Рис. 39: Порівняння шару загальної згортки та шару mlpconv. (Джерело: Lin et al. [2013])

9.3.3 Inception

Згортка 1x1 стає більш популярною, коли Szegedy et al. [2014] презентує Inception (Рисунок 40) в GoogLeNet. У цій роботі згортка 1x1 в основному використовується для зменшення розмірів, що дозволяє мережі рухатись глибше і ширше. Крім того, додаткові згорткові шари з функціями активації ще більше збільшують нелінійність і здатність.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Рис. 40: Початковий модуль. (Джерело: Szegedy et al. [2014])

$s = \frac{1}{\beta} log \sum e^{\beta x}$
Такий вид об'єднання може бути дорогим, тому, можливо, ви не захочете його використовувати.

Хитрощі в розпізнаванні облич: як розпізнати обличчя різного розміру: ви зменшуєте зображення (до квадратного кореня з 2), потім проганяєте зображення через згорткову мережу.

Зробіть це кілька разів, і вартість буде тільки в 2 рази більше первісної вартості. Тому що кожен раз розмір зображення зменшується на 1/2, тому загальна вартість є геометричній сумою.

Негативний аналіз: ви спочатку навчаєте свою модель, а потім запускаєте модель на дуже великому масиві зображень, в якому ви знаєте, що немає облич. Таким чином можна уникнути помилкових спрацьовувань.

Повернення хибнопозитивного і хибнонегативного, вам необхідно встановити поріг для отримання балансу / повернення.

Якщо ви порівнюєте 2 системи, порівняйте, яка крива нижче на графіку FN-FP.

Precision-Recall схожий на FN-FP.

Сіамська архітектура: навчайте розпізнавач облич всього на декількох прикладах.

Любіть метричне навчання.

Ви хочете щось, що маємо ненульовий градієнт в 0.

10 Загальні примітки про архітектуру моделей

10.1 Огляд та руйнування міфів

ConvNets (Згорткова нейронна мережа - ЗНМ) добре працює над проблемами класифікації зображень через припущення про місцевість (близькі сигнали корелюються) і стаціонарність (подібні схеми можна знайти на всьому зображенні). Вона складається з різної кількості різних шарів. Наприклад, ЗНМ може бути Ввід→ Згортання→ Випрямлення→ Згортання→ Випрямлення→ Обєднання→ Зв'язування.

За останні роки мережі поглибленого навчання швидко розвинулися. Тож чому воно зростає зараз? Можливою причиною для цього питання є те, що ми отримуємо все більше і більше даних в наш час. Якщо порівняти продуктивність підходу глибокого навчання з іншими загальними машинними підходи до навчання, ми бачимо, що це невелика область даних, два методи мають однакову ефективність, і цілком можливо, що загальні алгоритми машинного навчання працюють навіть краще. Однак, якщо розглядати великі об'єми даних, ми можемо досягти значно вищої продуктивності за допомогою методів глибокого навчання порівняно із загальними алгоритмами машинного навчання. Крім того, розробка більш прогресивних центральних процесорів і графічних процесорів дозволили людям тренувати більші мережі. Так було на початку підйому поглибленого навчання через масштаб даних та масштаб обчислювальної потужності. Пізніше також з'являються нововведення з боку алгоритмів, які дозволяють людям тренувати нейронні мережі набагато швидше, типовим прикладом є використання шару випрямлення замість сигмовидного, щоб набагато швидше зробити тренування з градієнтним спуском.

1. **Шар згортування**
   У цьому шарі ми використовуємо різні фільтри, щоб визначити місцеві та низькорівневі функції. Крок: сума за допомогою якого фільтр зміщується. Доповнення: додайте нульові відступи до границі вводу, що допоможе зберегти розмір вхідного обсягу.

2. **Шар вирівнювання**
   Впроваджує нелінійність після згортання (лінійність, через елементне множення та підсумовування).

3. **Шар обєднання**
   Як тільки ми дізнаємося, що деяка особливість знаходиться в початковому введеному обсязі, її точне місце розташування не настільки важливо, як її відносне розташування до інших об'єктів. Включення об'єднання в ЗНМ допускає незмінність класу об'єктів до перекладів, що корисно для ідентифікації об'єкта (наприклад: яблуко - це яблуко, незалежно від того, де воно відображається на зображенні). Об'єднання також різко зменшує просторовий розмір вхідного об'єму, а за разом, величину параметрів та обчислювальну вартість.

4. **Шар зв'язування**
   Дізнається глобальну статистику та інформацію про зображення. Він розглядає які особливості високого рівня найбільш сильно співвідносяться з певним класом (у разі класифікації).

5. **Шар скидання**
   Скидає випадковий набір активацій, встановивши їх на нуль. Це перевіряє надійність мережі - вона повинна мати можливість забезпечити правильну класифікацію, або результати для конкретного прикладу, навіть якщо деякі активації скинуті. Це гарантує, що мережа не надто «пристосовується» до тренувальних даних.

Питання: Яка оптимальна кількість шарів чи архітектура та як вибрати гіперпараметри? **Відповідь:** "Немає безкоштовного обіду" - єдиний алгоритм не може добре виконувати всі завдання. Ми повинні вибрати алгоритм та архітектуру на основі наших попередніх знань та припущень про проблему. В ЗНМ, ми вибираємо гіперпараметри (наприклад, кількість шарів, які типи шарів) залежно від вхідних даних та типу завдання.

Питання: Чи дасть ЗНМ подібний результат, якщо ми перестановимо пікселі на вхідних зображеннях? **Відповідь:** Якщо ми застосуємо ЗНМ до різних наборів даних, які не містять припущень про місцевість, ця архітектура не допоможе нам отримати кращий результат. Натомість ми могли б досягти подібної продуктивності, використовуючи повністю зв'язані шари. Експеримент з налаштування функції розупорядкування на пікселях показує це ефект.

10.2 Переваги архітектур глибокого вивчення

Машинне навчання - це, по суті, функціональна апроксимація завдань, які ми хочемо виконувати. В теорії, ми можемо апроксимувати будь-які функції за допомогою нейронної мережі. Перевага глибинних архітектур полягає в тому, що ми можемо апроксимувати деякі функції ефективніше, використовуючи глибинні архітектури, ніж використовуючи поверхневі, і наслідком цього є те, що ми матимемо кращий резульат. Однак сильно контрольовані нейронні мережі, як і раніше важко піддаються навчанню і потребують великої обчислювальної потужності. Щоб подолати це, натхненний структурою зорової системи, і за припущеннями, про які ми говорили раніше,

дослідник придумав Згорткову Нейронну Мережу. Вони мають набагато менше з'єднань, ніж звичайні нейромережі прямої передачі, тому їх набагато легше тренувати, при цьому жертвуючи лише невеликою продуктивністю.

10.3 Пропуск зв'язків

Глибокі згорткові нейронні мережі призвели до низки проривів у завданнях обробки зображення. З приходом графічних процесорів, навчання більш глибинної архітектури ставало дедалі доцільнішим. Дослідники почали вивчати глибшу архітектуру та отримати покращені результати. Однак глибинні мережі спричиняють кілька проблем. Основними проблемами навколо "глибини"є поширення відповідної інформації протягом прямої передачі та проблеми з оптимізацією через зникаючі/вибухові градієнти. Пропуск зв'язків забезпечує певне вирішення цих питань та суттєво вдосконалило різні методи в завданнях поглибленого навчання. Пропущенні зв'язки дозволяють пересилати інформацію на наступний шар. Вони сприяють кращому поширенню градієнта в глибинних мережах.

ResNet (Залишкова мережа) Основною перевагою ResNet є те, що він дозволяє тренувати дуже глибоку згорткову нейронну мережу, не страждаючи від проблеми зникнення градієнта. Без пропускання з'єднань, градієнтні сигнали можуть зникати і не досягати початку мережі під час проходження десятки нелінійних шарів. Пропуск з'єднань, як показано на 41, створює найкоротший шлях проходження для градієнта. Будучи найпростішою можливою функцією (identity), вона легко передає сигнал градієнта на початок мережі.

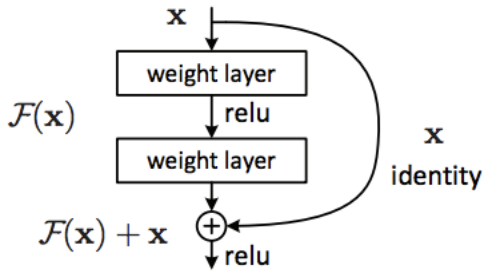


Рис. 41: Будівельний блок ResNet

DenseNet (щільно пов'язана згорткова мережа) Одним із варіантів пропуску зв'язків є DenseNet. Замість підсумовування результату шару з його введенням, їх можна об'єднати вздовж осі ознаки. Це дозволяє нам залишатися на низькому рівні особливостей вздовж шарів і може бути дуже корисним, коли зображення мають дуже високу роздільну здатність і тому потрібні глибші моделі.

Unet Іншим варіантом є Unet, який вводить пропущенні зв'язки в сегментовані мережі. Пропуск зв'язку здійснюється між рівнями мережі кодера / декодера. Ця архітектура зіграли

роль у покращенні результатів сегментації.

10.4 Рекурсивні ConvNets

У більшості ЗНМ вхідний розмір і кількість шарів є гіперпараметрами, які повинні бути визначені або відрегульовані вручну (не вивчені). Рекурсивні ЗНМ (Рис. 42) натомість приймають вхід змінного розміру. Це дерево згортання зі спільною вагою, яке зменшує введення рекурсивно, поки вихід не матиме розміру 1. Отже, кількість шарів також є змінною. Максимальна кількість шарів буде log(довжина введення).

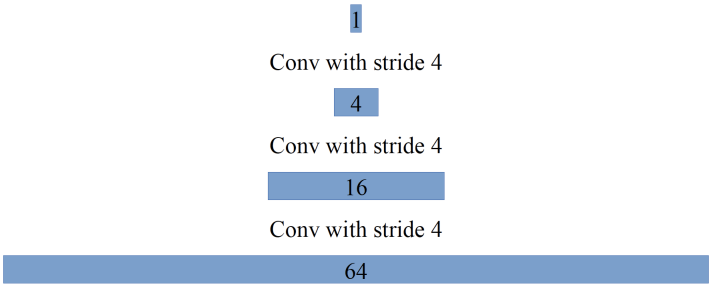


Рис. 42: Рекурсивне застосування згортання для зменшення вхідних даних до вихідного розміру 1
(Посилання: Deep Learning 2018 by Y LeCun at NYU)

Рекурсивні ConvNets додатки Вони добре працюють з послідовністю та деревовидними структурами в NLP(Natural Language Processing) завданнях. Вони може бути використані при семантичному моделюванні текстових послідовностей різної довжини до вектора з фіксованою довжиною. Потім вектор з фіксованою довжиною можна використовувати в інших завданнях, таких як класифікація тексту.

10.5 LeNet

10.5.1 Вступ

LeNet-5 - це новаторська 7-шарова (не включаючи вхідний рівень) згорткова нейронних мережа, розроблена Янном ЛеКун та співавторами у 1998 р. "Y. LeCun, L. Bottou, Y. Bengio та P. Haffner: Gradient-Based Learning Applied to Document Recognition, Збірник IEEE,86(11):2278-2324, Листопад 1998".
Це була одна з перших згорткових нейронних мереж, яка допомогла просунути поле глибинного навчання. Вона використовувалася в основному для розпізнавання символів. Зверніть увагу, що існують інші варіанти LeNet, такі як: LeNet1, LeNet2, LeNet3, LeNet4, Підсилений LeNet4. Це інші варіанти LeNet, кожен з деякими відмінностями в дизайні архітектури.

10.5.2 Архітектура

LeNet-5 містить 1 вхідний шар, 3 згорткові шари, 2 шари субдискретизації, 1 обєднувальний шар і 1 вихідний шар.

1. Вхідний рівень: зображення 32x32 пікселів
2. Згортковий шар 1: 6 карт характеристик розміром 28x28
3. Субдискретизаційний шар 1: 6 карт характеристик розміром 14x14
4. Згортковий шар 2: 16 карт характеристик з розмірами 10x10
5. Субдискретизаційний шар2: 16 карт характеристик з розмірами 5x5
6. Згортковий шар 3: 120 функціональних карт розміром 1x1
7. Обєднувальний шар: 84 повністю підключені блоки
8. Вихідний рівень.

10.6 AlexNet

10.6.1 Вступ

Alexnet був використаний для перемоги у 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge). Це була перша поява моделі нейронних мереж, яка так добре працювала на наборі даних ImageNet. Прийоми використані в моделі використовуються і донині, такі як збільшення даних та відсутність даних. Документ "Класифікація ImageNet з глибокими ЗНМ"показав переваги ЗНМ і підтримав їх рекордними показниками на змаганнях у 2012 році.

10.6.2 Архітектура

Alexnet містить 5 згорткових шарів і 3 обєднувальні.

1. Перше згортання створює шар глибиною 64, використовуючи функцію ядра 11×11, за яким слідує ReLU та MaxPooling з розміром ядра 3. 2. Друге згортання створює шар глибиною 192 із використанням функції ядра 5×5, за яким слідує ReLU та MaxPooling з розміром ядра 3. 3. Третє згортання створює шар глибиною 384 із використанням функції 3×3 ядра, за яким слідує ReLU та MaxPooling з розміром ядра 3. 4. Четверте згортання створює шар глибиною 256 із використанням функції 3×3 ядра, за яким слідує ReLU та MaxPooling з розміром ядра 3. 5. П'яте згортання створює шар глибиною 256, використовуючи функцію ядра 3×3, за яким слідує ReLU та MaxPooling з розміром ядра 3. 6. Перший обєднувальний шар відображає вектор розміром 9216 у вектор розміром 4096. 7. Другий обєднувальний відображає вектор розміром 4096 у вектор розміром 4096. 8. Третій обєднувальний шар відображає вектор розміром 4096 у вектор розміром 1000.

10.6.3 Хитрощі, що застосовуються для запобігання переповненню

Модель мала 60 мільйонів параметрів для вивчення. Були використані два методи, використані нижче (в основному), щоб запобігти переповненню:

1. Збільшення даних

   Найпростіший і найпоширеніший спосіб запобігти переобладнанню - це штучне збільшення набору даних. Перетворене зображення створюється з вихідного зображення з невеликими обчисленнями, так що його не потрібно зберігати на диску.

   Перший спосіб полягає у випадковому вирізанні нового зображення розміром 224×224 із зображення розміром 256×256 та випадковим горизонтальним поворотом зображення. Таким чином, ми збільшуємо набір даних у 2048 разів.

   Другий спосіб - змінити інтенсивність RGB-каналів у навчальних наборах. Так що кожен RGB піксель

   $$I_{xy} = [I^R_{xy}, I^G_{xy}, I_3 xy^B]^T$$

   буде додано

   $$[p_1, p_2, p_3][a_1\lambda_1, a_2\lambda_2, a_2\lambda_2]^T$$

   де p_i та λ_i це i^{th} - власний вектор і власне значення матриці коваріації пікселя RGB, та a_i є випадковою величиною, отриманою з Гаусса із середнім нулем та стандартним відхиленням 0,1.

2. Виключення

   Модель використовувала коефіцієнт виключення рівний 0,5.

10.6.4 Деталі тренування

Модель тренується за допомогою SGD (стохастичний градієнтний спуск) з розміром групи 128, імпульсом 0,9 та розпадом ваги 0,0005. Вага ініціюється з розподілу Гауса із середнім нулем та стандартним відхиленням 0,1. Модель навчається близько 90 циклів завдяки навчальному набору з 1,2 мільйона зображень, які займають п'ять-шість днів на двох 3 ГБ графічних процесорах NVIDIA GTX 580.

10.6.5 Результат

Результати ILSVRC-2010 дають рейтинги помилок топ-1 та топ-5 на рівні 37,5 відсотка та 17 відсотків. Результати ILSVRC-2012 шляхом усереднення ЗНМ дають 15,3 відсотка в топ-5.

10.6.6 Порівняння AlexNet з іншими моделями

Архітектура

AlexNet використовує 5 згорткових шарів та 3 повністю з'єднаних шари. Архітектура виглядає наступним чином:

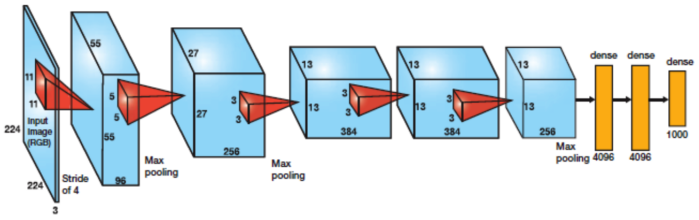


Рис. 43: Архітектура AlexNet

Функція активації

Порівняно з насичуючими нелінійностями, які використовують гіперболічну дотичну функцію або сигмоподібну функцію, ненасичуюча нелінійність, така як ReLU, є дуже ефективно обчислювальною і порівняно швидкою у навчанні.

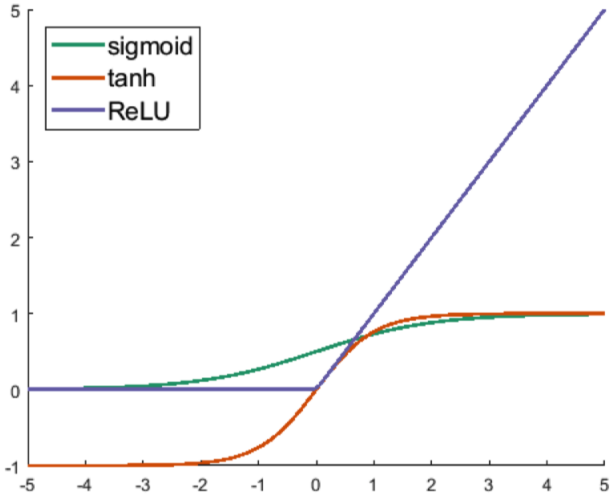


Рис. 44: Функція активації

Навчання на декількох GPU

Половина нейронів певного шару знаходиться на кожному графічному процесорі, і графічні процесори спілкуються лише на певних шарах. Такий прийом покращує продуктивність.

Локальна нормалізація відповідей

Локальна нормалізація відповіді допомагає в узагальненні, навіть якщо ReLU не потребують нормалізації, щоб запобігти їх насиченню. Нормалізації виконуються на перших двох шарах згортки, за якими слідує об'єднання максимальних.

Об'єднання

Для того, щоб зменшити дисперсію та складність обчислень та витягти низькорозмірні об'єкти із сусідства, ми використовуємо об'єднання. AlexNet використовує об'єднання максимальних та перекривне об'єднання. Перекривне об'єднання, може дещо зменшити рівень помилок порівняно з неперекриваючим, а також ускладнити переобладнання моделі.

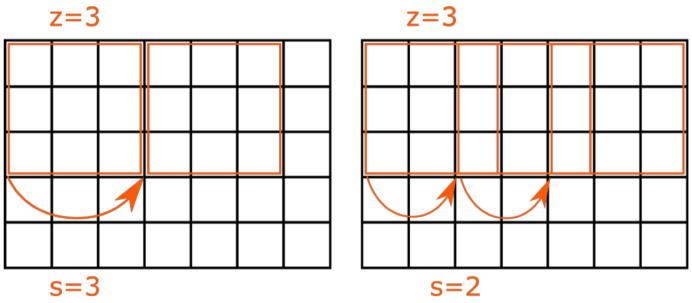


Рис. 45: Неперекривне об'єднання порівняно з перекривним

Об'єднання максимальних виділяє найвидатніші функції, такі як ребра, тоді як середнє об'єднання іноді не може виділити хороші функції, оскільки воно враховує все.

Зменшення переобладнання

Тут є дві методики для запобігання переобладнанню; перша - це збільшення даних, а друга - випадання, що означає, що певний набір нейронів, вибраних навмання, не враховується під час певного проходження вперед або назад. Випадання - це підхід до регуляризації, який допомагає зменшити взаємозалежне навчання між нейронами.

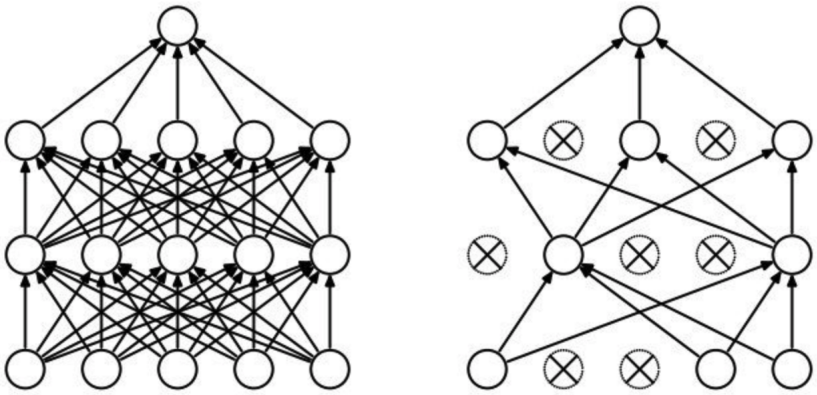


Рис. 46: Нейронна мережа без випадання (a) та з випаданням (b)

Висновки

Comparison					
Network	**Year**	**Salient Feature**	**top5 accuracy**	**Parameters**	**FLOP**
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Рис. 47: Порівння AlexNet з іншими нейронними мережами

AlexNet та ResNet-152, обидва мають приблизно 60 мільйонів параметрів, але в їхній точності з найкращих результатів існує приблизно 10% різниці. Але для навчання ResNet-152 потрібно багато обчислень (приблизно в 10 разів більше, ніж у AlexNet), що означає більше часу на навчання. В порівнянні з Inception, навчання AlexNet займає приблизно той самий час, але в Inception вимоги до пам'яті в 10 разів менші і точність краща приблизно на 9%.

10.7 VGGNet

10.7.1 Представлення

VGGNet став переможцем у ILSVRC 2014 (широкомасштабне завдання візуального розпізнавання ImageNet), розробленому Карен Сімонян та Ендрю Ціссерман [Simonyan and Zisserman, 2014]. Сімонян та інші ретельно оцінили мережі зростаючої глибини, використовуючи архітектуру з дуже малими(3×3) фільтрами згортки, що показує, що суттєве покращення конфігурацій рівня техніки може бути досягнуто шляхом просування глибини до 16-19 вагових шарів [Simonyan and Zisserman, 2014]. Ці архітектури не тільки показали найсучаснішу точність в завданнях класифікації та локалізації під час ILSVRC 2014, але і при застосуванні до інших наборів даних розпізнавання зображень.

10.7.2 Архітектура

Вхідні дані для ConvNets це зображення фіксованого розміру 224×224 RGB які передаються через стопку згорткових шарів, де використовуються фільтри з дуже малим полем сприйняття: 3×3 (що є найменшим розміром для охоплення поняття ліворуч/праворуч, вгору/вниз, по центру)). Швидкість згортки зафіксована на рівні 1 пікселя; просторове заповнення вхідного згорткового шару є таким, що просторова роздільна здатність зберігається після згортки. Просторове об'єднання здійснюється за допомогою п'яти шарів об'єднання максимумів, які слідують за деякими згортковими шарами (не всі згорткові шари супроводжуються об'єднанням максимумів). Об'єднання максимумів здійснюється у вікні 2×2 пікселі з кроком 2.
За стопкою згорткових шарів слідують три повністю з'єднаних шари: перші два мають по 4096 каналів, третій містить 1000 каналів (по одному для кожного класу). Кінцевим шаром є шар soft-max. Конфігурація повністю зв'язаних шарів однакова у всіх мережах. Усі приховані шари оснащені нелінійним випрямлячем (ReLU). Жодна з мереж (за винятком однієї) не містить локальної нормалізації відповіді (LRN), що не покращує продуктивність набору даних ILSVRC, але призводить до збільшення споживання пам'яті та витрати часу. Архітектура VGG-16(13 згорткових шарів та 3 повністю з'єднаних) показана на рисунку 48.

10.7.3 Налаштування

Усі налаштування відповідають загальному дизайну, представленому в розділі архітектура, і відрізняються лише глибиною, від 11 шарів (8 згорткових шарів, 3 повністю з'єднаних шари) до 19 шарів (16 згорткових шарів, 3 повністю з'єднаних шари). Ширина згорткових шарів (кількість каналів) досить мала, починаючи з 64 у першому шарі, а потім збільшуючись у 2 рази після кожного шару об'єднання максимумів, поки не досягне 512. Конфігураційний план та кількість параметрів для кожної конфігруації показані на рисунку 49.

10.7.4 Обговорення

Ми знаємо, що стопка з трьох згорткових шарів розмірністю 3×3, з кроком 1 рівнозначна 7×7 сприйнятливим шарам. Навіщо використовувати дуже маленькі рецептивні поля з розмірністю 3×3 по всій мережі, а не 11×11 (використовуються в переможці ILSVRC-2012) або 7×7 (використовуються в переможці ILSVRC-2013)? Порівняємо стопку з трьох згорткових шарів

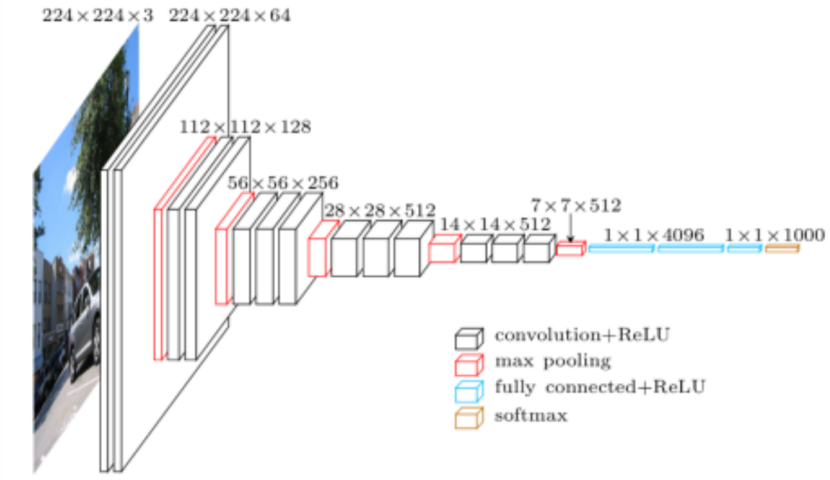


Рис. 48: Архітектура VGG-16 ConvNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Рис. 49: Налаштування VGG

розмірністю 3×3 з одним згортковим шаром розмірністю 7×7:
1. Включення трьох шарів з нелінійних випрямлячів замість одного робить функцію прийняття рішення більш диференціальною.
2. Зменшення кількості параметрів: припускаючи що як вхід так і вихід трьох згорткових шарів розмірністю 3×3 має C каналів, в стопці буде $3(3^2C^2) = 27C^2$ параметрів; в той же час, один згортковий шар розмірністю 7×7 вимагає $7^2C^2 = 49C^2$ параметрів, ща на 81% більше.

10.7.5 Деталі навчання

1. Налаштування
Модель навчається шляхом оптимізації мультиноміальної логістичної регресії з використанням міні-пакетного градієнтного спуску з імпульсом [Krizhevsky et al., 2012]. Розмір партії встановлено на 256, імпульс на 0,9, множник штрафу L_2 на $5 * 10^{-4}$, коефіцієнт відсіву для перших двох повністю зв'язаних шарів до 0,5. Швидкість навчання була ініціалізована до 10^{-2}, а потім зменшилась у 10 разів, коли точність набору перевірки перестала покращуватися. Зрештою, навчання було припинено після 370 тисяч ітерацій (74 епохи). Хоча VGGnets мають глибші шари та більше параметрів для вивчення порівняно з [Krizhevsky et al., 2012], сітки сходилися швидше завдяки неявній регуляризації, накладеної більшою глибиною та меншими розмірами фільтра згортки та попередньою ініціалізацією певних шарів.

2. Ініціалізація
Почнемо з навчання конфігурації A (11 вагових шарів) із випадковою ініціалізацією, потім при навчанні більш глибоких архітектур перші чотири згорткові шари та останні три повністю зв'язані шари ініціалізуються шарами мережі A, а проміжні шари ініціалізуються випадковим чином. Швидкість навчання для попередньо ініціалізованих шарів дозволялося змінювати під час навчання, а ваги відбирали із звичайного розподілу із середнім значенням нуля та дисперсією 10^{-2} для випадково ініціалізованих шарів. Похибки ініціалізувались нулем.

3. Розмір зображення для навчання
Два підходи до встановлення навчальної шкали S, яка є найменшою стороною ізотропно масштабованого навчального зображення:

3.1. Виправимо, що S має значення 256 або 384. Враховуючи конфігурацію ConvNet, спочатку навчіть мережу, використовуючи S=256. Для прискорення навчання мережі S=384 вона була ініціалізована вагами, попередньо навченими S=256, а потім використана менша початкова ставка навчання в 10^{-3}.

3.2. Випадково відбирають S з певного діапазону $[S_{min}, S_{max}]$, де $S_{min} = 256$ і $S_{max} = 512$. Це дозволяє краще тренувати предмети на зображеннях з різним розміром. З міркувань швидкості багатомасштабні моделі навчаються шляхом тонкої настройки всіх шарів одномасштабної моделі з однаковою конфігурацією, попередньо навчених із фіксованим S=384.

10.7.6 Деталі реалізації

Навчання та оцінка виконуються на декількох графічних процесорах, встановлених в одній системі. Навчання з декількома графічними процесорами використовує паралельність даних і здійснюється шляхом розділення кожної партії навчальних зображень на кілька пакетів графічних процесорів, які паралельно обробляються на кожному графічному процесорі. Час

навчання в готовій системі 4-GPU в 3,75 рази швидше, ніж при використанні одного графічного процесора.

10.7.7 Оцінювання

1. Одномасштабне оцінювання
Розмір тестового зображення встановлюємо наступним чином: $Q = S$ для фіксованого S та $Q = 0.5(S_{min} + S_{max})$ для змінних S $\in [S_{min}, S_{max}]$. Результати наведені нижче в Таблиця3.
По-перше, використання локальної нормалізації відповіді (мережа A-LRN) не покращує показники верхнього та верхнього 5. помилка на моделі A, тому нормалізація не застосовується на більш глибоких архітектурах.
По-друге, конфігурація C (яка містить три рівні конверсії 1×1) працює гірше, ніж конфігурація D, яка використовує 3×3 конв. рівні по всій мережі. Це вказує на те, що важливо фіксувати просторовий контекст, подаючи позов на конв. фільтри з нетривіальними рецептивними полями. Частота помилок не падала, коли було використано 19 шарів. Крім того, більш глибока сітка з маленькими фільтрами перевершує дрібну сітку з більшими фільтрами. Це пов'язано з тим, що перша помилка неглибокої мережі з п'ятьма 5×5 конв. рівнів був на 7% вищим, ніж у B (який використовує 3×3 конв. рівнів).
Нарешті, зміна масштабу на тренувальному наборі допомагає отримувати статистику багатомасштабних зображень, оскільки зміна масштабу під час тренувань досягає значно кращих результатів, ніж тренування на зображеннях із фіксованою найменшою стороною.

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

2. Багатомасштабне оцінювання
Найкраща продуктивність однієї мережі на наборі перевірки - помилка 24.8% top-1 /7.5% top-5. Це показує, що зміна масштабу під час тесту призводить до кращих показників.

10.8 Нейронна мережа глибокого залишкового навчання(ResNet)

10.8.1 Вступ

Глибокі згорткові (Convolutional) нейронні мережі(ConvNets or CNNs) [LeCun et al., 1989] були застосовані для аналізу візуальних образів та виявилися успішною для розпізнавання та класифікації зображень. CNNs можуть захоплювати характеристки високого рівня та передавати їх до класифікаторів. Попередні дослідження [Simonyan and Zisserman, 2014] та Szegedy et al. [2015] показують що якість характеристик(особливостей) має сильну залежність від глибини нейронної мережі. Більш того, Eldan and Shamir [2016] математично доводить перевагу глибоких мереж над широкими мережами. У використанні глибоких нейроних мереж, зникаючі/вибухаючі градієнти стали основною завадою, яку буде детально описано у підрозділі 1.2 Деградація. Нейронна мережа глибокого залишкового навчання(ResNet) опублікована у 2015 дослідниками з Microsoft Research [He et al., 2016a], що займалися процесом деградації. Як результат, ResNet досягає найкращих результатів на складному ImageNet наборі даних [Russakovsky et al., 2015] з безпрецендентною глибиною.

10.8.2 Деградація

Одна головна перевага Нейронних мереж глибинного навчання це їх можливість навчатися більш абстрактним особливостям вхідних даних із зростанням глибини [Zeiler and Fergus, 2013, Goodfellow et al., 2016]. This hierarchical nature робить Нейронні мережі глибинного навчання ідеальними для розпізнавання зображень, оскільки це дозволяє Нейроній мережі захоплювати прості особливості як вершини та ребра на ранніх шарах та об'єднати їх на глибшихі шарах для абстрактного представлення. Отже, збільшення глибини мережі повинно дозволяти мережі вивчати більш абстрактні уявлення [Bengio et al., 2013]. Проте, було помічено, що продуктивність мережі падає після певної глибини не лише коли мережа застосовується на тестових данних а також для даних авчання. Іншими словами, обидві тестова та точність тренування падає якщо у мережу додається більше шарів, це називається ефектом деградації в глибоких мережах. networks[He et al., 2016a, Srivastava et al., 2015, 201, 2018]. Графік на Figure 50 є типовим прикладом [He et al., 2016a].

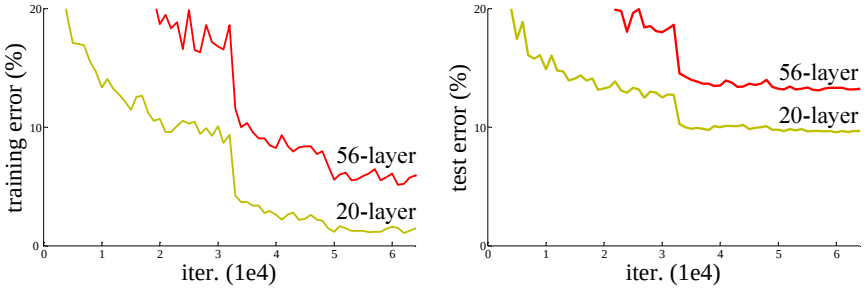


Рис. 50: Помилки тренування (ліво) і помилки тесту (право) на CIFAR-10 з 20-шарами та з 56-шарів "прості" мережі. Глибша мережа має вищі показники помилок для тренування, та слідством помилку тестування.

Деградація не є інтуітивно зрозумілою так як із додаванням шарів, очікується що мереж= має можливість(вміння) інтегрувати особливості високого рівня для тренувальних даних, та результатом надавати кращі тренувальні результати. Крім того, це також природньо думати, що якщо додати шари відображення ідентичності(identity mapping), глибша мережа має підходити для тренувальних щонайменше так само як до цього. Одне з можливих пояснень, ми можемо бачити у assignment 1, може бути зникаючі/вибухаючі градієнти [Bengio et al., 1994, Glorot and Bengio, 2010] які є перешкодами на шляху до зближення, особливо коли мережі дуже глибокі. Тим не менш, ці проблеми були вирішені застосуванням нормалізованої ініціалізації [LeCun et al., 1998, Glorot and Bengio, 2010, Saxe et al., 2013, He et al., 2015] та проміжними шарами нормалізації(пакетна нормалізація) [Ioffe and Szegedy, 2015b].

Для спостереженням явища деградації, уявимо, що ми маємо дрібніше мережу з k шарів та ми додаємо m шарів відображення ідентичності (identity mapping layers) до цієї мережі та порівнюємо помилки тренування обох мереж. Такий експеремент передбачає, що такі мережі важко навчати відображеню перехресно на кілька нелінійних шарів, тобто не всі системи однаково легко оптимізувати. Переважно, складніше підходити до відображення ідентичності (identity mapping) $F(x) = x$ ніж нульове відображення $F(x) = 0$ группи нелінійних шарів. Для вирішення проблеми деградації, *deep residual learning framework* представлено [He et al., 2016a].

