Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Випускна кваліфікаційна робота магістра

на тему:

"Багатозадачні нейронні мережі як багатозадачна оптимізація"

Студента 2-го курсу магістратури
кафедри обчислювальної математики
Чан Ха Ву

Науковий керівник:
професор, доктор фізико-математичних наук
Клюшин Дмитро Анатолійович

Робота заслухана на засіданні кафедри обчислювальної математики
та рекомендована до захисту в ДЕК
протокол №11 від 15 травня 2019 року
Завідувач кафедри обчислювальної математики
проф. Ляшко С.І.

Київ – 2019

Анотація

В цій роботі, ми провели детальний аналіз існуючих методів тренування багатозадачних нейронних мереж. На базі цих методів, ми запропонували простий гібридний алгоритм, який працює швидше існуючих методів але все одно гарантує потраплення в точки стаціонарності за Парето.

Основним результатом цієї роботи є запропонований евристичний алгоритм автоматичного пошуку архітектури багатозадачної нейронної мережі. Цей алгоритм позбавлений від деяких обмежень попередніх робіт, та наші експерименти показують, що наша евристична оцінка відповідає візуальним інтуїтивним представленням про схожі задачі серед обраних датасетів.

В рамках цієї роботи, ми розробили універсальний фреймворк Hydra[1] для побудови, тренування, та автоматичного пошуку архітектури багатозадачних нейронних мереж.

[1]Доступний за посиланням github.com/hav4ik/Hydra під відкритою ліцензією MIT.

Зміст

1 Вступ

1.1 Глибинне навчання

Сучасні алгоритми та системи роспізнавання образів спираються на конволюційних нейроних мережах, які уперше були використані в роботах [1] та набрали популярність після прориву [2]. З тих пір, дуже велика варіація архітектур були запропоновані для вирішення різних задач переважно комп'ютерного зору, таких як класифікації [2, 3, 4, 5, 6, 7], детекції об'єктів [8, 9, 10, 11], а також загальні архітектури які оптимізовані для роботи на мобільних пристроїв [12, 13, 14]. Ці мережі найчастіше зустрічаються під спільною назвою *глибокі нейронні мережі*, а галузь що їх вивчає — *глибинне навчання*.

1.2 Багатозадачне навчання

Одним з важливих підрозділів навчання глибинних мереж є *багатозадачне навчання* [15], де замість оптимизації однієї цільової функції (або однієї *задачі*), нейронна мережа навчається оптимізувати декілька різних цільових фукнції (в літературі це називають *різними завданнями*).

Основна мотивація такого класу методів є тим, що допоміжні набори даних можуть бути використані для поліпшення загальної продуктивності за рахунок використання закономірностей, що існують у різних задачах. Так, наприклад, цю ідею використовували в задачах комп'ютерного зору [16, 17, 18, 19, 20, 21, 22], роспізнавання тексту [23, 24, 25], та навчання з підкріпленням [26, 27, 28, 29]. В цих роботах, завдяки парадігми *багатозадачних нейронних мереж*, вдалося значно покращити результати навчання основної задачі.

1.3 Мотивація

Основною мотивацією цієї роботи є покращення практичного аспекту алгоритмів навчання *багатозадачних глибоких нейронних мереж*, а саме:

- Отримати прискорення (за часом тренування та за кількістю епох) для найкращих серед існуючих методів *багатозадачного навчання*.
- Систематизувати процес побудови архітектури *багатозадачної нейронної мережі*.

1.4 Основні результати

В цій роботі, комбінуючи існуючі методи оптимізації *багатозадачних нейронних мереж*, а також методів з розділів *прунінгу* та *інтерпретативності* нейронних мереж, ми зробили наступні внески:

- У розділі 4, ми сформулювали простий гібридний алгоритм навчання *багатозадачних нейронних мереж*, який значно швидше ніж [30], та водночас гарантовано досягає точки стаціонарності за Парето.
- Ми запропонували у розділі 5 показник *внутрішнього стресу* блоків *багатозадачної нейронної мережі* з деревоподібною архітектурою (розділ 3). Експериментально показуємо, що такий показник у мережах з однаковою складністю (за кількістю обчислень) корелюється з якістю роспізнавання цієї мережі.
- Ми розробили фреймворк Hydra[2] для ефективного тренування *багатозадачних нейронних мереж* з узагальними архітектурами (розділ 3). Вона є достатньо гнучкою, щоб в неї можно було легко реалізувати майже будь-який з методів, перелічених у списку літератур цієї роботи.

[2]Доступний за посиланням github.com/hav4ik/Hydra під відкритою ліцензією MIT.

2 Огляд літератури

Один з найдивніших результатів у статистиці — парадокс Штейна. В своїй роботі [31] Штейн показав, що краще оцінювати середнє значення трьох або більше гауссівських випадкових величин використовуючи зразки з усіх них, ніж оцінювати їх окремо, навіть коли гаузіани незалежні. Парадокс Штейна став мотивацієї багатозадачних нейронних мереж [15] — парадигм машиного навчання у которому дані для різних задач комбінуються з надією покращення якості навчання для кожного з задач.

Потенційні переваги *багатозадачного навчання* виходять за рамки прямих наслідків парадокса Штейна, оскільки навіть не пов'язані між собою завдання реального світу мають деякі залежності через спільних процесів, що породжують ці дані. Наприклад, незважаючи на те, що автономне водіння і маніпулювання об'єктом є незв'язаними, основні дані регулюються тими ж законами оптики, властивостями матеріалу і динамікою. Це мотивує використання декількох завдань як індуктивного зміщення в системах навчання.

У цьому розділі, ми розглянемо останні підходи навчання *багатозадачних нейронних мереж*, які стали мотивацією для цієї роботи. Зацікавленим читачам раджемо ознайомитися з більш повним оглядом літератури *багатозадачного навчання* [32], [33], та [34].

2.1 Оптимізація багатозадачних мереж

У роботі [35], проблему *багатозадачного навчання* розглядається як взаємодія індивидуальних агентів та мета-алгоритму: кожний агент відповідає за одну задачу, а мета-алгоритм вирішує як оновлювати спільні параметри кожного агенту.

Роботи [16, 17, 18, 19, 20, 21, 22, 23, 24, 25], не зважаючи на те що вони отримали значне покращення якості роспізнавання, використовують просте зважене підсумування як мета-алгоритм. Недолік такого підходу полягає у тому, що воно ігнорує *проблему несбалансування*: градієнти завдань, які є занадто домінуючим під час навчання, обов'язково матиме відносно великі величини.

Цей недолік був успішно ліквідований емпіричними методами [36, 37] та більш повільним, але теоретично обгрунтованим методом [30] який пока-

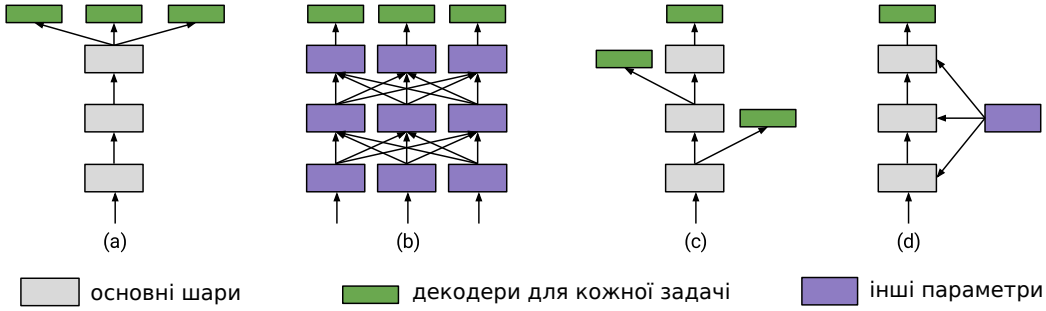


Рис. 1: Основні різновиди архітектури *багатозадачної нейронної мережі*. (a) класичні, (b) стовпчик для кожної задачі, (c) розгалудження на різних рівнях, (d) універсальні представлення

зує як можна досяги стаціонарності за Парето в проблемах *багатозадачного навчання*. Ці роботи є основою для нашого швидкого алгоритму у розділі (4). Зацікавленим читачам раджується ближче та більш детально ознайомитись з цими методами у повному обзорі [38].

2.2 Архітектури

Проектування глибокої системи *багатозадачного навчання* вимагає відповіді на ключове питання: яка найкраща архітектура нейронної мережі? Існуючі підходи можна класифікувати на 4 види, схематично представлені на Рис. 1, в залежності від того як вони відповідають на це запитання.

2.2.1 Класичні

Ці підходи дуже схожі на [15]: мають спільний *енкодер* (сірим кольором) що вивчає спільне представлення на шарі високого рівня, за яким слідують *декодери*, специфічні для кожного завдань, які повертають мітки для кожного завдання. Прикладами цього підходу є роботи [26, 39, 23, 28, 40, 20, 25, 41].

2.2.2 Стовпчик для кожної задачі

Ідея цих підходів полягає у тому, що призначається кожному завданню власний шар параметрів, на кожній загальній глибині. Потім вони

визначають механізм для спільного використання параметрів між завданнями на кожному рівні глибини мережі, наприклад за допомогою спільного тензора [21], або дозволяють деяку форму зв'язку між стовпцями [17, 19, 29]. Спостереження за негативними ефектами спільного використання в методах на основі стовпців [29] можна віднести до невідповідності між фічами, необхідними на одній глибині між завданнями, які занадто різноманітні.