10.8.3 Глибоке залишкове навчання

Залишкове навчання

Враховуючи $\Omega(x)$ як базове зіставлення для відповідності, яке можна розмістити кількома накладеними шарами (не обов'язково усю мережу), з $\boldsymbol{x}$ позначають входи для першого з цих шарів[He et al., 2016a]. Введемо залишкову функцію: $F(x) := \Omega(\boldsymbol{x}) - \boldsymbol{x}$ (У оригінальному тексті автора, $F(\boldsymbol{x})$ це функція активації Relu, але використання Relu не обов'язково; ми думаємо інші диференційні активації мають працювати теж). Оригінальна функції mapping може бути представлена, як $\Omega(\boldsymbol{x}) := F(\boldsymbol{x}) + \boldsymbol{x}$. Порівняно з навчанням $\Omega(\boldsymbol{x})$ у традиційних нейронних мережах, переформулювання вводить ідентифікаційні зв'язки між шарами.

Identify Mapping by Shortcuts

Автор He et al. [2016a] застосував залишок до кожного будівного блока який складається з декількох накладених шарів. Будівні блоки визначено як:

$$\boldsymbol{y} = F(\boldsymbol{x}, \{\boldsymbol{W}_i\}) + \boldsymbol{x} \tag{57}$$

де $\boldsymbol{x}$ і $\boldsymbol{y}$ є входом та виходом векторів будівних блоків. Функція $F(\boldsymbol{x}, \{\boldsymbol{W}_i\})$ представляє залишкове відображення, яке необхідно навчити. З'єднання між функцією $F(\boldsymbol{x}, \{\boldsymbol{W}_i\})$ та вводом $\boldsymbol{x}$ має назву shortcut connection. Приклад на Fig.1 [He et al., 2016a] представляє будівельні блоки з двома шарами навчання, $F = \boldsymbol{W}_2 F(\boldsymbol{W}_1 \boldsymbol{x})$.

Згідно з визначеннямм будівельних блоків у Eqn.(1), розмірність $\boldsymbol{x}$ та F має бути рівними. Якщо нам необхідно узгодити ввід/вивід вимірів, ми представляємо інший лінійну проекцію $\boldsymbol{W}_s$ завдяки найкоротшим з'єднанням (which we have done in lab 4):

$$\boldsymbol{y} = F(\boldsymbol{x}, \{\boldsymbol{W}_i\}) + \boldsymbol{W}_s \boldsymbol{x} \tag{58}$$

Дослідники у He et al. [2016a] показали, що не обов'язково використовувати квадратну матрицю $\boldsymbol{W}_s$ у Eqn.(1). Якщо ми обираємо багато шарів із згорткою, як у функції $F(\boldsymbol{x}, \{\boldsymbol{W}_i\})$, поелементне додавання буде виконуватися на двох картах, канал за каналом. Такі зміни є ефективним розрахунком та вільним від параметрів.

Переважно, блоки будівництва з об'єднаними шарами може бути названа "Residual Unit"та бути представлена у формі [He et al., 2016b]:

$$\boldsymbol{y}_l = h(\boldsymbol{x}_l) + F(\boldsymbol{x}_l, \boldsymbol{W}_l), \boldsymbol{x}_{l+1} = f(\boldsymbol{y}_l) \tag{59}$$

де $h(\boldsymbol{x}_l) = \boldsymbol{x}_l$ це ідентичний мапінг (the identical mapping) та f це ReLU функція.

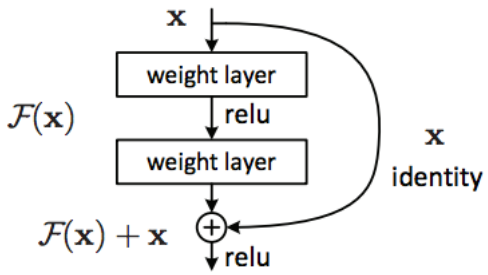


Рис. 51: Залишкове навчання: блок будівництва

Мережеві архітектури

Автор He et al. [2016a] використовує VGG-19модель, просту мережу та одну ResNet в якості прикладів для порівняння та обговорення. Архітектури прикладів мереж показано на Fig.2 [He et al., 2016a].

Залишкова мережа: Найкротші шляхи(shortcuts) з'єднань додані для конвертації простої мережі у ResNet, показано на Fig.2. Шляхи що позначені лініями означають ідентичне відображення. Пунктирні шляхи представляють дві потенційні опції коли збільшується розмірність. (A) Найкоротший шлях(shortcut) виконує зіставлення ідентифікаторів з нульовим заповненням для збільшення розмірів. (B) Найкоротші шляхи(shortcuts) забезпечують відповідність розмірам згортки 1x1.

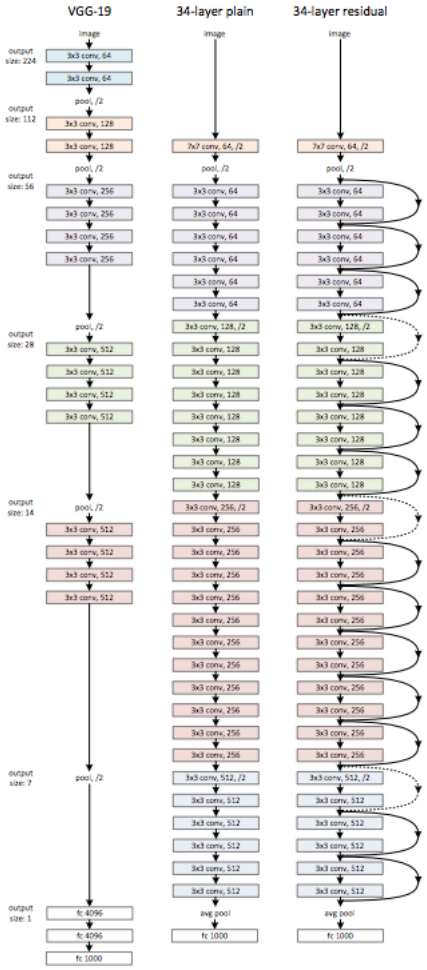


Рис. 52: Приклад архітектур для мереж ImageNet. **Ліворуч:** Модель VGG-19 [Simonyan and Zisserman, 2014] (19.6 billion FLOPs) як силка. **Центр:** проста мережа з 34 шарами (3.6 billion FLOPs). **Праворуч:** залишкова мережа з 34 шарів із параметрами (3.6 billion FLOPs). Пунктирні шляхи(shortcut connection) збільшують розмірність.

10.8.4 Переваги

ResNet найбільш використовувана у завданнях класифікації зображень. Експеременти у [He et al., 2016a], показують що ResNet має три головні переваги.

1. 34-шарова ResNet виконується краще за 18-шарову ResNet; обидві показник тренувальної та валідаційної помилки зменшуються, що доводить у ResNet деградація для точності тренування відсутня. Тож, додавання шарів може підвищити точність розпізнавання зображень.

2. З такою самою кількістю параметрів, ResNet зменшує помилку навчання у порівнянні з 34-гарами простої нейронної мережа (AlexNet, VGG .etc).

3. Для 18-шарової моделі з точністю ResNet сходиться швидше ніж інші прості нейронні мережі. Таке перевага має бути особливо корисним для зображень з низькорівневими характеристиками (такі як лінії, край та контури для роспізнавання зображень).

10.8.5 Додаткові матеріали

18-рівнева мережу - це просто підпростір в 34-рівневої мережі, і вона все ще працює краще. ResNet виграє зі значним відривом, якщо мережа глибше:

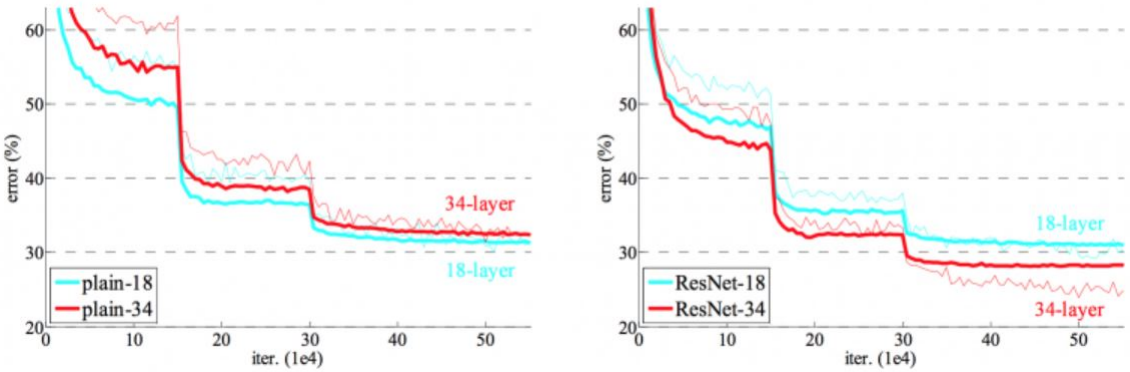


Рис. 53: Навчання в ImageNet. Тонкі криві позначають помилку навчання, а жирні криві позначають помилку валідації. **Ліворуч:** прості мережі з 18 і 34 шарів. **Праворуч:** ResNets з 18 і 34 шарів. На цьому графіку залишкові мережі не мають додаткових параметрів у порівнянні з їх звичайними аналогами.

10.8.6 Результати досліджень нейронної мережі із глибоким залишковим навчанням

Пропуск з'єднань і активація після додаваня

Хоча оригінальний ResNet, що реалізує He et al. [2016a] апроксимує індентичні зв'язки та чомусь вирішує проблему деградації, шари все ще використовують нелінійну активацію, як ReLU, після ідентичних з'єднань. Ці нелінійності перешкоджають розповсюдженню градієнта. [He et al., 2016b] запропонував прямі і зворотні сигнали, які можуть передаватися безпосередньо з одного блоку до будь-якого іншого блоку, коли використовується зіставлення ідентифікаторів, такі як пропуск з'єднань та активація після додавання, див. на Рисунок 3(b)[He et al., 2016b]. Реконструйована "Залишковий Юніт"("Residual Unit") може бути представлена у вигляді:

$$\boldsymbol{x}_{l+1} = \boldsymbol{x}_l + F(\boldsymbol{x}_l, \boldsymbol{W}_l) \tag{60}$$

Рекурсивно, $(\boldsymbol{x}_{l+2} = \boldsymbol{x}_{l+1} + F(\boldsymbol{x}_{l+1}, \boldsymbol{W}_{l+1}) = \boldsymbol{x}_l + F(\boldsymbol{x}_l, \boldsymbol{W}_l) + F(\boldsymbol{x}_{l+1}, \boldsymbol{W}_{l+1}), etc.)$ ми маємо

$$\boldsymbol{x}_L = \boldsymbol{x}_l + \Sigma_{i=l}^{L-1}(\boldsymbol{x}_i, \boldsymbol{W}_i) \tag{61}$$

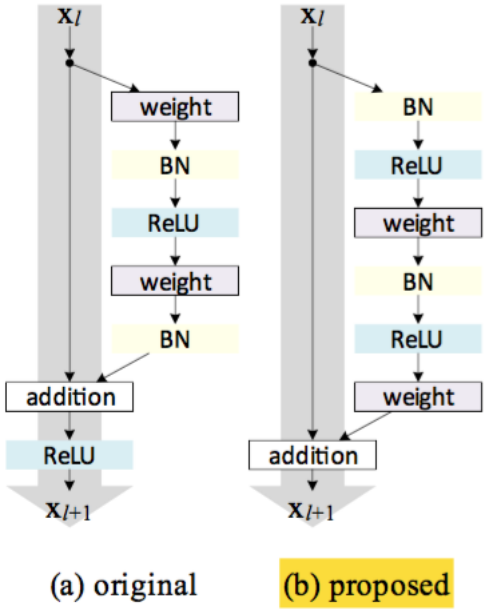


Рис. 54: (a) оригінальний Залишковий Юніт; (b) реконструйований Залишковий Юніт

Resnet в Resnet

[Targ et al., 2016] запропонував Resnet в Resnet (RiR), який встановлює традиційний згортковий шар поруч із кожним шаром ResNet. Ця архітектура дозволяє паралельним шарам обмінюватися інформацією між собою. RiR схожий на LSTM [Hochreiter and Schmidhuber, 1997], який може навчитися довгостроковим структурам ResNet.

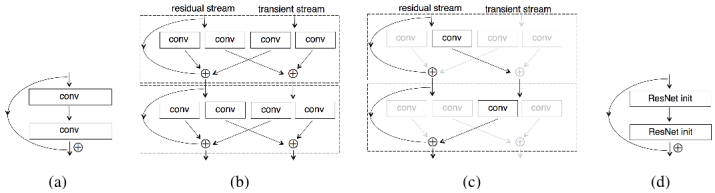


Рис. 55: (a) 2-шаровий блок ResNet. (b) 2 узагальнені залишкові блоки (ResNet Init). (c) 2-шаровий блок ResNet з 2 узагальнених залишкових блоків (сірі з'єднання - 0). (d) 2-шаровий блок RiR.[Targ et al., 2016]

ResNet з експоненціально лінійним юнітом

[Shah et al., 2016] комбінує експоненціальні лінійні юніти з ResNets, щоб показати покращену продуктивність навіть без нормалізації групи.

10.8.7 Виклики

Неможливе вивчення параметрів

Подібно до інших простих нейронних мереж, кожен навчальний рівень має свою вагу. У першому домашньому завданні ми побачили повністю зв'язану лінійну модель, яка має зникаючі або вибухаючі градієнти, що може спричинити непрактичне вивчення ваг. Це також може з'явитися в ResNet.

В одному з ResNet, який ми тренували минулого семестру, ми спостерігали або вибухаючі градієнти, які не дали результатів, або дуже мінливі градієнти, що призводять до великої похибки в навчанні, і це дуже часто зустрічається і в роботі інших дослідників. Ми розуміємо, що порівняно з іншими звичайними мережами додавання залишкових блоків у навчанні ResNet дійсно може зменшити помилку в навчанні, але це також може призвести до додаткових параметрів, які можуть значно збільшити час навчання. Більше того, кожен доданий шар повинен зберігати вивчену інформацію, що призводить до додаткових витрат пам'яті.

Генерація надлишкових особливостей

Кожен навчальний рівень ResNet повинен витягувати різну інформацію з попередніх шарів, тому наступні шари можуть мати подібні особливості, до попередніх. Таке формування надлишкових особливостей зазвичай поширене в класифікаціях зображень, особливо коли вхідне зображення має лише функції низького рівня. Нещодавно було розроблено кілька методів, таких як відсівання, Групова Нормалізація(Batch Normalization), які можуть полегшити цю проблему, але занадто велика регуляризація під час навчальних процесів може також спричинити втрату інформації.

10.9 Опитування про найкращі мережі на ImageNet

Цей підрозділ підсумовує Аналіз Глибоких Нейромережевих Моделей Для Практичного Застосування Canziani et al. [2016], який в основному описує результати опитування про найкращі мережі на ImageNet.

Мотивацією Canziani et al. [2016] є оцінка продуктивності глибоких нейронних мереж на ImageNet, інші показники, крім точності, такі як розмір пам'яті, параметр, операції, також мають вирішальне значення в практичних розгортаннях та додатках.

Причини полягають у тому, що якщо кінцевою метою моделей є лише досягнення максимально можливої точності прогнозування в багатокласному класіфікаційному завданні, виникнуть інші занепокоєння.

Перш за все, поширена практика в Глибоких нейронних мережах, яка називається усередненням моделей або ансамблем, збільшить обчислювальну суму, необхідну для часу навчання. По-друге, різні методи вибірки матимуть різну точність. По-третє, важливість часу виводу не враховується належним чином. Для вирішення згаданих вище питань, Canziani et al. [2016] порівняли передові архітектури глибоких нейронних мереж, які були подані до ImageNet challenge протягом останніх чотирьох років з точки зору обчислювальних вимог, а також точності. У статті повідомлялося про їх результати та порівняння.

1. **Точність**

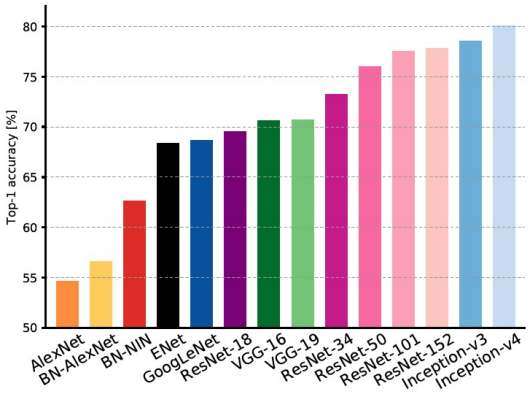


Figure 1: **Top1 *vs.* network.** Single-crop top-1 vali-

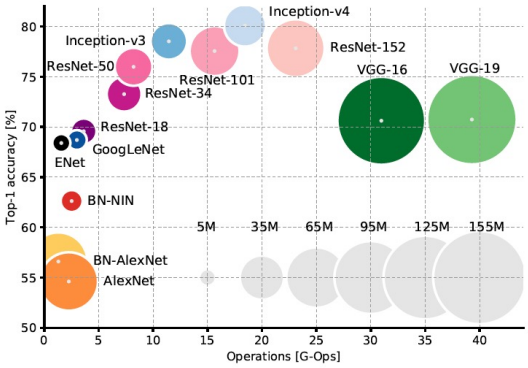


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.**

Як показано на Figure 1 вище, ResNet та Inception Architecture перевершують інші майже на 7% у точності.

Figure 2 не тільки візуалізує точність, але також обчислювальні витрати та кількість параметрів мережі. Цифра показала, що VGG - найдорожча архітектура.

2. **Час Виводу & Енергія**

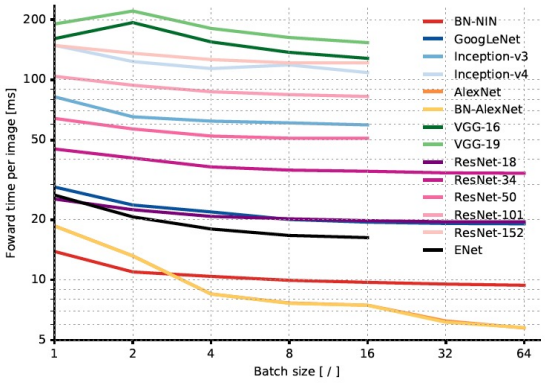


Figure 3: **Inference time *vs.* batch size.** This

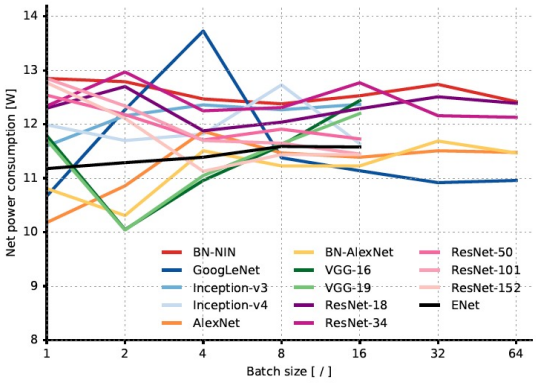


Figure 4: **Power *vs.* batch size.** Net power consump-

Figure 3 показав, що VGG має лідируючий час за зображення, а AlexNet прискорюється втричі від розміру групи від 1 до 64. Тим часом з Figure 4, споживання енергії не залежить від розміру групи.

3. **Пам'ять**

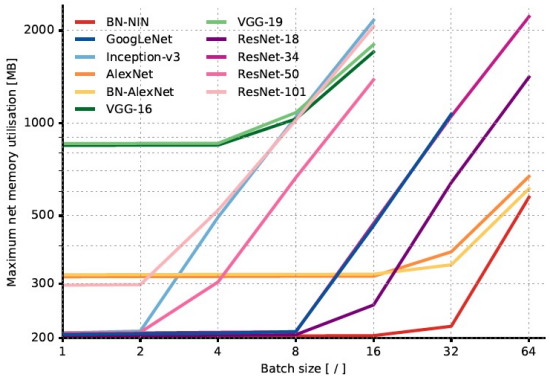


Figure 5: **Memory *vs.* batch size.** Maximum sys-

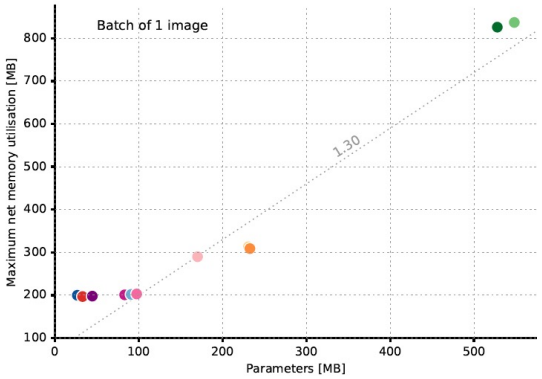


Figure 6: **Memory *vs.* parameters count.** De-

На Figure 5 показано, що спочатку пам'ять є постійною, а потім збільшується зі збільшенням розміру групи. На Figure 6 мінімальний об'єм пам'яті становить 200 МБ, потім він лінійно збільшується з нахилом 1,3.

4. **Операції**

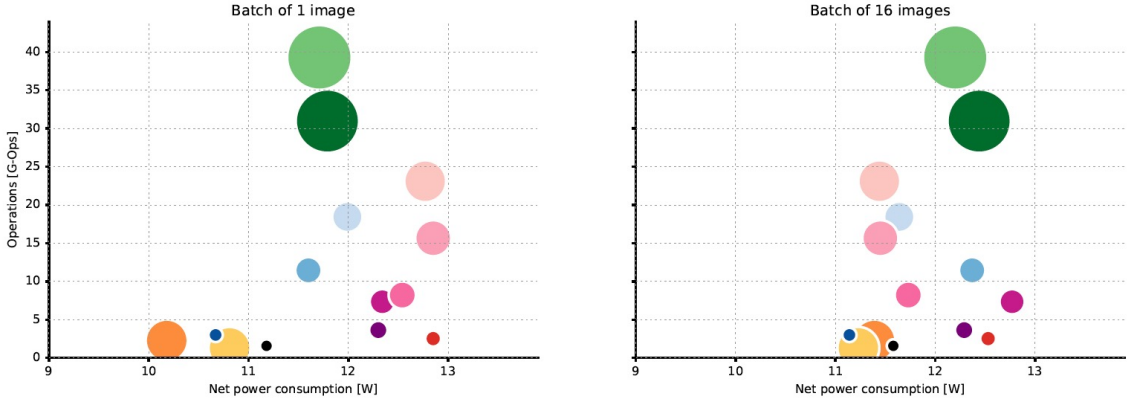
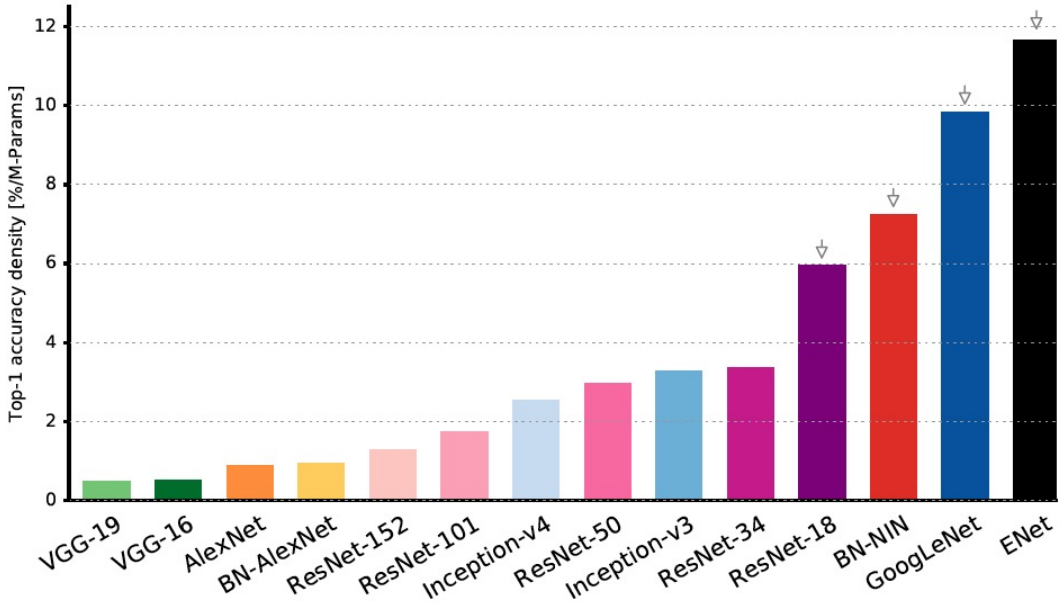


Figure 8: **Operations *vs.* power consumption, size $\propto$ parameters.** Independency of power and operations is

У статті зазначено, що підрахунок операцій є важливим при оцінці часу виводу, а також розміру апаратної схеми. На Figure 7 ми бачимо, що зв'язок між кількістю операцій та часом висновку на одне зображення є лінійним. Таким чином, обмежуючи кількість операцій, ми можемо підтримувати швидкість обробки в корисному діапазоні.

5. **Використання параметрів**



Глибокі нейроні мережі відомо що не є ефективними у використаті всіх їхніх потужностей, тобто кількості поділених параметрів на ступені свободи. З картинки вище ми бачимо, що, хоча VGG має кращу точність чим AlexNet(з картинки 1), він також має гіршу щільність інформації. ENet досягає найвищих балів.

10.10 DenseNet

10.10.1 Вступ

Нейронна мережа - це парадигма програмування, яка дозволяє комп'ютеру вчитися на данних із спостереженнь. Тенденції останніх років вказують на більш глибоке направлення. Згорткові нейроні мережі (ЗНМ) є домінуючими методом навання машин вже 20 років, який дозволяє глибоке навчання. Однак, ЗНМ стає глибше і глибше, інформація о вхідних або градієнти може зникати і зникати, коли вони досягають певного глибинного шару. Більш конкретно, шляхом простого складання шарів разом, сигнал градієнта повинен поширюватись назад по всіх шарах, щоб забезпечити успішне оновлення мереж, а отже виникає проблема зникнення градієнта. В цьому випадку було запропоновано нові роботи з різними способами зробити згорткові мережі глибшими, більш точними та ефективними у навчанні, забезпечивши градієнт який може легко досянути всі шари мережі в глибокій мережі.Тому метою сучасної архітектури останніх запропонованих підходів є створення коротких шляхів від ранніх шарів до подальших шарів, щоб забезпечити доставку інформації.

В підсумку, ми узагальнимо відомий документ *Щільно пов'язані конволюційні мережі*[11], яка представяє нову сучасну архітектуру, щільно пов'язаної конволюції мережі. DenseNets

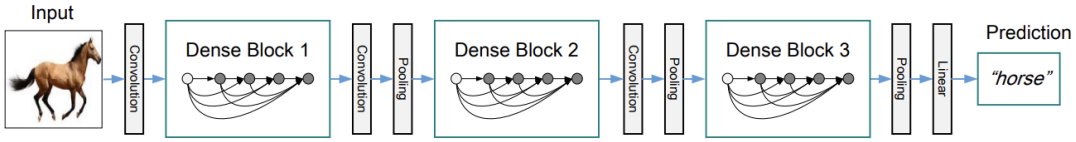
[11]Хуанг, Гао, Чжуан Лю, Кіліан К. Вайнбергер і Лоренс ван дер Маатен. "Щільно пов'язані конволюційні мережі."У матеріалах конференції IEEE з комп'ютерного бачення та розпізнавання образів, том 1, №. 2, с. 3. 2017.

використовує ідею скорочення шляхів, щоб сформувати просту схему зв'язку, тобто об'єднати кожен шар з усіма попередніми шарами та передаючи на власній карті-особливостей у всі наступні шари.

10.10.2 Супутні роботи

Серед усіх згорткових нейронних мереж, Highway Networks [12] були першими ефективими методами для тренерування понад 100 шарів. Обхідні шляхи, що використовуються в Highway Network, відіграють валживу роль для підготовки дуже глибоких мереж. ResNet підтримує ці обхідні методи, які успішно покращують навчання мереж з розміром більше ніж 1000 шарів. Традиційні ResNet підключають вихідний шар до наступного [Huang et al., 2016]. Градієнт проходити крізь функція identity між зєднаними шарами. Інший підхід тренерувати глибокі мережі є збільшення ширини мережі. GoogLeNet і FractalNet досягають конкурентних результатів на навчальному наборі даних, використовуючи широку структуру[13]. Однак, результати вищезазначених методів підсумовуються до кінцевого стану, який може вплинути на роботу мережі. Для покращення інформаційного потоку, DenseNet представлена для використання мережі за через з'єднання шарів за іншим шаблоном для того щоб повторно використовувати.

10.10.3 Архітектура DenseNets



- Щільність зв'язку

  У DenseNets функції всіх попередніх шарів використовуються вхідні дані для наступних шарів. Кожен шар отримує інформацію про функції всіх попередніх шарів, а не лише від попереднього шару. Для кожної реалізації кілька входів шарів H об'єднуються в один тензор. Всього в DenseNets можна знайти $L(L+1)/2$ з'єднаннь.

$$x_l = H_l([x_o, x_1, ..., x_{l-1}]) \tag{62}$$

  де $[x_0, x_1, ..., x_{l-1}]$ об'єднанн карт-особливостей створенних на кожному шарі.

- Композиційна функція

  H представляється як функція-композиція з трьох операцій: групова нормалізація (BN), ректифікована лінійна одиниця(ReLU) та супроводжується $3X3$ згорткою(Conv).

[12] Шрівастава, Рупеш К., Клаус Грефф та Юрген Шмідхубер. "Навчання дуже глибоких мереж."У досагненні нейронних систем обробки інформації, ст. 2377-2385. 2015.

[13] Сугуді, Крістіан, Вей Лю, Янцин Цзя, П'єр Сермане, Скотт Рід, Драгомір Ангуелов, Думітру Ерхан, Вінсент Ванхоук та Ендрю Рабінович. "Заглиблення у згортки."Cvpr, 2015.

- Об'єднання шарів

  Зменшення вибірки шарів для того щоб змінити розмір карти-особливостей є важливим в згорткових нейронних мережах. Для DenseNets, мережі поділені на кілька з'єднанних блоків. Шари між блоками відомі як перехідні шари, які задіяні в згортках і об'єднанні.

- Темпи росту

  На відміну від існуючих мереж, DenseNet може мати дуже вузькі карти-особливостей, які позначаються як k, швидкість росту мережі. Темп росту відображає, наскільки більше нової інформації додається до існуючої інформації. В експериментах, малий темп росту є достатнім для ефективного тренування набору даних.

10.10.4 Експерименти

Автори протестували DenseNet на трьох наборах даних, щоб порівняти їх моделі з іншими архітектурами. Зокрема, вони використали CIFAR набір даних який містить 32*32 кольорові пікселіьні картинки, Street View House Number(SVHN) набір даних який містить 32*32 кольорові цифрові зображення та ImageNet(ILSVRX 2012 класифікаційний набір даних) з більш ніж 1.2 мільйонами зображень.

Всі мережі вони тренерували основуючись на стохастичному градієнтному спуску з початковою швидкістью 0.1 Варто зазначити, що наївна реалізація DenseNet може містити неефективність пам'яті. Таким чином, ефективна реалізація використання пам'яті для DenseNet є важливим аспектом та є детально розробленою в поточній сессії.

Автори тренерували DenseNet модель з різними значеннями глибини(L) та швидкістю росту(k) на вибраних наборах даних. Вони показали, що DenseNet має хороші показники порівняно з іншими архітектурами.

Класифікація результатів на CIFAR і SVHN показали що:

- DenseNet зазвичай виконується краще при більшому L і k без компресії та вузьких місцях. Це означає, що DenseNet має потенціал поглиблюватись та збільщуватись, уникаючи проблем із перенавчанням та оптимізацією ResNet.

- Порівнюючи з ResNet, DenseNet має більше ефективні параметри.

- Через ефективнісств параметрів DenseNet, вона менше страждає від перенавчання. DenseNet-BC вузьке місце та компресія шарів згаданих вище вважаються ефективими для протидії перенавчання.

Класифікація результатів на ImageNet показує що:

- На відміну від ResNet, DenseNet використовує значно меншше параметрів і обраховує поки не досягне того ж самого рівня продуктивності як ResNet.

- Продуктивність DenseNet може бути покращенн за допомогою більш широкого пошуку гіперпараметрів.

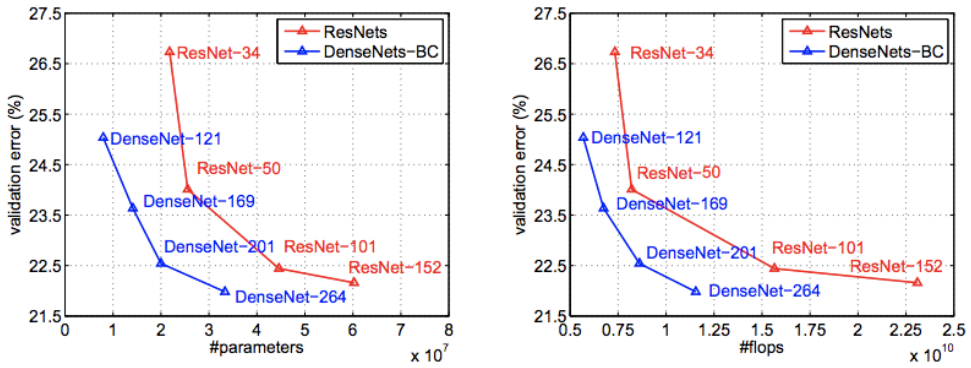


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

10.10.5 Ефективна реалізація пам'яті DenseNet

Як і зазначено вище, DenseNet має вищу ефективність обчислення. Проте неефективність пам'яті є проблемою при тренуванні надзвичайно глибоких DenseNets. Можливим рішенням може бути використанні буфери спільної пам'яті і переобчислення деяких перетворень.

- Наівна реалізація

  Входи для кожного шару є функціями попередніх шарів. Елементи, що походять з різних шарів, не зберігаються суміжно. Дана реалізація виділяє пам'ять для об'єднання і нормалізації виводів так як градієнт проміжні ознаки. Ріст пам'яті є квадратичним.

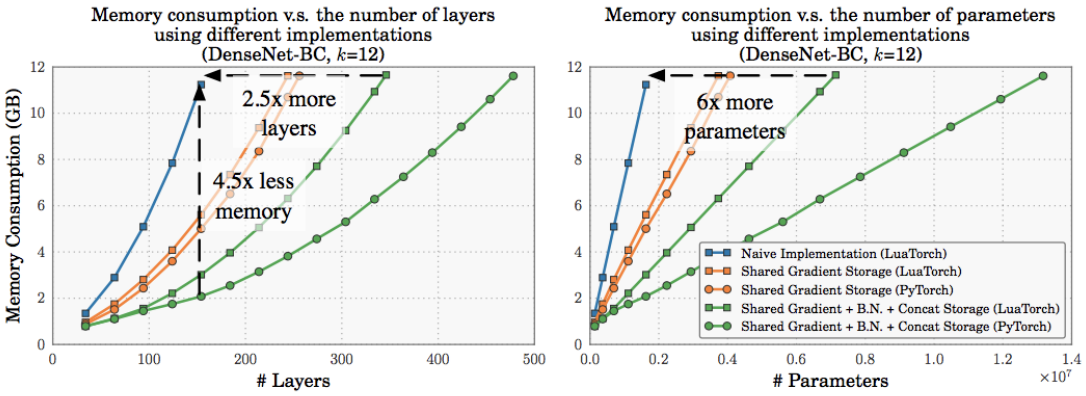
- Ефективна реалізація пам'яті

  Місця зберігання спільної пам'яті стовренні для того щоб уникнути квадратичний ріст пам'яті. Операціїї пакетної нормалізації беруть об'єднанні признаки з спільної пам'яті сховища 1 як вхідні дані та зберігають вихідні дані у сховищі спільної пам'яті 2. Для двох сховищ, їх дані не є постійними і будуть замінені наступним шаром. Під час зворотнього росповсюдження, об'єднанні і нормалізовані особливості перераховуються, оскільки перерахунок є дешевим. Таким чином, ця ефективна реалізація зберігає лише вихідні особливості в пам'яті, які ростуть лінійно.