2.2.3 Розгалудження на різних рівнях

Може існувати інтуїтивна ієрархія, що описує, як зв'язаний набір завдань. Кілька підходів інтегрують контрольований зворотний зв'язок від кожного завдання на рівні, що відповідає такій ієрархії [42]. Цей метод може бути чутливим до вибору ієрархії [43] і до вибору завдань [42]. Один з підходів вивчає ієрархію відносин між завданнями під час навчання [18], незважаючи на те, що параметри тільки діляться на відповідній глибині.

2.2.4 Універсальні представлення

Цей клас підходів поділяє всі основні параметри моделі, крім коефіцієнтів масштабування *батч нормалізації* [44]. Коли кількість класів однакове для всіх завдань, навіть вихідні шари можуть бути спільними, а невелика кількість специфічних для кожного завдання параметрів дозволяє мінімізувати кількість памяті для зберігання моделі. Остання робота у цьому напрямку [45] ще дозволяє *блокам* багатозадачної нейронної мережі бути переставленими у різному порядку для кожного з задач.

2.3 Алгоритми пошуку архітектури

Усі роботи, що були нами розглянуті до тих пір (окрім роботи [45]) використовують *статично побудовані* архітектури, тобто їх побудували опираючись на людській інтуіції. Клас методів, що дозволяють автоматизувати процес підбору оптимальної архітектури нейронної мережі для розв'язування конкретних задач, називають методами *пошуку архітектур нейронних мереж* [46, 47, 48]. Для *багатозадачного тренування*, прикладами методів автоматичного пошуку архітектури є роботи [18, 45, 49, 50, 51].

3 Деревоподібні архітектури

В цьому розділі, введемо деякі позначення, які будуть достатньо зручними для опису процесу маніпуляції архітектурами, а також для програмної реалізації.

3.1 Узагальнена архітектура

Нехай маємо простір вхідних даних $\mathcal{X}$ та простори бажаних вихідних даних $\{\mathcal{Y}^t\}_{t\in[T]}$, де T — кількість задач. Також маємо *датасет* $\mathcal{D} = \{\boldsymbol{x}_i, y_i^1, \dots y_i^T\}_{i\in[N]}$, де N — кількість *семплів датасету* (на практиці це кількість *батчів*), $\boldsymbol{x}_i$ — i-й *семпл* датасету, y_i^t — бажаний вихід нейронної мережі для t-й задачі для i-го *семплу датасету*.

У підрозділі 2.2, ми розглянули основні 4 типи архітектур *багатозадачної нейронної мережі*. Усі ці типи архітектур можна узагальнити таким зручним для обчисленні та реалізації чином:

Означення 3.1 *Багатозадачною нейронною мережею з узагальненою архітектурою назвемо* $\mathcal{F} = (\boldsymbol{B}, \boldsymbol{P}, \boldsymbol{\theta})$*, де:*

- $\boldsymbol{B} = \{f_1, f_2, \dots, f_n\}$ — *множина блоків мережі, де кожний блок* f_i *це деяка функція (також виражається нейронною мережею) з параметрами* $\boldsymbol{\theta}_i$.
- $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_n\}$ — *множина параметрів (ваг) мережі.*
- $\boldsymbol{P} = \{p^1, p^2, \dots, p^T\}$, *де* $p^t = \{s_i^t\}_{i=1}^{l(t)}$ — *впорядкований ланцюг індексів елементів* $\boldsymbol{B}$ *без циклу, для кожної задачі* t.
- *Вихід мережі для* t*-й задачі над семплом* $\boldsymbol{x}$ *обчислюється проходом по ланцюгу:* $\mathcal{F}^t(\boldsymbol{x}) = f_{s_{l(t)}^t} \circ \dots \circ f_{s_2^t} \circ f_{s_1^t}(\boldsymbol{x}) \in \mathcal{Y}^t$.

Відображення f_i на практиці найчастіше обираються такими, що за архітектурою схожі на один з [2, 3, 4, 5, 6, 7, 12, 13, 14]. Зауважимо, що фреймворк Hydra[3] дозволяє ефективно працювати саме з такими нейронними мережами з узагальненою архітектурою. Зауважимо також, що ланцюги з $\boldsymbol{P}$ може змінюватись під час тренування (наприклад, як наслідок алгоритмів пошуку архітектури).

Для кожного блоку з індексом i, позначимо $\mathcal{O}_{\mathcal{F}}(i) = \{t \colon i \in p^t\}$, тобто це множина усіх задач, що містять блок з індексом i як елемент свого лан-

[3] Доступний за посиланням github.com/hav4ik/Hydra під відкритою ліцензією MIT.

цюга обчислення. У розділі 4, покажемо як можна узагальнити методи [37] та [30] до цих архітектур.

3.2 Деревоподібна архітектура

У цьому підрозділі, розглянемо клас архітектур, до якої будемо застосовувати алгоритми пошуку архітектури у розділі 5.

Означення 3.2 *Багатозадачна нейронна мережа з узагальненою архітектурою $\mathcal{F} = (\boldsymbol{B}, \boldsymbol{P}, \boldsymbol{\theta})$ назвемо деревоподібною, якщо:*

(а) Існує єдиний $f_r \in \boldsymbol{B}$ такий, що для всіх шляхів $p^t \in \boldsymbol{P}$, f_r є першим елементом цього шляху, тобто $f_{s_1^t} \equiv f_r$. Цей елемент назвемо коренем.

(б) Для кожного елементу $f_k \in \boldsymbol{B}$ що не є коренем, виконується наступне: існує єдиний елемент f_j такий, що для всіх шляхів $\{s_i^t\}_{i=1}^{l(t)} \in \boldsymbol{P}$ що містить f_k, тобто $s_{i(k)}^t = k$, попередній елемент шляху завжди f_j, тобто $s_{i(k)-1}^t = j$.

З такими додатковими умовами, таке означення описує класичні архітектури (2.2.1) та архітектури з розгалудженням на різних рівнях (2.2.3). Позначимо також $\mathcal{C}_{\mathcal{F}}(i)$ множину індексів усіх блоків, для яких f_i є попереднім елементом (інакше кажучи, $\mathcal{C}_{\mathcal{F}}(i)$ — множина індексів нащадків блоку з індексом i).

За допомогою означення 3.2, задамо наступні операції над такою деревоподібною архітектурою:

- $peel(\mathcal{F}, t) = (\boldsymbol{B}^t, \boldsymbol{\theta}^t, \boldsymbol{P}^t)$ — склонована однозадачна нейронна мережа, що еквівалентна підмережі $\mathcal{F}$ що відповідає задачі t, тобто складається з блоків $\boldsymbol{B}^t = \{f_i' \equiv f_i | i \in p^t \in \boldsymbol{P}\}$ з параметрами $\boldsymbol{\theta}^t = \{\boldsymbol{\theta}_i' \equiv \boldsymbol{\theta}_i | i \in p^t \in \boldsymbol{P}\}$ та тільки одним шляхом $\boldsymbol{P}^t = \{p^t\}$.
- $split(\mathcal{F}, i, \pi) = \mathcal{F}_{splitted}^{i,\pi} = (\boldsymbol{B}', \boldsymbol{\theta}', \boldsymbol{P}')$ — нейронна мережа, отримана шляхом заміни блоку f_i блоками $f_1', \ldots, f_{|\pi|}'$, де $\pi = \{\pi_1, \ldots, \pi_k\}$ — деяке покриття над множиною нащадків $\mathcal{C}_{\mathcal{F}_{splitted}^{i,\pi}}(i)$ блоку i, при чому у кожному ланцюгу p^t, якщо воно містить елемент з π_j, то в цьому ланцюзі f_i заміняється f_j'.

Ці дві операції є основними що будуть використані в процеці пошуку архітектури у розділі 5.

4 Багатокритеріальна оптимізація

В цьому розділі, ми розглянемо тренування багатозадачної глибинної нейронної мережі як задачу багатокритеріальної оптимизації, та узагальнимо результати робіт [30, 37, 52] до випадку архітектур у розділі 3. Також, сформулюємо гібридний метод який буде швидше сходитись до точки стаціонарності за Парето. В кінці цього розділу перевіримо єфективність нашого алгоритму емпіричним шляхом.

4.1 Постановка задачі оптимізації

Розглянемо задачу багатоцільової оптимізації над простором вхідних даних $\mathcal{X}$ та просторами бажаних вихідних даних $\{\mathcal{Y}^t\}_{t\in[T]}$, де T — кількість задач, тобто кількість цільових функції яких ми будемо оптимізувати. Також розглянемо *датасет* $\mathcal{D} = \{\boldsymbol{x}_i, y_i^1, \dots y_i^T\}_{i\in[N]}$, де N — кількість *семплів датасету* (на практиці це кількість *батчів*), $\boldsymbol{x}_i$ — i-й *семпл* датасету, y_i^t — бажаний вихід нейронної мережі для t-й задачі для i-го *семплу датасету*. Для кожної задачі, розглянемо цільову функцію $\mathcal{L}^t(\cdot,\cdot): \mathcal{Y}^t \times \mathcal{Y}^t \to \mathbb{R}^+$. Також, припустимо, що наша багатозадачна нейронна мережа параметризується набором ваг $\boldsymbol{\theta}$, та її вихід для кожної задачі позначимо як $\mathcal{F}^t(\boldsymbol{x};\boldsymbol{\theta})$.