10.10.6 Переваги

На основі рузультатів експерименту та архітектури DenseNet ми отримали наступні переваги для цієї мережі:

- DenseNet заохочує повторне використання особливостей, що призводить до більш компактної моделі. Карта-особливостей навчена на будь-яких попередніх шарах може бути доступна для всіх наступних шарів. Повторне використання особливостей допомагає менше страждати DenseNet від перенавчання.



- DenseNet має кращі показники продуктивності порівнянно з іншими архітектурами зазначених вище. Це є потенційно тому що неявний глубокий нагляд вбудований в модель архутектури. Всі шари в DenseNets мають однакову функцію втрат, і кожен шар DenseNet має прямий доступ до градієнтів від функції втрат.
- DenseNet підтримує низьку складність функцій.

10.11 Повністю згорткові мережі на семантичній сешментації

10.11.1 Семантична сегментація

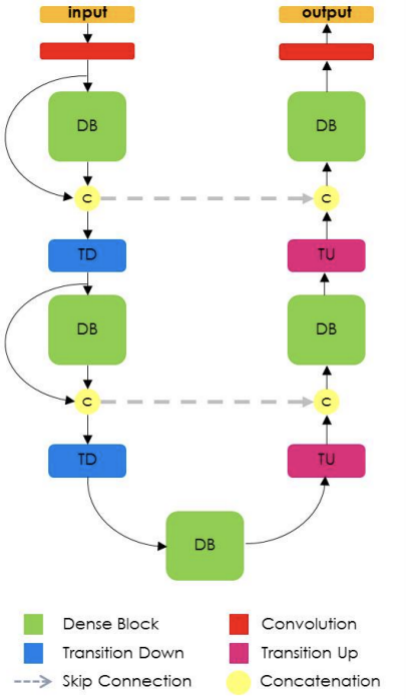
У роботі *Сто шарів Тірамісу: повністб згорнуті DenseNets для семантичної сегментаціїї*[14], ще одна розробка DenseNets представлена щодо застосування семантичної сегментації. Семантична сегментація включає три сполучні частини: шлях зменшення вибірки, який вилучає грубі семантичні особливості, шлях збільшення вибірки, який повинен повернути вихідну роздільну здатність до початкового стану, та модель постобробки для оптимізації кінцевих результатів у моделях, таких як умовне випадкове поле.

Як згадувалось раніше, сучасна архітектура DenseNets має декілька головних переваг. По-перше, вона вимагає менше параметрів ніж традиційна згорткова мережа, так як вона повинна навчати повторювані карти-особливостей. По-друге, DenseNets покращую потік інформації і градієнтів через короткий шлях до кожної карти особливостей. По-третє, шари можуть отримати доступ до карти-особливостей з усіх попередніх шарів для повторного використання інформації з раніше розрахованих карт-особливостей. Всі переваги DenseNets роблять їх придатними для семантичної сегментації.

10.11.2 Архітектура повністю згорткових мереж

DenseNet архітектура в подальшому розширюється до повністю згорткових мереж для семантичної сегментації. Архітектура повністю згорткових мереж побудована шляхом зменшення вибірки, шляхому збільшення вибірки і пропуском з'єднання.

[14] Джегу, Саймон, Міхал Дрозджал, Давід Васкес, Адріана Ромеро та Йоуша Бенджо. "Сто шарів тірамісу: повністю згорнуті мережі для семантичної сегментації."У семінарах "Комп'ютерне бачення та розпізнавання зразків"(CVPRW), 2017 IEEE Конференція , ст. 1175-1183. IEEE, 2017.



- Зменшення вибірки

  При зменшенні вибірки, кількість особливостей зростає лінійно. Однак, для компенсації, просторова роздільна здатність кожної карти-особливостей зменшується після об'єднання. Останній шар всього зменшення вибірки також відомий як вузьким місце, як і згадано раніше.

- Збільшення вибірки

  Протягом попереднього зменшення вибірки, роздільна здатність була зменшенна, для того щоб відновити роздільну здатність, збільшення вибірки представленно, як композиція згорток, операцій збільшення виборки та пропуску з'єднаннь. Зокрема, у моделі DenseNet операція згортки відноситься до щільного блоку, операцій збільшення вибоки - переходу в верх.

- Модуль переходу вгору

  Модуль переходу вгору - це перенесена згортка, яка здатна збільшити виборку попередніх шарів особливостей. Потім збільшена виборка особливостей об'єднується з пропущенними з'єднаннями в новий щільний блок. Однак, Однак, підчасу процесу збільшення виборки, роздільна здатність збільшується та простір пам'яті стає більше. Отже, щоб подолати цей ліміт пам'яті, вхід щілного блоку не об'єднується з виходом, тільки особливості з останнього щільного блоку замість карти-особливостей застосовується. Останній щільний блок не може містити інформацію з ранніх щільних блоків того ж самого розміру. Хоча

деяка інформація з ранніх блоків може бути втрачена, але вона інсує в попредньому в зменшеній виборці мережі і буде отримана через пропуск з'єднаннь. Загалом, весь процес підтримує оригінальну роздільну здатність вхідних особливостей.

10.11.3 Висновок

У цій роботі автори розширили DenseNets до повністю згорткових мереж, що мають глибокі шари та декілька параметрів, і, отже, зможуть поліпшити загальну ефективність стану художньої архітектури. Що ще важливіше, завдяки розумним моделям зв'язку повністю згорнутих мереж, він може представляти ансамбль мереж із змінною глибиною, що буде дуже цікавим для реалізації у проблемі семантичної сегментації.

10.12 Генеративні змагальні мережі (GANs)

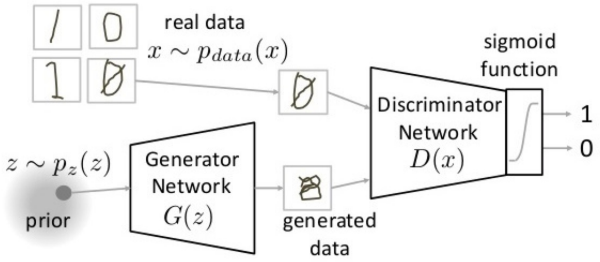
10.12.1 Вступ

Генеративні змагальні мережі (GAN) були представлені в роботі GAN (Goodfellow et al., 2014). Глибокі генеративні моделі відрізняються від традиційних дискримінаційних моделей глибокого навчання, які базуються на алгоритмах зворотного розповсюдження та відсіву, які добре виконують градієнт, забезпечений кусочно лінійними одиницями. Зокрема, це допоможе уникнути труднощів наближення багатьох нерозв'язних імовірнісних обчислень та використання переваг кусочно лінійних одиниць у генеративному контексті.

GAN - це глибокі нейромережеві архітектури, що складаються з двох мереж, які протистоять одна одній (тобто "конкурентність"). Основну ідею того, як працює модель, можна описати як змагання між двома компонентами, в якій вони будуть бити один одного повторними кроками, поки обидва показники не стануть достатньо хорошими, і, таким чином, різниця між ними досягне мінімальної стадії.

10.12.2 Архітектура

Є два основні компоненти GAN - генератор та дискримінатор. Генератор $G(\mathbf{z}; \theta_g)$ приймає ведення $\mathbf{z}$ з попереднього розподілу ймовірностей $p_{\mathbf{z}}(\mathbf{z})$ де G це диференційована функція, представлена багатокористувацьким персептроном з параметрами θ_g. Другий шар персептрону $D(\mathbf{x})$ представляє ймовірність того, що $\mathbf{x}$ походить від даних, а не від розподілу генератора p_g.



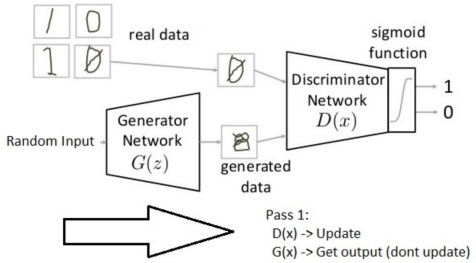
GAN виконується як мінімакс-гра для двох гравців із функцією значення $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

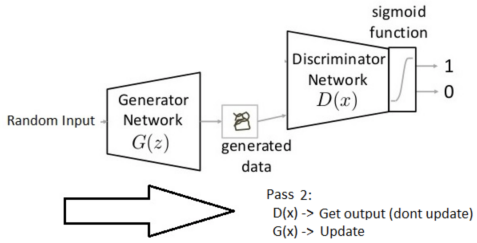
Іншими словами, генератор намагається максимізувати ймовірність того, що дискримінатор помилково вводить свої входи як реальні, тоді як завдання дискримінатора прямо протилежне, тобто він повинен вирішити, чи надходить вхід від генератора або від справжнього навчального набору.

Частини навчання GAN.

Є 2 частини, які виконуються послідовно на етапі навчання. Прохід 1: Тренування дискримінатора та заморожування генератора. Заморожування означає встановлення навчання як помилкове. Мережа робить лише прямий прохід, і зворотне розповсюдження не застосовується



Прохід 2: Тренування генератора та заморожування дискримінатора



Кроки для тренування GAN

Крок 1: Визначте проблему. Визначте ціль (тобто згенеруйте зображення або текст) і переконайтесь, що ви зібрали для цього достатньо даних.

Крок 2: Визначте архітектуру GAN. Вирішіть, яку архітектуру ви повинні використовувати як для генератора, так і для дискримінатора, багатошарових персептронів або згорткових нейронних мереж.

Крок 3: Навчіть дискримінатор на реальних даних за n проходів. Навчіть дискримінатор правильно прогнозувати реальні дані як реальні.

Крок 4: Створіть підроблені вхідні дані для генератора та навчайте дискримінатор на підроблених даних. Навчіть дискримінатор правильно передбачати сформовані дані як підроблені.

Крок 5: Навчіть генератор з виходом дискримінатора. Навчіть генератор обдурювати дискримінатор.

Крок 6: Повторіть крок 3 - крок 5 кілька разів.

Крок 7: Перевірте, чи неправдиві дані видаються законними. Якщо це здається доречним, припиніть тренування, інакше перейдіть до кроку 3.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

10.12.3 Результат

Теоретичні результати

1. Глобальна оптимальність $p_g = p_{data}$

   Для фіксованого G , оптимальний дискримінатор D це

$$D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$

2. Збіжність алгоритму 1

   Якщо G та D мають достатню потужність, і на кожному кроці алгоритму 1 дискримінатору дозволяється досягти свого оптимального значення G, а p_g оновлюється з метою покращення критерію

   $$\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D_G^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_g}[1 - \log D_G^*(\mathbf{x})]$$

   тоді p_g конвертується у p_{data}.

Експериментальні результати

Коли і G, і D є нейронними мережами прямого зв'язку, результати, які ми отримуємо від навчання з набору даних MNIST та набору даних TFD, є такими.

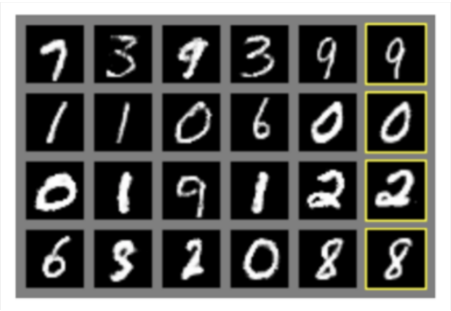


Рис. 56: MNIST

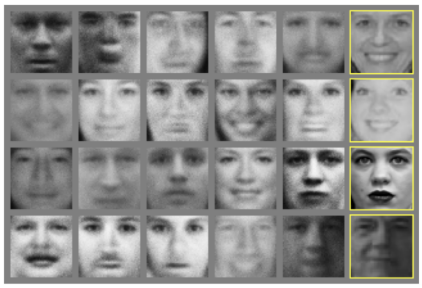


Рис. 57: TFD

10.12.4 Переваги & Недоліки

GANs є потужним інструментом для навчання класифікаторів напівнаглядом, коли недостатньо позначених прикладів. Він генерує зразки швидше, ніж повністю видимі мережі довіри (NADE, PixelRNN, WaveNet тощо), оскільки немає необхідності генерувати різні записи у зразку послідовно. GANs не покладаються на наближені значення Монте-Карло для навчання, які погано працюють у великого розміру просторах, тому, наприклад, GANs працюють ефективніше, ніж машини Больцмана, які покладаються на методи Монте-Карло під час навчання на ImageNet.

Однак GANs іноді нестабільні, оскільки навчання моделі GAN вимагає пошуку рівноваги Неша

в грі. Якщо думате про те, що дискримінаційна частина є потужнішою, ніж її колега-генератор на етапі навчання, тоді генератор не зможе ефективно тренуватися. З іншого боку, якщо генератор набагато потужніший за дискримінатор, тоді генерується будь-яке зображення. Хоча іноді градієнтний спуск може сприяти підвищенню стійкості, він не працює весь час, і на цьому етапі немає хорошого алгоритму пошуку рівноваги. Більше того, GAN важко навчитися генерувати дискретні дані, такі як текст.

10.13 Глибокі згорткові генеративні змагальні мережі

Оригінальна робота про Генеративні змагальні мережі від Goodfellow et al. [2014] запропонувала нову архітектуру для підготовки генеративної моделі. Ця робота залишила багато місця для подальшої роботи. Створення хорошої генеративної моделі є мотивацією для GANs, а при тому, що генерація зображень виконується експериментальним шляхом, було вражаюче побачити створені зображення, враховуючи, що реалізація GAN використовувала лише повністю зв'язані шари. У домені контрольованого навчання, коли зображення використовуються як вхідні дані, згорткові архітектури показали надзвичайну ефективність (просто див. Розділ 10.9).

Враховуючи потенціал, який демонструють GAN, і величезний успіх згорткових архітектур в області комп'ютерного зору, Редфорд, Мец та Чінтала запровадили новий клас CNN, заснований на обох цих ідеях, під назвою Deep Convolutional Generative Adversarial Network (DCGAN). у їхній роботі Radford et al. [2015].

Представлення зображень

Робота у Goodfellow et al. [2014] показала, що хороші зображення в MNIST та інших наборах даних малих зображень можуть бути створені за допомогою моделі лінійних шарів. Однак, як було показано, зображення є чудовим кандидатом для оператора згортки через локалізацію об'єктів на зображеннях та той факт, що часто об'єкти присутні в декількох місцях зображення. Тому при генерації зображень було б корисно мати пул багаторазових функцій, на які можна посилатися. Крім того, загальні риси присутні у багатьох природних зображеннях, і їх не потрібно спеціально маркувати, щоб їх можна було ідентифікувати за допомогою некерованого алгоритму навчання. Нарешті, хоча ImageNet та інші набори даних забезпечують безліч зображень із мітками, кількість немічених зображень набагато більша, і об'єднання позначених та немічених зображень можна використовувати для навчання без нагляду.

Враховуючи багатство доступних немаркованих зображень та індуктивні упередження, наявні в природних зображеннях, має сенс поєднувати згорткові оператори з некерованим алгоритмом навчання, таким як GAN. Після вивчення хороших екземплярів, у мережах генераторів та дискримінаторів з'являться хороші генератори та екстрактори характеристик. Потім вони можуть бути використані як екстрактори функцій для більшої контрольованої моделі, тобто ця модель матиме переваги для завдань такі як класифікація зображень.

Архітектура

Три основні архітектурні компоненти DCGAN:

1. Всі згорткові: замість того, щоб використовувати пул для подальшої вибірки, використовуйте поетапні згортки. Це дозволяє мережі навчитися власному зменшенню вибірки. Цей прийом використовується як в дискримінаторі, так і в генераторі (див. фігуру 58) де

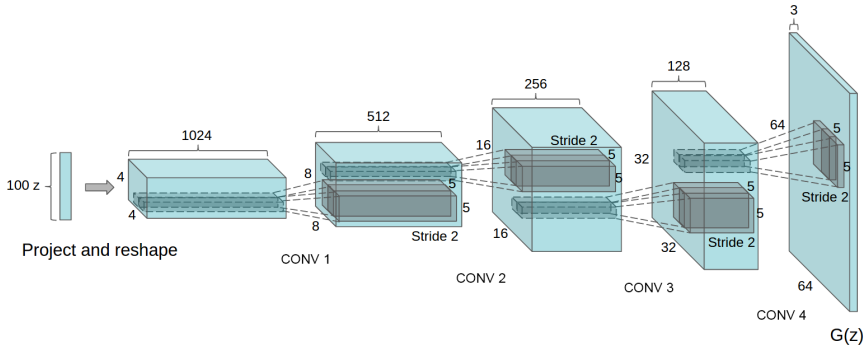


Рис. 58: Generator design for the DCGAN architecture

кроки стають дробовими (вірніше, перенесена згортка з кроками) і викликається збільшення вибірки.

2. No fully connected layers: remove fully connected layers from the top of the components. Hence the output of the discriminator (which is spatial with channels) is used directly as the input to the discriminator. The authors of Radford et al. [2015] motivated this architectural choice by indicating that it lead to faster convergence.

3. Batch Normalization Ioffe and Szegedy [2015a]: the authors found that Batch Normalization lead to more stable learning as it allowed gradients to flow with more ease. This was applied to every layer with the exception of the output of the generator and input to the discriminator.

In addition to the architectural choices listed above the authors suggested using ReLU Nair and Hinton [2010] at every layer in the discriminator, minus the last layer which used `tanh` (ostensibly to more easily allow the model to reach all the possible colors). The discriminator, on the other hand, used leaky ReLUs Xu et al. [2015] which allowed more gradient to flow through the discriminator (especially important at the beginning of training when the generator produces images obviously off the data manifold).

Performance

The DCGAN model was able to help train a classifier to near state-of-the-art performance on CIFAR-10 (compared to other unsupervised feature extraction models). Here the layers from the discriminator were used as a feature extractor, with a linear model on top to do the actual classification. This model achieved 82.8% accuracy on CIFAR-10 (and, notably, the discriminator was trained on ImageNet, not CIFAR-10).

Lastly, and qualitatively, the generator model was able to produce very realistic images of both bedrooms (for the LSUN dataset) and faces (from the celebrity faces dataset). The reader should refer to the original paper Radford et al. [2015] for these examples.

10.14 Цикл GAN

10.14.1 Історія

Архітектура Cycle GAN пропонується для виконання завдань перекладу зображення в зображення. Метою перекладу зображення в зображення, як випливає з назви, є вивчення відображення між доменом вихідного зображення X та цільовим доменом Y. Він має широкий спектр застосувань в комп'ютерному зорі та графіці, наприклад, його можна використовувати для редагування фотографій для перетворення зображення в стиль олійного живопису.

Кілька випробувань для цього завдання включають наступне:

- Парні дані часто обмежені чисельністю або взагалі недостатньо чітко визначені, а це означає, що зв'язок неможливо дізнатись безпосередньо.
- Характеристики колекції зображень важко захопити без нагляду

Документ Zhu et al. про цикл GAN вводить зворотне відображення та втрату послідовності циклу для Генеративної мережі супротивників (GAN) для вирішення цих проблем.

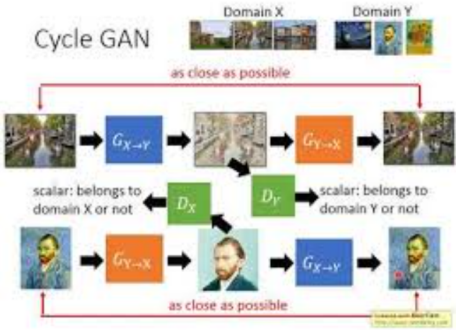


Рис. 59: Cycle GAN

10.14.2 Архітектура

Окрім звичайного GAN, який має лише відображення $G : X \to Y$, у CycleGAN введено зворотне відображення $F : Y \to X$.

Функція збитків для GAN є наступною

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x))]$$

Також введено дискримінатор для зворотного відображення, який позначається як D_X. Подібний термін втрат введено для зворотного відображення, яке становить $\mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$. З цими двома термінами у змагальної мережі все ще залишається занадто багато свободи для функції відображення. Таким чином, автори ввели третій термін втрат $\mathcal{L}_{\text{cyc}}(G, F)$, відомий як консистенція циклу, щоб накласти на нього подальше обмеження. Тоді повна мета є наступною:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \mathcal{L}_{\text{cyc}}(G, F)$$

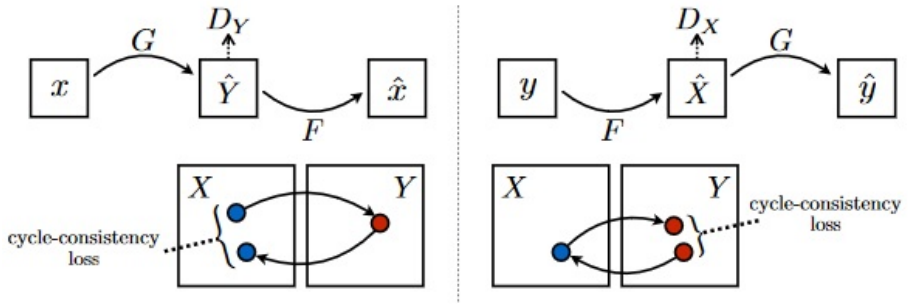


Рис. 60: CycleGAN Architecture

10.14.3 Результати

Результати, які вони продемонстрували у своїй роботі, справді дуже вражаючі, як показано в Figure 61. Результат є гарним візуально, автори також оцінили його кількісно та повідомили, що вони мають найсучасніші результати.

Рис. 61: CycleGAN Результат

11 Рекуррентна нейронна мережа

11.1 Мовна модель

Нейронна мережа досягає вражаючого прогресу у розв'язанні нагальних проблем, пов'язаних із природними мовами, включаючи перевірку якості, переклад мов, класифікацію тестів, тощо. Перш ніж ми поговоримо про те, які моделі можуть допомогти нам розв'язувати проблеми ОБМ (Обробка природної мови), нам потрібно знати, як змоделювати речення.

Речення, побудоване за словами, і правильне речення повинні містити правильні слова та їх правильний порядок. Тому для призначення можливості вироку ми можемо мати:

$$P(W) = \mathbb{P}(w_1, w_2...w_n)$$

Потім ми застосовуємо ланцюгове правило до цієї можливості:

$$\mathbb{P}(w_1, w_2...w_n) = \prod_i \mathbb{P}(w_i|w_1...w_{i-1})$$

Хоча, для речення *ця вода така прозора* ми можемо не хотіти обчислювати умовну можливість $P(\text{transparent}|\text{itswaterisso})$шляхом підрахунку та ділення, оскільки у нас недостатньо даних для обчислення. Використовуючи припущення Маркова, ми можемо використати Спрощене припущення:

$$\mathbb{P}(w_1, w_2...w_n) \approx \prod_i \mathbb{P}(w_i|w_{i-k}...w_{i-1})$$

Отже, умовну можливість наступного слова ми можемо виразити його найближчими k словами. Тому доцільно використовувати групу слів для обчислення слів у наступній групі у документах. Розглянемо просту задачу генерації тексту. У текстовій послідовності *abcdefghijklm nopqrstuvwxyz*, ми застосовуємо пакетну перевірку та використовуємо групу слів як вхід для подачі клітинки рекуррентної нейронної мережі (РНМ).

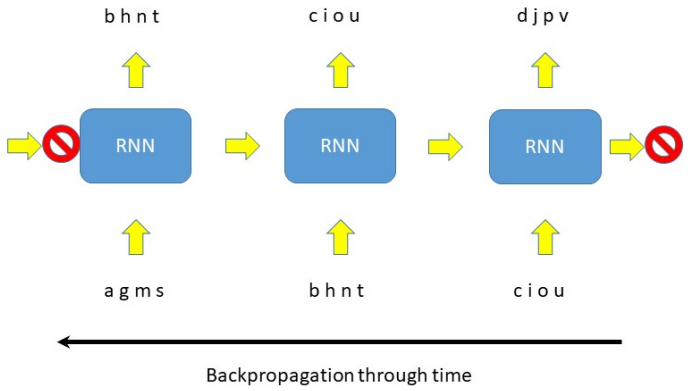


Рис. 62: Мовна модель для генерації тексту

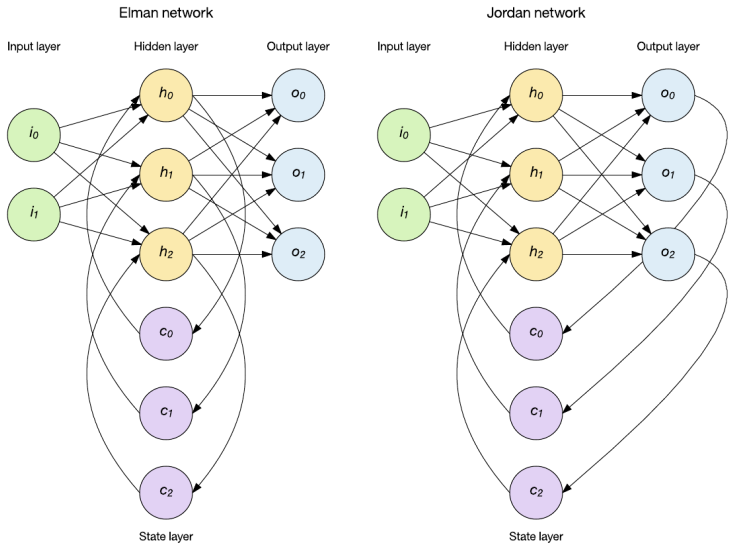


Рис. 63: Мережі Елмана (зліва) та мережі Джордана (праворуч)

11.2 Проста періодична мережа

Рекуррентна нейронна мережа (РНМ) - це клас штучної нейронної мережі, де зв'язки між нейронами утворюють спрямований графік. Це означає, що рівень на виході або результати прихованого рівня зберігаються в пам'яті (так званий рівень стану) і повертаються до мережі як вхідні дані.

Рекурретні нейронні мережі дуже корисні для послідовних даних, зокрема, вони мають великі перспективи у багатьох завданнях НЛП. Це пояснюється тим, що кожен нейрон або одиниця може використовувати свою внутрішню пам'ять для підтримки інформації про попередній контекст, даючи кращі результати навчання.

Як правило, рекурентні мережі можна навчити за допомогою зворотного розповрсюдження. Існує два підходи до побудови найпростішої форми рекурентних мереж. У випадку мережевого підходу Ельмана, канали прихованого шару зберігаються в пам'яті і використовуються як рівень стану, і вони повинні надходити назад у систему.

Інший підхід називається мережами Джордана. Мережі Джордана замість того, щоб зберігати історію прихованого рівня, вони зберігають історію вихідного рівня до рівня стану і передають це назад до мережі. Загалом, мережа Джордана отримує кращу продуктивність, оскільки вхідні дані на рівні стану є результатами вихідного рівня, також часто простіше інтерпретувати інтуїтивне значення результатів рівня стану рівня.

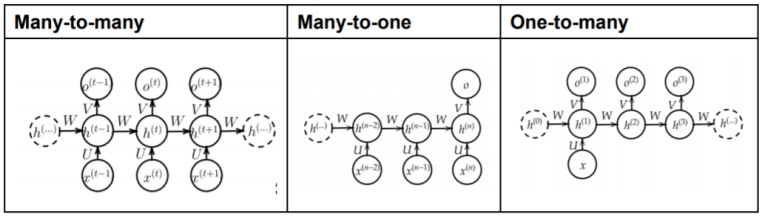


Рис. 64: Також є розділення рекуррентних мережа на три види

11.3 Нейронна мережа Гопфілда

Нейронна мережа Гопфілда — це тип рекурентної, повнозв'язної, штучної нейронної мережі з симетричною матрицею зв'язків. У процесі роботи динаміка таких мереж сходиться (конвергує) до одного з положень рівноваги. Ці положення рівноваги є локальними мінімумами функціоналу, що називається енергія мережі (у найпростішому випадку — локальними мінімумами негативно певної квадратичної форми на n-вимірному кубі). Така мережа може бути використана як автоасоціативна пам'ять, як фільтр, а також для розв'язання деяких завдань оптимізації. На відміну від багатьох нейронних мереж, що працюють до отримання відповіді через певну кількість тактів, мережі Хопфілда працюють до досягнення рівноваги, коли наступний стан мережі дорівнює попередньому.

11.4 Рекуррентна мережа з довшою пам'яттю

Підхід до довгої короткочасної пам'яті (ДКЧП) - це особливий тип рекурентних мереж, здатних вивчати довготривалі залежності, вперше представлений Хохрейтером та Шмідхубером (1997).

ДКЧП має можливість видаляти або додавати інформацію до рівня стану за допомогою регульованих структур, або по іншому шлюзів. Шлюзи складаються з сигмовидних функцій, де вони використовують нуль та одиницю для контролю, якщо сигнал повинен проходити через рівень стану. Більш конкретно, ДКЧП має троє шлюзів для захисту та управління, а саме: вхідні, вихідні та забувальні шлюзи. Всі ці шлюзи містяться в комірці, тому інформацію можна зберігати, переписувати чи читати, а управління шлюзами вивчається за допомогою нейронних мереж, налаштованих за допомогою процесу навчання рекурретних мереж.

У простій рекурентній мережі на рівні стану ми маємо:

$$f_t = \boldsymbol{W_f} \cdot [h_{t-1}, x_t] \tag{63}$$

де стан складається з вхідних даних із рівня стану в $t-1$, і поточний вхід x_t. У ДКЧП

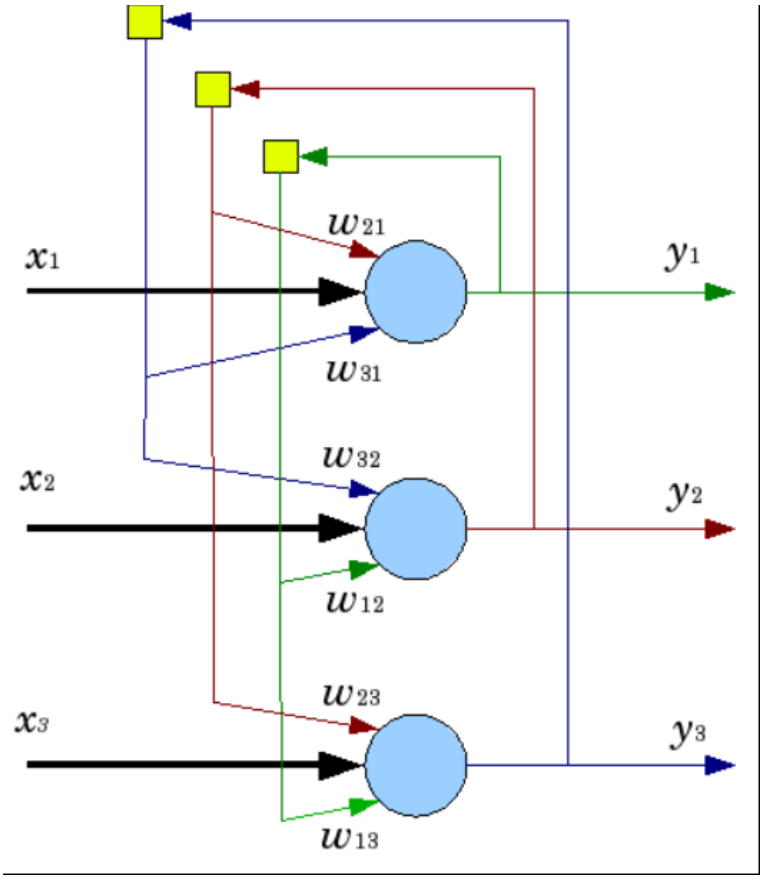


Рис. 65: Нейронна мережа Гопфілда

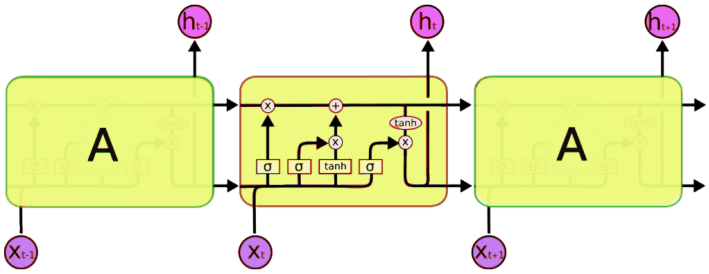


Рис. 66: Керуюча комірка ДКЧП

це передається через сигмоподібну функцію і використовується для управління забувними шлюзами, де 1 представляє утримання, тоді як 0 представляє забування (відкидає поточне збережене значення).

$$f_t = \sigma(\boldsymbol{W_f} \cdot [h_{t-1}, x_t] + b_f) \tag{64}$$

Забувний шлюз дозволяє тренувати окремі нейрони, що важливо і як довго це залишатиметься важливим, тому ДКЧП може працювати з даними, де важливі події можуть бути розділені на тривалі періоди часу. Аналогічно, для вхідного шлюзу, який контролює те, що записано на рівень стану, ми мали б:

$$i_t = \sigma(\boldsymbol{W_i} \cdot [h_{t-1}, x_t] + b_i) \tag{65}$$
$$h_t = \tanh\left(\boldsymbol{W_{hh}} \boldsymbol{h_{t-1}} + \boldsymbol{W_{xh}} \boldsymbol{x_t}\right) \tag{66}$$
$$y_t = \boldsymbol{W}_{hy} h_t \tag{67}$$

Вхідні дані визначають, які значення слід додати до стану комірки, залучаючи сигмоподібну функцію. Це схоже на забувальний шлюз і діє як фільтр для всієї інформації від h_{t-1} і x_t.

Вхідний шлюз також передбачають створення вектора, який містить усі можливі значення, які можна додати (з h_{t-1} і x_t) до стану клітини за допомогою tanh функції, яка виводить значення з -1 до $+1$. Помножуючи значення регуляторного фільтра (сигмоподібний затвору) на створений вектор (tanh функції), а потім додає їх до стану комірки за допомогою операції додавання.

Нарешті, вихідний шлюз, який контролює зчитування з рівня стану:

$$o_t = \sigma(\boldsymbol{W_o} \cdot [h_{t-1}, x_t] + b_o) \tag{68}$$

Фільтр побудований з використанням значень h_{t-1} і x_t, а також сигмовидної функції. Вихідні ворота відправляють це як вихід і в прихований стан наступної комірки.

ДКЧП є дуже перспективним рішенням проблем, пов'язаних з послідовністю та часовими рядами, насправді, мережевий дизайн Ельмана та Джордана сьогодні використовується рідко. Існує багато сучасних досліджень, побудованих на основі підходу ДКЧП, таких як Grid ДКЧП від Кальхбреннер, та ін. (2015), або інша робота з використанням РНМ в генеративних моделях Грегор та ін. (2015), Чунг та ін. (2015), або Байєр & Осендорфер (2015).

11.5 Зникаючі та вибухові градієнти

Нейронні мережі, що повторюються, схильні до зникнення та вибуху градієнтів. Градієнт має тенденцію "вибухати"або "зникати"під час зворотного поширення в часі.

Розглянемо рекуррентну нейронну мережу на послідовності, довжиною k + 1, позначену x_0 через x_k. У наступних формулах: s_i відноситься до прихованого стану в момент i, $\hat{y_i}$, відноситься до виводу в точці даних (x_i, y_i) і E_i посилається на помилку у (x_i, y_i). Щоб знайти градієнт E_k ваги W, нам потрібно було б розрахувати $\frac{dE_k}{dW} = \sum_{i=0}^{k} \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{ds_k} \frac{ds_k}{ds_i} \frac{ds_i}{dW} = \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{ds_k} \sum_{i=0}^{k} \frac{ds_k}{ds_i} \frac{ds_i}{dW}$.

Ми можемо розширити це рівняння, написавши $\frac{ds_k}{ds_i}$ as $\prod_{j=i}^{k-1} \frac{ds_{j+1}}{ds_j}$, отже, загальне рівняння зворотного поширення РНМ є:

$\frac{dE_k}{dW} = \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{ds_k} \sum_{i=0}^{k} (\prod_{j=i}^{k-1} \frac{ds_{j+1}}{ds_j}) \frac{ds_i}{dW}$ [Britz, 2016]

Ключовим моментом є те, що градієнт є результатом багатьох множень матриць, чим довша довжина послідовності, тим більше множень потрібно для обчислення градієнта. Далі, градієнт відносно одного кроку часу залежить від усіх попередніх кроків часу. Отже, якщо будь-який з попередніх кроків часу має нульовий градієнт, вони рухають градієнти в наступні часові кроки до 0, створюючи зникаючий градієнт. Подібним чином, якщо будь-який з попередніх кроків часу має великий градієнт, вони ведуть наступні градієнти до вищих значень.

Вибір функції активації може зробити вірогідним зникнення або вибух градієнтів. Функції активації tanh і sigmoid мають похідну, близьку до 0 на обох кінцях, де вони наближаються до рівної лінії. Функція активації ReLU менш сприйнятлива до зникаючих градієнтів.

11.5.1 Далекі залежності

Один із наслідків проблеми зникаючого градієнта це коли рекуррентні нейронні мережі намагаються вивчити залежності великого діапазону. Градієнт помилки на наступному елементі послідовності x_k прихованого стану після попереднього елемента послідовності x_a полягає в наступному:

$\frac{dE_k}{ds_a} = \frac{dE_k}{d\hat{y_k}} \frac{d\hat{y_k}}{ds_k} \sum_{i=a}^{k} (\prod_{j=i}^{k-1} \frac{ds_{j+1}}{ds_j})$

Чим далі один від одного ранній x_a і наступні x_k один від одного, тим менший вплив s_a має на градієнт на кроці k, тому що в рівнянні є все більше і більше членів, що заглушують вплив s_a.

Це має практичні наслідки для продуктивності моделі. Розгляньте запитання, відповідаючи на модель обробки природної мови. Як приклад, припустимо, модель намагається відповісти "Коли народився Бетховен?"Вхідним реченням: "Бетховен, відомий композитор, який здійснив революцію в класичній музиці, народився в 1770 році". Модель повинна мати можливість зв'язати Бетховена з роком народження, незважаючи на довгу фразу, що розділяє два споріднені слова. Отже, йому потрібно навчитися залежності на великій відстані.

ДКЧП та Керований Рекурентний Блок(КРБ) призначені для кращого вивчення великого діапазону залежностей за допомогою окремого вектора для представлення довготривалої пам'яті.

11.5.2 Обмеження вибуху градієнтів

"Вибухові градієнти"можна виправити за допомогою "Обмеження вибуху градієнтів". Існує багато способів відсікання градієнтів, але одним із загальних прикладів є нормалізація градієнтів, коли норма L2 перевищує поріг.

Якщо проблема вибуху градієнта досить серйозна, програма може взагалі вибухнути через значення NaN.

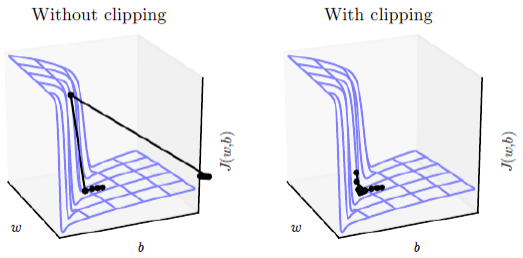


Рис. 67: Обмеження вибуху градієнтів Goodfellow et al.

11.6 Ортогональна РНМ

За визначенням, матриця $\boldsymbol{W}$ є ортогональною, якщо $\boldsymbol{W}\boldsymbol{W}^\top = \boldsymbol{W}^\top\boldsymbol{W} = \mathbb{I}$. Ортогональні матриці мають цікаву властивість, де власні значення мають абсолютне значення 1 або, що еквівалентно, властивість, що зберігає норму. Це робить ортогональні матриці надзвичайно корисними при зворотному розповсюдженні РНМ. У нас є формула зворотного розповсюдження через час (ЗРЧЧ), як показано нижче для РНМ з $\boldsymbol{W}$, ініціалізованим як ортогональна матриця.

$$\delta^{(n-1)} = \delta^{(n)}\mathbf{w}^T\mathbf{D}^{(n)} \tag{69}$$

де локальні градієнти $\delta^{(n)}$ визначаються як $\frac{\partial L}{\partial \mathbf{w}}$ і $\mathbf{D}^{(n)} = diag(\sigma'(\mathbf{w}h_{t-2}+b))$ [15]

Можна легко сказати, що властивість, що зберігає норми, має велику перевагу тут, в рівнянні(69). З ортогональністю, незалежно від того, скільки разів ми повторюємо множення матриці під час ЗРЧЧ, отримана матриця ваги не вибухає і не зникає.

11.7 Вступ до ДКЧП

11.7.1 Навіщо нам ДКЧП

При зворотному розповсюдженні РНМ, нам потрібно помножити частину транспонованої матриці ваги в кожному шарі РНМ, щоб обчислити градієнт втрат відносно h_0. Остаточне вираження градієнта на h_0 буде включати багато факторів вагової матриці. Коли найбільше сингулярне значення вагової матриці більше 1, ми матимемо проблему вибуху градієнта. Коли

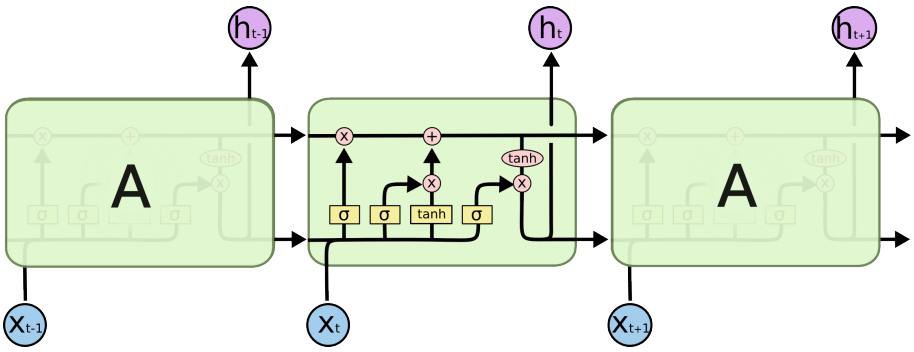
[15] diag перетворює вектор у діагональну матрицю

найбільше сингулярне значення вагової матриці менше 1, ми матимемо проблему зникаючого градієнта.

ДКЧП - це архітектура РНМ, яка базується на цій проблемі. Він призначений для вирішення проблеми зникнення та вибуху градієнта. ДКЧП дозволяє зберігати градієнти. Осередок пам'яті запам'ятовує перший вхід, доки забувальний шлюз відкритий, а вхідний шлюз закритий. ДКЧП формує градієнтну магістраль, яка дозволяла градієнтному потоку перешкоджати від втрати до початкового стану комірки на початку моделі. Забувальні шлюзи в ДКЧП можуть варіюватися в різних шарах, тобто, уникати проблеми зникнення та вибуху градієнта.

11.7.2 Архітектура ДКЧП

ДКЧП не змінює загальної структури РНМ. Натомість ДКЧП просто замінює клітину, яка використовується в інших поширених нейронних мережах. Ми використовуємо зображення від відомого підручника з ДКЧП для ілюстрації архітектури ДКЧП.



Перш ніж йти далі, нам потрібно визначити два важливі поняття: вхід / вихід і стан комірки. Як і комірка в РНМ, ДКЧП приймає введення від попередньої комірки (стрілка вліво внизу) та поточний вхід (синій X), і генерує вихід (фіолетовий h) що також використовується наступною коміркою (стрілка внизу праворуч). ДКЧП підтримує ще одну властивість: стан комірки, який можна інтерпретувати як "інформація, яку ми хочемо зберігати по ланцюжку РНМ яка представлена стрілками у верхній частині кожної комірки.

Функція комірки ДКЧП функціонує у три етапи:

1. Вирішіть, яку інформацію слід забрати від стану клітини. Вхідні дані проходять через сигмоподібний шар (крайній лівий жовтий квадрат) і помножуються (по елементам) на попередній стан комірки. Запам'ятайте, що значення сигмоїдів відображає значення 0 або 1, що означає, що ці значення, зіставлені з 0, будуть забуті в стані комірки. Цей сигмоподібний шар також називається "Забувальний шлюз".

2. Оновіть стан комірки новою інформацією. Вхідні дані передаються через інший шар - сигмоподібний , щоб визначити, які значення будуть додані (оновлені), а також через шар "tanh щоб сформувати вектор-кандидат для оновлення. Ці два шари представлені двома жовтими квадратами посередині. Кандидат-вектор множиться із сигмоподібним висновком і додається (за елементами) до стану клітини. Також сигмоподібний шар називається "вхідним шлюзом". Після оновлення поточний стан комірки готовий до передачі в наступну комірку.

3. Визначте вихід. Наш вхід знову проходить через третій сигмоподібний шар (так званий "вихідний шлюз") і помножується на стан комірки, який пройшов через "tanh". Кінцевий вихід також подається в наступну комірку в ланцюзі РНМ.

11.7.3 Застосування ДКЧП

ДКЧП використовується в багатьох завданнях. Наприклад, машинний переклад, субтитри до зображень, генерація зображень, відповіді на запитання, розпізнавання мови, управління роботом, прогнозування часових рядів, розпізнавання рукописного вводу тощо.

12 Мережа, доповнена пам'яттю

РНМ і ДКЧП страждають від обмеженого обсягу пам'яті. Крім того, вони читають дані лише в послідовному порядку. Мережі, розширені пам'яттю, прагнуть мати набагато більшу зовнішню пам'ять, ніж РНМ та ДКЧП, маючи при цьому можливість робити вибіркові читання та записи, щоб вона могла отримувати доступ до входів у послідовних замовленнях.

12.1 Мережі пам'яті

Для більшості моделей, заснованих на ДКЧП,РНМ та УРН використовується прихований стан як механізм пам'яті, який обмежений об'ємом пам'яті. Тобто їх пам'ять надто мала, щоб точно запам'ятати факти з довготривалих залежностей, так що вони не можуть встановити взаємозв'язок між кількома реченнями.. Крім того,УРН, ДКЧП і РНМ зчитують дані в послідовному порядку і кодують речення до одного щільного вектора низької розмірної пам'яті. Такі механізми пам'яті можуть призвести до втрати величезної кількості інформації. Для вирішення цих питань, Weston et al. [2014] запропонована мережа пам'яті (ЗМН), яка дозволяє глибокій нейронній мережі зберігати велику пам'ять, а також ефективно читати та записувати пам'ять.

Мережа пам'яті містить слот пам'яті (m), а також чотири компоненти: вхід (I), узагальнення (G), вихід (O) та відгук (R) відповідно. Приклад відповіді на запитання, який використовується для ілюстрації архітектури мережі пам'яті, можна знайти на малюнку 68. Спочатку модуль I перетворює введений текст у вектор функції, а потім модуль G оновлює слоти пам'яті на основі вектора ознак. Далі модуль O обчислює та знаходить відповідну інформацію відповідно до введення функції та тексту запитання, потім модуль R перетворює відповідну інформацію, що виводиться, у відповідь в бажаному форматі природної мови. Що стосується базової архітектури мережі пам'яті, ми можемо розглядати модуль I як процес пошуку вбудовування. Модуль G - це не що інше, як зберігання вбудованого вектора у слот пам'яті. Отже, модулі O і R обробляють більшу частину роботи з умовиводу. Модуль O виробляє вихідні функції, знаходячи верхню частину k, що підтримує пам'яті заданої вхідної функції x. Наприклад $k = 2$, найвищу оцінку допоміжної пам'яті отримують:

$$o_1 = O_1(x, m) = \underset{i=1,..,N}{\operatorname{argmin}} s_O(x, m_i) \tag{70}$$

where, s_O - функція оцінки, яка відповідає парі речень x і m_i. N представляє загальну кількість слотів пам'яті. Потім дано o_1,ми можемо продовжувати обчислювати o_2 наступним чином,

$$o_2 = O_2(x, m) = \underset{i=1,..,N}{\operatorname{argmin}} s_O([x, m_{o_1}], m_i) \tag{71}$$

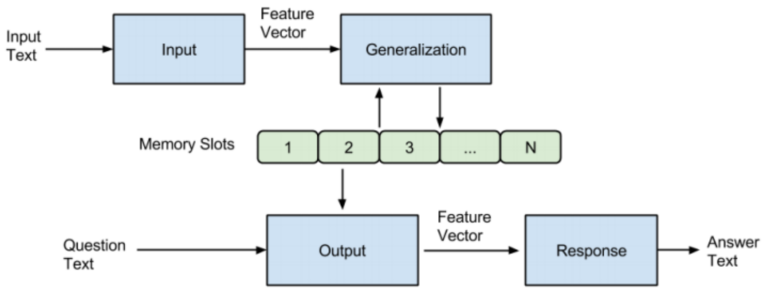


Рис. 68: Архітектура мережі пам'яті у завданні, що відповідає на питання

де, $s_O([x, m_{o_1}], m_i)$ також, може бути написано як $s_O(x, m_i) + s_O(m_{o_1}, m_i)$.

Остаточний результат - для $k = 2$ is $[x, m_{o_1}, m_{o_2}]$, тоді модуль R приймає вихідні дані і видає текстову відповідь r наступним чином,

$$r = O_2(x, m) = \underset{w \in W}{\operatorname{argmin}}\, s_R([x, m_{o_1}, m_{o_2}], w) \tag{72}$$

де W - набір усіх слів у словнику.

ЗМНО проходить навчання в повністю контрольованому середовищі, де нам надаються вхідні дані, правильні відповіді та підтримка вибору. Однак деякі варіанти мережі пам'яті були вивчені з меншим наглядом під час навчання. Ми представимо один із них у наступному розділі.

12.2 Наскрізні мережі пам'яті

Наскрізні мережі пам'яті [Sukhbaatar et al., 2015] - це вдосконалення мереж пам'яті, запропо -нованe Weston et al. [2014]. Основний внесок наскрізної мережі пам'яті полягає в тому, що модель може тренуватися шляхом градієнтного спуску від вихідного сигналу до вхідного, без необхідності тренувати шари окремо.

Тут речення представлені як вектори пам'яті. Вхідні речення $x_1, .., x_n$ кодуються у вектори пам'яті розмірності d за допомогою матриці вбудовування A^k на кожному шарі k. Вони використовують або мішок слів, або кодування позицій, де порядок слів кодується у вектор пам'яті. Мішок слів - це місце, де подання речення є простим підсумовуванням усіх його векторів слів $m_i = \sum_j Ax_{ij}$,тоді як кодування позиції є $m_i = \sum_j l_j \cdot Ax_{ij}$ де $\cdot$ є елементарним множенням і l_j - вектор стовпця, де $l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$де J - кількість слів у реченні i, a d - розмір вбудовування. Показано, що ефективність роботи вища, ніж домен відповідей на питання.

Вектори пам'яті не впорядковані, але якщо використовується часове кодування, то порядок вхідних речень кодується як у вхідних, так і у вихідних поданнях наступним чином: $m_i = \sum_j Ax_{ij} + T_A(i)$ and $c_i = \sum_j Ax_{ij} + T_C(i)$ де T_A і T_C - це тимчасова інформація, засвоєна під час навчання.

Запит q також вбудовується за допомогою вбудовування матриці B з тими ж розмірами, що і A, щоб отримати внутрішній стан u. Крім того, коженx_i має відповідний вихідний вектор c_i з використанням матриці вбудовування C^k на кожному шарі k.

На кожному стрибку вектори пам'яті m створюються крапкою з вектором стану контролера u і застосовується softmax для розрахунку ваг уваги p, де $p_i = Softmax(u^T m_i)$. Потім цей результат стає зваженою сумою o вихідних векторів c_i таким, що $o = \sum_i p_i c_i$ і додається до стану контролера $u^{k+1} = u^k + o^k$. Кінцевий вихід мережі мережі обчислюється наступним чином із використанням кінцевої матриці ваги W такої, що $a = Softmax(W(o^k + u^k))$.

Деякі обмеження на A^k і C^k можна використовувати для зменшення кількості параметрів у моделі. Один набір обмежень, який можна використовувати, називається "суміжним і він є таким: $A^{k+1} = C^k$, $W^T = C^k$, $B = A^1$. Інший набір обмежень, "пошарово полягає в наступному: $A^1 = A^2 = ... = A^K$, $C^1 = C^2 = ... = C^K$ і використовувати лінійне відображення H для оновлення u таким чином, що $u^{k+1} = Hu^k + o^k$.

Sukhbaatar et al. [2015] повідомляє про меншу кількість невдалих завдань і меншу середню помилку, ніж ДКЧП та Weston et al. [2014] на 20 bAbI QA tasks.

12.3 Нейронна машина Тьюрінга

12.3.1 НМТ та огляд рекурретних нейронних мереж

Нейронна машина Тьюрінга (НМТ) - це архітектура нейронної мережі, яка складається з контролера та банку пам'яті. Мережа спроектована та забезпечена для імітації основних функцій комп'ютерів, таких як логічне управління потоком та зчитування з зовнішньої пам'яті/ запису у зовнішню пам'ять. Включення цих фундаментальних можливостей в архітектуру нейронних мереж було натхнено їх відсутністю використання в сучасному машинному навчанні. Малюнок 1 нижче показує архітектуру мережі високого рівня.

Кожна частина НМТ є диференційованою, тому вона дозволяє зворотне розмноження, і її можна навчити з градієнтним спуском. На відміну від машини Тьюрінга, де операції зчитування та запису елементів пам'яті безпосередньо звертаються до одного елемента за раз, операції зчитування та запису НМТ визначаються як "розмиті". Ці розмиті операції складають механізм, що полегшує повну диференціацію НМТ. Як отримується ця "розмитість".
Моделі, які спочатку були включені в експерименти НМТ такі, (1) НМТ з контролером прямої передачі, (2) НМТ з контролером ДКЧП і (3) стандартна мережа ДКЧП, тип рекурретних нейронних мереж, які, як відомо, є Повними Тьюрінговими Мережами. Бажана властивість контролера полягає в тому, що він повинен мати можливість вибірково прослуховувати входи та зберігати інформацію відповідно. Це досягається шляхом приєднання програмованого шлюза, тобто функція контексту, до ідеального інтеграторного вбудовування ДКЧП ($x(t+1) = x(t) + i(t)$, де i (t) - вхід в систему в момент часу t). З додаванням програмованого затвора рівняння інтегратора стає

$$x(t+1) = x(t) + g(context)i(t)$$

де останній вираз $g(context)i(t)$ має на увазі, що інтегратор може здійснювати контекстно--вибіркове зберігання інформації.

Нижче ми включили розділи для:

- Основи неврології [12.3.2], де ми пояснюємо аналогії між робочою пам'яттю людини та НМТ,
- Модулі нейронних машин Тьюрінга [12.3.3], де ми пояснюємо операції в контролері [12.3.4], пам'яті [12.3.5], механізм адресації [12.3.6], і голови читання та запису [12.3.9].

- Навчання [12.3.12] та Порівняння з іншими архітектурами [12.3.13].

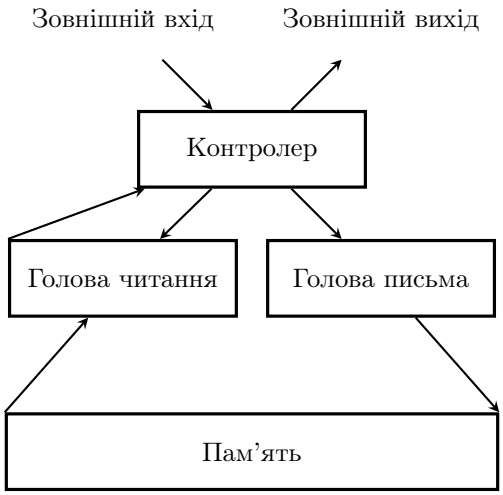


Figure 1: Нейронна архітектура машини Тьюрінга

12.3.2 Основи пізнання та нейронауки

У галузях когнітивної психології та поведінки людини термін "робоча пам'ять"вперше був використаний для визначення компонента системи обробки інформації мозку людини, який безпосередньо доступний для короткострокового усвідомлення [12.3.14], і це дозволяє зберігати та обробляти інформацію, використовуючи набір правил. З тих пір ця концепція використовується для характеристики тимчасового зберігання та когнітивної обробки інформації в мозку людини за допомогою таких завдань, що включають прості правила, такі як розуміння мови та проста арифметика. [12.3.14][12.3.14].

Ще однією важливою властивістю робочої пам'яті є те, що ресурси, доступні для цього короткочасного механізму зберігання та обробки пам'яті, визнаються обмеженими сферою психології. Ця властивість також передбачає, що робоча пам'ять та моделі, натхненні робочою пам'яттю, повинні мати механізм, який регулює „увагу". У розділі "Розвиваючі концепції робочої пам'яті"книги "Робоча пам'ять і пізнання людини"представлений всебічний та детальний аналіз основоположних та останніх робіт у цій галузі. [12.3.14].

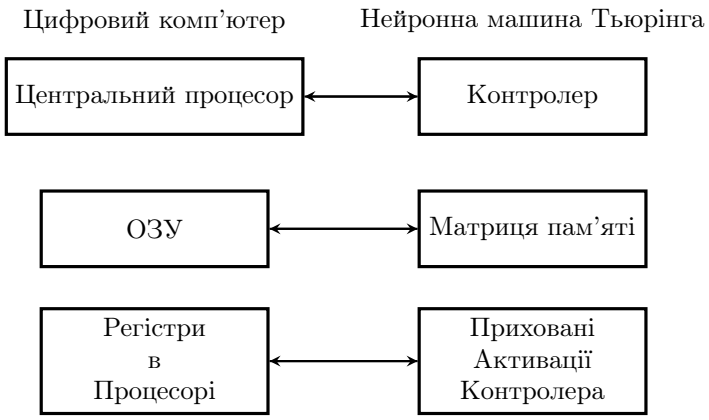
Операції зчитування та запису нейронної машини Тьюрінга регулюються процесом уваги, який буде детально описаний пізніше, і завдяки цим операціям машина нейронного Тьюрінга може вирішувати основні завдання за допомогою "швидко створюваних змінних, які визначаються як дані, що швидко прив'язані до слотів пам'яті в контексті когнітивного представлення змінних.[12.3.14]. Залучення процесу уваги до фундаментальних операцій та вирішення основних завдань шляхом застосування наближених правил до швидко створюваних змінних - дві основні властивості нейронної машини Тьюрінга(НМТ), якими вона ділиться з робочою пам'яттю[12.3.14].

12.3.3 Модулі нейронних машин Тьюрінга

12.3.4 Контролер

Контролер, який є нейронною мережею, яка може бути як довгостроковою короткочасною пам'яттю, так і нейронною мережею прямого зв'язку, виступає в ролі інтерфейсу між вхідними даними та пам'яттю.

Нижче наведена схема, що показує порівняння між цифровими комп'ютерами та нейронною машиною Тьюрінга, яка представлена авторами в оригінальній роботі:



Нейронна машина Тьюрінга вчиться управляти власною пам'яттю за допомогою адресації. Контролер зчитує вхідні дані та видає вихідні дані, необхідні підмодулям. Вихідними параметрами контролера є:

- k_t: Ключовий вектор. Порівняний з кожним вектором $M_t(i)$ матриці пам'яті M_t, при використанні контент-адресації.
- β_t: Ключова сила. Використовується для посилення або послаблення фокусу на розташуванні в пам'яті під час використання контент-адресації.
- g_t: Коефіцієнт змішування або значення інтерполяційного затвора. Зважування на основі вмісту в момент часу t інтерполюється з вагами попереднього кроку часу відповідно до цього параметра.
- s_t: Зваження зсуву. Регулює градус о повертання зважування за допомогою кругової згортки. Він просто визначає розподіл за цілочисельними значеннями зсуву.
- γ_t: Показник різкості. Цей параметр визначав ступінь посилення зважування доступу до пам'яті.

Детальні рівняння щодо використання вищезазначених параметрів при адресації наведені в Розділі 12.3.6 нижче.

12.3.5 Пам'ять

Пам'ять - це 2D-матриця M, яка доступна контролеру для читання та запису. Мережа обирає місце для читання та запису за допомогою адресації на основі вмісту та розташування. Термін "голова"використовується для опису місць читання та запису.

12.3.6 Механізм Адресації

Вагові коефіцієнти, що використовуються операціями зчитування та запису Нейронної машини Тьюрінга, виробляються двома механізмами адресації, а саме адресацією на основі контент-адресації. (12.3.7) та адресації на основі місцезнаходження. (12.3.8).

Механізм адресації на основі вмісту враховує схожість вмісту ділянок у пам'яті зі значеннями, що видаються контролером. Тобто він порівнює ключовий вектор k_t виданий контролером в момент часу t, з векторами матриці пам'яті M_t. З іншого боку, використовуючи адресацію на основі місцезнаходження, контролер може зберігати змінні, які будуть вхідними даними для будь-якої операції в різних місцях, і адресувати їх, коли це необхідно. Адресація на основі місцезнаходження використовується, коли вхідними даними для операції є довільні змінні, які потребують належного доступу до заздалегідь визначеного ідентифікатора та адреси. Будь-яка арифметична функція форми $f(x, y)$ вимагатиме адресації на основі місця розташування.

12.3.7 Адресація за схожістю вмісту

Ключовий вектор k_t виданий кожною головою, порівнюється з векторами матриці пам'яті $(M_t(i))$ за допомогою міри подібності, позначеної K[.,.]. Тут може використовуватись будь яка відповідна послідовність мір подібності, однак автори використовують косинусну подібність. Нормоване значення ваги, отримане за допомогою адресації на основі вмісту:

$$w_t^c(i) \leftarrow \frac{exp(\beta_t K[k_t, M_t(i)])}{\sum_j exp(\beta_t K[k_t, M_t(j)])}$$

де β_t є ключовою силою, яка регулює точність фокусування, k_t є ключовим вектором, M_t є матрицею пам'яті, і K є подібністю косинусів заданою за:

$$K[u, v] = \frac{\langle u, v \rangle}{||u||.||v||}$$

12.3.8 Адресація за місцем розташування

Прості ітерації або стрибки з довільним доступом, створені за допомогою обертального зсуву, забезпечують механізм адресації на основі розташування. Отриманий орієнтований ваговий коефіцієнт, який є вихідним показником механізму в момент часу t, слід інтерпретувати як суміш між ваговим коефіцієнтом, сформованим у момент часу t-1 (w_{t-1}) та вагою на основі вмісту:

$$w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1}$$

де g_t - скалярний інтерполяційний затвор, випромінюваний контролером, w_t^c - вага, що виробляється системою адресації на основі вмісту, та w_{t-1} - вага, вироблена головою на попередньому кроці часу. З цього рівняння затвора видно, що g_t має коливатися від 0 до 1, щоб мати можливість

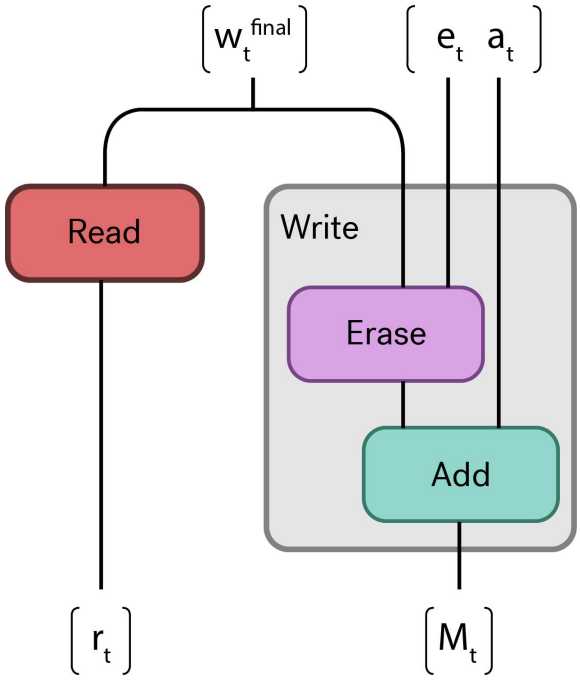
створити двоступеневу суміш між t-1 і t. Після w_t^g інтерполюється з w_t^c and w_{t-1}, зважувальний зсув s_t згенерований контролером, визначає розподіл по цілочисельних зсувах, дозволених до адресації на основі розташування. Вираз для обертання, застосований до ваги закритого типу w_t^g за ваговою зміною s_t - задано через:

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g(j) \star s_t(i-j)$$

що відповідає згортці.

Наприклад, якщо фокус уваги не змішується (тобто зважування фокусується на одному місці без будь-якого синтезу), обертання на 1 повністю змістить фокус уваги на наступне місце, а негативний зсув навпаки, у тій же величині змістить фокус на безпосереднє попереднє місце.

12.3.9 Читання та запис



12.3.10 Читання

Вказівники зчитування працюють за допомогою механізму уваги, щоб визначити, звідки читати. Нормовані зважування, сформовані механізмом адресації, можна розглядати як розподіл по N рядків матриці пам'яті. Тому на нього поширюються обмеження будь-якого розподілу ймовірностей:

$$\sum_{i \in N} w_t(i) = 1$$

$$w_t(i) \in [0,1] \; \forall i$$

Тому вектор читання, повернутий вказівником читання, може бути записаний як

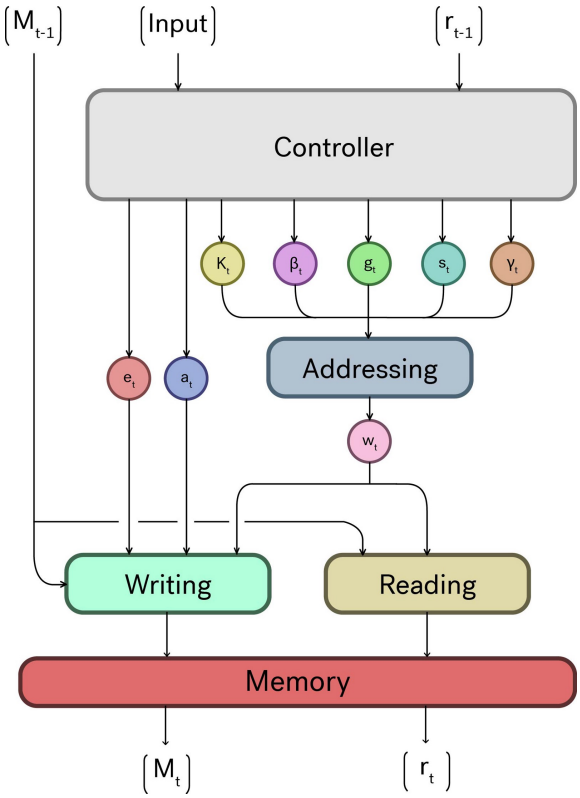
$$r_t \leftarrow \sum_i w_t(i) M_t(i)$$

Зверніть увагу, що це опукла комбінація матричних векторів пам'яті: $M_t(i)$, таким чином він диференціюється.

12.3.11 Запис

У вказівниках запису виконується два набори операцій: стирання та додавання. Операції виконуються відповідно до вектора ваги w_t, вектора стирання e_t, та вектора додавання a_t виданих вказівником запису в момент часу t. Однак час випромінювання не слід плутати з часом, коли вектори фактично додаються до матричних векторів пам'яті. Процес відбувається наступним чином;

- Вектор матриці пам'яті з попереднього кроку часу (M_{t-1}) оновлєються відповідно до рівняння:
$$\tilde{M}_t \leftarrow M_{t-1}(i)[1 - w_t(i)e_t]$$
- Після завершення етапу стирання за допомогою w_t та e_t, вказівник запису також генерує вектор додавання a_t,
- Оновлення кроку додавання виконується для попереднього оновлення:
$$M_t(i) \leftarrow \tilde{M}_t(i) + w_t(i)a_t$$



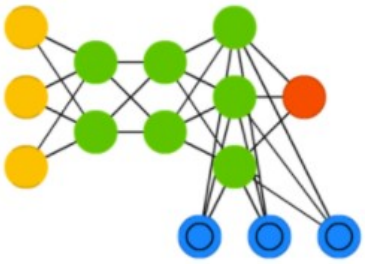
12.3.12 Навчання

НМТ навчений за замовчуванням за допомогою зворотного пропагування та стохастного градієнтного спуску. Але градієнт не розраховується по відношенню до конкретних індексів, і контролер виробляє зважування над місцями пам'яті, щоб зробити їх різними. Це викликано тим, що ми не можемо взяти градієнт конкретного індексу в матриці, таким чином контролер взаємодіє з цими індексами стовпців, що виробляють зважування і роблять їх різними.

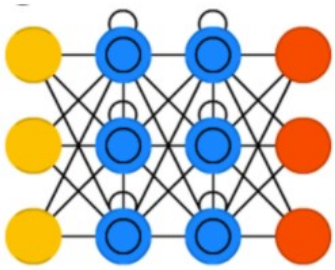
12.3.13 Порівняння з іншими архітектурами

У статті також порівнюється архітектура НМТ з різними контролерами та ванільним МДКП, щоб визначити, чи має матриця зовнішніх даних призводить до підвищення продуктивності.

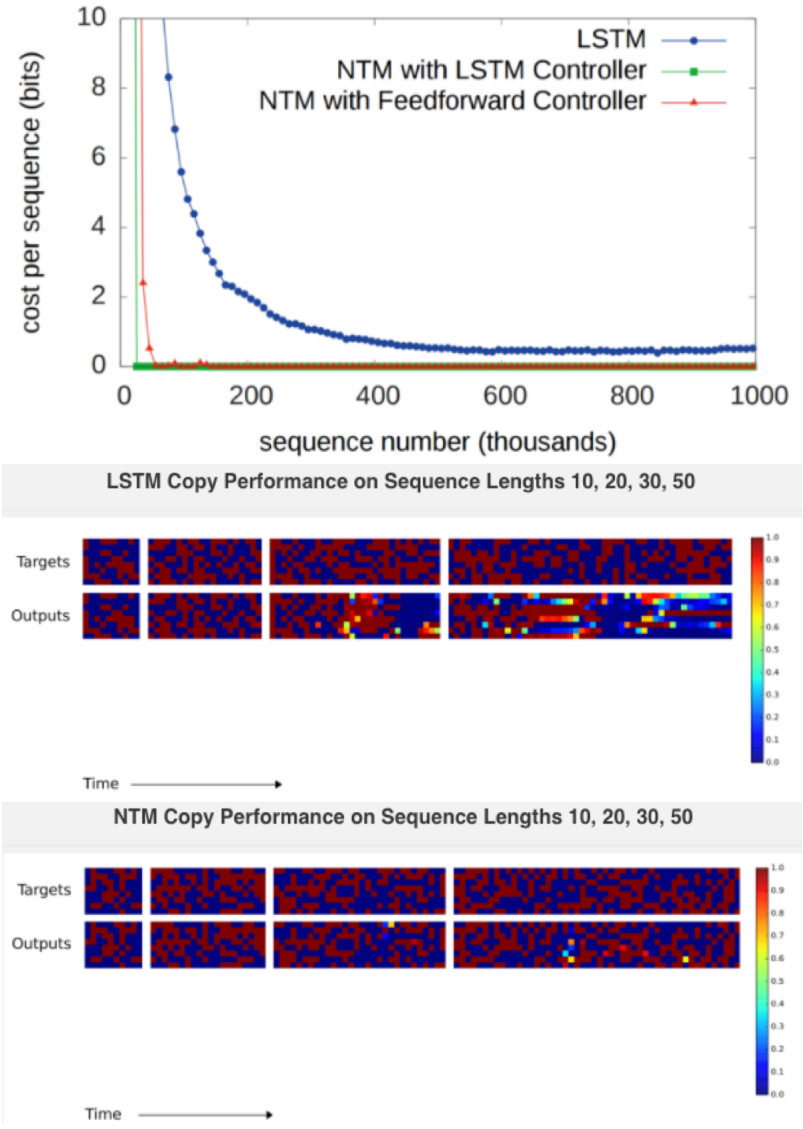
Нейронні машини Тьюринга (НМТ) можна розглядати як абстрактну модель МДКП і спробу показати, що насправді відбувається всередині нейронної мережі. Осередок пам'яті не поміщен у нейрон, а розміщен окремо з метою об'єднати ефективність звичайного сховища даних і міць нейронної мережі. Власне, тому такі мережі і називаються машинами Тьюринга - в силу здатності читати і записувати дані і змінювати стан в залежності від прочитаного вони є Тьюринг-повними.