Формулювання задачі оптимізації параметру $\boldsymbol{\theta}$ у більшості попередніх робіт де використовуються багатозадачні нейронні мережі [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], незважаючи на те що вони використовують різні архітектури та цільові функції для кожної задачі, можна описати узагальнено у наступній формі:

$$\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} c^t \hat{\mathcal{L}}^t(\boldsymbol{\theta}) \tag{4.1}$$

для деякого статичного чи динамічно обчислюваного набору коефіцієнтів c^t для кожної задачі, де $\hat{\mathcal{L}}^t(\boldsymbol{\theta})$ — емпіричне значення цільової функції задачі t, яка обчислюється як середнє його значення на деякому *батчі датасету*:

$$\hat{\mathcal{L}}^t(\boldsymbol{\theta}) \triangleq \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \mathcal{L}^t\left(\mathcal{F}^t(\boldsymbol{x}_i;\boldsymbol{\theta}), y_i^t\right) \tag{4.2}$$

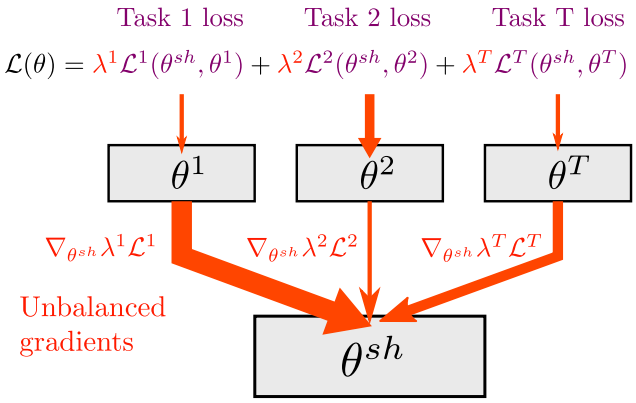


Рис. 2: Проблема балансування градієнтів. Нехай маємо багатозадачну нейронну мережу, з параметрами декодера $\boldsymbol{\theta}^sh$ та параметрами енкодерів $\boldsymbol{\theta}^t$. Може трапитись така ситуація, що градієнти від однієї задачі у багато разів більше ніж градієнти від інших задач, у наслідку чого нейронна мережа буде ігнорувати інші задачи при тренуванні.

де $\mathcal{B}$ — набір індексів цього *батчу*. Не зважаючи на те що таке формулювання є досить інтуїтивною, питання вибору вдалого набору таких коєфіцієнтів є достатньо складною.

Таке формулювання звичайно потребує додаткової оптимізації гіперпараметрів $\{c^t\}_{t\in[T]}$, що зазвичай призводить до використання еволюційних алгоритмів або алгоритмів перебору. Такий процес може займати досить багато часу — одна нейронна мережа може тренуватись декілька днів, тому перебір гіперпараметрів може займати декілька тижнів. В роботах [36, 37] були використані евристичні оцінки для динамічного обчислення коєфіцієнтів $\{c^t\}_{t\in[T]}$ — це набагато швидше ніж пошук гіперпараметрів перебором.

Проблема формулювання (4.1) полягає у тому, що вона не дозволяє задавати критерії глобальної оптимальності. Розглянемо набори параметрів $\boldsymbol{\theta}$ та $\bar{\boldsymbol{\theta}}$ для яких:

$$\hat{\mathcal{L}}^{t_1}(\boldsymbol{\theta}) < \hat{\mathcal{L}}^{t_1}(\bar{\boldsymbol{\theta}}) \quad \text{та} \quad \hat{\mathcal{L}}^{t_2}(\boldsymbol{\theta}) > \hat{\mathcal{L}}^{t_2}(\bar{\boldsymbol{\theta}}) \ , \tag{4.3}$$

для деяких задач t_1 та t_2. Іншими словами, набір $\boldsymbol{\theta}$ є кращими за критерієм $\hat{\mathcal{L}}^{t_1}$ та набір $\bar{\boldsymbol{\theta}}$ є кращим за критерієм $\hat{\mathcal{L}}^{t_2}$. Отже, ці набори параметрів не можна беспосередньо порівнювати без попарної значущості задач, що в практичних задач звичайно не передбачається. Для уникнення цієї проблеми, у роботі [30] пропонується інший підхід, де використовується

поняття *оптимальності за Парето* — економічний термін, який описує такий стан системи, при якому значення кожного окремого критерію не може бути покращено без погіршення становища інших критеріїв.

Означення 4.1 (домінація за Парето) *Будемо говорити, що набір параметрів $\boldsymbol{\theta}$ домінує за Парето (або просто домінує) над набором $\bar{\boldsymbol{\theta}}$, якщо $\hat{\mathcal{L}}^t(\boldsymbol{\theta}) \leq \hat{\mathcal{L}}^t(\bar{\boldsymbol{\theta}})$ для усіх задач t, та існує t^* такий, що $\hat{\mathcal{L}}^t(\boldsymbol{\theta}) < \hat{\mathcal{L}}^t(\bar{\boldsymbol{\theta}})$.*

Інакше кажучи, якщо набір параметрів $\boldsymbol{\theta}$ домінує за Парето над набором $\bar{\boldsymbol{\theta}}$, то вона не гірше за усіма критеріями, та за деяким критерієм вона навіть краща.

Означення 4.2 (оптимальність за Парето) *Набір параметрів $\boldsymbol{\theta}^*$ будемо називати оптимальною за Парето, якщо не існує такого набору $\boldsymbol{\theta}$, що домінує над $\boldsymbol{\theta}^*$.*

Зауважимо, що множина усіх наборів параметрів що є оптимальною за Парето називають *множиною парето* та позначають $\mathcal{P}_{\boldsymbol{\theta}}$.

Таке формулювання оптимальності дозволяє нам розглядати не повний порядок зваженого значення критерії (4.1) для наборів параметрів, а більш сильний порядок. Отже, можемо поставити задачу багатоцільової оптимізації у розумінні Парето оптимальності наступним чином:

$$\min_{\boldsymbol{\theta}} \boldsymbol{L}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \left(\hat{\mathcal{L}}^1(\boldsymbol{\theta}), \hat{\mathcal{L}}^2(\boldsymbol{\theta}), \ldots, \hat{\mathcal{L}}^T(\boldsymbol{\theta})\right)^\intercal \tag{4.4}$$

Саме такий критерій оптимізації ми будемо розглядати надалі у цієї роботі.

4.2 Багато-градієнтний спуск

Аналогічно випадку оптимізації однієї цільової функції, задача багатоцільової оптимізації теж можна вирішити градієнтним спуском. У цьому підрозділі, опишемо один з таких методів, який називається *алгоритмом багато-градієнтного спуску* [53].

Нехай $\mathcal{F} = (\boldsymbol{B}, \boldsymbol{P}, \boldsymbol{\theta})$ — багатозадачна нейронна мережа з узагальненою архітектурою (см. розділ 3).

Означення 4.3 *Набір параметрів $\boldsymbol{\theta}_i \in \boldsymbol{\theta}$ задовольняє умові Каруша–Куна–Таккера, якщо існує $\{\alpha^t\}_{t\in\mathcal{O}_\mathcal{F}(i)} \geq 0$ таке, що $\sum_{t\in\mathcal{O}_\mathcal{F}(i)} \alpha^t = 1$ та $\sum_{t\in\mathcal{O}_\mathcal{F}(i)} \alpha^t \nabla_{\boldsymbol{\theta}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) = 0$.*

Таке формулювання узагальнює [30] до випадку узагальнених архітектур (см. розділ 3). Набори параметрів, що задовольняють умові 4.3, називають стаціонарними за Парето. Усі точки оптимальності за Парето є також стаціонарними за Парето, а навпаки — не завжди.

Для кожного блоку $f_i \in \boldsymbol{B}$ багатозадачної мережі $\mathcal{F}$ з параметрами $\boldsymbol{\theta}_i$ розглянемо наступну проблему оптимізації:

$$\begin{aligned} &\text{мінімізувати} && \left\| \sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{\theta_i}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) \right\|_2^2 \\ &\text{при умові} && \sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t = 1, \;\; \alpha^t \geq 0 \end{aligned} \tag{4.5}$$

У дисертації [53] було доведено наступну теорему, який у випадку багатозадачної мережі $\mathcal{F}$ виглядає так:

Теорема 4.1 *Якщо для блоку $f_i \in \boldsymbol{B}$ з параметрами $\boldsymbol{\theta}_i$ розв'язком задачі мінімізації (4.5) є коефіцієнти $\{\alpha_t\}_{t \in \mathcal{O}_{\mathcal{F}}(i)}$, то виконується одне з наступних умов:*

(а) $\sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{\theta_i}} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) = 0$ *і отримані коефіцієнти* $\{\alpha_t\}_{t \in \mathcal{O}_{\mathcal{F}}(i)}$ *задовольняють умовам Каруша–Куна–Таккера (4.3).*

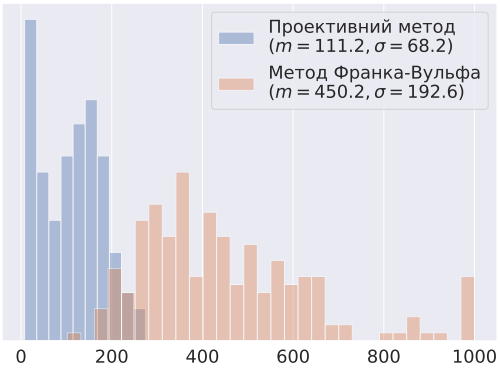
(б) $\sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{\theta_i}} \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ *є напрямком спуску, що зменшує усі цільові функції.*

Формулювання 4.5 є задачею опуклої оптимізації на симплексі. Модифікований алгоритм Франка–Вульфа для розв'язку саме цієї задачі детально описаний у роботі [30].