Мережі з довгої короткостроковою пам'яттю (МДКП) намагаються вирішити проблему втрати інформації, використовуючи фільтри і явно задану клітку пам'яті. У кожного нейрона є клітина пам'яті і три фільтра: вхідний, вихідний і забуваючий. Метою цих фільтрів є захист інформації. Вхідний фільтр визначає, скільки інформації з попереднього шару буде зберігатися в клітці. Вихідний фільтр визначає, скільки інформації отримають наступні шари. Ну а забуваючий фільтр, яким би дивним не здавався, також виконує корисну функцію: наприклад, відеодзвінки вивчає книгу і переходить на нову главу, якісь символи зі старої можна забути. Такі мережі здатні навчитися створювати складні структури, наприклад, писати як Шекспір або складати просту музику, але і ресурсів вони споживають чимало.

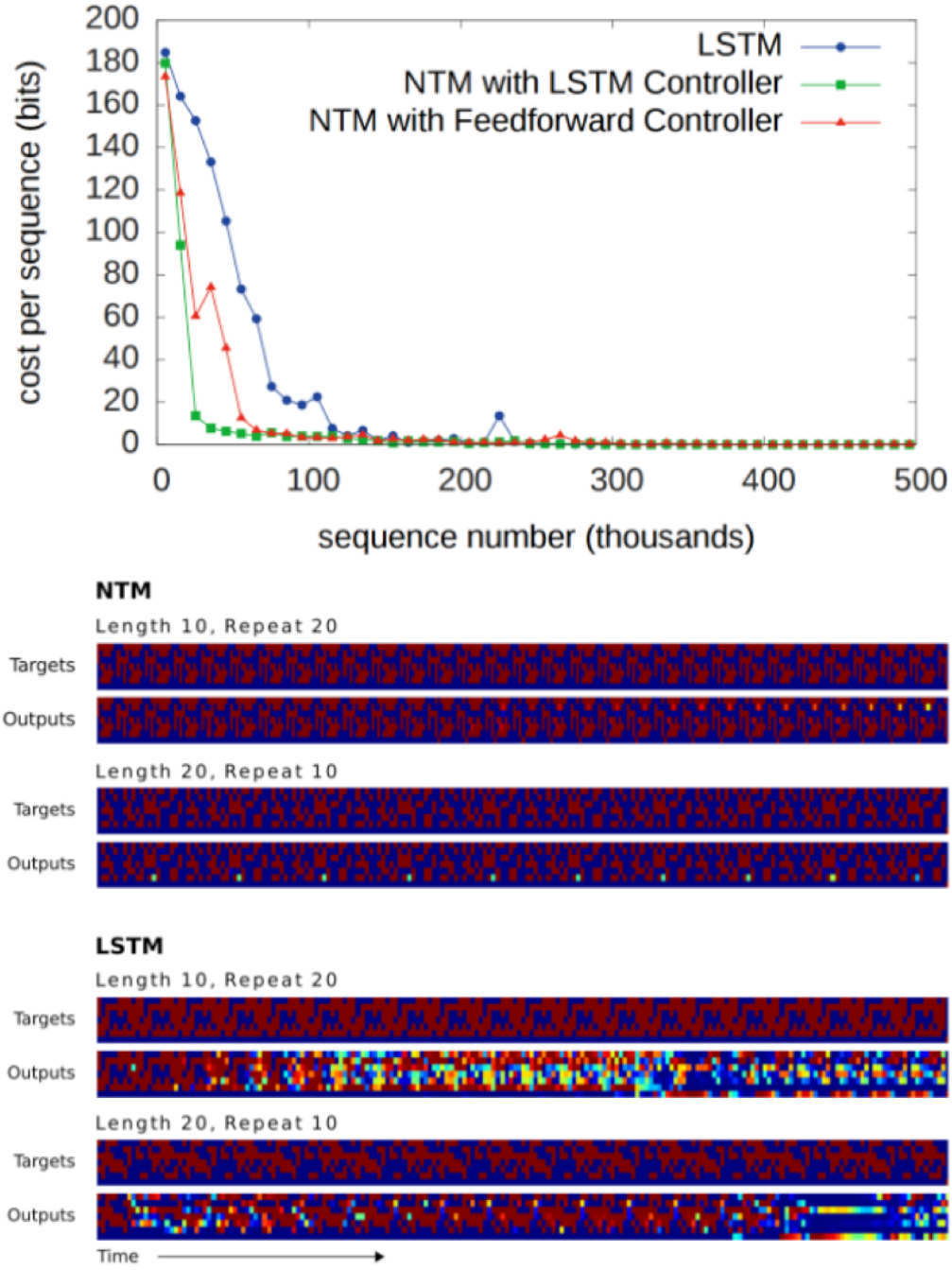


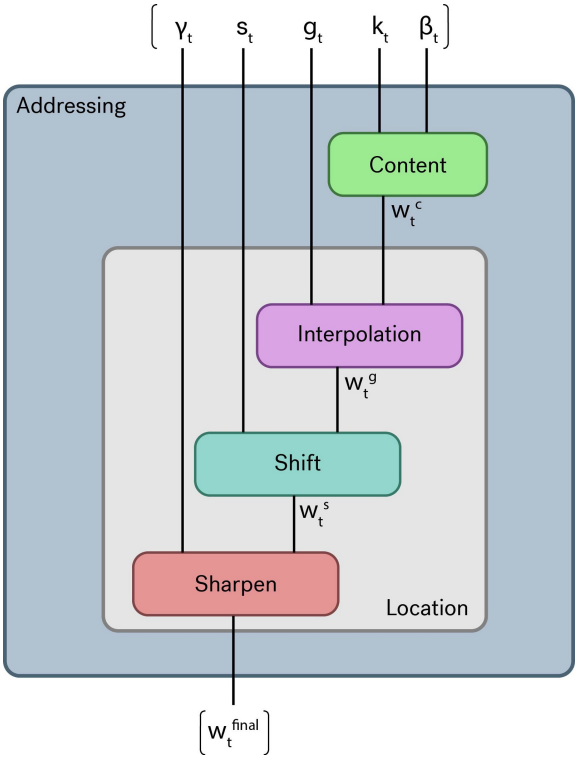
Мережам була надана послідовність випадкових восьми біт векторів, а потім буде запропоновано скопіювати вхідну послідовність після роздільника. Результати відображаються нижче, де вартість за послідовність відноситься до кількості бітів, які були неправильно скопійовані, і яскраво забарвлені біти неправильно скопійовані біти:



Ми чітко бачимо, що НМТ виробляють набагато менше помилок, коли використовуються довші послідовності.

Крім того,НМТ були протестовані, щоб побачити, чи можуть вони дізнатися вкладені функції (для циклу), де послідовність повторюється протягом ряду разів, який визначається скалярний, який передається НМТ. У наведених нижче графіках ми можемо побачити, як конфігурації НМТ перевершують ванільний МДКП:


cost per sequence (bits)
200
180
160
140
120
100
80
60
40
20
0
0
100
200
300
400
500
sequence number (thousands)
LSTM
NTM with LSTM Controller
NTM with Feedforward Controller
NTM
Length 10, Repeat 20
Targets
Outputs
Length 20, Repeat 10
Targets
Outputs
LSTM
Length 10, Repeat 20
Targets
Outputs
Length 20, Repeat 10
Targets
Outputs
Time



Однак, НМТ не має механізму, щоб запобігти перекриття і звільнити пам'яті в блоках. НМТ з довгостроковою короткостроковою пам'яттю (МДКП) мережевий контролер може викликати прості алгоритми, такі як копіювання, сортування.

Різні нейронні комп'ютери - це переростання нейронних машин Turing, з механізмами уваги, які контролюють, де активна пам'ять, і підвищення продуктивності.

12.3.14 References

[2] Miller GA, Galanter E, Pribram KH. Plans and the structure of behavior. New York: Holt, Rinehart and Winston, Inc; 1960.
[3] Baddeley, A., Graham,H. 1974, Working Memory, *Psychology of Learning and Motivation*, p. 47-89.
[4] Baddeley, A., 1996, The Fractionation of Working Memory, PNAS, p-93.
[5] Richardson, J. et. al. 1996, Working Memory and Human Cognition, *Oxford Scholarship On-line*
[6] Hadley,R., 2009, The Problem of Rapid Variable Creation, *Neural Computation*, 21-2-510.
[7] Graves,A. et al., 2014, Neural Turing Machines, *Google DeepMind*

13 Attention and Augmented Recurrent Neural Network

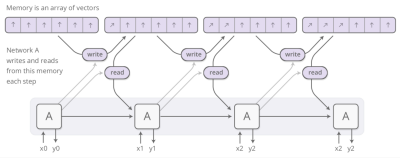
13.1 Введення

Завдання навчання на послідовностях, такі як машинний переклад, обробка сигналів та мовні моделі, є популярною областю досліджень у сфері рекуррентних нейронних мереж (RNN). Вони можуть використовуватися розкладання послідовностей для розуміння мови на вищому рівні, для анотування послідовностей та для створення нових послідовностей.

Базовий блок RNN страждає при роботі із довгими послідовностями, але такі моделі, як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit) виявилися більш ефективними при навчанні на подібних завданнях. У цьому розділі ми спеціально зупинимось на інших перспективних розширеннях RNN, Нейронних машинах Тюрінга та інтерфейсах уваги.

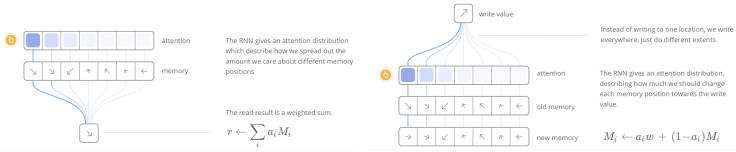
13.2 Нейронні машини Тюрінга (NTM)

Нейронні машини Тюрінга поєднують RNN із зовнішнім банком пам'яті. Пам'ять - це масив векторів, оскільки вектори є звичною мовою нейронних мереж.



Завдання читання та писання полягає в тому, щоб зробити їх диференційованими щодо місця, з якого ми читаємо чи пишемо, щоб ми могли навчитися, де читати та писати. Незважаючи на те, що адреси пам'яті здаються принципово дискретними, NTM вирішують цю проблему, читаючи та записуючи скрізь на кожному кроці, але в різній мірі.

- При читанні, RNN надає "розподіл уваги що описує, як дана величина поширюється по різних комірках пам'яті замість того, щоб вказати одну. Отже, результатом операції зчитування є зважена сума.
- Подібно до читання, ми використовуємо розподіл уваги, щоб описати, скільки ми пишемо в кожному місці. Ми робимо це за рахунок того, що нове значення в позиції пам'яті є опуклою комбінацією старого вмісту пам'яті та нового значення, а положення між ними визначається вагою уваги.



Поєднання двох різних методів - уваги на основі контенту та уваги на основі розташування - використовується для того, щоб NTM вирішували, на яких комірках в пам'яті слід зосередити свою увагу. Увага на основі контенту дозволяє NTM шукати у своїй пам'яті та зосереджуватись на місцях, які відповідають тому, що вони шукають, тоді як увага на основі розташування дозволяє відносно рухатись у пам'яті, дозволяючи NTM циклічно працювати.

Здатність читати та писати дозволяє NTM виходити за межі нейронних мереж у деяких простих завданнях. Наприклад, вони можуть навчитися імітувати таблицю пошуку; розуміти,

що робить алгоритм, коли вивчає довгі послідовності. Однак не на таких завданнях, як додавання або множення чисел.

13.3 Інтерфейси Уваги

Використовуючи увагу, нейронні мережі фокусуються на частині підмножини інформації. Наприклад, RNN може брати участь у виведенні іншій RNN, яка на кожному моменті часу фокусується на різних позиціях в іншій RNN. Подібно до Нейронних машин Тюрінга, відвідуюча RNN використовує увагу на основі контенту. Вона генерує запит, який описує обрану частину. На основі скалярного добутку кожного елемента та запиту була обчислена оцінка, яка описувала, наскільки вона відповідає запиту. Крім того, розподіл уваги буде обчислений функцією softmax, застосованою до оцінок.

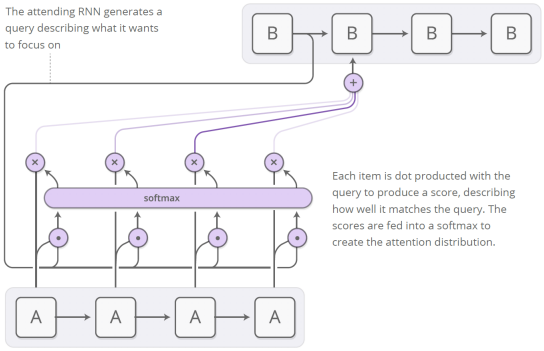


Рис. 69: Attention in RNNs

Одне із основних застосовувань уваги між RNN є переклад. У традиційному sequence-to-sequence моделюванні вхідні дані відображаються в єдиний вектор і перенаправляються назад як вихідні дані. Цього складного та трудомісткого процесу уникають за допомогою уваги. Одна RNN обробляє вхідні дані для передачі інформації про кожне слово, а інша RNN генерує вихідні дані для зосередження на відповідних словах.

Інші застосування включають розпізнавання голосу та парсинг тексту. При розпізнаванні голосу одна RNN обробляє аудіо, а інша RNN переглядає поверхнево аудіо, фокусуючись на відповідних частинах, при перетворенні їх у слова. Під час парсингу тексту модель генерує дерево синтаксичного парсингу, поглядаючи на слова. Розмовна модель зосереджується на попередніх частинах розмови при формуванні відповіді.

Окрім згаданих вище застосувань, увага також корисна на сполученні між згортковою нейронною мережею та RNN. Увага дозволяє моделі RNN сфокусуватись на різному положенні у зображення на кожному кроці. Наприклад, у підписуванні зображень, процесі згортки та вилученні високорівневих ознак у зображенні. У процесі формування підпису зображення модель RNN зосередилася на інтерпретації результатів згорткових мереж відповідних частин зображення.

Як підсумок, слід зазначити, що інтерфейси уваги широко використовуються в різних сферах завдяки своїй загальним та потужним можливостям.

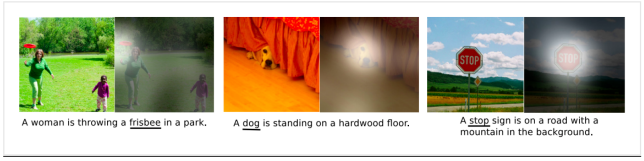


Figure from Xu, *et al.*, 2015

Рис. 70: Інтерфейси Уваги у Зображеннях

13.4 Проблематика

Основними недоліками моделей на основі уваги, є те, що кожен крок навчання повинен бути зосередженим, що лінійно збільшує обчислювальні витрати, оскільки збільшується обсяг пам'яті в Нейронній машині Тюрінга. Потенційним вирішенням цього недоліку є використання розрідженої уваги, так що береться лише певна частина пам'яті. Однак, це все ще є складною задачею, оскільки такий вид уваги залежить від вмісту пам'яті і наївно змушує модель дивитися на кожну комірку. Наразі, деякі спроби в напрямку вирішення були виконанні, але для кінцевого вирішення цих проблем потрібно провести більше досліджень.

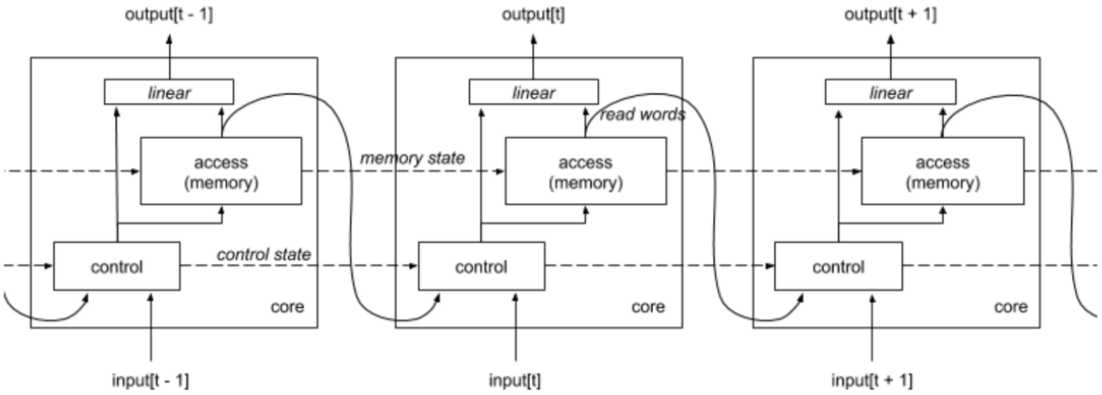
13.5 Диференційовані нейронні комп'ютери

Ця архітектура була запропонована Грейвсом та ін. у 2016 р. у статті під назвою "Гібридні обчислення за допомогою нейронної мережі з динамічною зовнішньою пам'яттю". Її можна розглядати як тип RNN з авто-асоціативною пам'яттю. Нейронні мережі неявно зберігають знання, отримані в результаті навчання у вагах, і інформація піддається деградації, оскільки в систему додається більше інформації. LSTM намагаються уповільнити цю короткочасну деградацію інформації, використовуючи вентилі, але все ще не дуже успішно. DNC є диференційованими та наскрізними, тому їх можна навчити за допомогою SGD. Мережа також була протестована на наборі даних bAbI від Facebook. RNN та її різновиди можуть обробляти послідовності, однак у задачах в області комп'ютерних наук графові структуровані дані є більш поширеними.

DNC можна розглядати як розширення Нейронних машин Тюрінга (NTM). DNC мають механізми пам'яті на основі уваги, які контролюють, де зберігається пам'ять, і тимчасову увагу, яка реєструє порядок подій. На кожному кроці часу пам'ять має доступ до стану мережі, що містить поточний контент пам'яті. На кожному кроці часу беруться вхідні дані, переглядається матриця пам'яті із корисною інформацією, що там зберігається, і записується в дану матрицю для подальшого використання. Виділення та вивільнення пам'яті здійснюється динамічно. Крім того, обробляється тимчасовий зв'язок між комірками пам'яті.

DNC продемонстрували здатність обробляти такі прості завдання, як копіювання послідовностей, які могли б бути виконані за допомогою простої комп'ютерної програми. Різниця полягає в тому, що DNC можна натренувати окремо для кожної проблеми і навчитися визначати власні символічні міркування. Наприклад, DNC може навчитись вчитися орієнтуванню в транзитних системах, і дана робота показує успішні результати для лондонського метро. Нейронну мережу без пам'яті потрібно навчити для кожної транзитної системи, проте для DNC цього не потрібно.

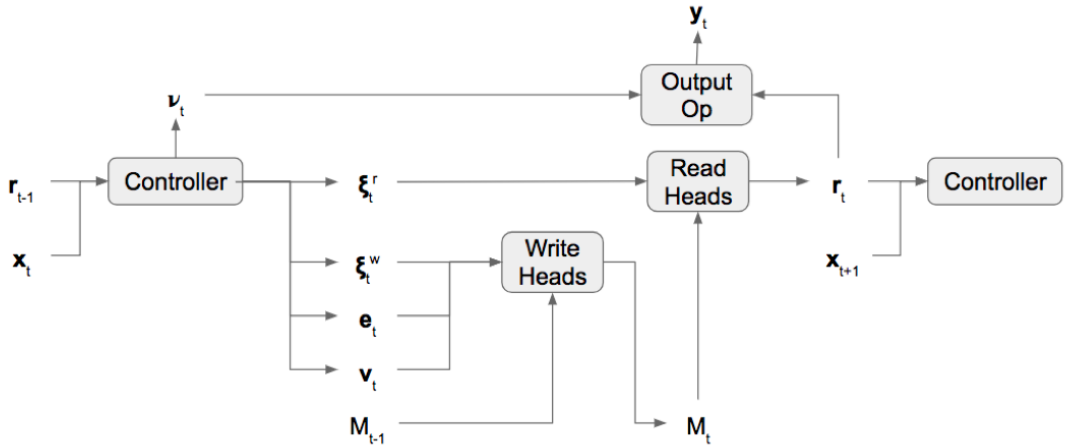
Загальна архітектура DNC:



Подібність до LSTM

Архітектура DNC має багато подібностей до архітектури LSTM. Забувальні та вхідні вентилі в архітектурі LSTM схожі на операцію запису в DNC, де операція забуття аналогічна запису нулів в пам'ять. Вихідний вентиль подібний до операції зчитування. У LSTM стан стирається з кожним кроком, що спричиняє неконтрольоване "забування однак у DNC пам'ять відокремлена від мережі, тому вона не страждає від цього.

Один крок у DNC



Контролер

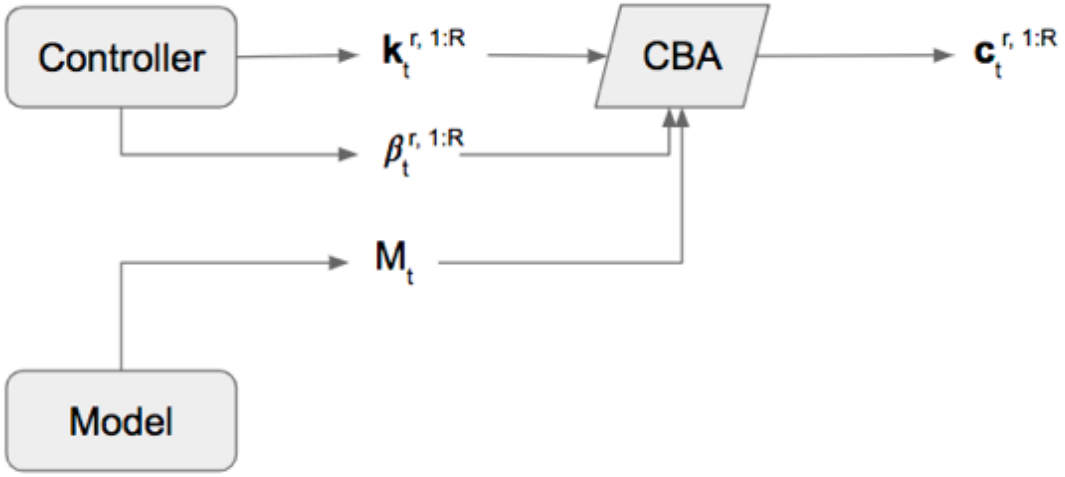
Контролер, як правило, являє собою повнозв'язну або рекурентну нейронну мережу. Однак контролером може бути все, що є диференційованим. Його назва походить від того, що він контролює доступ до пам'яті.
Входи контролеру - це входи в DNC, а зчитані дані з пам'яті та виведені з попереднього кроку, об'єднуються з ним. Це відображається у вектор. Вихідні дані - це список змінних, які визначають, яка дія над пам'яттю (читання або запис) буде на наступному кроці. Вихідний вектор подається на наступний крок без будь-яких змін і використовується як вхідний сигнал на наступному кроці. Вектор зчитування використовується головкою зчитування для перенаправлення елементу в матриці пам'яті. Вектор стирання використовується для вибору та стирання елемента з матриці пам'яті. Вектор запису - це вектор, який використовується для запису (а не перезапису) деякої інформації в пам'ять.
Потім контролер взаємодіє з головками читання та запису, щоб записувати або зчитувати елементи з пам'яті. Ці взаємодії вивчаються за допомогою певного методу градієнтного спуску, оскільки він диференційований.

Увага

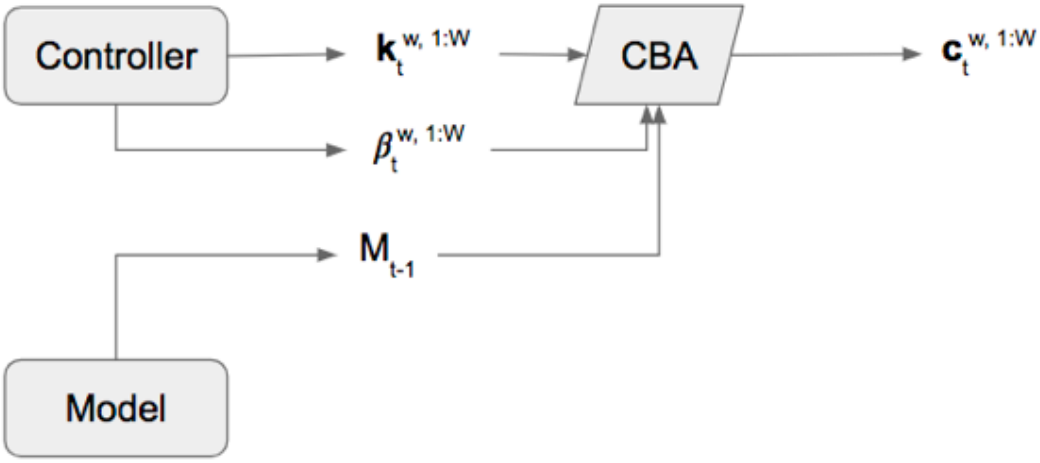
DNC використовує вибіркову увагу у своїх процесах читання та запису. Виходи контролера перетворюються на розподіл і параметризуються за допомогою моделі уваги. Цей розподіл складається з ваг над рядками (позиціями) в матриці пам'яті.
Механізм уваги складається з трьох компонентів, заснованих на контенті, виділенні пам'яті та часовому порядку операцій, які здійснюються над матрицею пам'яті. Контролер використовує ці три методи уваги з використанням інтерполяції на основі скалярного перетворення. Саме ці диференційовані механізми уваги визначають розподіл по рядках матриці пам'яті. Ключовий вектор отримується з контролера і порівнюється з кожним рядком пам'яті певною мірою подібності, а саме косинусною відстанню. Потім значення подібності нормується за допомогою softmax.

Увага на основі контенту - Content Based Attention (CBA)

Ваги після softmax формують метод CBA. CBA використовує ключові вихідні вектори з контролера, ваги після softmax та матрицю пам'яті, щоб отримати результат того, наскільки схожі вихідні вектори до рядків матриці пам'яті.
Операція читання:



Операція запису:



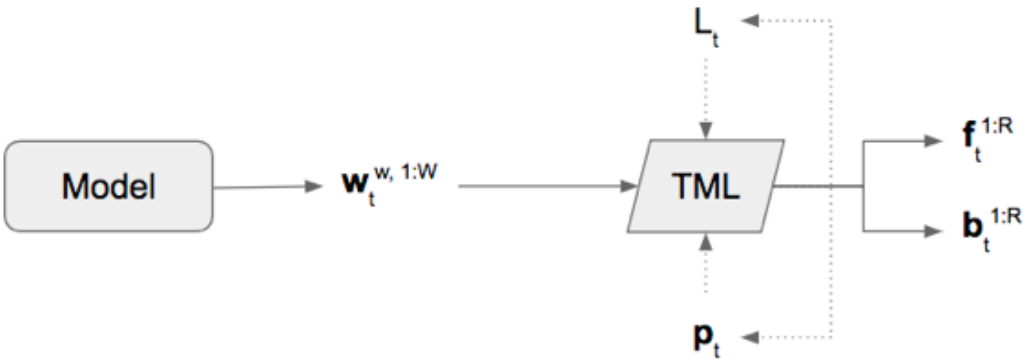
Увага на основі виділення пам'яті

Ця увага відстежує доступність пам'яті, таким чином відслідковуючи, в які комірки пам'яті щось записано, а які - вільні.

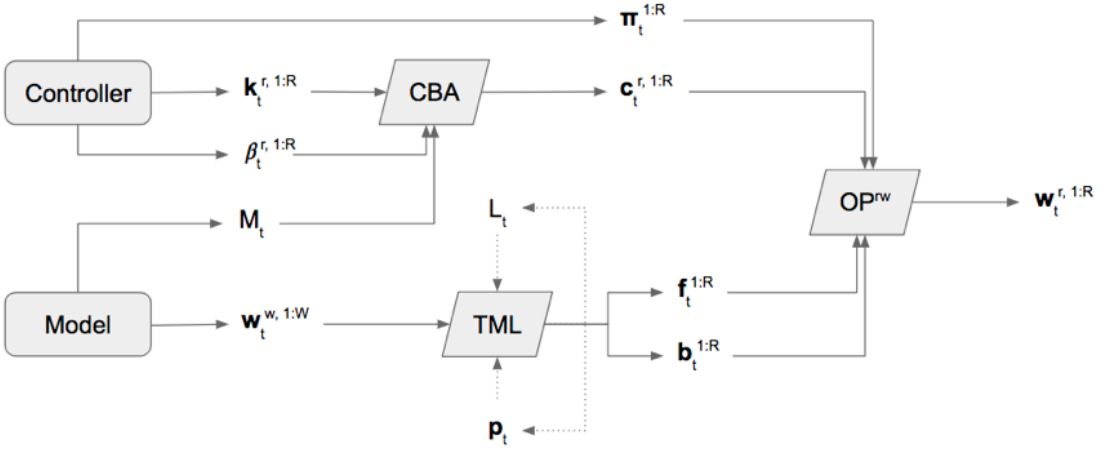
Лінкування тимчасової пам'ятті - Temporal Memory Linkage (TML)

Це лінкування відстежує порядок запису в пам'ять. Генерує два результати: прямого та зворотного зважування. Ці два зважування у поєднанні з результатом CBA формують зважування для запису. Модель Маркова першого порядку L використовується для відстеження ступеня зв'язності

наступних записів між кожною позицією. На практиці модель L є розрідженою матрицею для досягнення більшої ефективності.

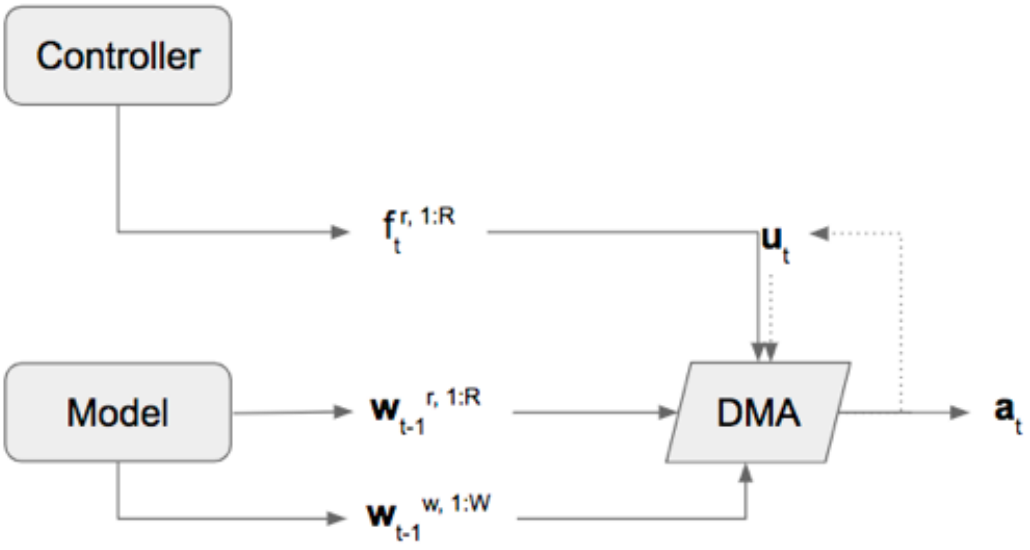


Ці три результати використовуються для отримання зважувань для читання:

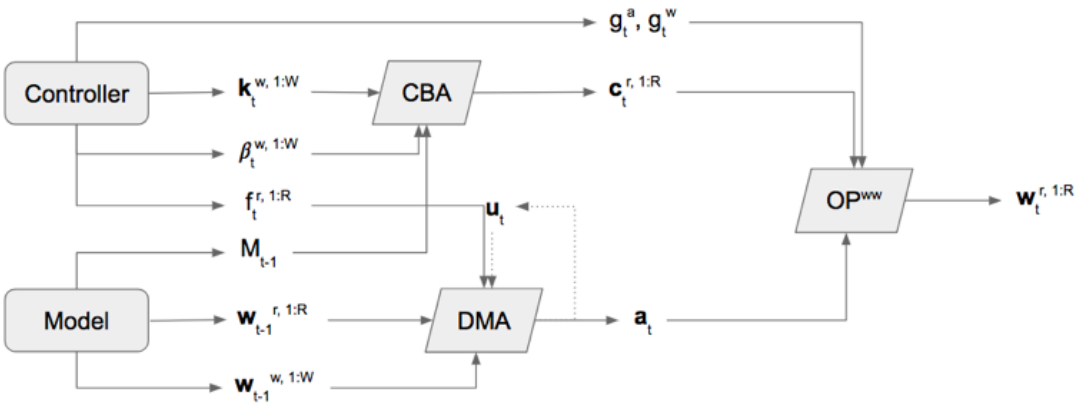


Динамічне виділення пам'яті - Dynamic Memory Allocation (DMA)

DMA генерує зважування розподілу. Він складається зі зв'язного списку доступних комірок у пам'яті та керується додаванням та видаленням адрес з нього. На кожному кроці міра використання кожної комірки пам'яті визначається вектором використання u_t. Метод формує вектор a_t, що визначає, чи можна відкинути чи залишити адреси комірок пам'яті.

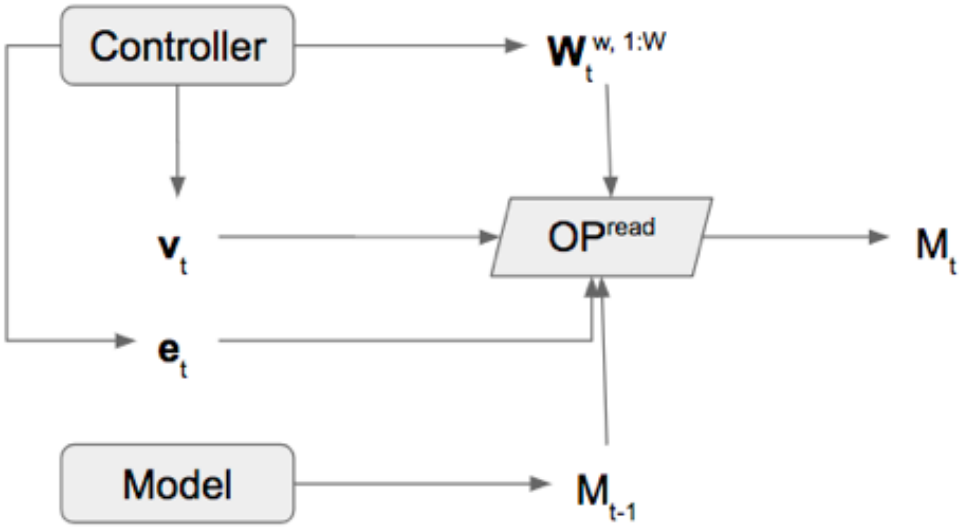


Два вихідні вектори з DMA та CBA використовуються для побудови зважувань для запису:



Пам'ять

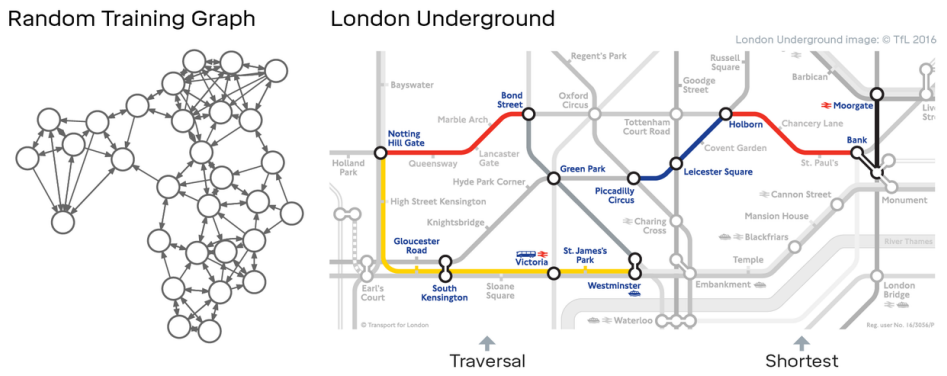
Контролер обирає: чи слід щось фіксувати в пам'яті, перезаписувати чи взагалі не фіксувати зміни. Пам'ять оновлюється наступним чином:



На кожному кроці записується та зчитується єдина комірка в пам'яті, і фокус змінюється від кроку до кроку.

Приклад з статті - Лондонське метро

Архітектура DNC була навчена на безлічі випадково згенеровано графів, і коли її перевірили на орієнтуванні в лондонському метро, вона показала 98,8 відсоткову точність порівняно з 37 відсотковою точністю звичайної нейронної мережі, яка була навчена приблизно на двох мільйонах прикладів.



Приклад з статті - Сімейство дерев QnA

Нижче наведено відео від Google DeepMind, яке демонструє здатність DNC вирішувати задачі питання - відповідь на графі генеалогічного дерева:
Задачі генеалогічних дерев у DNC

Порівняння DNC та NTM

- Мають однаковий спосіб генерації зважувань для читання та запису.
- DNC виділяє та звільняє пам'ять, використовуючи DMA, а NTM - ні.
- DNC зберігає тимчасовий лінкування між комірками пам'яті через модуль TML, тоді як NTM - ні, отже, не зберігає порядок запису рядків в пам'ять.
- DNC включає вектор зчитування на кожному кроці для отримання вихідних даних, однак NTM цього не робить, принаймні не так явно, як у DNC.
- NTM можуть отримувати інформацію з матриці пам'яті у відповідності до її індексу, але не в тому порядку, в якому вони записували. DNC здатні на це, що й необхідно для багатьох завдань.

13.6 Мережі рекурентних сутностей

13.6.1 Вступ

Мережі рекурентних сутностей (EntNet) - це модель, запропонована для вирішення задач відповідь-питання. Проблема пошуку відповіді вирішується спочатку надаючи контекст (абзац або кілька речень) моделі. Ознайомившись з усім контекстом, модель отримує запитання, пов'язане з поточним контекстом, і, як очікується, вона дасть правильну відповідь. EntNet - це нова сучасна модель завдань bAbI, яка стала еталоном для оцінки нейронних мереж з доповненою пам'яттю.
EntNet оснащена LSTM, яка допомагає підтримувати та оновлювати свій стан після отримання нових даних. Модель зберігає фіксовану кількість комірок пам'яті, і кожна комірка пам'яті містить два елементи: ключ w_j, і контент h_j. Кожна комірка пам'яті також має свої окремі вентилі, які контролюють, скільки інформації оновлювати та яким чином, враховуючи нові дані.
Таким чином, модель можна розглядати як набір RNN, де кожну можна вважати пов'язаною із певною сутностю в контексті через вагу w_j. Прихований стан h_j в комірці пам'яті записує або запам'ятовує всю інформацію, пов'язану з конкретною сутністю. Прихований стан оновлюється лише тоді, коли отримана нова інформація пов'язана з ключем або пов'язана з інформацією, записаною у прихованому стані, інакше вона залишається незмінною.

13.6.2 Архітектура моделі

Модель складається з трьох частин: вхідного кодера, динамічної пам'яті та вихідного рівня. Він розроблений у якості відповідей на питання на короткі історії, де вхідними даними для мережі є послідовності слів. Архітектура моделі виглядає наступним чином:

- Вхідний кодер: Вхід на кроці t - це послідовність слів із відповідними ембедингамислів $e_1, e_2, ...e_k$. Векторне представлення вхідних даних:

  $$s_t = \sum f_i \odot e_i$$

  Один і той же набір $f_1, f_2...f_k$ використовується на кожному кроці і навчається разом із іншими параметрами моделі.

- Динамічна пам'ять: Це керована рекурентна мережа з (частково) структурованою блоковою схемою зв'язки вагів. Приховані стани мереж поділені на блоки $h_1, h_2...h_m$, і повністю прихований стан - це поєднання блоків h_j. На кожному кроці t, h_j оновлюється наступним чином:

  $$g_j \leftarrow \sigma(s_t^T h_j + s_t^T w_j)$$
  $$\hat{h_j} \leftarrow \phi(U h_j + V w_j + W s_j)$$
  $$h_j \leftarrow h_j + g_j \odot \hat{h_j}$$
  $$h_j \leftarrow \frac{h_j}{\|h_j\|}$$

  Де σ - це функція сигмоїда, U,V,W параметри, що піддаються навчанню, спільні для всіх блоків. ϕ можу бути будь-якою функцією активації, у даному випадку ReLU. g_j керуюча функція, яка визначає скільки пам'яті необхідно оновити. $\hat{h_j}$ це потенційне значення пам'яті цього блока.

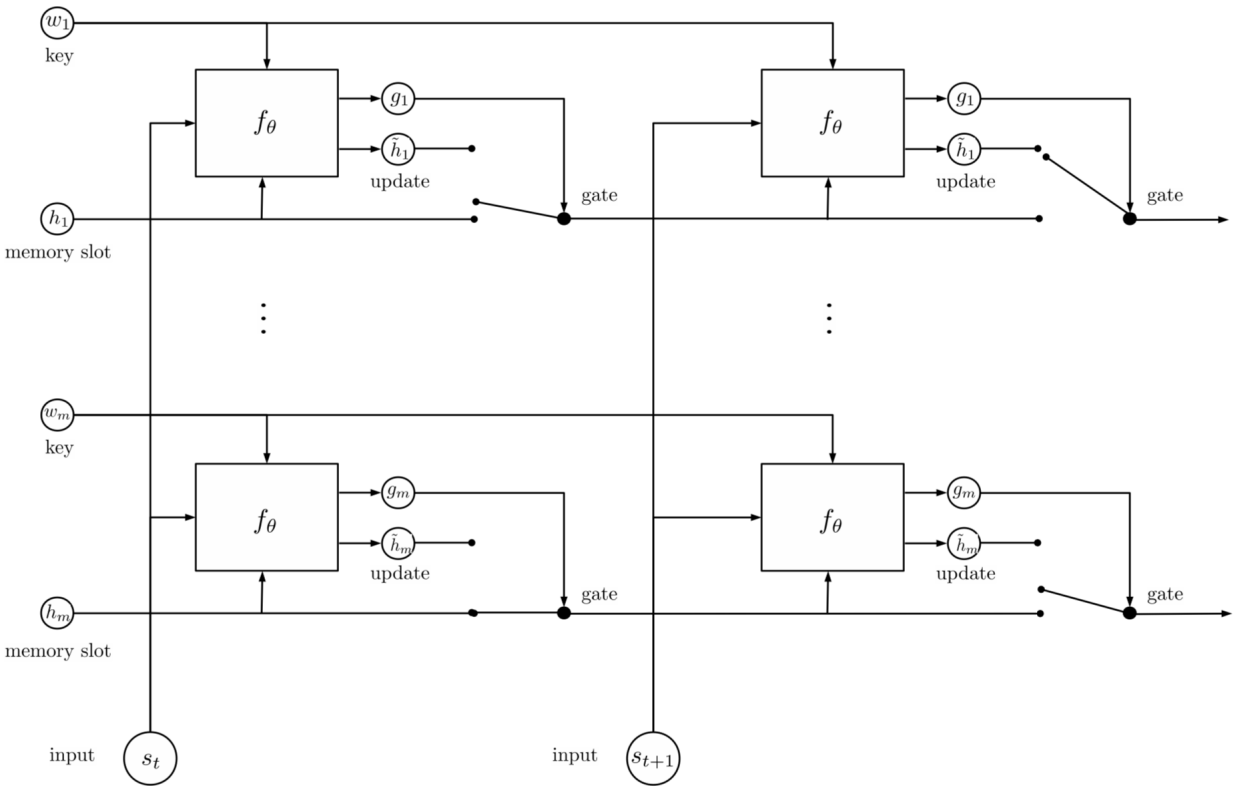


Рис. 71: Архітектура EntNet

- Вихідний шар: Щоразу, коли модель генерує результат, він представляється вектором запиту q. Результат обчислюється наступним чином:

$$p_j = Softmax(q^T h_j)$$

$$u = \sum_j p_j h_j$$

$$y = R\phi(q + Hu)$$

Параметри H і R - додаткові параметри, які можна навчати. Якщо комірки пам'яті відповідають певним словам, які формують відповідь, p може бути представлено як розподіл потенційний відповідей і може бути використано для формування безпосередньо прогнозу або передано до loss функції, відкидаючи потребу двох останніх кроків.

13.6.3 Мотивуючі приклади і результати

Розглянемо, як модель читає такі два речення: Мері взяла м'яч. Мері пішла в сад. Модель матиме дві комірки пам'яті, значення яких відповідно "Мері"та "м'яч". В першому реченні

згадувалась Мері, тому комірка пам'яті, що відповідає "Мері буде оновлюватися, а інформація про те, що Мері несе "м'яч буде записана в прихований стан комірки "Мері". Те саме для комірки пам'яті, що відповідає значенню "м'яч". Його прихований стан також буде оновлено після перегляду першого речення, і прихований стан повинен записати інформацію про те, що "Мері"несе "м'яч".
Побачивши друге речення, комірка "Мері"повинна оновитися та включити інформацію про те, що "Мері"знаходиться в саду. Цікаво, що комірка пам'яті "м'яч"також буде оновлюватися, оскільки її прихований стан містив інформацію для "Мері а модель зможе виявляти поняття "м'яч"пов'язаним з "Мері і тому значення "м'яч"також буде оновлено.
З цього прикладу ми бачимо, що модель перспективна в тому, що вона може робити узагальнення та догадки, виходячи із обмеженого обсягу інформації. Навчена на задачах bAbI з використанням 10 тисяч навчаних прикладів, модель є першим рішенням всіх завдань bAbI. Для покращення ефективності моделі зроблено більше допрацювань, щоб таку саму якість можна було досягти за меншу кількість навчальних прикладів.

14 Автоенкодер

Автоенкодер - це штучна нейронна мережа, яка використовується для безконтрольного вивчення ефективних кодувань, і вона вміє копіювати свої вхідні дані у вихідні дані. Зазвичай автоенкодер використовується для

- зменшення розмірності
- вивчення особливостей

Останнім часом автоенкодери застосовуються для генеративного моделювання на основі теоретичних зв'язків між автоенкодерами та прихованими змінними моделями.

Структура

В архітектурному відношенні найпростіша форма автоенкодера - це пряма, не повторювана нейронна мережа - що має вхідний рівень, вихідний шар та один або кілька прихованих шарів, що їх з'єднують. Крім того, вихідний рівень має таку ж кількість вузлів, що і вхідний рівень, метою якого є реконструкція вхідних даних моделей.

Як правило, автокодер складається з двох частин, енкодера та декодера. Вхідний простір становить $\mathcal{X}$, і відображення для двох частин:

- енкодер, $f : \mathcal{X} \to \mathcal{F}$
- декодер, $g : \mathcal{F} \to \mathcal{X}$

Вихідний результат $g(f(x))$ для вводу x, а мета якого знайти g і f які мінімізують різницю між входом і виходом, наприклад $\min_{g,f} \|g(f(x)) - x\|_2$.

Як показано на малюнку 39 z зазвичай називають "кодом прихованим поданням. Як правило, z має меншу розмірність, ніж x, і, отже, є стислим поданням вхідних x. Якщо є

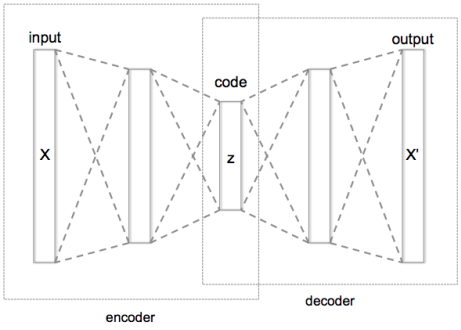


Рис. 72: Автоенкодер

лише один прихований шар, $z = f(x) = \ sigma(Wx + b)$, де $\ sigma$ - це функція активації, а W - вагова матриця (b - вектор зміщення). І $x' = g(z) = \ sigma'(W'z + b')$.

На основі базового моделювання автокодерів існує багато вдосконалених методів вивчення кращих уявлень.

14.1 Затихаючий автоенкодер

Затихаючі автоенкодери фокусуються на надійних представленнях входів. Окрім вилучення функцій, корисних для представлення розподілу вхідних даних, автоматичні кодектори шуму повинні відключати вхідні дані. Іншими словами, хорошим поданням для шумозаглушення автокодерів є те, що його можна надійно отримати з пошкодженого вводу і яке буде корисним для відновлення відповідного чистого вводу.

Щоб дезонізувати вхідні дані, функція втрат у дезоанізуючих автокодерах повинна трохи змінитися. Чисті дані складають x, а вхідні дані - x', що пошкоджено. Результат - $\ widehatx$, а функція втрат $l(x, \ widehatx)$ замість $l(x', \ widehatx)$.

Тут ref denoisingAE є прикладом автоматичного кодування denoising cite Vincent. Зображення цифри 9, x', звучить, а c - це вивчене подання з x'. Завдання полягає в мінімізації різниці між x і $\ widehatx$.

14.2 Розріджений автоенкодер

Розріджений автоенкодер використовується для створення розрідженого представлення ознак входів, щоб їх можна було використовувати для попередньої обробки або розробки елементів для завдань класифікації. Ідея полягає в тому, щоб накласти обмеження на ймовірність того, що прихований нейрон може бути активований. Наприклад, якщо необхідна розрідженість становить 5%, кожен прихований нейрон може активуватися лише 5% часу. За такої розрідженої установки функції, як очікується, представлятимуть функції низького рівня лише невелику підгрупу спільного використання даних. Таким чином, це може допомогти у класифікації.

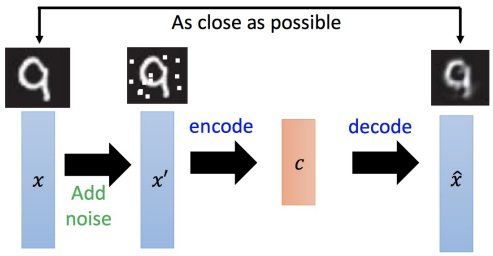


Рис. 73: Затихаючий автоенкодер

Є два шляхи досягнення розрідженості. Можна поставити додатковий термін обмеженості в цільовій функції, щоб покарати не рідко активовані нейрони cite AndrewNg. Штраф за розрідженість обчислює, як часто кожен прихований нейрон активується $hat\ rho_j$, і обмежує його до бажаної розрідженості при розбіжності KL.

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m} a_j(x_i) \tag{73}$$

$$KL(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\rho_j} + (1-\rho)\log\frac{1-\rho}{\hat{\rho}_j} \tag{74}$$

$$L(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||\hat{y}_i - y_i||_2^2 + \lambda||W||_F^2 + \beta\sum_{j=1}^{s_2} KL(\rho||\hat{\rho}_j) \tag{75}$$

Можна також зберегти $k\%$ найбільшої активації, а решту встановити на нуль для досягнення розрідженості Makhzani and Frey [2013].

14.3 Передбачуване розріджене розкладання

Попередня обробка зображення з розрідженим поданням є вигідною, оскільки ці функції, швидше за все, можуть бути лінійно відокремлюваними у великих розмірах і більш надійними проти шуму.

Проблему пошуку розрідженого розкладання можна формально сформулювати наступним чином. Для заданого сигналу $Y\ in\ mathbbR^n$ знайдіть подання $Z\ in\ mathbbR^m$ у даному надмірному базисі B:

$$\min \|Z\|_0\ s.t.\ Y = BZ \tag{76}$$

Де норма l^0 тут позначає кількість ненульових елементів у Z. Якщо застосувати опуклу релаксацію l^0 доданок, це еквівалентно необмеженій задачі оптимізації

$$\mathcal{L}(Y,Z;B) = \frac{1}{2}\|Y - BZ\|_2^2 + \lambda\|Z\|_1 \tag{77}$$

Є й інші алгоритми, які також можуть вивчити базові функції, стовпці B, чергуючи оптимізацію щодо Z і B.

Однак до передбачуваного розрідженого розкладання (PSD) алгоритми, які обчислюють розріджене представлення, обчислювальні надмірно дорого. PSD вдається вирішити проблему

вивчення розріджених подань у 100 разів ефективніше, ніж попередній найкращий існуючий алгоритм, і забезпечити настільки ж гарне представлення ознак. Зокрема, PSD мінімізує дещо іншу функцію втрат:

$$\mathcal{L}(Y, Z; B, P_f) = \frac{1}{2}\|Y - BZ\|_2^2 + \lambda\|Z\|_1 + \alpha\|Z - F(Y; P_f)\|_2^2 \tag{78}$$

де $P_f = \{G, W, D\}$ - це набір параметрів, які слід вивчити, і

$$F(Y; P_f) = G\tanh(WY + D) \tag{79}$$

є нелінійним відображенням. Мінімізація функції збитків щодо Z сил

1. $F(Y; P_f)$ навчитися реконструювати введені дані
2. Z бути розрідженим
3. $F(Y; P_f)$ дає плавне наближення до розрідженого подання

Вище викладено основну настройку та ідею для передбачуваного розрідженого кодування.

14.4 Варіаційний автоенкодер

Декодерна мережа автокодера приймає прихований вектор і переводить його в зображення. Якщо хтось хоче генерувати зображення за допомогою мережі декодера, він повинен знати прихований вектор, перш за все. Однак немає конкретного правила щодо прихованого представлення стандартних автокодерів. Ідея варіаційного автокодера полягає в тому, щоб змусити розподіл прихованих векторів слідувати стандартному розподілу Гауса. Це має сенс, оскільки такі реальні особливості, як висота, приблизно відповідають нормальному розподілу. Потім, коли ми хочемо генерувати зображення, ми просто беремо вибірку зі стандартного гауссового розподілу як вхід і подаємо на декодер. Математика, що лежить в основі, полягає в тому, що кодер апроксимує задній розподіл декодера. Таким чином, цільова функція має вигляд:

$$\mathcal{L}(\phi, \theta, x) = D_{KL}(q_\phi(z|x)||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}(\log p_\theta(x|z)) \tag{80}$$

де $q_{\ phi(z|x)}$ - приблизний розподіл мережі кодера, $p_{\ theta(z)}$ - бажаний стандартний нормальний розподіл, а $p_{\ theta(x|z)}$ - задній розподіл декодування мережі. Мережі кодерів також називають мережею розпізнавання або мережею умовиводу, оскільки вони вивчають наближення $p_{\ theta(x|z)}$. Мережі-декодери також називають генераційними мережами.

Основним недоліком варіаційного автокодера як генеративної моделі є те, що він чутливий до шуму через випадкові вибірки як вхідні дані. Згенеровані зображення зазвичай розмиті та низької якості.

Основними причинами використання варіацій автокодера є запобігання вивченню функцій ідентифікації та покращення здатності фіксувати важливі функції та вивчати багатші уявлення.

14.5 Складені що-де автокодери

Складені що-де автокодери (Stacked What-Where Auto-encoders, SWWAE) – це нова архітектура, представлена Zhao et al. [2015], що запроваджує єдиний підхід до контрольованого, напівконтрольованого та неконтрольованого навчання. Архітекутра має на меті задовольнити дві цілі. Перша з

них полягає у вивченні факторизованого подання вхідних даних, що кодує як інваріантність (тобто виявляти "що незалежно від "де"), так і еквіваріантність (тобто зміни в "де"на вході дають еквівалентні "де"зміни в площинах ознак). На рисунку 75 показані експериментальні результати інваріантності та еквіваріантності. Друга ціль – використовувати як марковані, так і немарковані дані, щоб засвоїти це подання в єдиній системі.

Для досягнення цих цілей SWWAE інтегрує дискримінаційний шлях, що складається із згортки та об'єднання, та генеративний шлях, який використовує деконволюцію та роз'єднання. У дискримінаційному шляху вхідні дані подаються через згортку та об'єднання шарів. Кожен об'єднуючий рівень на шляху передачі просуває інформацію "що"наступному дискримінаційному шару, а інформацію "де ідповідному роз'єднуючому шару генеруючого декодера. Починаючи з мітки класу, генеративний шлях має на меті реконструювати як вхідні, так і проміжні площини ознак дискримінаційного шляху. Він використовує інструкції щодо роз'єднання, передані з дискримінаційного шляху, щоб вказати "де"для реконструкції та деконволюції, з метою генерації "що"на кожному рівні. Ця архітектура відображена на рисунку 74.

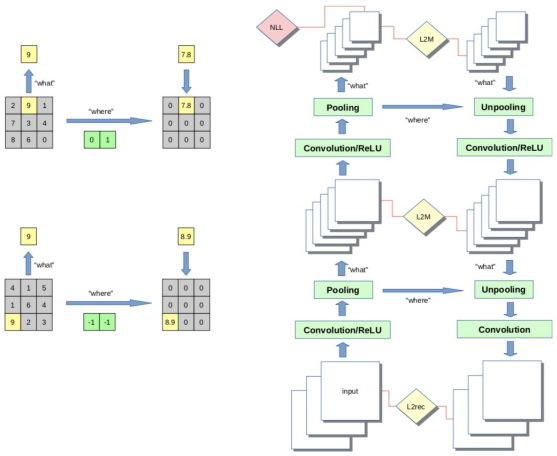


Рис. 74: Ліворуч (a): з'єднання-роз'єднання. Праворуч (b): модель архітектури. Для стислості на цьому малюнку пропущені повністю з'єднані шари. Zhao et al. [2015]

Архітектура SWWAE навчається з функцією втрат, що складаєтсья з трьох частин:

$$L = \lambda_{NLL} L_{NLL} + \lambda_{L2rec} L_{L2rec} + \lambda_{L2M} L_{L2M} \tag{81}$$

де L_{NLL} - дискримінаційна втрата (від'ємна логарифмічна ймовірність) між виходом дискримінаційного шляху та найкращим очікуваним результатом, L_{L2rec} - $L2$ втрати відновлення на вході, а L_{L2M} - $L2$ втрати відновлення на проміжних конволюційних площинах ознак. Таким чином, $L2M$ є показником відновлення на кожному шарі, який обмежує приховані стани шляху зворотного зв'язку, аби наблизити їх до прихованих станів шляху прямого зв'язку, у той час як $L2rec$ гарантує, що реконструйований вхід є близьким до початкового входу. $\lambda-$ є гіперпараметрами, що регулюють ваги компонентів втрат.

$$L_{L2rec} = \|x - \widetilde{x}\|_2, \quad L_{L2M} = \|x_m - \widetilde{x}_m\|_2 \tag{82}$$

Роз'єднуючий крок у процесі генерації є особливо важливою характеристикою SWWAE, що відрізняє його від попередніх архітектур, які уніфікували контрольоване та неконтрольоване навчання, таких як Глибока Машина Больцмана (DBM) Hinton et al. [2006], і автори демонструють його важливість вивчення хороших відображень за допомогою експериментальних результатів. Ця методика вирішує проблему генеративного шляху створення відображення з відношенням один до багатьох, наприклад зіставлення міток класів з реконструкціями зображень. У попередніх методах, таких як DBM, цю проблему зазвичай вирішували, розглядаючи відображення реконструкції як імовірнісне і генеруючи зображення із зображення категорії шляхом вибірки. Однак вибірка непрактична для навчання великомасштабних мереж, оскільки вона часто створює шумні градієнти, чого вже не можна сказати про SWWAE.

SWWAE може бути реалізований за допомогою "м'якої"версії "що"і "де"за допомогою "м'якої"версії max і arg max запропонованої Goroshin et al. [2015].

$$m_k = \sum_{N_k} z(x,y) \frac{e^{\beta z(x,y)}}{\sum_{N_k} e^{\beta z(x,y)}} \approx \max_{N_k} z(x,y) \tag{83}$$

$$\boldsymbol{p_k} = \begin{bmatrix} x \\ y \end{bmatrix} \frac{e^{\beta \ z(x,y)}}{\sum_{N_k} e^{\beta \ z(x,y)}} \approx \arg\max_{N_k} z(x,y) \tag{84}$$

$z(x,y)$ - активація карти ознак, а x, y показують просторове розташування, що є нормаізованим значенням від -1 to 1. N_k показує k^{th} об'єднуючу область, а β є гіперпараметром на множині значень $(0, \infty]$, для якого малі значення призводять до апроксимації середнього об'єднання, а великі значення - апроксимації максимального об'єднання. М'яке М'яке об'єднання та роз'єднання має дві переваги. По-перше, воно робить можливим зворотне розповсюдження через об'єднання шарів, на відміну від жорсткого максимального об'єднання, яке не диференційовано щодо розташування arg max. По-друге, м'яке об'єднання може точніше представляти інформацію про місцезнаходження, що дозволяє ознакам захоплювати більше деталей про вхід.

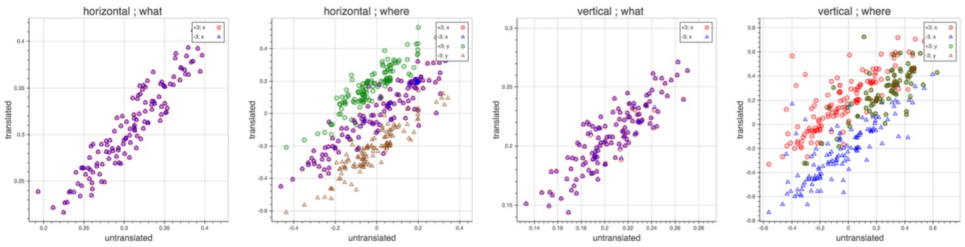


Рис. 75: Точкові графіки, що відображають реакцію на ознаку шляхом переміщення вхідних даних. Горизонтальні осі представляють вихід з однієї площини ознак для незміщеного зображення. Вертикальна вісь представляє вихідні дані для перекладеного зображення (+/- 3 пікселі). (a) та (c) показують виходи "що"для зміщених відносно оригінальних входів. (b) та (d) показують виходи "де"для зміщених відносно оригінальних зображень. Zhao et al. [2015]

SWWAE може легко змінювати модальність між контрольованим, напівконтрольованим та неконтрольованим навчанням, включаючи або вимикаючи певні шляхи та втрати.

- Для контрольованого навчання легко деактивувати генеративний шлях, за допомогою встановлення λ_{L2M} та λ_{L2rec} в 0.
- Для неконтрольованого навчання, встановлюєтья λ_{NLL} в нуль, щоб вимкнути дискримінаційні втрати.
- Для напівконтрольованого навчання, усі ці втрати залишаються активними.

15 Трансферне навчання

Трансферне навчання - це проблема дослідження в машинному навчанні, яка зосереджена на зберіганні знань, отриманих під час розв'язання однієї проблеми, та їх застосуванні до пов'язаних проблем.

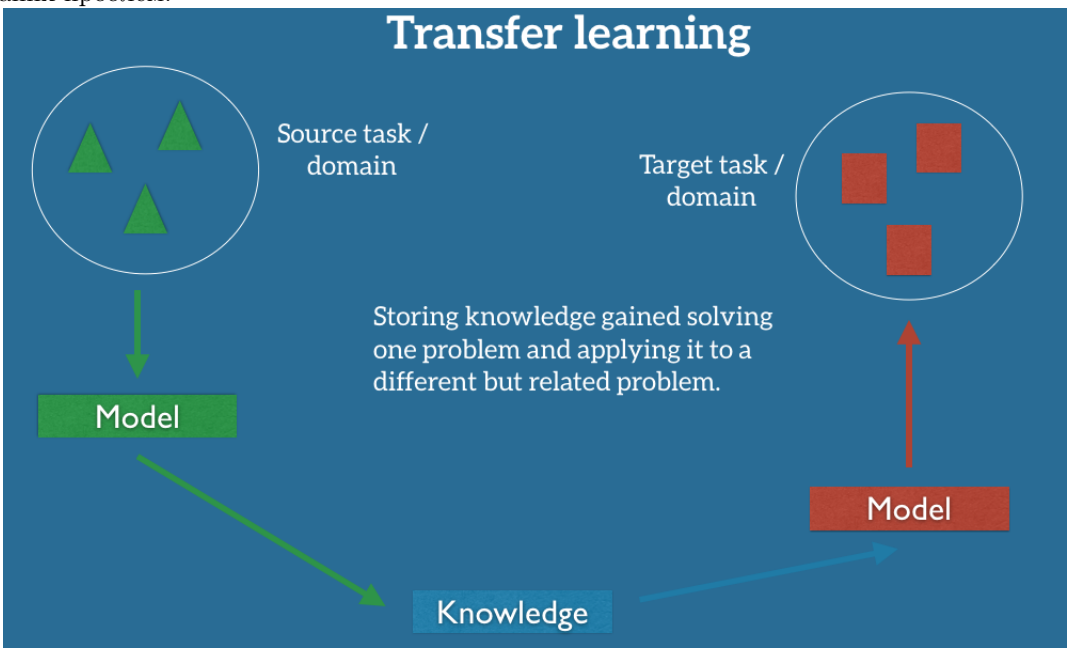


Рис. 76: Трансферне навчання (Джерело: Transfer Learning - Machine Learning's Next Frontier)

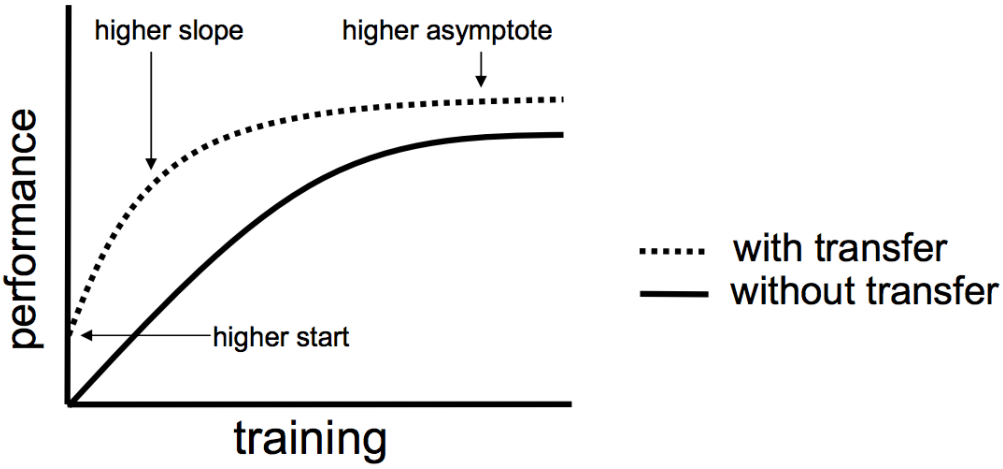


Рис. 77: Ефект трансферного навчання

15.1 Застосування

Існує багато сценаріїв, де трансферне навчання може бути застосоване.

- **Навчання на імітаціях**: Імітації є моделями реального світу, але не відображають його ідеально. Отже, будь-які знання, отримані при їх переживанні, будуть корисними в реальності. Витрати також зменшуються, оскільки ми не витрачаємо ресурси на збір величезних наборів даних, а також скорочується час навчання, оскільки тренування прискорюється за умови використання трансферного навчання. Наприклад, навчання самокерованих автомобілів та роботів.

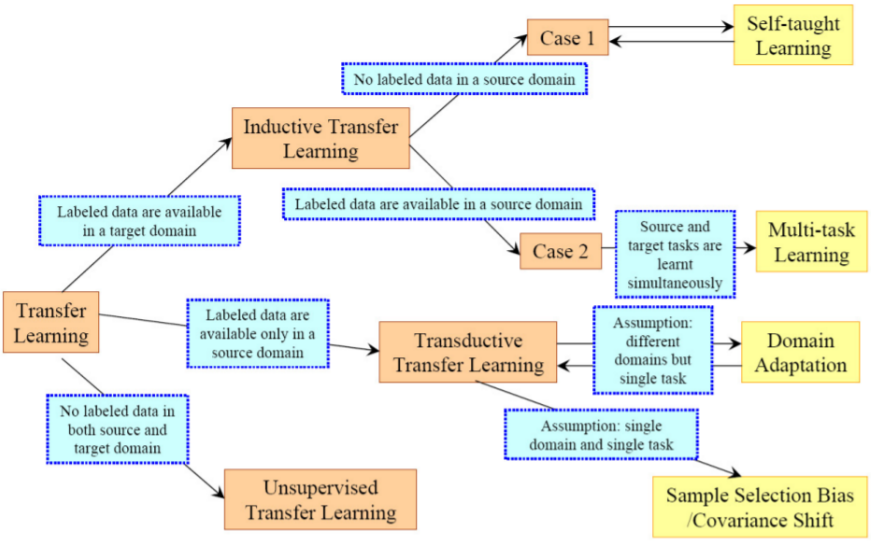


Рис. 78: Застосування трансферного навчання

- **Навчання в різних доменах**: Моделі, навчені даними з новин, важко справляються з новими текстовими формами, такими як повідомлення в соціальних мережах та проблеми, що з них витікають. Таким чином, якщо ми придумаємо більш узагальнені подання та вивчимо інваріантні ознаки домену, ми можемо використати це в інших сферах.

15.2 Трансферне навчання для комп'ютерного зору

Трансферне навчання було вдало адаптовано до роботи з комп'ютерним зором.

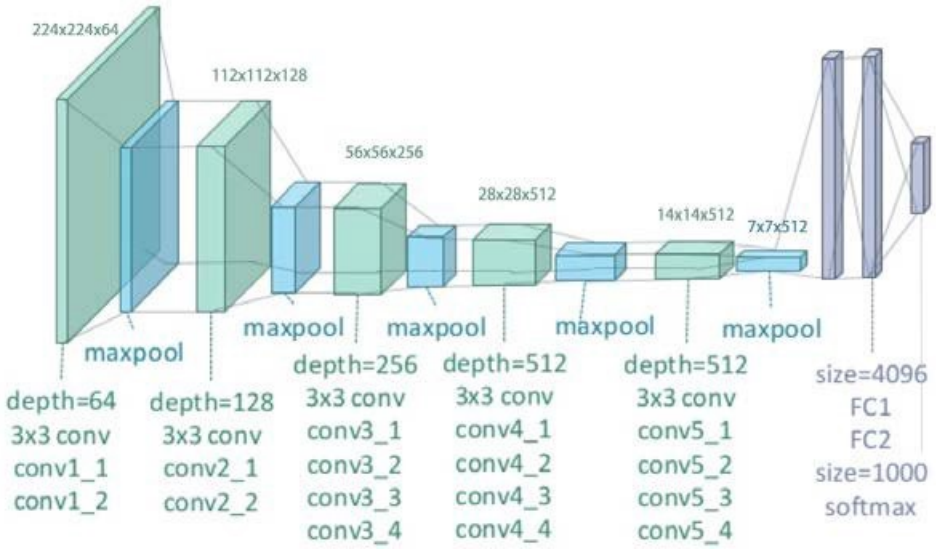


Рис. 79: Мережа VGG-19 для розпізнавання широкомасштабних зображень

15.2.1 Реалізація

Спочатку необхідно навчити модель згорткової нейронної мережі (CNN) на завданні з великою кількістю загальнопозначених навчальних даних, таких як ImageNet. ПоДля відображення поточного завдання треба оновити класифікатор, що складається з повністю зв'язаних шарів. Наприклад, якщо початковим завданням було розпізнавання зображень із десятьма класами, а поточне завдання полягає лише у визначенні присутності кота, то в кінцевому шарі тепер буде лише два юніти замість десяти, а попередні повністю зв'язані шари можна змінити відповідно до архітектури.