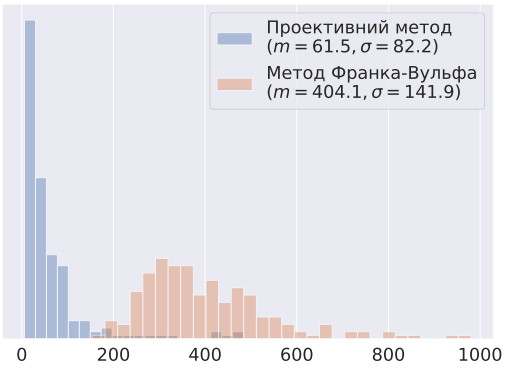
Слід помітити, що метод Франка–Вульфа працює дуже повільно для цій задачі. На гістограммах 3а та 3б показано, як повільно працює метод Франка–Вульфа у порівнянні з проективним методом.

4.3 Оптимізація верхньої межі

У випадку простих архітектур, у роботі [30] описується алгоритм оптимізації верхньої межі. Поширюємо йодо до випадку узагальнених архітектур 3. Для кожного блоку $f_i \in \boldsymbol{B}$ з параметрами $\boldsymbol{\theta}_i$ розглянемо вихід



(а) 40 точок у $\mathbb{R}^{1000}$



(б) 40 точок у $\mathbb{R}^{10000}$

Рис. 3: Порівняння методів Франка–Вульфа та проективного методу за кількістю ітерації для розв'язку задачі (4.5) та (4.7).

$\boldsymbol{Z}_i$ цього блоку у ланцюгу обчислень. Тоді верхня межа 4.5:

$$\left\| \sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{\theta}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) \right\|_2^2 \leq \left\| \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{\theta}_i} \right\|_2^2 \left\| \sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{Z}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) \right\|_2^2 \tag{4.6}$$

де $\| \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{\theta}_i} \|_2$ — матрична норма якобіану $\boldsymbol{Z}_i$ відносно $\boldsymbol{\theta}_i$. Оскільки $\nabla_{\boldsymbol{Z}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ обчислюється під час алгоритму зворотнього поширення помилки і $\| \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{\theta}_i} \|_2^2$ не є функцією від $\{\alpha_t\}_{t \in \mathcal{O}_{\mathcal{F}}(i)}$, можемо сформулювати наступне:

$$\begin{aligned} &\text{мінімізувати} && \left\| \sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{Z}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) \right\|_2^2 \\ &\text{при умові} && \sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t = 1, \;\; \alpha^t \geq 0 \end{aligned} \tag{4.7}$$

При цьому, теорема о гарантованості досягнення точок стаціонарності за Парето набуває наступної форми (доведення подібна до [30]):

Теорема 4.2 *Нехай якобіан $\frac{\partial Z}{\partial \boldsymbol{\theta}^{sh}}$ має повний ранг. Якщо розв'язком задачі мінімізації верхньої межі (4.7) є коефіцієнти $\{\alpha_t\}_{t \in \mathcal{O}_{\mathcal{F}}(i)}$, то виконується одне з наступних умов:*

(а) $\sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{\theta}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta}) = 0$ і отримані коефіцієнти $\{\alpha_t\}_{t \in \mathcal{O}_{\mathcal{F}}(i)}$ задовольняють умовам Каруша–Куна–Таккера (4.3).

(б) $\sum_{t \in \mathcal{O}_{\mathcal{F}}(i)} \alpha^t \nabla_{\boldsymbol{\theta}_i} \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ *є напрямком спуску, що зменшує усі цільові функції.*

Така теорема дозволяє нам тренувати багатозадачні нейронні мережі з більш загальною архітектурою (3) ефективним чином.

4.4 Прискорений гібридний алгоритм

Описані в підрозділах (4.2) та (4.3) алгоритми мають досить серьозний недолік — на кожному кроці оптимизації *батчу*, вони потребують додаткового розв'язку опуклої задачі оптимізації коефіцієнтів градієнту $\alpha_1, \ldots, \alpha_T$, тим самим уповільнюють процес в декілька десятків разів. На практиці, цей процес робить алгоритм майже неможливим для використання. В цьому підрозділі, сформулюємо емпіричний алгоритм який дозволяє швидше досягти точки стакціонарності за Парето (4.2).

Спостереження 4.1 *На початку тренування, майже ні на якому батчі не попадаємо у стаціонарну за Парето (на цьому батчі) точку (як показано на Рис. 4).*

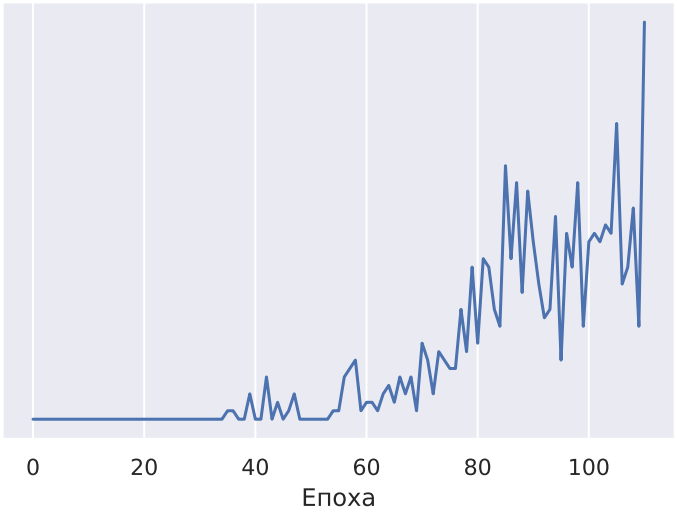


Рис. 4: Процентне відношення кількості міні-батчів, які задовольняють умову Каруша–Куна–Таккера (4.3), а отже, і є стаціонарними за Парето. Можна побачити, що на початок тренування, кількість таких батчів незначне, а отже, більш доцільно використовувати більш швидкі евристичні алгоритми.

Це означає, що більшість часу ми даремно розв'язуємо задачу опуклої оптимізації (4.7).

З геометричної точки зору, формулювання задачі оптимізації (4.5) та (4.7) для блоку $f_i \in \boldsymbol{B}$ з параметрами $\boldsymbol{\theta}_i$ знаходять точки з найменшою нормою на поверхні симплексу з вершинами $\nabla_{\boldsymbol{\theta_i}} \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ де $t \in \mathcal{O}_{\mathcal{F}}(i)$. Отже, у процесі тренування, $\boldsymbol{\theta}_i$ буде рухатись максимально повільно до поверхні симплекса.

Така поведінка має сенс тільки у одному випадку: коли ми вже знаходимось досить близько до точки стаціонарності за Парето. Оскільки градієнти $\nabla_{\boldsymbol{\theta_i}} \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ — стохастичні та обчислюється на кожному мінібатчі, бажано щоб після потраплення до точки стаціонарності за Парето на деяких батчах, мінімально рухатись на інших мінібатчах щоб не руйнувати те що маємо.

Коли ми знаходимось далеко від точок стаціонарності за Парето, така точніть у вибору напрямку руху нам не обов'язкова.

Згідно з цими міркуваннями та як показує спостереження (4.1), на початок тренування доречно замінити емпіричним алгоритмом, який б дозволяло швидко тренуватись на перших епохах. Серед еврістичних методів балансування градієнтів, найкращий за результатами тренування на різних датасетах є метод [37]. У цьому методі розглядаються таке формулювання:

$$\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} c^t \hat{\mathcal{L}}^t(\boldsymbol{\theta}) \tag{4.8}$$

де параметри балансування цільових функції багатокритеріальної задачі $\{c^t\}_{t\in[T]}$ також навчаються в процеці тренування нейронної мережі.

Основна ідея цього методу [37] полягає в тому, що на кожному кроці тренування на мінібатчі, ми намагаємося корегувати градієнти до параметрів енкодера таким чином, щоб дозволити задачам з меншою якістю роспізнавання мати більші значення градієнтів. Для кожного блоку $f_i \in \boldsymbol{B}$ з параметрами $\boldsymbol{\theta}_i$, Розглядаються наступне:

- $G^t_{\boldsymbol{\theta}_i} = \|\nabla_{\boldsymbol{\theta_i}} c^t \hat{\mathcal{L}}^t(\boldsymbol{\theta})\|$ — норма градієнту для зваженої цільової функції $c^t \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ відносно параметру i-го блоку.
- $\overline{G}_{\boldsymbol{\theta}_i} = \mathbb{E}_{t\in\mathcal{O}_{\mathcal{F}}(i)}[G^t_{\boldsymbol{\theta}_i}]$ — середнє значення норми градієнту по всім задачам $t \in \mathcal{O}_{\mathcal{F}}(i)$ блоку f_i з параметром $\boldsymbol{\theta}_i$.

- $\tilde{\mathcal{L}}^t = \hat{\mathcal{L}}^t(\boldsymbol{\theta})/\hat{\mathcal{L}}^t_0(\boldsymbol{\theta})$ — відносне значення цільової функції для задачі t, де $\hat{\mathcal{L}}^t_0$ це значення цієї функції на початку тренування. Це також вважається оберненою швидкістю тренування.
- $r^t_i = \tilde{\mathcal{L}}^t/\mathbb{E}_{t\in\mathcal{O}_{\mathcal{F}}(i)}[\tilde{\mathcal{L}}^t]$ — відносна обернена швидкість тренування для задачі t, обчислене для блоку f_i з параметром $\boldsymbol{\theta}_i$.

Відносна зворотна швидкість навчання завдання може бути використана для того, щоб оцінити баланс градієнтів. Конкретно, чим вище значення r^t, тим більшою повина бути величина градієнта $\nabla_{\boldsymbol{\theta_i}} c^t \hat{\mathcal{L}}^t(\boldsymbol{\theta})$ для завдання, щоб заохотити нейронну мережу швидше тренуватися на цій задачі. Тому наша бажана норма градієнта для кожного завдання t:

$$G^t_{\boldsymbol{\theta}_i} \mapsto \overline{G}_{\boldsymbol{\theta}_i} \times [r^t]^\alpha \tag{4.9}$$

де α - додатковий гіперпараметр. α встановлює силу відновлення, яка повертає градієнти параметру $\boldsymbol{\theta}_i$ до загальної швидкості навчання.

Отже, цільова функція для оптимізації параметрів балансування c^t виглядає наступним чином:

$$\hat{\mathcal{L}}_{grad}(c^t) = \sum_{t\in[T]} \left| G^t_{\boldsymbol{\theta}_i} - \overline{G}_{\boldsymbol{\theta}_i} \times [r^t]^\alpha \right|_1 \tag{4.10}$$

Цей критерій регуляризації градієнтів обчислюється після кожного кроку методу зворотнього поширення помилок, та залежить тільки від c^t (усі інші параметри вважаємо константною на цьому етапі).

Аналогічно до (4.6) та (4.7), можна цей алгоритм прискорити, обчислюючи замість (4.10) наступне:

$$\hat{\mathcal{L}}_{grad}(c^t) = \sum_{t\in[T]} \left| G^t_{\boldsymbol{Z}_i} - \overline{G}_{\boldsymbol{Z}_i} \times [r^t]^\alpha \right|_1 \tag{4.11}$$

Цей алгоритм працює досить швидко та, за результатами обчислень на датасетах [54] та [55] є кращою за усіх інших методів, окрім [30], якому вдається робити малі покращення під кінець тренування.

Отже, гібридний алгоритм тренування багатозадачних нейронних мереж

у загальному випадку архітектури (3) виглядає наступним чином:

Algorithm 1: Гібридний алгоритм

$step \leftarrow 0$;
while $\hat{\mathcal{L}}^t(\boldsymbol{\theta}^{s-1}) - \hat{\mathcal{L}}^t(\boldsymbol{\theta}^s) > \epsilon \ \ \forall t \in [T]$ **do**
    $step \leftarrow step + 1$;
    виконати крок алгоритму (4.11);
end
while $step < epochs$ **do**
    $step \leftarrow step + 1$;
    виконати крок алгоритму (4.7);
end

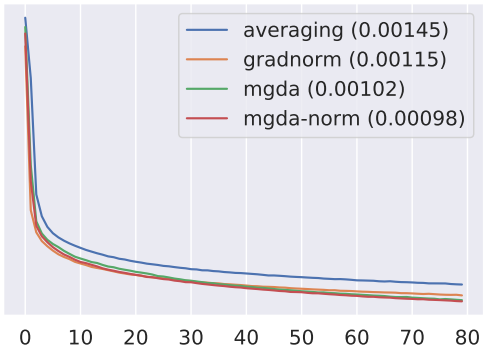
4.5 Експерименти

Ми продемонструємо ефективність гибрідного алгоритму навчання багатозадачної нейронної мережі з архітектурою у загальному вигляді (3) на прикладі трьох датасетах: MNIST [56], Fashion-MNIST [57], та Kuzushiji-MNIST [58]. Перша є стандартною для тестування нових алгоритмів машиного навчання, а інші дві — відносно нові датасети (на момент написання цієї роботи), які по структурі дуже схожі на першу, але значно складніші. Приклади семплів цих датасетів зображено на Рис. 5.

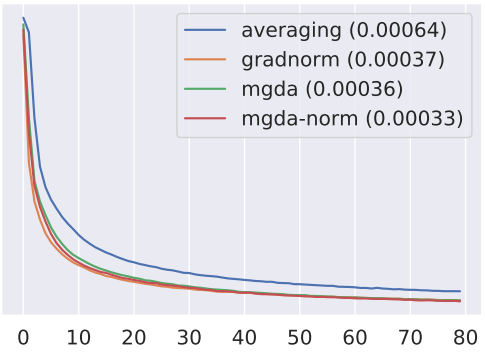
Датасети Kuzushiji-MNIST [58] та Fashion-MNIST [57] були обрані спеціально: перший дуже схожий на MNIST [56] (теж символи), а другий —

(а) MNIST [56] (б) Fashion-MNIST [56] (в) Kuzushiji-MNIST [58]

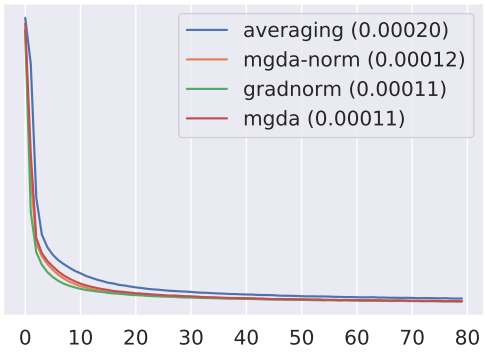
Рис. 5: MNIST–подібні датасети що використовувались для тренування багатозадачної нейронної мережі.



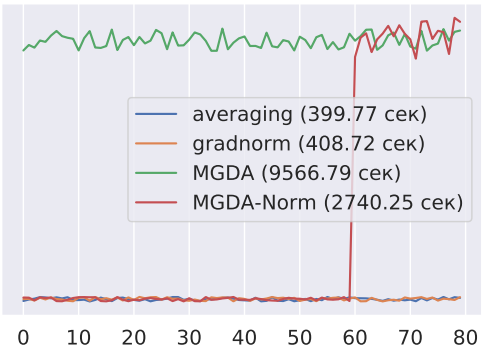
(а) Fashion-MNIST [57]



(б) Kuzushiji-MNIST [58]



(в) MNIST [56]



(г) Швидкість тренування

Рис. 6: Візуальне порівняння алгоритмів навчання багатозадачних нейронних мереж. (6а), (6б), та (6в) показують графік цільової функції. показує скільки часу працює алгоритм.

схожий за структурою датасета (теж чорно-біле, той ж розмір зображення, та ж кількість семплів в датасеті, теж 10 класів для класифікації), але візуально дуже відрізняється, як можна побачити на Рис. 5.

У якості мінімального експерименту, використаємо урізану архітектуру LeNet, де присутні дві конволюційні шари та тільки один повнозв'язний шар до виходу кожної з задач. Причиною того, чому ми обрали урізану архітектуру, просте — на відмінну від LeNet, де більша частина обчислень залежить від перший з повнов'язних шарів, така архітектура зобов'язує нейронну мережу перенести більшу частину репрезентації на конволюційні шари.

Також, між кожними шарами, на відмінку від LeNet, ще додамо шари батч нормалізації [59]. Цей шар був доданий, оскільки наступний роз-

діл 5 потребує батч нормалізації, отже треба продемонструвати що наш гібрідний алгоритм працює і в такому випадку.

Для тренування, використуємо простий SGD з шляхом 10^{-3} з використанням методу Нестерова та імпульсом 0.9 (стандартні параметри цього алгоритму). Незважаючи на те що цей метод працює гірше ніж Adam [60], ми обрали саме цей алгоритм, оскільки він має приблизно однаковий коефіцієнт множення на градієнт на кожному кроці тренування, а отже, може продемонструвати переваги та недоліки кожного з алгоритмів вибору напрямки градієнту.

Порівнюються наступні алгоритми: (а) усереднення градієнтів з наперед заданими коефіцієнтами; (б) модифікований GradNorm (4.11); (в) модифікований MGDA [30] для загального випадку архітектур (3); та (г) гібрідний метод, описаний в (1).

Тренування проводилося на відеокарті NVIDIA GeForce 1070ti на домашньому кластері студента. Було проведено більш ніж 20 експериментів для кожного алгоритму, що займало більш ніж 48 годин на обох відеокартах.

Результати експериментів можна побачити на Рис. 6. На початку тренування, можно помітити що GradNorm працює не гірше ніж MGDA, не зважаючи на те що перший з них — еврістичний, а другий — теоретично обгрунтований. На останніх епохах, графік цільової функції MGDA "відривається"з графіку GradNorm. Бачимо, що наш гібридний алгоритм MGDA-Norm дає таку ж якість роспізнавання що й MGDA [30], але працює значно швидше, як показано на Рис. 6.

5 пошук архітектур

В цьому розділі, ми розглянемо різні методи пошуку оптимальної архітектури для багатозадачної нейронної мережі, обговоримо їх обмеження, запропонуємо новий емпіричний алгоритм який не має тих обмежень, та експериментально перевіримо її ефективність.

5.1 Мотивація

У попередньому розділі, ми розглянули різні методи оптимизації багатозадачних нейронних мереж [30, 37, 36], та запропонували простий спосіб прискорення тренування без втрати якості. Отже, маючи фіксовану архітектуру, ми вже вміємо ефективно його тренувати. Питання у тому, чи є обрана архітектура оптимальною?

У багатозадачному глибинному навчанні, найбільш поширеним та природним підходом є класичні архітектури, з спільним енкодером та декодером для кожної задачі (2.2.1). Прикладом таких архітектур є роботи [26, 39, 23, 28, 40, 20, 25, 41]. Проте, при такому підході, де усі підмережі $\mathcal{F}^t$ мають однакову спільну частину (енкодер), якість тренування може постраждати від проблеми негативного перенесення — ситуація, коли неякісні (неточні) предиктори для більш складних задач можуть негативно вплинути на якість предикторів на простих завданнях. У роботі [61] детально описується ця проблема, разом з інтуїтивними прикладами.