- **Згорткові нейронні мережі у виявленні ознак**: Необхідно перенавчити модель, зафіксувавши ваги конволюційних шарів. Популярний набір даних ImageNet розглядається як ідеальна відправна точка для більшості завдань, оскільки він дуже різноманітний, і згорткова мережа може дуже добре вивчити основні особливості в нижчих шарах.

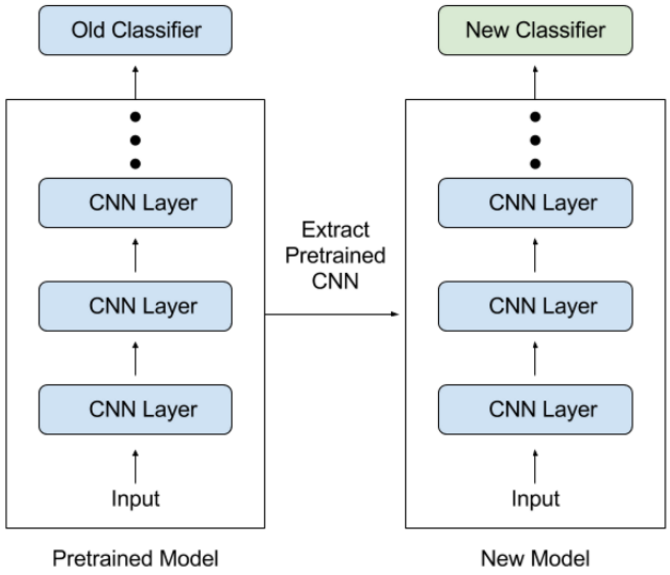


Рис. 80: Згорткові нейронні мережі у виявленні ознак

- **Точне налаштування згорткових нейронних мереж**: Необхідно перенавчити початкову модель. Зазвичай швидкість навчання регулюється по-різному для згорткових шарів та повністю зв'язаних шарів залежно від того, наскільки потрібно зберегти початкові набуті ваги.

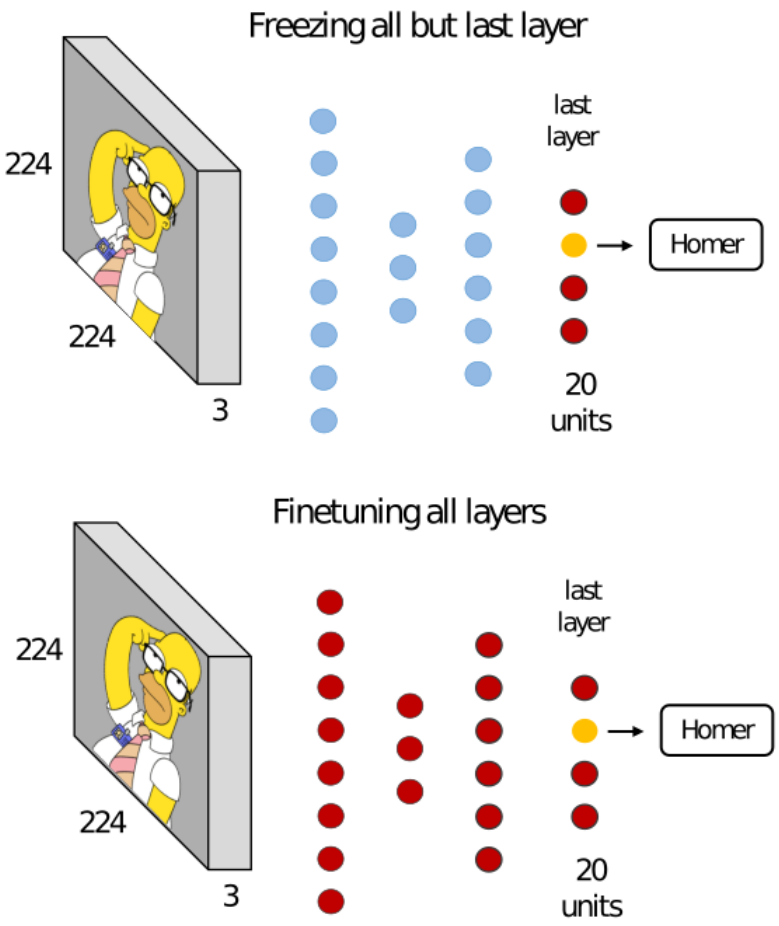


Рис. 81: Точне налаштування згорткових нейронних мереж

У будь-якому з двох методів можна безпосередньо використовувати попередньо навчену згорткову мережу, замість того, щоб самостійно навчати початкову модель, аби скоротити час навчання. Однак це обмежує архітектуру даної моделі.

15.2.2 Примітка

Кількість шарів із замороженими вагами та кількість шарів із вільними вагами залежить від розміру та подібності даних між двома завданнями.

- Якщо наявний обсяг даних великий, дозволяється розморозити більше шарів, оскільки модель не так легко пристосувати.
- Якщо подібність двох наборів даних невелика, припускається можливість розморожування більшої кількості шарів, оскільки можна очікувати, що ці дві задачі матимуть менш загальні спільні ознаки. Отже, в нейронній мережі шари нижче будуть більш корисними, ніж шари вище.

Чим більше вільних шарів у мережі, тим більше можливостей до налаштування архітектури, оскільки немає обмеженя на використання відповідних ваг. Можна навіть позбутися кількох проміжних шарів або додати їх, якщо існує така потреба. Якщо нове завдання абсолютно інше і з великою кількістю даних, існує можливість скористатися цілком новою моделлю або просто ініціалізувати поточну модель вагами старого завдання як хорошу вихідну точку і пройти наскрізне навчання.

Окрім комп'ютерного зору, попередньо навчені функції виявилися корисними в обробці природних мов. Зараз загальноприйнятим підходом є використання вбудованих слів, створених попередньою підготовкою, на великому немаркованому корпусі із наближеною метою мовного моделювання.

15.2.3 Загальні очікування щодо трансферного навчання

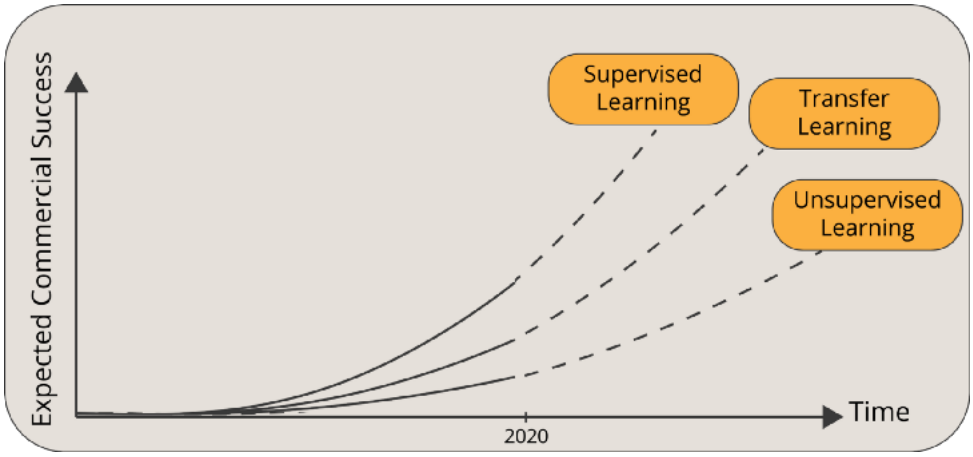


Рис. 82: Очікування за типом контролю

Якщо метою є створення додатку для комп'ютерного зору, а не навчання мережі з нуля, шляхом випадкової ініціалізації, то прогрес здобувається значно швидше, при використанні завантажуваних моделей/способів, для яких вже проводилося навчання в іншій архітектурі мережі, і використанні їх у якості попередньої підготовки та передачі до визначеного завдання. На сьогоднішній день у співтоваристві CV існує безліч наборів даних в Інтернеті, такі як типи наборів даних ImageNet, MS COCO або Pascal, це назви різних наборів даних, які люди розміщують в Інтернеті, і багато дослідників навчили на них свої алгоритми. Іноді це навчання займає кілька тижнів і може зайняти багато часу та ресурсів графічного процесора, а той факт, що дана робота вже зроблена і пройдено болісний високоефективний процес пошуку, означає, що є можливість завантажувати додакти з відкритим кодом, на які деяким дослідником було витрачено багато тижнів або місяців, зрозуміти і використовувати це як дуже гарну ініціалізацію для власної нейронної мережі.

У більшості випадків, розгортання чужої попередньої роботи краще, ніж власний проект, окрім випадку, коли проблема настільки унікальна, що знайти схожості з попередніми роботами не вихожить. Загальні рекомендації на практиці: якщо немає великої кількості даних під

рукою, то можна просто завантажити дуже добре навчену глибоку мережу, подібну за архітектурою, і змінити останній вихідний рівень softmax відповідно до заданого випадку. Як і зазначено вище, якщо необхідний класифікатор котів, і немає великого набору даних, достатньо просто змінити останній шар softmax і заморозити всі попередні шари, щоб продовжити тренувати набір даних. Якщо наявний помірний обсяг набору даних, який треба розмножити, можна розблокувати попередні два-три або більше конволюційних шарів і заморозити всі попередні шари. Виходячи з цього, починається тренування власної моделі. Якщо наявний великий обсяг даних, можна розглянути всі шари та тренувати мережу з самого початку. На щастя, усі ці операції можна легко виконати в сучасних Deep Learning фреймворках, таких як Tensorflow, Caffe, PyTorch.

16 Суміш експертів

16.1 Введення в суміш експертів

В цьому розділі ми поговоримо про суміш експертів. Ця модель була вперше розроблена в 1990-х роках. Ідея цієї моделі полягає в тому, щоб навчити ряд нейромереж, кожна з яких спеціалізується на різній частині даних. Тобто ми припускаємо, що дані надходять з різних режимів, і навчаємо систему, в якій одна нейромережа спеціалізується на кожному режимі. Для управління нейромережею ми дивимося на вхідні дані і вирішуємо, якому фахівцю це належить.

Така система не дозволяє ефективно використовувати дані, тому що дані дробові для кожної механіки. Тому для невеликих наборів даних, можливо, це не дуже добре працює. Але в міру того, як наборів даних стає більше, така система може розігріватися сама по собі, тому що вона може добре використовувати надзвичайно великі набори даних.

Для багатьох моделей після навчання кожна модель має однакову вагу для всіх тестових наборів. Ми не робимо індивідуальну модель залежною від випадку, з яким маємо справу. Так інтуїтивно зрозуміло, що ми можемо запитати: чи можемо ми зробити краще, щоб просто усереднення моделей не залежало від конкретного тренувального випадку? Суміш експертів може це зробити. Ми можемо розглянути вхідні дані для конкретного випадку, щоб вирішити, на яку модель покладатися. Це може дозволити певним моделям спеціалізуватися на підмножині навчальних кейсів. Вони не вчаться на кейсах, для яких вони не обрані. Тож вони можуть ігнорувати речі, які вони не вміють моделювати.

Ключова ідея полягає в тому, щоб змусити кожного експерта(модель) зосередитися на прогнозуванні правильної відповіді в тих випадках, коли він уже робить це краще, ніж інші експерти(моделі).

Різні моделі мають дуже різні властивості. Наприклад, порівняємо ці дві моделі: найближчі сусіди і поліноміальне наближення. Дуже швидко підганяються дані для найближчих сусідів. І це стабільно, тобто коливання одних точок входу не змінять передбачення інших точок входу. З іншого боку, апроксимація поліномів є різновидом повністю глобальних моделей. Це може бути повільним, а також нестабільним. Кожен параметр залежить від всіх даних. Невеликі зміни даних можуть привести до великих змін підгонки.

Замість використання однієї глобальної моделі або безлічі дуже локальних моделей, можна використовувати кілька моделей проміжної складності, особливо добре, якщо набір даних містить кілька різних режимів, які мають різні зв'язки між входом і виходом. Наприклад, фінансові дані, які залежать від стану економіки. Тоді ми повинні використовувати різні моделі для різних станів економіки. Але заздалегідь ми можемо не знати, які точки належать до одного й того ж економічного стану. Ми повинні навчитися і цьому.

Тому у нас є проблема, коли ми будемо використовувати цю різну модель, щоб відповідати різним режимам, а саме, як ми поділяємо набір даних на різні режими.

Для того щоб використовувати різні моделі в різних режимах, нам необхідно згрупувати навчальні кейси по підмножинах, по одному для кожної локальної моделі. Нас цікавить схожість

зіставлень введення-виведення. Ми не хочемо шукати кластери схожих вхідних векторів. Ми хочемо, щоб кожен кластер мав відношення між входом і виходом, який може бути добре змодельований однією локальною моделлю.

Як і в інших навчальних системах, ми повинні вибирати функції помилок для суміші експертів. Різні функції помилок по-різному впливають на розбиття наборів даних. Якщо ми хочемо заохочувати співпрацю, ми порівнюємо середнє значення всіх предикторів з метою і навчаємо, щоб зменшити розбіжність. Це може погано поєднуватися. Це робить модель більш потужною, ніж навчання кожного предиктора окремо. З іншого боку, якщо ми хочемо заохочувати спеціалізацію, ми порівнюємо кожного предиктора окремо з цільовим показником. Ми також використовуємо "менеджера"для визначення ймовірності підбору кожного експерта.

16.2 Мережа входу

Суміш експертів відноситься до техніки машинного навчання, в якій для поділу проблемного простору на однорідні регіони використовується кілька моделей. Це свого роду ансамблеві методи об'єднання простих учнів для поліпшення прогнозів. Вони охоплюють різні вхідні регіони з різними учнями

Перше завдання при впровадженні суміші експертів - знайти параметри для кожного учня. Це одне і те ж завдання, коли ми застосовуємо лише одного учня як модель. Але, якщо ми використовуємо змішування експертів, у нас багато різних учнів, ми повинні використовувати мережу входу, щоб вирішити, як їх комбінувати.

Функція входу з мережі входу дає можливість всім експертам. Вони підсумовуються як 1, і кожен з них може розглядатися як розділ для різних учнів. Як правило, ми можемо використовувати нормовану експоненційну модель або модель генеративного класифікатора для мережі входу.

Припустимо, що функція входу є $g_1(x)$, $g_2(x)$,..., $g_\infty(x)$,

$$0 \leq g_1(x), g_2(x), ..., g_\infty(x) \leq 1 \tag{85}$$

$$\sum_{i=1}^{\infty} g_i(x) = 1 \tag{86}$$

16.3 Ієрархія суміші експертів

Ми вже знаємо, що функція мережі входу може розділити всіх експертів по ймовірностям. Але вона не призначає кожну ймовірність кожному експерту, він визначає імовірнісний розподіл для надання ієрархії експертам. Це схоже на дерево імовірнісних рішень з фіксованою структурою. Таким чином, висновок суміші експертів обумовлений декількома рівнями суміші.

Припустимо, що у нас є набір лінійних учнів, і ми використовуємо нормовану експоненційну

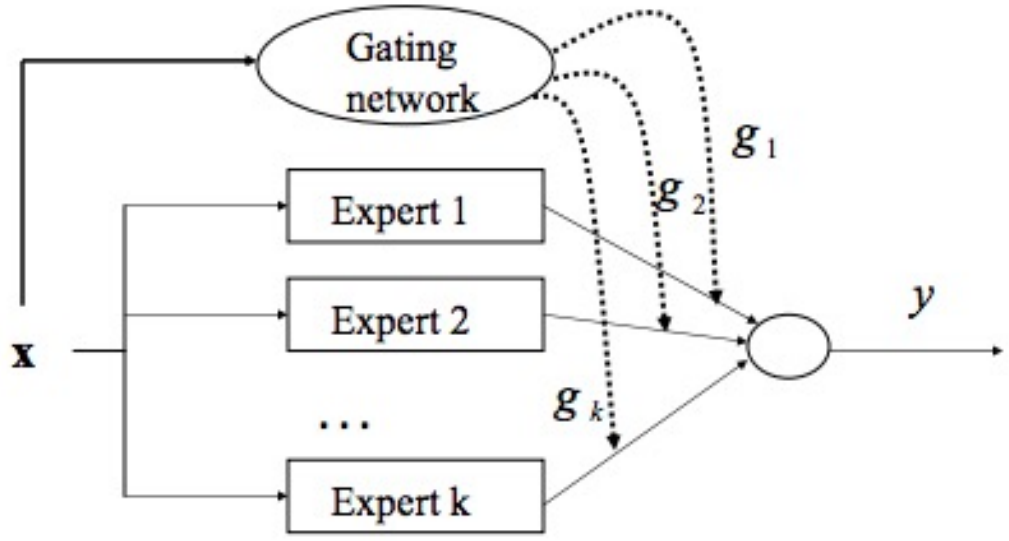


Рис. 83: Gating functions partition each expert by probabilities

мережу входу:

$$\mu_i = \theta_i^\top \boldsymbol{x} \tag{87}$$

$$g_i(x) = \frac{\exp(\eta_i^\top \boldsymbol{x})}{\sum_{u=1}^{k} \exp(\eta_u^\top \boldsymbol{x})} \tag{88}$$

$$P(y|\boldsymbol{x}, \Theta) = \sum_u P(\omega_u|\boldsymbol{x}, \eta) \sum_v P(\omega_{uv}|\boldsymbol{x}, \omega_u, \xi_u) ... \sum_s P(\omega_{uv...s}|\boldsymbol{x}, \omega_u, \omega_{uv}, ...) P(y|\boldsymbol{x}, \omega_u, \omega_{uv}, ..., \theta_{uv...s}) \tag{89}$$

$P(y|\boldsymbol{x}, \omega_u, \omega_{uv}, ..., \theta_{uv...s})$ є індивідуальними експертами.

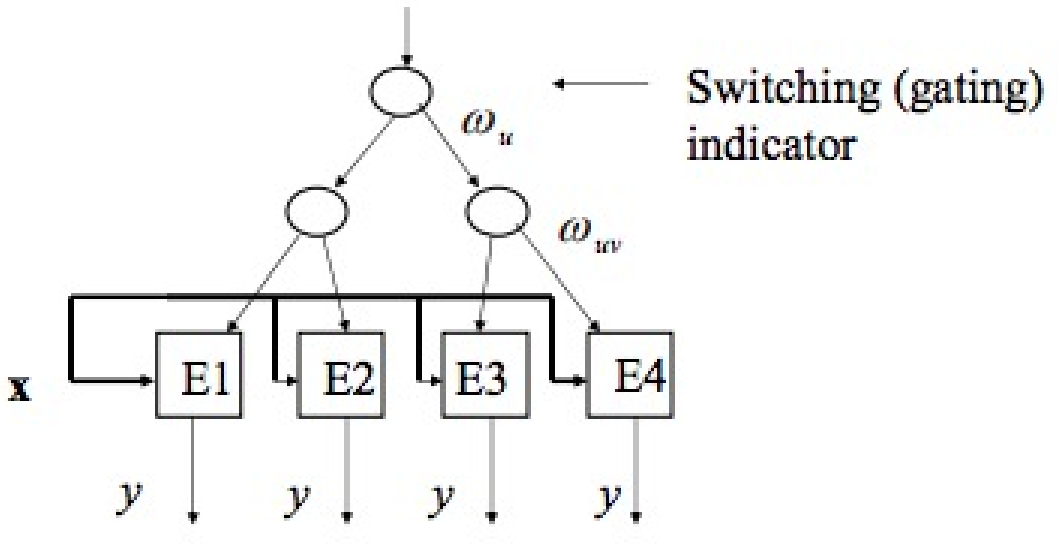


Рис. 84: An example of hierarchical mixture of experts is like a probabilistic decision tree

17 Просторові моделі

Просторова модель починається з визначення графічної моделі дерева, яка є випадковим полем Маркова над просторовими місцями на зображенні. Потім проводиться оцінка спільного розподілу такої проблеми. І з цього різні частини зображення підбираються до того, до якого просторового розташування вони найкраще підходять. Одним з популярних прикладів просторових моделей є оцінка пози людини. У слайдах лекції "Зворотні мережі, частина 2"наведено два різні методи; "MODEC: Мультимодальні розкладаються моделі для оцінки пози людини " Бен Сапп та Бен Таскар та "Вивчення ефективної оцінки пози людини на основі неточної анотації" Сем Джонсон та Марк Еверінгем.

17.1 MODEC

MODEC - це мультимодальна модель, що розкладається, яка визначає її режими шляхом кластеризації конфігурацій об'єднання людського тіла в нормалізованому просторі координат зображення. Модель MODEC має явні змінні вибору режиму, які виводяться спільно з найкращою конфігурацією корпусу частин тіла для зображення.

Щоб отримати уявлення про те, як працює MODEC, давайте розглянемо проблему, яку намагається вирішити модель. Для цієї проблеми вхідні дані (пікселі зображення) визначаються як x, вихідні змінні $y = \ left[y_1, \ ..., \ ; , y_P \ right]$ де P - кількість розміщень частин тіла в координатах зображення, а спеціальні змінні режиму $z = [z_1, \ ..., \ ; , z_K], \ z_i \ in [1, M]$, які слугують міткою для можливих конфігурацій суглобів для частин тіла. Функцію оцінки для моделі MODEC можна записати як

$$s(x, y, z) = s(x, z) + \sum_{c \in C} s_c(x, y_c, z_c) \tag{90}$$

Рівняння 90 оцінює приклад x при виборі вихідної змінної y і змінної z. Y_c позначають підмножини y, які індексуються однією клікою z, що позначається z_c. Завдяки такій функції оцінки 90 метою MODEC є:

$$z^*, y^* = \arg\max_{z,y} s(x, y, z) \tag{91}$$

За словами Бена Саппа та Бена Таскара, є три припущення, необхідні для того, щоб зробити висновок MODEC вдалим: (1) Ефективно обчислювати $\max_y \sum_{c \in C} s_c(x, y_c, z_c)$. (2) Ефективно обчислювати $\max_y s(x, z)$. (3) Існує взаємозв'язок "один до багатьох"від клік z_c до кожної змінної в y. Якщо ці умови відповідають цілі MODEC, описаній у **??**, це може бути ефективно вирішено.

Тепер, коли модель MODEC пристосована до оцінки людської пози, реалізовано дві змінні режиму, одну для лівої частини тіла, а другу для правої сторони: z_l і z_r, де кожна має $= 32$. Ліву та праву бічні моделі можна записати так:

$$s_l(x, y_l, z_l) \sum_{i \in \mathcal{V}_l} \mathbf{w}_i^{z_l} \cdot \mathbf{f}_i(x, y_i, z_l) + \sum_{i,jin\mathcal{E}_l} \mathbf{w}_{i,j}^{z_l} \cdot \mathbf{f}_{i,j}(y_i, y_j, z_l) \tag{92}$$

Де $\mathcal{V}_l$ і $\mathcal{E}_l$ - це вершини та ребра графіка взаємодії змінних відповідно. Тепер термін оцінки режиму $s(x, z)$ з 90 можна записати як:

$$s(x, z) = \mathbf{w}^{l,r} \cdot \mathbf{f}(z_l, z_r) + \mathbf{w}^l \cdot \mathbf{f}(x, z_l) + \mathbf{w}^r \cdot \mathbf{f}(x, z_r) \tag{93}$$

Перший член може розглядатися як вимірювання того, наскільки узгоджені обраний лівий і правий режими, тоді як останні два терміни дають оцінку, наскільки зображення узгоджується із заданим режимом кожної сторони. Поєднайте рівняння 92 та 93, щоб отримати загальну функцію оцінки:

$$s(x, y, z) = s_r(x, y_r, z_r) + s_l(x, y_l, z_l) + \mathbf{w}^{l,r} \cdot \mathbf{f}(z_l, z_r) + \mathbf{w}^l \cdot \mathbf{f}(x, z_l) + \mathbf{w}^r \cdot \mathbf{f}(x, z_r) \tag{94}$$

17.2 Вивчення ефективної оцінки пози людини на основі неточної анотації

Amazon Mechanical Turk (AMT) спостерігає сплеск у використанні нещодавно для збору великих наборів даних зору, що дозволяє ефективно і недорого виконувати прості завдання такі як анотація зображень. Потрібно використовувати AMT для збору великого набору даних для оцінки пози людини, принаймні на порядок більше, ніж було доступно раніше. Однак для більшості попередніх програм потрібні анотації порівняно прості - наприклад, теги зображень або позначення обмежувальної рамки. Будемо вимагати точного маркування 14 різних розташування на тілі разом із зазначенням видимості кожного. Тоді як AMT дозволяє швидко збирати великі номери таких анотацій іноді є неточними або навіть зовсім неправильний. Ми пропонуємо моделювати та враховувати ці помилки в ітеративній схемі навчання - покращуючи якість поганих анотацій до досягти великої кількості високоякісних навчальних даних. Ми прагнемо покращити ефективність роботи в першу чергу у найскладніших позах, які приймають люди. В набору даних LSP ми показали, що найскладніші пози полягають у діяльності з гімнастики, паркуру та менший ступінь легкої атлетики. Ми поставили запит до Flickr за допомогою цих тегів і вручну відібрав 10800 зображень різних людей у широкий спектр надзвичайно складних

поз. Поширюючи ці зображення в AMT - разом із спеціально створеним інструментом анотацій - ми можемо отримати приблизно 400 анотацій в годину. Однак ця ефективність залежить від точності - деякі анотації демонструють помилки локалізації, помилки в "Будова тіла"(наприклад, плутанина ліворуч / праворуч) або повністю непридатні. У деяких випадках користувачі здається, що вони випадково позначили зображення або позначили його усі пункти невидимі - ці анотації ми вважаємо «непридатними для використання». В останньому випадку ідентифікація є тривіальною. У колишньому виявлення справи вимагає подальших роздумів. Відхилення анотацій як непридатних призводить до того, що працівник AMT отримує неоплату - заохочення людей подавати високоякісні анотації у пізніші завдання. Завдяки цьому ми повинні подбати лише про те, щоб по-справжньому відкинути погані анотації. Для напівавтоматизації процесу ми класифікуємо анотації шляхом вивчення моделі Гауса суміші над частиною довжини та відносні орієнтації з набору даних LSP. Ми тоді замовити 10 800 анотацій за їх вірогідністю за цією моделлю та відхилити будь-які непридатні. Для остаточний набір даних ми випадково відбираємо 10 000 зображень із прийняті анотації.Ми пропонуємо взяти до уваги помилки, наявні в анотаціях AMT, обробляючи істинні розташування суглобів і структура тіла як приховані змінні, с Анотації AMT є галасливими спостереженнями за ними. З підмножини з 300 зображень, для яких ми маємо обидва «Експертні» ґрунтовні правдиві та анотації AMT, за нашими оцінками дві моделі типу помилок, допущених працівниками AMT щодо цього завдання: (i) ми припускаємо, що спільні місця, прокоментовані працівниками AMT, розподіляються відповідно до ізотропний розподіл Гауса по горизонтальному та вертикальному зсуву із середнім значенням у справжньому розташуванні; (ii) набір найпоширеніших конструктивних помилок (включаючи відсутність помилок) S визначається за зразком анотацій: Без помилки, лівий / правий перемикач, лівий / правий перемикач рук, лівий / правий ноги поміняли, руки / ноги змінили, руки / ноги змінили і ліва / права переключена, руки / ноги змінені та ліва / права руки переключено, руки / ноги переключені та ліва / права ноги змінені. Не маючи великого обсягу даних з експертною обґрунтованою правдою ми припустимо, що анотація працівників AMT демонструє кожну цих структурних помилок з однаковою ймовірністю. Ітеративний метод навчання. Використовуючи ці вивчені моделі помилка анотації, ми можемо ітеративно покращувати якість галасливі анотації AMT щодо того, що можна було б вважати "експертною"ґрунтовною правдою. По суті, навчальним завданням є представляється як одне з багаторазового навчання, де нотація AMT для зображення визначає пакет правдоподібних справжніх анотацій. Двохетапний процес, подібний до цього використовували Фельзеншвальб та ін. для уточнення застосовується анотація обмежувального вікна: (i) початковий набір моделей зовнішнього вигляду деталей вивчається з необроблених анотацій; (ii) приховані місця розташування суглобів і структура тіла оновлюються w.r.t. вивчені моделі помилок. Це можна зрозуміти як чергування між вивченням моделі та вибором «найкращої» анотації в кожній сумці, що є і правдоподібним, враховуючи шум анотація і яка узгоджується з поточною вивченою моделлю.

18 Ядра

18.1 Введення

Ця стаття надає базову інформацію про Функції Базису, Орієнтованих На Зразки—-Kernels та інші приклади

18.2 Розуміння Ядер

Що таке Ядра

Ядро — це функція подібності, яка приймає два входи і представляє наскільки вони схожі. У проблемі класифікації зображень замість обчислення властивостей і вхідних векторів ознак в алгоритм навчання, ми визначаємо єдину функцію ядра для обчислення подібності між зображеннями, і вводимо це ядро і зображення в алгоритм навчання і створюємо класифікатор.

Визначення "Ядреобразного"методу

Метод є "Ядреобразним якщо кожен вектор ознак $\psi(x)$ з'являється тільки всередині внутрішнього представлення з іншим вектором ознак $\psi(x')$. Це стосується як задачі оптимізації, так і функції прогнозування.

Функція Ядра

- Вхідний простір: χ
- Ознака простору: H (простір Гільберта, тобто внутрішній простір представлення з проекціями,наприклад R^d)
- Карта ознак: $\psi(x) :\rightarrow \chi$
- Функція ядра, що відповідає χ є

$$k(x, x') =< \chi(x), \chi(x') >$$

де <.,.> — це внутрішнє представлення, пов'язана з H

Чому ми використовуємо Ядро

1. Для відносно невеликих просторів дуже швидко оцінюється $k(x, x')$ без явних обчислень $\psi(x) i \psi(x')$
2. Ядро може уникнути будь-яких $O(d)$ операцій, що дозволяє отримати доступ до нескінченно-вимірних просторів ознак
3. Ядро дозволяє нам думати скоріш за все про "схожості ніж про ознаки, що простіше для розгортання і розуміння

Рис. 85: Exampled data inseparable in 2-dimension but separable in 3-dimension

18.3 Хитрощі Ядра

Коли виконуються умови Мерсера, в яких стан K має бути симетричний, а також позитивно визначеним, виконується:

- Функція ядра дозволяє працювати над внутрішнім представлення для нової функції ядра на основі заданого алгоритму ML, таким чином, нове ядро може відповідати високо-розмірному просторі ознак, а вартість обчислення залежить не від розмірності простору ознак, а від кількості точок даних.
- Ми можемо замінити точкове представлення ядер на ті алгоритми машинного навчання, в яких використовуються точкові представлення. Це дозволить нам працювати над потенційно нескінченним простором розмірних ознак.

18.4 Приклад Ядер

- Лінійне Ядро
  Карта ознак:
  $$\psi(x) = x$$
  Ядро:
  $$k(x, x') = x^\top x'$$
- Квадратичне Ядро
  Карта ознак:
  $$\psi(x) = (x_1, ...x_d, x_1^2, ..., x_d^2, \sqrt{2}x_i x_j, ...\sqrt{2}x_{d-1}x_d)^\top$$
  Ядро:
  $$k(x, x') =< \psi(x), \psi(x') >=< x, x' > + < x, x' >^2$$
- Ядро Гаусса
  $$k(w, x) = exp(-\frac{\|w - x\|_2^2}{2\sigma^2})$$

18.5 Застосування Методу Ядра

Метод ядра (Базові функції орієнтовані на зразок) широко використовується при вирішенні проблем навчання машин, а також при аналізі шаблонів. Найпопулярніші програми для функцій ядра застосовуються до Ядра Методу Опорних Векторів (МОВ). У цьому розділі ми в основному поговоримо про метод, який ми розробляємо для цієї моделі з функцією ядра.

18.5.1 Матриця Ядра та Теорема Представника

Для кращого розуміння програм ядра, спочатку ми вводимо дві концепції:

18.5.2 Матриця Ядра

Матриця Ядра K для функції ядра $k(x, x') = (\Psi(x), \Psi(x'))$ на $x_1...x_n \in \chi \in$:

$$K = (k(x_i, x_j))_{ij} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \cdots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{pmatrix}.$$

18.5.3 Теорема Представника

У дослідженні глибокого навчання ми завжди використовуємо теорему представника для регуляризованої емпіричної функції ризику, яка визначається в просторі ядром Гільберта. Ця теорема робить регуляризовану емпіричну функцію реформування функцією кінцевої лінійної комбінації продуктів ядра. Детальний вираз такий:

$$J(w) = R\left(\|w\|\right) + L\left(\langle w, \psi(x_1)\rangle, \ldots, \langle w, \psi(x_n)\rangle\right),$$

де w, x_1 ... x_n — це всі значення з простору Гільберта, $R : [0, \infty] \to R$ — це термін регуляризації, а $L : R^n \to R$ — це термін втрат.

18.5.4 Ядро Методу Опорних Векторів

Метод опорних векторів — це модель, яка забезпечує набір гіперплощин у великому розмірному просторі для класифікації та регресії, завдяки хитрощам ядра та вдосконаленій моделі ядра, ми можемо оцінити цільову функцію із меншими витратами часу та вартості. Для великих просторів об'єктів швидкість різко змінюється.

Цільова функція методу опорних векторів $J(w)$ можна виразити як:

$$J(w) = \min_{w \in R^n} \frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m} \max\left(0, 1 - y_i w^T x_i\right)$$

Завдяки двоїстості Лагранжа ми можемо знайти мінімізацію, вирішивши наступну задачу оптимізації:

$$\max_{\sigma \in R^n}(\sum_{i=1}^{n} \sigma_i - \frac{1}{2}\sum_{i=1}^{n} \sigma_i \sigma_j y_i y_j \Psi(x_j)^T \Psi(x_i))$$

при умові :

$$\sum_{i=1}^{n} \sigma_i y_i = 0, \sigma_i \in [0, \frac{c}{n}], i = 1...n$$

ми спочатку дозволили $s(w) = \begin{pmatrix} k(w, x_1) \\ \vdots \\ k(w, x_n) \end{pmatrix}$. Застосовуючи згадану вище теорему репрезентатора, форму подвійності методу опорних векторів можна переформулювати за допомогою:

$$
\begin{aligned}
J(w) &= R(||\sum_{i=1}^{n}\sigma_i x_i||) + L(s(\sum_{i=1}^{n}\sigma_i x_i)) \\
&= \sqrt{k(\sum_{i=1}^{n}\sigma_i x_i, \sum_{j=1}^{n}\sigma_j x_j)} + L(\begin{pmatrix} \sum_{i=1}^{n}\sigma_i x_1 \\ \vdots \\ \sum_{i=1}^{n}\sigma_i x_n \end{pmatrix}) \\
&= R(\sqrt{\sigma^T K\sigma}) + L(K\sigma)
\end{aligned}
\tag{95}
$$

Тоді нашу мінімізацію для цільової функції можна оновити до:

$$
J(w) = \min_{w\in R^n} \frac{1}{2}\sigma^T K\sigma + \frac{c}{n}\sum_{i=1}^{m}\max\left(0, 1 - y_i(K\sigma)_i\right)
$$

18.6 Висновок

У результаті, ми спочатку представляємо метод ядра, а також хитрощі ядра. Потім ми перелічили кілька прикладів хитрощів ядра та детального розширення для однієї програми: ядро MOB, який забезпечує менший час та обчислювальну вартість.

19 Енергетичне Навчання

19.1 Введення

Як для традиційного машинного навчання, так і для статистичного моделювання вкрай важливо фіксувати залежності між змінними, щоб модель могла давати значення неспостережуваних змінних, коли їй відомі значення спостережуваних змінних. Енергетичні моделі (Energy-Based Models, EBM) досліджують новий спосіб знайти це відношення залежності. Вона пов'язує заздалегідь визначену скалярну енергію з кожною конфігурацією змінних. Скалярну енергію також можна інтерпретувати як спосіб вимірювання сумісності.