Потенціальна причина цієї проблеми досліджується у роботі [62], в якому припускається, що потенційне обмеження більшості існуючих робіт полягає в тому, що спільний простір ознаки на кожному шарі лежить разом з приватними ознаками на цьому просторі (під просторами ознак мається на увазі простори змінних $\boldsymbol{Z_i}$, отримані в процесі обчислення виходу $\mathcal{F}$ мережі для $\boldsymbol{x}$ деяким ланцюгом p^t). Причиною такого розподілу параметрів є те, що усі підмережі усіх задач мають однакову спільну частину. В цій роботі [62] також приводиться експеримент, який підсилює цю гіпотезу.

Природним розв'язком цієї проблеми очевидне: треба вилучили приватні риси для кожної задачі і роз'єднати їх у окремі просторі. Однак на практиці, використовувані методи просто збирають всі ознаки в спільний простір, замість того, щоб вивчати спільні правила в різних завданнях. Більш детально ця проблема досліджується у роботі [63].

Деякі дослідження в цьому напрямку, тренуючі нейронні мережі в змагальному середовищі, показують, що явний розподіл ознаки на приватні та спільні простори покращують якість багатозадачної нейронної мережі [62, 64]. Для класу деревоподібних архітектур (3), пошук спільних ознак еквівалентне визначенні спільного префіксу ланцюгів p^{t_1} та p^{t_2} для кожної пари задач.

Основна складність полягає у тому, що вибір шарів (блоків мережі $\mathcal{F}$, які необхідно розділити між завданнями, є NP-повною задачею. Більш того, пошук оптимальної архітектури також слід робити з урахуванням переносного навчання — коли береться за основою мережу, що тренувалася на дуже великому датасеті (напр. [65]) та навчилась корисними ознаками.

Попередні роботи [18, 51, 49], що можна використовувати разом з попереднім переносним навчанням, намагалися вирішити цю проблему шляхом використання деяких евристик:

- В роботі [49] оптимальна архітектура отримується шляхом тренування дуже великої мережі, та відкидаються зайві блоки. Цей підхід є дуже громіздким, та, взагалі кажучі, не завжди така велика мережа може поміститись в пам'ять відеокарти.
- [18] за допомогою *спорідненості виходів* роблять припущення на схожість ознак. Але такий підхід має недолік у тому, що потребує наявність y^t для кожної задачі $t \in [T]$ для кожного семплу $\boldsymbol{x}$, що на практиці дуже рідко зустрічається.
- Нещодавно опублікована робота [51] позбавлений цих проблем, але він дуже повільний — потребує навчання $T^2 \cdot N$ нейронних мереж, де T це кількість задач, а N — кількість шарів в мережах.

Отже, наша основна мотивація — позбавитись від недолік цих методів. Для цього, введемо евристичну оцінку *сумісність* між двома мережами.

5.2 Сумісність між мережами

У даному підрозділі, намагаємося з'ясувати, які вузли є спільними для найбільш різних задач. Для цьоно, нам потрібні такі показники: (а) показник *важливості* ознак, та (б) показник *схожості* між ознаками різних шарів в різних мережах.

(а) Мінімізоване зображення (б) Приклад зображень датасету що мінімізує (в) Приклад зображень датасету що максимізує (г) Максимізоване зображення

Рис. 7: Візуалізація каналу №492 виходу (ознаки) шару *mixed 4a* мережі Inception V3 [4], взятої з роботи [66]. Зображення (а) та (г) отримані градієнтним спуском.

5.2.1 Оцінка подібності каналів між мережами

До цього моменту, ми дуже абстрактно говорили про поняття ознак. Взагалі кажучі, ознаки картинки, отримані нейронною мережею, називається вихід $\boldsymbol{Z}_i$ блоку f_i над зображенням $\boldsymbol{x}$.

Якщо шар, що створило цю ознаку, є конволюційним, $\boldsymbol{Z}_i \in \mathbb{R}^{h \times w \times c}$, де h та w — розмір мапи ознак, c — кількість каналів. Тоді, таку мапу ознак однозначно описується деякою функцією g_i, що породжує вектор ознак $v \in \mathbb{R}^c$. Отже, коли ми говоримо про оцінку подібності двох мереж $\mathcal{F}_1$ і $\mathcal{F}_2$ з однаковою архітектурою (тобто однакові $\boldsymbol{B}$ та $\boldsymbol{P}$) на i-му шарі, ми насправді порівнюємо функції $g_i^{\mathcal{F}_1}$ та $g_i^{\mathcal{F}_2}$ цих мереж.

На Рис. 5 показано візуалізація семантичного змісту деякого напрямку вектору ознак $v \in \mathbb{R}^c$ [66]. Такі візуалізації натягують на ідею, що насправді ознака $\boldsymbol{Z}_i$ — це сукупність *шаблонів*, кожен з яких розпізнає деякі семантично значні деталі вхідного зображення, та кожний шаблон представляє собою напрямок в $v \in \mathbb{R}^c$ — чим більший розмір вектору у цьому напрямку, тим більш ймовірно що вхідне зображення включає те, що роспізнає цей шаблон. Ось, наприклад, на Рис. 5 показано шаблон дахів. Нещодавно було показано, що сучасні нейронні мережі [2, 3, 4, 5, 6, 7] можна апроксимувати набором таких шаблонів [67].

Незважаючи на те, що [68] показав, що випадкові напрямки є також значущими, як і напрями базисних векторів в $\mathbb{R}^c$, [69] виявили, що напрямки базисних векторів можна інтерпретувати частіше, ніж випадкові напрямки. Отже ми можемо ще більше спростити нашу задачу оцінки

подібності ознак: замість розгляду функції g_i що породжують вектор $\mathbb{R}^c$, можна розглядати функції h_j, що породжують координати g_i, тобто $g_i = (h_1, \ldots, h_c)$. Функції h_j відображає зображення у $\mathbb{R}$, та можна їх інтерпретувати як значення певного нейрону (якщо $\boldsymbol{Z}_i$ розглядати як сукупність нейронних активації).

Також, зробимо припущення, що остання функція в блоці, що породжує $\boldsymbol{Z}_i$ — це батч нормалізація [59]. Таке припущення є досить доцільним — майже усі сучасні архітектури [5, 6, 12, 14, 13, 7, 4] використовують цей шар після кожного блоку. Будемо розглядати $\tilde{\boldsymbol{Z}}_i$ — значення перед лінійного перетворення в батч нормалізації.

У такому випадку, схожість між двома *нейронами* $z^{\mathcal{F}_1}_{n,m,k}$ та $z^{\mathcal{F}_2}_{n,m,k}$ з ознаки $\tilde{\boldsymbol{Z}}^{\mathcal{F}_1}_i$ та $\tilde{\boldsymbol{Z}}^{\mathcal{F}_2}_i$ можна виразити через розходження Єнсена–Шеннона:

$$\mathcal{S}\left(z^{\mathcal{F}_1}_{n,m,k} \parallel z^{\mathcal{F}_2}_{n,m,k}\right) = D_{JS}\left[\sigma\left(z^{\mathcal{F}_1}_{n,m,k}\right) \parallel \sigma\left(z^{\mathcal{F}_2}_{n,m,k}\right)\right] \tag{5.1}$$

Де $\sigma(z) = 1/(1+e^{-z})$ — сигмоїда. Оскільки остання функція в блоці, що породжує $\boldsymbol{Z}_i$ — це батч нормалізація, то можемо припустити що $z_{n,m,k}$ центрована (з матсподіванням 0) та дисперсією 1. D_{JS} — розходження Єнсена–Шеннона, що обчислюється наступним чином:

$$D_{JS}\left(P \| Q\right) = \frac{1}{2} D_{KL}\left(P \| M\right) + \frac{1}{2} D_{KL}\left(Q \| M\right) \tag{5.2}$$

де $M = (P+Q)/2$, а $D_{KL}(P\|Q)$ — росходження Кульбака–Лейблера, яка обчислюється для усіх семплів $\boldsymbol{x}$ датасету $\mathcal{X}$ наступним чином:

$$D_{\mathrm{KL}}(z \parallel z') = -\frac{1}{N} \sum_{\boldsymbol{x} \in \mathcal{X}} z(\boldsymbol{x}) \log\left(\frac{z'(\boldsymbol{x})}{z(\boldsymbol{x})}\right) \tag{5.3}$$

де N — кількість батчів у датасеті. Тоді, схожість між двома мапами ознак $\boldsymbol{Z}^{\mathcal{F}_1}_i$ та $\boldsymbol{Z}^{\mathcal{F}_2}_i$ можна виразити як зважена сума схожості їх нейронів:

$$\mathcal{S}\left(\boldsymbol{Z}^{\mathcal{F}_1}_i \parallel \boldsymbol{Z}^{\mathcal{F}_2}_i\right) = \sum_{k=1}^{c} \alpha_k \frac{1}{h+w} \sum_{n=1,m=1}^{h,w} \mathcal{S}\left(z^{\mathcal{F}_1}_{n,m,k} \parallel z^{\mathcal{F}_2}_{n,m,k}\right) \tag{5.4}$$

де $\alpha_k \geq 0$ та $\sum_{k=1}^{c} \alpha_k = 1$. Помітимо, що таке формулювання схожості не залежить від величини виходу завдяки батч нормалізації та сигмоїдальної функції.

5.2.2 Оцінка важливості каналів мережі

В обчисленні оцінки подібності між блоками $f_i^{\mathcal{F}_1}$ та $f_i^{\mathcal{F}_2}$, ми використали зважену суму (5.4). Треба якимось чином знайти хороші параметри $\alpha_1, \ldots, \alpha_c$. Просто присвоїти константи $\alpha_k \equiv 1/c$ погано, тому що таким чином ми не відрізняємо *важливі* нейрони (ті що більш впливають на прогнозування багатозадачної нейронної мережі) та неважливі.

Є ціла галузь навчання глибоких нейронних мереж, яка називається *прунінгом*. Це техніки, які дозволяють зменшити розмір нейронної мережі. Зацікавленим читачам рекомендується ознайомитись з обзором існуючих класів технік прунінгу [70].

Серед існуючих методів прунінгу глибоких нейронних мереж, нас цікавить саме ті методи, які дозволяють емпірично відповісти на питання: наскільки важливі кожен з каналів мапи ознак? В роботі [71] оцінюється верхня межа оцінки важливості, це нам не підходить тому що верхня межа може бути дуже великою. В роботі [72] апроксимується гессіана, але припускається що градієнти нульові (оскільки тренують до точки стаціонарності), в нашому випадку не можемо зробити такі припущення. Нові методи що використовують навчання з підкріпленням [73] занадто повільно працюють.

Серед усіх емпіричних методів, метод [74] дає найкращі результати, це також узгоджується з незалежними результатами обчислень [75]. Нехай шар батч нормалізації є останнім в ланцюзі обчислення $\boldsymbol{Z}_i$ та виглядає наступним чином:

$$\tilde{\boldsymbol{Z}}_i = \frac{\boldsymbol{Z}_{in} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}; \quad \boldsymbol{Z}_i = \gamma \tilde{\boldsymbol{Z}}_i + \beta \tag{5.5}$$

де $\boldsymbol{Z}_{in}$ — вхід до блоку f_i, $\mu_{\mathcal{B}}$ та $\sigma_{\mathcal{B}}$ це середня та стандартне відхилення значень $\boldsymbol{Z}_{in}$ над батчем $\mathcal{B}$. Параметри γ і β є вільними параметрами що тренуються. Нас цікавить саме параметр γ. Ідея методу [74] полягає у тому, що при тренуванні, ми ще додаємо наступну терму регуляризації:

$$\hat{\mathcal{L}} = \sum \hat{\mathcal{L}}^t(\boldsymbol{\theta}) + \lambda \sum_{\gamma \in \Gamma} \|\gamma\|_1 \tag{5.6}$$

Таким чином, будемо мінімізувати енергію виходу з блоку f_i, а отже, заставимо мережу залишити тількі найважливіші канали мапи ознак.

Отже, чим більше коефіцієнт γ_k для каналу $\boldsymbol{z}_{:,:,k}$, тим він важливіший для цієї задачі. Задамо оцінку важливості k-го каналу наступним чином:

$$\mathcal{I}_k = \frac{\gamma_k}{\sum_{j \in [c]} \gamma_j} \tag{5.7}$$

Така оцінка не буде залежити від абсолютних значень γ_k та задовольняє умовам, що $\gamma_k \geq 0 \forall k \in [c]$ та $\sum_{k \in [c]} \gamma_k = 1$, що дозволяє нам використати ці коефіцієнти для обчислення (5.4).

5.2.3 Оцінка стрессу вузлів

Маючи оцінку подібності ознак між мережами та оцінку важливості каналів однієї мережі, можемо задати оцінку сумісності між блоками двох нейронних мереж:

$$\mathcal{C}\left(\boldsymbol{Z}_i^{\mathcal{F}_1} \parallel \boldsymbol{Z}_i^{\mathcal{F}_2}\right) = \sum_{k=1}^{c} \frac{\mathcal{I}_k}{h+w} \sum_{n=1,m=1}^{h,w} \mathcal{S}\left(z_{n,m,k}^{\mathcal{F}_1} \parallel z_{n,m,k}^{\mathcal{F}_2}\right), \tag{5.8}$$

де $\mathcal{S}$ обчислюється за формулою (5.1), $\mathcal{I}_k$ обчислюється за формулою (5.7). Ця оцінка буде нормалізованою.

Нехай маємо деревоподібну мережу $\mathcal{F} = \{\boldsymbol{B}, \boldsymbol{\theta}, \boldsymbol{P}\}$. Припустимо, що ця мережа вже нетренована до стаціонарності за Парето (4.7). Оцінимо *показник стрессу* для i-го блоку наступним алгоритмом:

Algorithm 2: Показник стрессу i-го блоку

Покладемо $\boldsymbol{F}_{peeled} \leftarrow \{peel(\mathcal{F}, t)\}_{t \in \mathcal{O}_{\mathcal{F}}(i)}$;
for $\mathcal{F}_{peeled}^t \in \boldsymbol{F}_{peeled}$ **do**
    Натренувати $\mathcal{F}_{peeled}^t$ на декілька епох (файн тюнинг);
    Покласти $\mathcal{C}^t \leftarrow \mathcal{C}(\boldsymbol{Z}_i^{\mathcal{F}_{peeled}^t} \parallel \boldsymbol{Z}_i^{\mathcal{F}})$;
end
return показник стрессу $\mathcal{S}_i = \sum_{t \in \mathcal{O}} \mathcal{C}^t$,

5.3 Жадібний алгоритм пошуку архітектури

Маючи емпіричний спосіб оцінки показника стрессу для мережі, можемо сформулювати алгоритм для динамічного росширення нейронної мере-

жі. Перед цим, треба задати алгоритм обчислення матриці сумісності між вітками деревоподібної мережі.

5.3.1 Матриця спорідненості

Поки що будемо розглядати тільки i-й блок мережі $\mathcal{F} = \{\boldsymbol{B}, \boldsymbol{\theta}, \boldsymbol{P}\}$. Обчислимо *матрицю спорідненості* між задачами $t_1 \in \mathcal{O}_\mathcal{F}(i)$ та $t_2 \in \mathcal{O}_\mathcal{F}(i)$ для i-го блоку наступним алгоритмом:

Algorithm 3: Матриця спорідненості задач для i-го блоку

Покладемо $\boldsymbol{F}_{peeled} \leftarrow \{peel(\mathcal{F}, t)\}_{t \in \mathcal{O}_\mathcal{F}(i)}$;
for $\mathcal{F}^t_{peeled} \in \boldsymbol{F}_{peeled}$ **do**
  Натренувати $\mathcal{F}^t_{peeled}$ на декілька епох (файн тюнинг);
end
for $\mathcal{F}^{t_1}_{peeled} \in \boldsymbol{F}_{peeled}$ **do**
  for $\mathcal{F}^{t_2}_{peeled} \in \boldsymbol{F}_{peeled}, t_1 \neq t_2$ **do**
    $\mathcal{A}^i_{t_1,t_2} \leftarrow \mathcal{C}(\boldsymbol{Z}_i^{\mathcal{F}^{t_1}_{peeled}} \parallel \boldsymbol{Z}_i^{\mathcal{F}^{t_2}_{peeled}})$;
  end
end
return Матриця спорідненості задач $\mathcal{A}^i$

Маючи матрицю спорідненості між задачами, можемо задати матрицю спорідненості для блоку i між нащадками цього блоку:

Algorithm 4: Матриця спорідненості нащадків для i-го блоку

Обчислити матрицю спорідненості задач $\mathcal{A}^i$ алгоритмом (3);
for $c_1 \in \mathcal{C}_\mathcal{F}(i)$ **do**
  for $c_2 \in \mathcal{C}_\mathcal{F}(i), c_1 \neq c_2$ **do**
    Покладемо $\hat{\mathcal{A}}^i_{c_1,c_2} \leftarrow \underset{t_1 \in \mathcal{O}_\mathcal{F}(c_1)}{\text{mean}} \left(\underset{t_2 \in \mathcal{O}_\mathcal{F}(c_2)}{\max} \mathcal{A}^i_{c_1,c_2} \right)$;
    Покладемо $\hat{\mathcal{A}}^i_{c_2,c_1} \leftarrow \underset{t_2 \in \mathcal{O}_\mathcal{F}(c_2)}{\text{mean}} \left(\underset{t_1 \in \mathcal{O}_\mathcal{F}(c_1)}{\max} \mathcal{A}^i_{c_2,c_1} \right)$;
    Покладемо $\tilde{\mathcal{A}}^i_{c_2,c_1} \leftarrow (\hat{\mathcal{A}}^i_{c_1,c_2} + \hat{\mathcal{A}}^i_{c_2,c_1})/2$;
  end
end
Покладемо $\tilde{\mathcal{A}}^i \leftarrow \exp\left[-\tilde{\mathcal{A}}^i / \max_{c_1,c_2} \tilde{\mathcal{A}}^i_{c_1,c_2} \right]$;
return Матриця спорідненості нащадків $\tilde{\mathcal{A}}^i$;

Остання дія оберненої нормалізації в алгоритмі (4) було зроблено спеціально для того, щоб можна було використати методи спектральної кластерізації графів. В цій роботі, було використано алгоритм [76].

5.3.2 Остаточний алгоритм

Гіпотеза 5.1 *Багатозадачна нейронна мережа $\mathcal{F}'$ з мінімальним внутрішнім стрессом (2), отриманий з початкової мережі $\mathcal{F}$ операцією $split(\mathcal{F}, \cdot, \cdot)$, має найбільшу якість роспізнавання серед тих архітектур, що мають таку ж саму кількість блоків (тобто не більші мережі) і можуть бути отримані з $\mathcal{F}$.*

Отже, тепер можемо сформулювати кінцевий алгоритм жадібного пошуку архітектури з мінімальним стрессом вузлів. Нехай маємо деревоподібну мережу $\mathcal{F} = \{\boldsymbol{B}, \boldsymbol{\theta}, \boldsymbol{P}\}$.

Algorithm 5: Жадібний пошук архітектури з мінімальним стрессом

Натренуємо $\mathcal{F}$ до стаціонарності за Парето (1);
for $step = 1 \dots N$ **do**
  Порахувати показники стрессу $\mathcal{S}_i$ для кожного блоку $f_i \in \boldsymbol{B}$ алгоритмом (2);
  Покладемо $I \leftarrow \arg\max_i \{\mathcal{S}_i\}$ (тобто I — блок з найбільшим внутрішнім стрессом);
  Порахуємо матрицю спорідненості нащадків $\tilde{\mathcal{A}}^i$ блоку I використовуючи алгоритм (4);
  Знайдемо оптимальне розбиття нащадків $\mathcal{C}_{\mathcal{F}}(I)$ за допомогою спектрального алгоритму кластеризації [76], використовуючи $\tilde{\mathcal{A}}^i$, отримаємо розбиття $\pi = \{\{c_1^1, \dots, c_k^1\}, \{c_1^2, \dots, c_l^2\}\}$;
  Виконуємо операцію розділення $split(\mathcal{F}, I, \pi)$;
end

Цей алгоритм є значно швидшою ніж [51], оскількі в тій роботі потребує тренування (файн тюнинг) $T^2 \cdot N$ нейронних мереж, де T це кількість задач, а N — кількість шарів в мережах. В нашому випадку, можемо навіть обійтись тренуванням (файн тюнингом) лише T мереж.

Зауважимо, що за допомогою алгоритмів (2), (3), та (4), можна сформулювати більш ефективний алгоритм пошуку архітектур, аніж жадібний алгоритм (5).

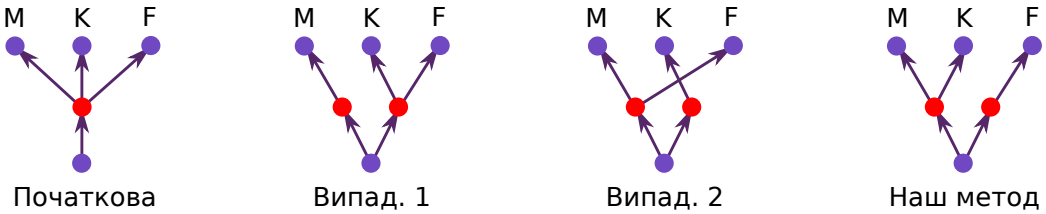


Рис. 8: Різні методи пошуку архітектур: дві випадкові та наш запропонований метод (5). Позначення: M — MNIST [56], K — Kuzushiji-MNIST [58], та F — Fashion-MNIST [57].

5.4 Експерименти

Проведемо експеримент з такими ж умовами що й у підрозділі 4.5. Початкова архітектура мережі оберемо той ж самий як і у попередньому розділі, тобто спількний повніст'ю конволюційний енкодер, та простий декодер для кожної з задач. На Рис. 8 зображено приклад роботи нашого алгоритму (5) на архітектурі LeNet та датасетами MNIST [56], Fashion-MNIST [57], та Kuzushiji-MNIST [58]. В табл. 1 приведені кінцеві результати тренувань.

	fashion	kmnist	mnist
fashion	0	0.0236	0.0243
kmnist	0.0236	0	0.0050
mnist	0.0243	0.0050	0

(а) Матриця спорідненості задач $\mathcal{A}^i$, отриманий алгоритмом (3) з 10 епохами файн-тюнингу.

метод	лосс ф-я	метрики
наш (5)	**0.00130**	**0.9119**
випад. 1	0.00132	0.9113
випад. 2	0.00149	0.8983

(в) Значення цільової функції та метрики для задачі Kuzushiji-MNIST [58]. Порівнюється алгоритм (5) з випадковими алгоритмами.

метод	лосс ф-я	метрики
наш (5)	**0.00124**	**0.8911**
випад. 1	0.00133	0.8795
випад. 2	0.00148	0.8666

(б) Значення цільової функції та метрики для задачі Fashion-MNIST [57]. Порівнюється алгоритм (5) з випадковими алгоритмами.

метод	лосс ф-я	метрики
наш (5)	0.00014	0.9879
випад. 1	0.00015	0.9877
випад. 2	**0.00012**	**0.991**

(г) Значення цільової функції та метрики для задачі MNIST [56]. Порівнюється алгоритм (5) з випадковими алгоритмами.

Табл. 1: Порівняння методів пошуку архітектур. Візуалізація на Рис. 8.

6 Висновки

Основним результатом цієї роботи є алгоритм автоматичного пошуку деревоподібної архітектури, що сформульований у розділі (5). Наш метод не має тих серьозних обмежень що мають попередні методи [18] (які робить їх майже не практичними, оскількі можуть вирішувати більш вузький клас задач), а також є значно швидшою та ефективною за пам'яті, ніж попередні результати [49], [51]. Ми також обговорювали, що отримана архітектура у результаті нашого методу відповідає нашої візуальної інтуїції про подібності задач.

В цій роботі, ми також провели детальний аналіз існуючих методів тренування багатозадачних нейронних мереж та проаналізували деякі їх недоліки. На базі цих попередніх робіт, ми сформулювали дуже простий, але ефективний гибрідний алгоритм навчання багатозадачних глибоких нейронних мереж, та продемонстрували якість та швидкість нашого методу в порівнянні з іншими методами на прикладі простого експерименту. Оскільки цей алгоритм був описаний у загальному вигляді для універсальних архітектур (3), його можна буде використати для дуже широкого класу задач багатозадачного навчання.

Плани для подальшого розвитку:

- Порівняти з існуючими методами шляхом експериментування на складніших датасетах (напр. [54, 55]) та на більших нейронних мережах (напр. [5, 6, 14]).
- Розробити більш ефективний алгоритм ніж жадібний (5).
- Зробити більш детальний аналіз нашої евристичної оцінки (5.8) та гіпотези (5.1) шляхом порівняння з випадково згенерованими архітектурами.

Література

[1] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[9] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[10] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[11] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[12] Bo Chen Dmitry Kalenichenko Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam Andrew G. Howard, Menglong Zhu. Mobilenets: Efficient convolutional neural networks for mobile vision. *arXiv preprint arXiv:1704.04861*, 2017.

[13] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[15] Rich Caruana. *Multitask Learning*, pages 95–133. Springer US, Boston, MA, 1998.

[16] Hakan Bilen and Andrea Vedaldi. Integrated perception with recurrent multi-task neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 235–243. Curran Associates, Inc., 2016.

[17] Brendan Jou and Shih-Fu Chang. Deep cross residual learning for multitask visual recognition. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, pages 998–1007, New York, NY, USA, 2016. ACM.

[18] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[19] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[20] R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):121–135, Jan 2019.

[21] Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391*, 2016.

[22] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3973–3981, 2015.

[23] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013.

[24] Michael L Seltzer and Jasha Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6965–6969. IEEE, 2013.

[25] Zhizheng Wu, Cassia Valentini-Botinhao, Oliver Watts, and Simon King. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4460–4464. IEEE, 2015.

[26] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2169–2176. IEEE, 2017.

[27] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. In *2017 The Genetic and Evolutionary Computation Conference (GECCO)*, 2017.

[28] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[29] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[30] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 527–538. Curran Associates, Inc., 2018.

[31] Charles Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 197–206, Berkeley, Calif., 1956. University of California Press.

[32] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[33] Jiayu Zhou, Jianhui Chen, and Jieping Ye. Malsar: Multi-task learning via structural regularization – user's manual version 1.1, 2012.

[34] Yu Zhang and Qiang Yang. A survey on multi-task learning. *CoRR*, abs/1707.08114, 2017.

[35] Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

[36] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

[37] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 794–803, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[38] Чан Ха Ву. Guide to instant noodles in multi-task learning, 2019.

[39] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1723–1732, 2015.

[40] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. 2015.

[41] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014.

[42] Kazuma Hashimoto, Yoshimasa Tsuruoka, Richard Socher, et al. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1923–1933, 2017.

[43] Shubham Toshniwal, Hao Tang, Liang Lu, and Karen Livescu. Multitask learning with low-level auxiliary tasks for encoder-decoder based speech recognition. *arXiv preprint arXiv:1704.01631*, 2017.

[44] Hakan Bilen and Andrea Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017.

[45] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *International Conference on Learning Representations*, 2018.

[46] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.

[47] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[48] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pages 4092–4101, 2018.

[49] Siyu Huang, Xi Li, Zhi-Qi Cheng, Zhongfei Zhang, and Alexander Hauptmann. Gnas: A greedy neural architecture search method for

multi-attribute learning. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 2049–2057. ACM, 2018.

[50] Jason Liang, Elliot Meyerson, and Risto Miikkulainen. Evolutionary architecture search for deep multitask networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 466–473. ACM, 2018.

[51] Simon Vandenhende, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: Deciding what layers to share. *CoRR*, abs/1904.02920, 2019.

[52] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2018.

[53] Jean-Antoine Désidéri. *Multiple-gradient descent algorithm (MGDA)*. PhD thesis, INRIA, 2009.

[54] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[55] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[56] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[57] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[58] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

[59] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[61] Hae Beom Lee, Eunho Yang, and Sung Ju Hwang. Deep asymmetric multi-task feature learning. In *International Conference on Machine Learning*, pages 2962–2970, 2018.

[62] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351, 2016.

[63] Pengfei Liu and Xuanjing Huang. Meta-learning multi-task communication. *arXiv preprint arXiv:1810.09988*, 2018.

[64] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–10, 2017.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[66] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[67] Wieland Brendel and Matthias Bethge. Approximating CNNs with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations*, 2019.

[68] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[69] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6541–6549, 2017.

[70] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[71] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[72] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[73] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018.

[74] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[75] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[76] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000.