Є дві основні задачі EBM: одна - висновок, інша - навчання. По-перше, давайте коротко розглянемо, що являють собою ці двоє в контексті машинного навчання. Висновок зазвичай використовується для передбачення. Навчання зазвичай означає процес оцінки параметрів.

Таким чином, з точки зору висновку, EBM намагаються знайти значення прихованих змінних / неспостережуваних змінних / невідомих змінних, які могли б мінімізувати енергію при фіксованому значенні спостережуваних змінних.

Що стосується навчання, EBM прагнуть знайти правильну функцію енергії, яка пов'язує нижчі енергії з правильними значеннями неспостережуваних змінних і пов'язує вищі енергії з неправильними значеннями. EBM використовують функціонал втрат як вимір якості функцій альтернативної енергії. Функціонал втрат є функцією параметра моделі, коли наводиться багато прикладів. Таким чином, функція енергії є функцією конфігурації неспостережуваних змінних і конфігурації вводів. Мала енергія означає високу сумісність значень введення X і виведення Y, що означає високу ймовірність коректності.

Одним словом, EBM інтерпретують висновок як мінімізацію енергетичної функції, а навчання як мінімізацію функції втрат. Отже, EBM демонструють загальну структуру висновку / навчання

в будь-яких типах статистичних моделей (імовірнісних або, особливо, не імовірнісних) і моделях машинного навчання.

Є чотири типи додатки, розроблені в роботі.

- Прогнозування, класифікація та прийняття рішень:
  Визначення найбільш сумісного y, найкращого y, якщо дано X. наприклад класифікація зображень, керуйте роботом, щоб уникнути перешкод.

- Рейтинг:
  Виявлення y (серед y1, y2, y3... ...), які є більш сумісними із заданим X. Порівняно з останньою, ця модель дасть кілька результатів. наприклад найкраща рекомендація до пункту.

- Виявлення:
  Визначимо, чи це значення y (серед багатьох інших значень) сумісне з X? наприклад розпізнавання облич.

- Оцінка умовної щільності:
  З'ясування умовного розподілу ймовірностей y, для заданого X.

EBM - це модельний фреймворку, який намагається відповісти на запитання "який Y найбільш сумісний з даним X?". Головною перевагою EBM є те, що він може оцінювати ймовірності, не вимагаючи нормалізації вхідних даних, що в багатьох випадках може бути дуже важким. Крім того, EBM також може застосовуватися до неімовірнісних завдань, що розширює його корисність.
В EBM модель розробляється шляхом мінімізації двох різних функцій. Одну називають енергетичною функцією E(Y, X). Енергетична функція подібна до цільової функції в інших алгоритмах машинного навчання, яка генерує оцінку для заданих вхідних даних X та вхідної мітки Y. У EBM функція енергії повинна призначати менші значення енергії для високосумісних конфігурацій змінних, і найкращий E(Y, X) знаходить процес виведення. Інша функція - це функція втрат, яка навчається в процесі навчання. (Глава 2)
Далі в главі 2 детально обговорюється ряд функцій втрат:
2.2.1 втрати енергії: найпростіші, але не можуть залучити енергію на небажані відповіді.
2.2.2 узагальнена втрата персептрону: завжди позитивна, але не може обіцяти енергетичний розрив між правильною відповіддю та неправильною.
2.2.3 узагальнена втрата націнки (втрата шарніра, втрата журналу, LVQ2, MCE тощо): ця категорія втрат здатна створити енергетичний розрив, але кожна з них має певні вимоги до даних.
2.2.4 негативна втрата вірогідності журналу: широко застосовується, але вимагає інтегральних обчислень.
Глава 3 представляє основні архітектури, а глава 4 - нову ідею під назвою „прихована змінна" із прикладами застосування. Приховані змінні, які також називаються прихованими змінними, дуже схожі на результати розробки особливостей в інших алгоритмах.
У розділі 5 обговорюються достатні умови для вибору хорошої функції втрат. Тобто, 1. Алгоритм виведення енергії повинен давати надійно стабільну правильну оцінку на входах. 2. Між правильною відповіддю та неправильною існує позитивна межа.
Глава 6 вводить поняття неімовірнісних графіків факторів та ефективних алгоритмів висновку.

Глава 7 зосереджена на маркуванні послідовностей та структурованих вихідних моделях із переглядом відповідних літератур.
Зауважимо, енергетичні результати від EBM знаходяться у довільних одиницях. Таким чином, калібрування необхідне для поєднання двох або більше результатів EBM. Одним із поширених способів є нормалізація збору енергії для всіх можливих результатів, але ця процедура може бути дуже дорогою і її слід уникати, якщо це можливо.

19.2 Тематичне дослідження машинного перекладу

Деякі причини, чому машинний переклад є складним завданням, включають наступне: (1) він повинен взяти змінну довжину вводу і вивести змінну довжину результату, (2) слово однією мовою може перетворитися на кілька слів іншої, (3) вихідний простір дуже великий, (4) може бути багато еквівалентних перекладів, (5) модель, вивчена за допомогою даних в одному домені, може не працювати для даних з іншими доменами, (6) може бути недостатньо даних для деяких мовних пар.

Моделі машинного перекладу, як правило, дотримуються такої архітектури: (1) вхідне речення кодується вбудованими словами, (2) кожне вбудоване вихідне слово отримує оцінки уваги за допомогою softmax, і (3) підсумовує вектор джерела, нормований за його балами уваги, і використовує його для генерування таргету. Під час навчання модель вчиться передбачати по одному цільовому маркеру за раз і мінімізувати наступні перехресні ентропійні втрати $\mathcal{L}_{TokNLL} = -\sum_{i=1}^{n} logp(t_i|t_1, ..., t_{i-1}, x)$. Під час тестування ми апроксимуємо найбільш вірогідне цільове речення за допомогою пошуку променя, де передбачення $\hat{u} = argmin(-logp(u|x))$. І ми оцінюємо цю сформовану ціль, використовуючи наступну оцінку BLEU:

$$BLEU = BPe^{\sum_{n=1}^{N} \frac{1}{n} logp_n}$$

$$p_n = \frac{\sum_{\text{generated sentences}} \sum_{\text{ngrams}} Clip(Count(\text{ngram matches}))}{\sum_{\text{generated sentences}} \sum_{\text{ngrams}} Count(\text{ngram})}$$

Проте, вище модель має кілька проблем: (1) вона має ухил експозиції, де навчання і тестування непослідовні, тому що модель ніколи не навчена для створення всієї послідовності, але тільки маркер в той час, і (2), тому що синій НЕ диференційовані, ми оптимізуємо для різних втрат. Для вирішення цих проблем було запропоновано ряд рішень [Ranzato et al., 2015, Bahdanau et al., 2016, Wiseman and Rush, 2016, Kim and Rush, 2016].

Ми також можемо використовувати основу модель енергії на основі того, щоб на рівні послідовності складання NLL, які могли б подолати деякі із зазначених труднощів:

$$\mathcal{L}_{SeqNLL} = -logp(u^*|x) + log \sum_{i=1}^{n} p(u|x)$$

де u - гіпотеза, породжена моделлю, $u^* = argmin_{u \in U(x)} cost(u, t)$ - гіпотеза оракула (серед гіпотез, породжених моделлю, та, що має найвищий BLEU або найнижчу вартість) і $\hat{u} = argmin_{u \in U(x)}(-logp(u|x))$ є найбільш вірогідною гіпотезою (результат пошуку променя, гіпотеза, яка найкраще оцінюється відповідно до імовірності журналу). Є дві помітні відмінності: (1) u^* використовується замість цілі, щоб навчання було стабільнішим, коли модель насправді може навчитися досягати заданої сурогатної цілі, і (2) ми нормалізуємося за досяжним набором замість всіх можливих гіпотез, щоб обчислення було більш зручним.

Розподіл Болтсмана для переходу від енергії до ймовірності є

$$\tilde{p}(u|x) = \frac{e^{E(u,x)}}{sum_{\bar{u}\in U(x)} e^{-E(\bar{u},x)}}$$

where

$$E(u,x) = -logp(u|x) = -log \prod p(u_i|u_{i-1},...,u_1,x)$$

. (Лектор сказав, що $\tilde{p}$ є частиною рівняння для $\mathcal{L}_{SeqNLL}$, але ми не впевнені, що він має на увазі лише перший енергетичний член або також і другий.)

20 GAN на енергетичній основі

20.1 Вступ

Як випливає з назви, енергетичний GAN (EBGAN) можна вважати похідним від GAN. Загальна структура EBGAN та GAN досить схожа. На малюнку нижче ви можете знайти структуру EBGAN. Давайте спочатку розглянемо буквальне значення EBGAN, енергія означає скаляр як вихід з моделі, що є визначенням з теорії про енергетичну модель Лекуна. cite EN [Lecunn et al., 2006] І що є скалярним у цій моделі, і це буде обговорено пізніше. Ідея енергії полягає в тому, що вона буде призначати меншу енергію справжнім даним, а нереальним даним - більшу енергію.LeCun et al. [2006]

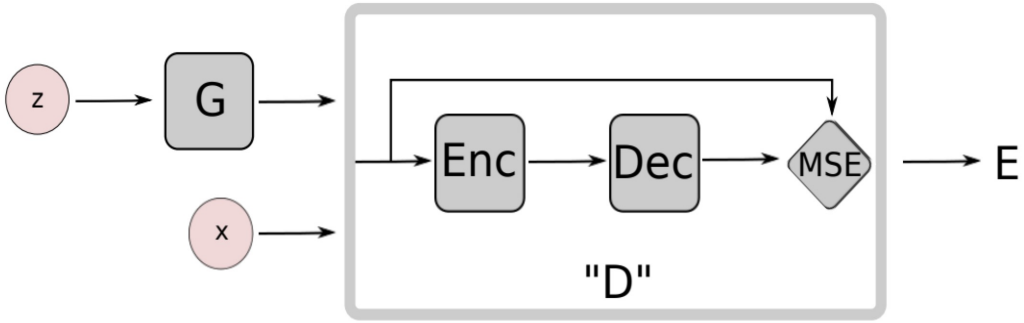


Рис. 86: The structure of EBGAN. [Zhao et al. ,2016]

20.2 EBGAN механізм та покращення

Однією з основних відмінностей енергетичного GAN та звичайного GAN є те, що EBGAN використовує автокодер як дискримінатор, який складається з кодера та декодера. Наприклад, коли цифра вводиться в дискримінатор, реконструйована цифра буде виходом дискримінатора. Можна уявити, що дискримінатор більше не використовується для безпосереднього отримання вірогідності чи оцінки, як це є у звичайному GAN. висновок дискримінатора для EBGAN, що суперечить GAN. Формула дискримінатора наведена нижче. Витік - це помилка MS реальних даних та реконструйованих даних з D.

$$D(x) = ||Dec(Enc(x)) - x||$$

Давайте розглянемо структуру EBGAN. Z є випадковим вектором з будь-яким попереднім розподілом, і попередній розподіл тут не дуже важливий, оскільки ми намагаємося нарешті генерувати Z з тих же розподілів, що і справжні дані, що важко. Подивіться на функцію втрат EBGAN, там - граничний термін M для функції дискримінатора втрат, і за цим існує дуже інтуїтивна причина.

Тепер давайте розглянемо функцію втрат EBGAN від [Zhao et al. 2017]. D (G (Z) не буде вищим за межу m, оскільки термін $[*]^+$ не має сенсу мати вищу енергію. Без запасу функція втрат може просто присвоїти D високу енергію (G (Z)), насправді не турбуючись про те, яку енергію слід дати першому доданку у формулі, який є енергією для реальних даних. Функція втрат для генератора досить проста, оскільки ідея подібна до GAN. хоче мінімізувати енергію, що подається генератором, щоб вихід генератора міг успішно обдурити дискримінатор.

$$L_D(x, z) = D(x) + [m - D(G(z))]^+$$

$$L_G(z) = D(G(z))$$

Інший зауважив у роботі Чжао, що навчання кодеру та декодеру, швидше за все, може вчитися лише на реальних даних і виробляти нульову енергію скрізь без будь-якої регуляризації.Zhao et al. [2016][Zhao et al.,2017] Щоб уникнути цієї проблеми, він реалізує термін регуляризації під назвою "Витягування який має на меті зробити вихідні вектори з кодера максимально різними.

21 Навчання без вчителя

21.1 Машини Больцмана з обмеженим доступом

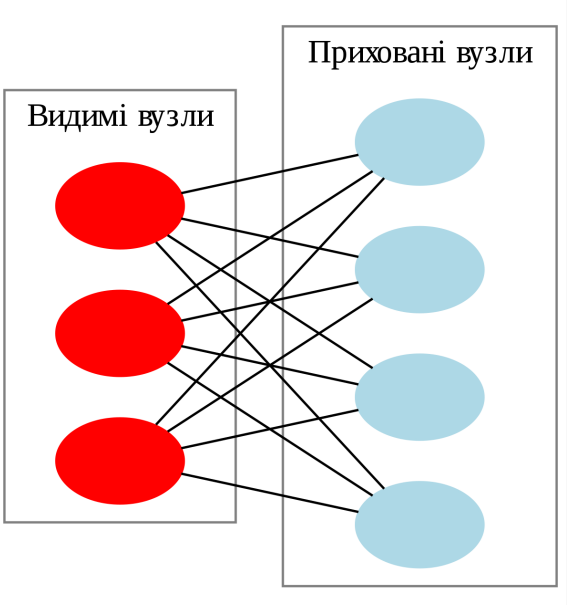
21.1.1 Вступ

Обмежені машини Больцмана (ОМБ) вперше винайдені Джеффрі Хінтоном, і ни розглядаются як одні з перших нейронних мереж.

Таку назву машина придбала як модифікація звичайної машини Больцмана, в якій нейрони розділили на видимі і приховані, а зв'язку допустимі тільки між нейронами різного типу, таким способом обмеживши зв'язку. Значно пізніше, в 2000-х роках, обмежені машини Больцмана набули неабиякої популярності і стали розглядатися вже не як варіації машини Больцмана, а як особливі компоненти в архітектурі мереж глибинного навчання. Об'єднання декількох каскадів обмежених машин Больцмана формує глибоку мережу довіри, особливий вид багатошарових нейронних мереж, які можуть самонавчатися без вчителя за допомогою алгоритму зворотного поширення помилки.

Обмежені машини Больцмана мають широкий спектр застосувань - це завдання зниження розмірності даних, завдання класифікації, коллаборативна фільтрація, виділення ознак (англ. Feature learning) і тематичне моделювання.

В обмеженою машині Больцмана нейрони утворюють двочастковий граф, з одного боку графа знаходяться видимі нейрони (вхід), а з іншого боку - приховані, причому перехресні зв'язки встановлюються між кожним видимим і кожним прихованим нейроном. Така система зв'язків дозволяє застосувати при навчанні мережі метод градієнтного спуску з контрастивної дивергенцией.



Особливістю обмежених машин Больцмана є можливість проходити навчання без учителя,

але в певних додатках обмежені машини Больцмана навчаються з учителем. Прихований шар машини являє собою глибокі ознаки в даних, які виявляються в процесі навчання.

ОМБ може мати багато шарів, з двома підшарами в кожному шарі: видимі одиниці ($\boldsymbol{x}$) називається введенням, а інші називаются прихованими одиницями ($\boldsymbol{h}$).Він переходить від видимого до прихованого, і реконструюється без нагляду, що від прихованого до видимого. Ітераційно вивчає і оновлює вагу, поки не сходиться до мінімальної помилки[16].

21.1.2 Перспективи в графічних моделях

ОМБ є прикладом абабілістичних графічних моделей[17].

Непрямі графічні моделі (НГМ) Не прямі графічні моделі захоплюють симетричні парні кореляції. НГМ являє собою дистрибутив $P(x_1, \cdots, x_n)$ визначається ненаправленим графіком $\mathcal{H}$, та набір позитивних потенційних функцій ϕ_c відповідає кожній клітці $c \in C$ of $\mathcal{H}$ як ця

$$P(x_1, \cdots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c),$$

де $Z = \sum_{x_1, \cdots, x_n} \prod_{c \in C} \phi_c(x_c)$ — це функція розділу. Модель має два компоненти: графічну структуру, яка використовується для отримання властивостей незалежності, і потенційну функцію, яка використовується для отримання представлення дистрибутива.

Машини Больцмана Машини Больцмана - це марковские випадкові поля, де все підключено до всього. Суть в тому, щоб визначити енергію мережі і знайти мінімальне енергетичний стан, зовсім як в фізичних системах. Це повністю пов'язаний графік з парними потенціалами на бінарних вузлах, які можуть бути виражені як підкліки. Спільним розподілом є:

$$\begin{aligned} p(x_1, \cdots, x_n) &= \frac{1}{Z} \exp\left(\sum_{i,j} \phi_{ij}(x_i, x_j)\right) \\ &= \frac{1}{Z} \exp\left(\sum_{i,j} \theta_{ij} x_i x_j + \sum_i \alpha_i x_i + C\right) \\ &= \frac{1}{Z} \exp\left((\boldsymbol{x} - \boldsymbol{\mu})^\top \Theta (\boldsymbol{x} - \boldsymbol{\mu})\right) \end{aligned}$$

Моделі Изинга Моделі Изинга є однією з моделей мережі Маркова, яка вперше виникла в енергії фізичних систем й включає взаємодію атомів. У цих системах кожен атом має двійкову випадкову змінну $X_i \in \{-1, +1\}$, визначення обертання атома. Енергетична функція визначає розподіл:

$$P(\xi) = \frac{1}{Z} \exp\left(-\sum_{i<j} \theta_{ij} x_i x_j - \sum_i u_i x_i\right).$$

[16] Для детального пояснення: https://deeplearning4j.org/restrictedboltzmannmachine
[17] Reference: http://www.cs.cmu.edu/~epxing/Class/10708-14/lecture.html

У дистрибутиві Больцмана змінні приймаються значення у проміжку $\{0, +1\}$ замість$\{-1, +1\}$, але все ще мають цю енергетичну форму. Ненульовий внесок у модель з краю (X_i, X_j) трапляється лише тоді, коли $X_i = X_j = 1$.

Машини Больцмана з обмеженим доступом З видимими одиницями$\boldsymbol{x}$ й схованими одиницями $\boldsymbol{h}$, імовірність функції для ОМБ є:

$$P(\boldsymbol{x}, \boldsymbol{h}|\boldsymbol{\theta}) = \exp\left(\sum_i \theta_i \phi_i(x_i) + \sum_j \theta_j \phi_j(h_j) + \sum_{i,j} \theta_{ij} \phi_{ij}(x_i, h_j) - A(\boldsymbol{\theta})\right).$$

Фактори є незначно залежними, але умовно незалежними заданими спостереженнями на видимих вузлах..

Машини Больцмана з безперервним обмеженим доступом: Машини Больцмана з безперервним обмеженим доступом - це форма ОМБ, яка приймає безперервний вхід (тобто числа, вирізані більш тонкими, ніж цілими числами) за допомогою іншого типу контрастної вибірки дивергенції. Це дозволяє БОМБ обробляти такі речі, як пікселі зображення або вектори підрахунку слів, які нормалізуються до десяткових від нуля до одного. Слід зазначити, що кожен шар глибоковивчої сітки вимагає чотирьох елементів: введення, коефіцієнти, ухил і перетворення (алгоритм активації). Введення є числовими даними, вектором, подається йому з попереднього шару (або як вихідні дані). Коефіцієнти - це ваги, що даються різним особливостям, які проходять через кожен шар вузла. Ухил гарантує, що деякі вузли в шарі будуть активовані незважаючи ні на що. Перетворення є додатковим алгоритмом, який стискає дані після того, як він проходить через кожен шар таким чином, що полегшує обчислення градієнтів (і градієнти необхідні для того, щоб мережа вивчалася). Ці додаткові алгоритми та їх комбінації можуть змінюватися за шаром. Ефективна машина Больцмана з обмеженим доступом використовує гаусове перетворення на видимий (або вхідний) шар і виправлене перетворення лінійного блоку на прихованому шарі. Це особливо корисно при реконструкції обличчя. Для ОМБ обробки двійкових даних, просто зробити обидва перетворення двійкові. Гаусові перетворення не дуже добре працюють на прихованих шарах ОМБ. Виправлені трансформації лінійних блоків, які використовуються замість цього, здатні представляти більше функцій, ніж двійкові перетворення, які ми застосовуємо в глибоких сітках рішень.

21.1.3 Обмежені машини Больцмана для спільної фільтрації:

Видимі одиниці softmax: Це не що інше, як одиниці, серед яких 1 і тільки 1 вузол повинен бути активним. У вашому випадку, це різні рівні рейтингу для кожного елемента. Ідея полягає в тому, що на основі значень на всіх одиницях, ви вирішите, який з них повинен бути активований. Іншими словами, ухил, який ви бачите в інших нейронних мережах, не є динамічним. Ось інший погляд на те ж саме. Ви запитаєте 10 людей, якщо на основі їх досвіду, вам сподобається фільм. Всі вони дадуть різні відповіді на кшталт - Однозначно, Зовсім ні, А небагато і т.д. Функція softmax допомагає вам вибрати один з них з усіх відповідей, які ви отримали.

Відображення оцінок Двійкові карти оцінок в такому форматі:
1. Оцінка 1? Так/Ні
2. Оцінка 2? Так/Ні

й так далі. Кожен вузол активується або деактивується на основі значення, яке він шукає. Вузол один піклується тільки про рейтинг 1 чи ні 1

Відсутні оцінки Ви не маєте справу з відсутніми рейтингами. Залиште всі пов'язані вузли вимкненими / порожніми. Математика подбає про це. Як? це просто. Навчання мережі базується на тому, наскільки хорошим є матч, але саме одне значення має бути обране мережею. Мережа буде намагатися розвиватися, щоб отримати якомога більше відповідних значень, але не вдасться незалежно від того, яке значення він прогнозує. Загальна "помилка"не зміниться. Це означає, що це відсутнє значення не надаватиме жодної інформації як "дедиференціал". Таким чином, алгоритм в кінцевому підсумку буде впливати на це відсутнє значення. І саме тут входить частина спільної фільтрації. Вся потужність RBM полягає в тому, що оскільки він знайшов відсутні значення, тепер він призначатиме значення на основі оцінок інших подібних людей.

22 Навчання з підкріпленням

В той час як навчання з вчителелем і навчання без вчителя не сильно відрізняються одне від одного, навчання з підкріпленням використовує зовсім інший підхід до навчання. В начанні з вчителем ми маємо наступний інтерфейс:

1. Підбір функції від X,Y
2. Передбачення результату в X.

В навчанні без вчителя ми маємо наступний інтерфейс:

1. Підбір функції від X.
2. Перетворення X

У випадках навчання з вчителем і без вчителя ми маємо загальну ідею: Інтерфейс це навчальні дані, і ми або робимо прогнози відносо невідомих даних або ми дізнаємося деяку цікаву інформацію з відомих.

Замість цього в навчанні з підкріпленням ми хочемо навчити агента діяти в реальному (або змодельованому) середовищі. Інтерфейс набагато ширший ніж просто вектор даних. Навчання з підкріпленням проходить по іншому: Метою агента в навчанні з підкрвпленням є **ціль**. При навчанні з вчителем ми намагаємося максимізувати ймовірність або мінімізувати ціну. Агентам при навчанні з підкріпленням надається зворотній зв'язок(у вигляді винагороди), коли вони взвємодіють з навколишнім середовищем. Таким чином метою агента є максимізація винагороди або мінімізаія штрафів. При навчанні з вчителем нам потрібні ярлики, але при навчанні з підкріпленням зворотній зв'язок(винагорода) автоматично надається навколишнім середовищем. Так, наприклад, мета ШІ відеогри - виграти або набрати найвищий бал. Є декілька важливих компонентів, які використовуються в навчанні з підкріпленням(далі - НП):

1. Агент: Те, чому ми намагаємося вчити, і це поміщається в навколишнє середовище.
2. Середовище: Реальний або змодельований світ в якому знаходиться агент.

3. Стан: Різні конфігурації середовища які може сприймати агент.

4. Винагорода: Агент отримує винагороду за дії. Агент хоче максимізувати не тільки негайну винагороду, але і майбутню. Концепція майбутніх винагород є важливою, до неї ми ще повернемося. Структура винагороти має велике значення для навчання агента. Наприклад, ми намагагаємося навчити робота проходити лабіринт. Мета: пройти лабіринт. Нагрода = 1 якщо пройшов, нагорода = 0, якщо не пройшов. Це хороші налаштування? Ні, оскільки робот буде рухатися випадково поки не знайде вихід. Не дуже вигідна стратегія. ми хочемо щоб робот знаходив вихід за мінімальну можливу кількість дій.

   Замість цього ми повинні давати винагороду = -1 за кожен крок. Для максимізації винагороди, роботу потрібно навчитися шукати вихід з лабіринту за мінімально можливу кількість кроків.

5. Дії: це те що агент робить в своєму середовищі в деякому теперішньому стані. Ми можемо думати про стан, дію і нагороду як про трійку (s,a,r). Ми починаємо в деякому стані S(t), ми виконуємо дію A(t), ми отримуємо винагороду R(t+1) і переходимо в стан S(t+1).

22.1 Марковські процеси прийняття рішень

Марковська властивість:

Дана послідовність $x_1, x_2..., x_t$ припустимо $p_{x_t}|x_{t-1}, x_{t-2}, ..x_1$ Зазвичай це не можна спростити, але з допомогою властивості Маркова можна.

1. Перший Марковський порядок: $p_{x_t}|x_{t-1}, x_{t-2}, ..x_1 = p_{x_t}|x_{t-1}$

2. Другий марковський порядок: $p_{x_t}|x_{t-1}, x_{t-2}, ..x_1 = p_{x_t}|x_{t-1}, x_{t-2}$.

Отже, перший порядок означає, що x_t залежить тільки від x_{t-1}.
Марковська властивість в навчанні з підкріпленням:

$$P[S_{t+1}, R_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ..S_0, A_0] = P[S_{t+1}, R_{t+1}|S_t, A_t] \tag{96}$$

В загальному вигляді:

$$p(s', r|s, a) = P[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a] \tag{97}$$

Будь-яка задача НП з набором станів, дій і нагород, що відповідає Марковській властивості являє собою Марковський процес вирішування(МПВ). МПВ визначається як сукупність:

1. набір станів

2. набір дій

3. набір винагород

4. ймовіронсті переходу між станами, ймовірності винагороди (як показано вище)

5. фактор знижки

22.2 Політика

Політика, позначена π , представляє алгоритм, який агент буде використовувати для навігації по середовищу.

22.3 Імовірності переходу станів

$p(s', r|s, a)$ Виділене середовище походить лише від того, що виокремлює агент, і не є самим середовищем. Виділена зона це неідеальне уявленням про навколишнє середовище. Напр. Блекджек: якщо ви агент, наступна карта може не бути частиною вашої виділеної зони, але є частиною середовища.

22.4 Майбутні винагороди

Ми зацікавлені у вимірі загальної майбутньої винагороди, яку ми можемо накопичити, перебуваючи в деякиому станоі S (t) (Майбутня винагорода - це все, що починається з t + 1 і далі. Ми називаємо це **return**, G(t).

Примітка: Не враховується поточна винагорода R (t). Це винагорода, отримана від поточного стану.

$$G(t) = \sum_{\tau=1} \gamma^{\tau}(t + \tau + 1) \tag{98}$$

22.4.1 Фактор знижки

Gamma = 1: не має значення, як далеко в майбутньому буде отримана винагорода. Ваги всіх нагород однаково.

Gamma = 0; Воістину жадібна стратегія. Спробуйте лише максимізувати негайні винагороди. Зазвичай ми вибираємо щось ближче до 1. тобто 0.9

22.5 Функція значення та рівняння Беллмана

У кожному стані s, я отримаю винагороду R. Загальне величина яку повертаємо G - це сума виграшів, які отримано. Ми хочемо знати: Якщо я перебуваю в стані s, яка середня сума винагород, яку я отримаю в майбутньому?

$$V(s) = E(G|s) \tag{99}$$

Враховуючи стан s, яка очікувана вартість майбутніх винагород?

$$V(s) = E(r + \gamma V(s')) \tag{100}$$

s = поточний стан, s '= наступний стан, r = винагорода від переходу від s до s '.

Ми бачимо, що функція значення - це рекурсивна функція. Це дорівнює винагороді від перебування в штаті плюс майбутні винагороди, які ми отримуємо в середньому від перебування у штаті. Тому ми можемо навіть розширити рівняння:

$$V(s) = E(r + \gamma E[r' + \gamma V(s'')]) \tag{101}$$

Розширенням функції значення V є ще одна функція значення Q. Q не тільки залежить від стану s, але й дії a.

1. V(s) = функція стану-значення
2. Q(s, a) = функція значення дії

$$Q(s,a) = E[G|s,a] = E[r + \gamma V(s')|s,a] \tag{102}$$

V (s) показує нам, яка очікувана майбутня віддача, враховуючи те, що я перебуваю у штаті s. Q (s, a) повідомляє нам, якою є моя майбутня очікувана віддача, враховуючи те, що я перебуваю у стані s I Я враховую a. Для того, щоб зрозуміти, як Q і V пов'язані, нам потрібно поглянути на політику. (Щось, що підказує нам, що робити, враховуючи те, що ми отримали дані у станах)

22.5.1 Розширена функція значення

Функція значення визначається політикою та має параметр стану s. Будь-які майбутні переходи станів залежать від вашої політики. Тільки майбутні винагороди сприяють функції вартості. Тому значення термінального стану дорівнює 0.

$$V_\pi(s) = E_\pi[G(t)|S_t = s] = E_\pi[\sum_{\tau=0} \gamma^\tau R(t+\tau+1)|S_t = s] \tag{103}$$

Оскільки очікуване значення перевищує пі, це означає, що ми можемо виразити це як розподіл ймовірностей. Імовірність зробити дію a, враховуючи, що ми перебуваємо в стані s.

$$\pi = \pi(a|s) \tag{104}$$

Оскільки очікувані значення є лінійними операторами, ми можемо знайти кожен доданок по одному:

$$E_\pi[R(t+1)|S_t = s] = \sum_a \pi(a|s) \sum_r rp(r|s,a) \tag{105}$$

Оскільки все стохастичне, наведене рівняння регулюється двома розподілами ймовірностей: По-перше, існує політика $\pi(a|s)$, Імовірність зробити дію a, враховуючи, що ми перебуваємо в стані s. Як тільки у нас є пара станів, ми також маємо $p(s', r|s, a)$ які ми можемо маргіналізувати до $p(r|s, a)$.

22.5.2 Рівняння Беллмана

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)r + \gamma V_\pi(s') \tag{106}$$

Отже, якщо я агент, який грає в хрестики-нулики, і я хочу знати, що мені робити далі, я хочу знати, яка винагорода, якщо я піду на s і яке значення s '.

Література

Avoiding degradation in deep feed-forward networks by phasing out skip-connections. 2018.

D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. C. Courville, and Y. Bengio. An actor-critic algorithm for sequence prediction. *CoRR*, abs/1607.07086, 2016. URL http://arxiv.org/abs/1607.07086.

Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *In Advances in Neural Information Processing Systems*, 14:585–591, 2002.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994. ISSN 1045-9227. doi: 10.1109/72.279181.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.50.

D. Britz. Recurrent neural networks tutorial, part 3 – backpropagation through time and vanishing gradients, Apr 2016. URL http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/.

A. Canziani. t.1 – capsules and routing techniques (part 1/2). *Youtube*, 2017. URL https://www.youtube.com/watch?v=EATWLTyLfmc.

A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016. URL http://arxiv.org/abs/1605.07678.

D. Cole. The chinese room argument. 2004.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research 12 (2011) 2493-2537*, 2011.

D. Crevier. *AI: the tumultuous history of the search for artificial intelligence*. Basic Books, 1993.

Demartines and J. Herault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, 1997.

J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848.

V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

R. Eldan and O. Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

M. Espi, M. Fujimoto, K. Kinoshita, and T. Nakatani. Exploiting spectro-temporal locality in deep learning based acoustic event detection. *Springer*. URL https://link.springer.com/content/pdf/10.1186%2Fs13636-015-0069-2.pdf.

A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. ISBN 9781491962244. URL https://books.google.com/books?id=bRpYDgAAQBAJ.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

R. Goroshin, M. Mathieu, and Y. LeCun. Learning to linearize under uncertainty. *CoRR*, abs/1506.03011, 2015. URL http://arxiv.org/abs/1506.03011.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016b.

G. Hinton and S. Roweis. Stochastic neighbor embedding. *In Advances in Neural Information Processing Systems*, 15:833–840, 2002.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach and D. M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015a. URL http://dblp.uni-trier.de/db/conf/icml/icml2015.html#IoffeS15.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015b.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016. URL http://arxiv.org/abs/1607.01759.

Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Y. Kim and A. M. Rush. Sequence-level knowledge distillation. *CoRR*, abs/1606.07947, 2016. URL http://arxiv.org/abs/1606.07947.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. J. Huang. A tutorial on energy-based learning. 2006. URL http://yann.lecun.com/.

X. Li, S. Chen, X. Hu, and J. Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *Cornell University Library*, 2018. URL https://arxiv.org/abs/1801.05134.

M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

L. v. d. Maaten. Learning with embeddings. 2018.

A. Makhzani and B. J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013. URL http://arxiv.org/abs/1312.5663.

MatlabTricks. 3x3 convolution kernels with online demo. URL http://matlabtricks.com/post-5/3x3-convolution-kernels-with-onli-ne-demo.

P. McCorduck. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. AK Peters/CRC Press, 2009.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL http://arxiv.org/abs/1301.3781.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, pages 3111–3119, 2013b.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

J. Park and Y. Ni. Sparse autoencoder project report, unpublished. 2017.

K. Pathy. Stanford cs231n class. 2018.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

A. Pritchard, R. Horne, and S. Sangwine. Rational arithmetic representation of colour image pixels. *Electronics Letters. 30 (18)*, 1994.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1): 145–151, 1999.

T. M. Quoc Le. Distributed representations of sentences and documents. 2014.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732, 2015. URL http://arxiv.org/abs/1511.06732.

Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 (5500):2323–2326, 2000.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017. URL http://arxiv.org/abs/1710.09829.

Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18 (5):401–409, 1969.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. URL http://arxiv.org/abs/1312.6229.

A. Shah, E. Kadam, H. Shah, and S. Shinde. Deep residual networks with exponential linear unit. *CoRR*, abs/1604.04112, 2016. URL http://arxiv.org/abs/1604.04112.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. *CoRR*, abs/1507.06228, 2015. URL http://arxiv.org/abs/1507.06228.

S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015. URL http://arxiv.org/abs/1503.08895.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

S. Targ, D. Almeida, and K. Lyman. Resnet in resnet: Generalizing residual architectures. *CoRR*, abs/1603.08029, 2016. URL http://arxiv.org/abs/1603.08029.

V. d. S. Tenenbaum and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Torgerson. Multidimensional scaling i: Theory and method. *Psychometrik*, 17:401–419, 1952.

UFLDL. Pooling. *Unsupervised Feature Learning and Deep Learning Tutorial*. URL http://ufldl.stanford.edu/tutorial/supervised/Pooling/.

L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

J. Wang, L.-C. Yu, K. R. Lai, and X. Zhang. Dimensional sentiment analysis using a regional cnn-lstm model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 225–230, 2016.

M. Wattenberg, F. Viegas, and I. Johnson. How to use t-sne effectively, Oct 2017. URL https://distill.pub/2016/misread-tsne/.

J. Weston, S. Chopra, and A. Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. *CoRR*, abs/1606.02960, 2016. URL http://arxiv.org/abs/1606.02960.

B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

W. Yin, K. Kann, M. Yu, and H. Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.

S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015a.

X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015b. URL http://arxiv.org/abs/1509.01626.

J. Zhao, M. Mathieu, R. Goroshin, and Y. LeCun. Stacked What-Where Auto-encoders. *ArXiv e-prints*, June 2015.

J. J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *CoRR*, abs/1609.03126, 2016. URL http://arxiv.org/abs/1609.03126.

J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks.