

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп'ютерних наук та кібернетики



**Звіт**

з дисципліни

«Інтелектуальна обробка текстів»

на тему

**«Question Answering System based on SQuAD»**

Виконала

студентка 1-го курсу магістратури

спеціальність «Інформатика»

Yelyzaveta

Київ — 2019

# ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ	3
АКТУАЛЬНІСТЬ ОБРАНОЇ ТЕМИ	4
ІСНУЮЧІ НАБОРИ ДАНИХ ДЛЯ QA СИСТЕМ ТА ЇХ НЕДОЛІКИ	5
ПОСТАНОВКА ЗАДАЧІ	7
ТЕОРЕТИЧНІ ВІДОМОСТІ	8
РОЗДІЛ 1. SQuAD: 100,000+ Questions for Machine Comprehension of Text	8
1.1 Аналіз набору даних SQuAD (Stanford Question Answering Dataset)	8
1.2 Колекція наборів даних (Dataset Collection)	10
1.3 Аналіз типів запитань	11
РОЗДІЛ 2. AI2 Reasoning Challenge (ARC) Dataset	17
2.1 Аналіз набору даних	17
2.2 Типи питань	19
2.3 Розширений корпус	21
2.4 Алгоритм інформаційного пошуку	21
2.5 Алгоритм знаходження вказівника з точковою інформацією	21
2.6 Набір складних питань	
ПРАКТИЧНА ЧАСТИНА	24
РОЗДІЛ 1. Bi-Directional Attention Flow (BIDAF) Reading Comprehension Model	24
1.1 Модель двонаправленого потоку для машинного розуміння	24
1.2 Багатошарова нейронна модель	25
1.3 Багатошарова нейронна модель	19
ВИСНОВКИ	26
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	27
ДОДАТОК А	29
ДОДАТОК Б	31

## СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

- MC — machine comprehension, машинне розуміння;
- QA — question answering, відповіді на запитання;
- SQuAD — Stanford Question Answering Dataset;
- ARC — AI2 Reasoning Challenge (ARC) Dataset;
- IR Solver — Information Retrieval (IR) Solver, алгоритм інформаційного пошуку;
- PMI — The Pointwise Mutual Information (PMI) Solver, алгоритм знаходження вказівника з точковою інформацією;
- BIDAF — Bi-Directional Attention Flow (BIDAF) Reading Comprehension Model, модель двонаправленого потоку для машинного розуміння;
- RNN — Recurrent neural network, рекурентна нейронна мережа;
- CNN — Convolutional Neural Networks, згорткова нейронна мережа;
- LSTM — Long Short-Term Memory Network, довга короткочасна пам'ять;
- C2Q — Context-to-query, увага від контексту до запиту;
- Q2C — Query-to-context attention, увага запиту до контексту.

## **АКТУАЛЬНІСТЬ ОБРАНОЇ ТЕМИ**

Задачі машинного розуміння (MC) та побудови питально-відповідних довідкових систем або відповіді на запитання (QA) отримали значну популярність протягом останніх кількох років в обробці природної мови та спільнотах комп’ютерного зору.

Системи, що натренировані неперервним ланцюгом (end-to-end) скрізь досягають перспективних результатів в різноманітних завданнях як в текстових , так і в графічних областях. Одним з ключових факторів розвитку в даному напрямі є використання механізму нейронної уваги (attention mechanism), який дозволяє диференційованій системі зосередитися на цільовій області речень (для MC) або зображення (для Visual QA) (Weston et al., 2015; Antol et al., 2015; Xiong et al., 2016a).

Щорічні оцінки на конференції Text REtreival (TREC) привели до багатьох досягнень у сфері забезпечення якості відкритої області (open-domain question answering), багато з яких використовувалися в IBM Watson для Jeopardy (Ferrucci et al., 2013).

# ІСНУЮЧІ НАБОРИ ДАНИХ ДЛЯ QA СИСТЕМ ТА ЇХ НЕДОЛІКИ

Існує безліч наборів даних, які забезпечують прогрес у відповіді на запитання в питально-відповідних довідкових системах (Question answering system). В більш ранніх наборах даних для розуміння написаного, наприклад, MCTest (Richardson, 2013), SQuAD (Rajpurkar et al., 2016), NewsQA (Trischler et al., 2016), і CNN / DailyMail (Hermann et al., 2015), місяться питання, чиї відповіді можна було б визначити виключно на підставах поверхневого рівня (тобто відповіді були “чітко визначені”). TriviaQA (Joshi et al., 2017) розширили це завдання, надавши декілька статей із запитанням, і використовували запитання, написані незалежно від цих статей. Знову ж таки, питання були в значній мірі фактичними, наприклад, *"Хто виграв Нобелівську премію миру в 2009 році?"*. Незважаючи на те, що системи тепер можуть добре працювати на цих наборах даних, навіть збігаючись з людськими показниками (Simonite, 2018), їх можна легко обдурити (Jia i Liang, 2017); ступінь, до якої вони дійсно розуміють мовні або специфічні поняття — все ще залишається неясним.

Yang et al. (2015) створили набір даних WikiQA, який, подібно до SQuAD, використовує уривки Вікіпедії як джерело відповідей, але їхнє завдання — вибір речення, в той час як SQuAD — вимагає вибору певного відрізу у реченні.

Щоб перейти до більш складних завдань із забезпечення якості, одним з підходів було обрано створення синтетичних наборів даних, найпомітнішим прикладом яких є набір даних bAbI (Weston et al., 2015). Він був створений з використанням простого світового симулятора та генератора мов, що дає дані !4 для 20 різних завдань. Він стимулював роботу з використання мережі пам'яті нейронних архітектур (Weston et al., 2014), підтримуючи форму багатоступеневого розуміння, коли нейронна пам'ять поширює інформацію з одного кроку на інший (наприклад, Henaff et al., 2016; Seo et al., 2017a). Проте використання синтетичного тексту та «синтетичного світу» обмежує

реалістичність і складність завдання, оскільки багато систем, досягають 100% точності на більшості завдань (наприклад, Weston et al., 2014).

Дослідники побудували закриті набори даних (cloze datasets), в яких мета полягає в тому, щоб передбачити відсутнє слово (часто названий суб'єкт) у пасажі. Оскільки ці набори даних можуть автоматично генеруватися з природних даних, вони можуть бути надзвичайно великими. Дитячий тест книги (CBT) (Hill et al., 2015), наприклад, передбачає передбачення вилученого слова речення з урахуванням 20 попередніх речень. Hermann et al. (2015) побудували корпус запитань стилю скульптури, відкинувши об'єкти в абстракційних резюмех статей CNN / Daily News; мета полягає в заповненні об'єкта на основі оригінальної статті.Хоча розмір цього набору даних вражає, Chen et al. (2016) показали, що набір даних вимагає менше міркувань, ніж вважалося раніше і прийшли до висновку, що продуктивність достатня.

Загалом, ризик використання великих наборів даних синтетичного QA полягає в тому, що нейронні методи є надзвичайно потужними в процесі зворотного проектування (“reverse-engineering”), за допомогою якого було створено набір даних, або так підібрані особливості, щоб перевершити та досягти успіху в цьому, без обов'язкового просування розуміння мови або міркування.

## **ПОСТАНОВКА ЗАДАЧІ**

Метою даної роботи є дослідити декілька різних наборів даних (а саме: AI2 Reasoning Challenge (ARC), SQuAD (Stanford Question Answering Dataset), Shaping Answers with Rules through Conversation (ShARC)) для створення питально-відповідних довідкових систем (Question answering system): з яких саме даних складається набір, розподіл на тренувальну вибірку, для перехресної перевірки (cross-validation dataset) та тестову, які саме проблеми виникають при побудові моделей та їх точність на цих наборах даних.

Обрати сучасну модель нейронної багатошарової мережі та на одному із цих наборів даних побудувати та натренувати її.

## ТЕОРЕТИЧНІ ВІДОМОСТІ

### РОЗДІЛ 1. SQuAD: 100,000+ Questions for Machine Comprehension of Text

Стенфордський набір відповідей на запитання (**Stanford Question Answering Dataset, SQuAD**) — набір даних для читання, що складається з 100'000+ запитань, поставлених роботодавцями з набору статей Вікіпедії, де відповідь на кожне запитання є сегментом тексту з відповідний читання. В даному наборі проаналізовано набір даних для розуміння типів міркувань, необхідних для відповіді на запитання, в значній мірі спираючись на дерева залежностей та виборчих дільниць. Для даного набору даних побудовано сильну модель логістичної регресії, яка досягає F1-score в 51,0%, що є значним поліпшенням порівняно з простою базовою лінією (20%). Однак, продуктивність людини (86,8%) набагато вища, що свідчить про те, що набір даних є важливою проблемою для майбутніх досліджень. Даний набір даних вільно доступний за адресою — <https://stanford-qa.com>. [2]

#### 1.1 Аналіз набору даних SQuAD (Stanford Question Answering Dataset)

Набір запитань SQuAD містить 107'785 пар питань-відповідей у 536 статтях і майже на два порядки більше попередніх ручних міток даних, таких як MCTest. [2]

На відміну від попередніх наборів даних, SQuAD не надає список варіантів відповідей для кожного питання. Навпаки, система повинна сама вибирати відповідь з усіх можливих варіантів, таким чином, потрібно впоратися з досить великою кількістю кандидатів (Рисунок 1.1.1 та Рисунок 1.1.2). Щоб оцінити складність SQuAD спочатку було запроваджено модель логістичної регресії з низкою особливостей. Особливості шляху до дерев лексикализованих і дерев залежностей є важливими для продуктивності моделі. Також виявлено, що продуктивність моделі погіршується із збільшенням складності типів

відповідей і синтаксичної розбіжності між питанням і пропозицією, що містить відповідь. Цікаво, що такої деградації для людини немає. Найкраща модель досягала 51,0% балів F1, що набагато краще за базову лінію ковзних вікон (sliding window base-line) — 20%. Ці результати все ще сильно відстають від людських показників, що становить 86,8% F1-score. Це говорить про те, що в наборі даних SQuAD є багато можливостей для просування в моделюванні та навчанні. [2]

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

What causes precipitation to fall?

**gravity**

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?

**graupel**

Where do water droplets collide with ice crystals to form precipitation?

**within a cloud**

Рисунок 1.1.1 — Пари (питання та відповідь), як зразок проходу в наборі даних SQuAD. Кожна з відповідей — це фрагмент тексту з уривка.

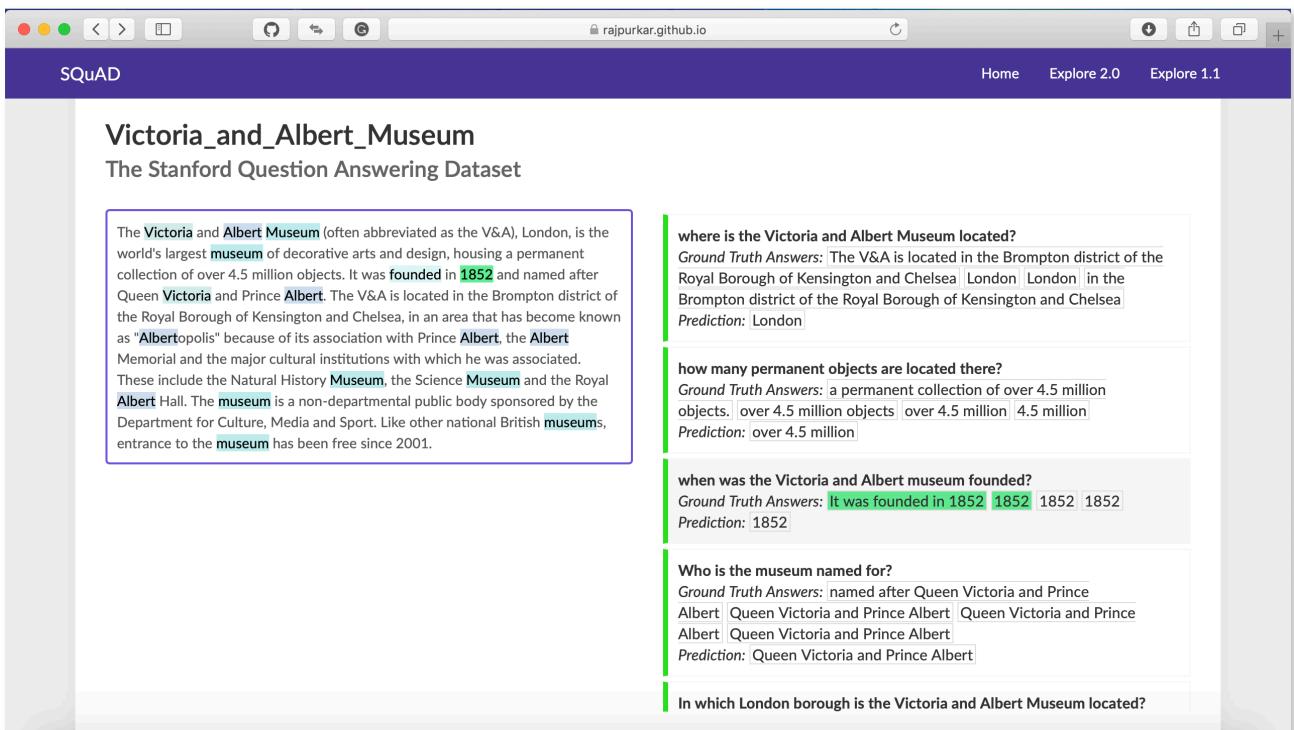


Рисунок 1.1.2 — Кожна з відповідей — це фрагмент тексту з уривка.

Відмінність між питаннями SQuAD і запитаннями з закритого набору даних є те, що відповіді на запитання в закритому наборі — є єдиними словами або сутностями, тоді як відповіді в SQuAD часто включають не-сутності і можуть бути набагато довшими фразами.

## 1.2 Колекція наборів даних (Dataset Collection)

Даний набір даних зібрано у три етапи: кураторські уривки, краудсорсинг-відповіді на ці уривки і отримання додаткових відповідей.

Щоб отримати високоякісні статті, було використано внутрішню сторінку PageRanks Вікіпедії, щоб одержати 10'000 статей з англійської Вікіпедії, з яких вибірково випадково вибрано 536 статей. З кожної з цих статей було витягнуто окремі абзаци, виключаючи зображення, малюнки, таблиці та відкидаючи пункти, коротше ніж 500 символів. Результат склав 23'215 пунктів для 536 статей, що охоплюють широкий спектр тем, від музикантів до абстрактних концепцій. Статті випадковим чином було розділено на **навчальний набір (80%)**, **набір для розробки (10%)** і **тестовий набір (10%)**.

Далі було використо людей для створення запитань. Було використано платформу Daemo (Gaikwad et al., 2015) та Amazon Mechanical Turk — як бекенд. Працівників попросили витратити 4 хвилини на кожен пункт і заплатили за кількість годин, необхідних для завершення статті.

На кожному пункті робітникам було доручено запитати і відповісти на 5 запитань щодо змісту цього пункту. Питання повинні були бути введені в текстове поле, і відповіді повинні бути виділені в абзаці. Для керівництва робітників завдання містили зразок абзацу, а приклади хороших і поганих питань і відповідей на цей пункт разом з причинами, які вони класифікували як такі. Крім того, робітникам було запропоновано задавати питання власними словами, не копіюючи словосполучення з абзацу.

Щоб отримати індикацію продуктивності людини на наборі даних SQuAD і зробити оцінку більш надійною, було отримано принаймні 2 додаткові відповіді на кожне питання в наборі розробки та тестування та створено колекцію додаткових відповідей. У задачі генерації вторинних відповідей кожному працівнику було показані лише питання разом з абзацами статті, і було запропоновано вибрати найкоротший проміжок у абзаці, який відповів на запитання. Якщо питання не відповідає певному пункту, працівників попросили подати питання без відмітки відповіді. Приблизно 2,6% питань були відмічені невідповідними принаймні одним з додаткових працівників.

### 1.3 Аналіз типів питань

Щоб зрозуміти властивості SQuAD, проаналізовано питання і відповіді в наборі розробок (development set). Зокрема, досліджено різноманітність типів відповідей, складність питань з точки зору типу міркувань, необхідних для відповіді на них і ступінь синтаксичного розходження між питаннями і відповідями.

#### Різноманітність відповідей

Автоматично класифіковано відповіді наступним чином: спочатку відокремлено числові та нечислові відповіді. Нечислові відповіді розподіляються за категоріями з використанням синтаксичних розрахунків і тегів POS, створених за допомогою Stan-Cored CoreNLP. Правильні фрази іменників далі розбиваються на людину, місце розташування та інші об'єкти, використовуючи теги NER. У Таблиці 1.2.1 наведено дати та інші цифри, що становлять 19,8% даних; 32,6% відповідей є власними іменниками трьох різних типів; 31,8% — це звичайні відповіді на іменники; решта 15,8% складаються з фрази прикметників, фрази дієслів, статті та інші типи.

Таблиця 1.2.1 — Розподіл питань по типам в наборі даних SQuAD

Answer type	Percentage	Example
Date	9 %	19 October 1512
Other Numeric	11 %	12
Person	13 %	Thomas Coke
Location	4 %	Germany
Other Entity	15 %	ABC Sports
Common Noun Phrase	32 %	property damage
Adjective Phrase	4 %	second-largest
Verb Phrase	6 %	returned to Earth
Clause	4 %	to avoid trivialization
Other	3 %	quietly

### Обґрунтування необхідного для відповіді на запитання

Щоб краще зрозуміти міркування, необхідні для відповіді на запитання, відібрано 4 запитання з кожної з 48 статей у наборі розробок, а потім вручну позначили приклади категоріями, наведеними у Таблиці 1.2.2, якась лексична або синтаксична розбіжність між питанням і відповіддю в уривку. Зауважимо, що деякі приклади підпадають під більш ніж одну категорію.

Таблиця 1.2.2 — Слова, що мають відношення до відповідного типу міркувань, виділено жирним шрифтом, а відповідь, надана краудсорсингу, підкреслена.

Reasoning	Description	Example	Percentage
Lexical variation (synonymy)	Major correspondences between the question and the answer sentence are synonyms.	Q: What is the Rankine cycle sometimes <b>called</b> ? Sentence: The Rankine cycle is sometimes <b>referred</b> to as a practical Carnot cycle.	33.3%
Lexical variation (world knowledge)	Major correspondences between the question and the answer sentence require world knowledge to resolve.	Q: Which <b>governing bodies</b> have veto power? Sen.: <b>The European Parliament and the Council of the European Union</b> have powers of amendment and veto during the legislative process.	9.1%
Syntactic variation	After the question is paraphrased into declarative form, its syntactic dependency structure does not match that of the answer sentence even after local modifications.	Q: What Shakespeare scholar is <b>currently on the faculty</b> ? Sen.: <b>Current faculty include</b> the anthropologist Marshall Sahlins, ..., Shakespeare scholar <b>David Bevington</b> .	64.1%
Multiple sentence reasoning	There is anaphora, or higher-level fusion of multiple sentences is required.	Q: What collection does <b>the V&amp;A Theatre &amp; Performance galleries</b> hold? Sen.: <b>The V&amp;A Theatre &amp; Performance galleries</b> opened in March 2009. ... <b>They</b> hold the UK's biggest national collection of material about live performance.	13.6%
Ambiguous	We don't agree with the crowd-workers' answer, or the question does not have a unique answer.	Q: What is the main goal of criminal punishment? Sen.: <b>Achieving crime control via incapacitation and deterrence</b> is a major goal of criminal punishment.	6.1%

### Стратифікація за синтаксичною дивергенцією

Також розроблено автоматичний метод для кількісного визначення синтаксичної розбіжності між питанням і реченням, що містить відповідь. Це забезпечує інший спосіб вимірювання складності питання та стратифікації набору даних, який ми повертаємо до розділу.

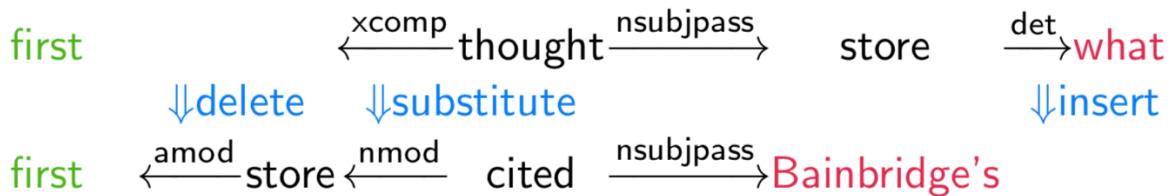
Проілюстровано, як вимірюється дивергенція на рисунку 1.2.1:

---

Q: What department store is thought to be the first in the world?  
S: Bainbridge's is often cited as the world's first department store.

---

Path:



Edit cost:

1

+2

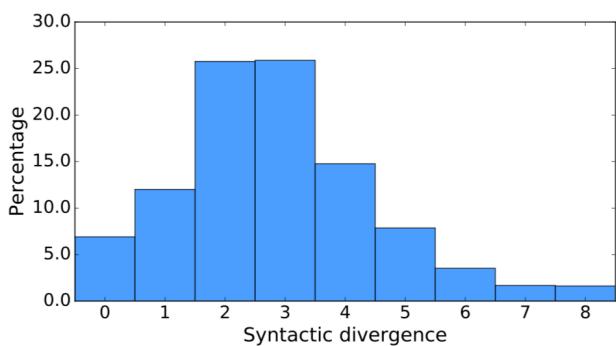
+1=4

---

Рисунок 1.2.1 — Приклад ходьби через обчислення синтаксичної розбіжності між питанням Q і відповіддю на вирок S

Спочатку виявлено якорі (слова-леми, спільні як для питань, так і для відповідей); у прикладі якір є “першим”. Два нелексикалізованих шляху, один від якоря “перший” у питанні до wh-word “what”, та інший з якоря у реченні відповіді та діапазону відповіді “Bainbridge's”, витягаються з розбору залежності дерев. Вимірюється відстань редагування між цими двома шляхами, яку визначену як мінімальну кількість видалень або вставок для перетворення одного шляху в інший. Синтаксична дивергенція визначається як мінімальна відстань редагування від усіх можливих якорів.

Гістограма на рисунку 1.2.2 показує, що в даному наборі даних є широкий діапазон синтаксичних розбіжностей. Також показано конкретний приклад, де відстань редагування дорівнює 0, а інша — 6. Синтаксична дивергенція ігнорує лексичну зміну. Крім того, невелика дивергенція не означає, що питання є простим, оскільки можуть бути інші кандидати з аналогічно малими розбіжностями.

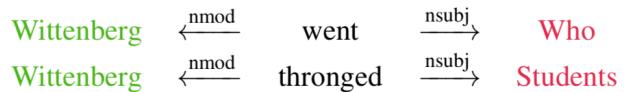


(a) Histogram of syntactic divergence.

Q: Who went to Wittenberg to hear Luther speak?

S: Students thronged to Wittenberg to hear Luther speak.

Path:

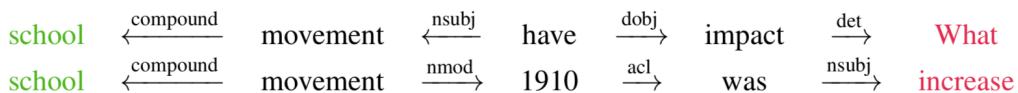


(b) An example of a question-answer pair with edit distance 0 between the dependency paths (note that lexical variation is ignored in the computation of edit distance).

Q: What impact did the high school education movement have on the presence of skilled workers?

S: During the mass high school education movement from 1910 – 1940 , there was an increase in skilled workers.

Path:



(c) An example of a question-answer pair with edit distance 6.

Рисунок 1.2.2 — Використовується відстань редагування між шляхами незалежних залежностей у запиті та реченні, що містить відповідь на вимірювання синтаксичної дивергенції.

## Генерація відповідей кандидата

Для всіх чотирьох методів, замість того, щоб розглядати всі проміжки О (L2) як відповіді, де L — кількість слів у реченні, використовуються лише проміжки, які є складовими в розрізі округів, сформованим Stanford CoreNLP. Ігноруючи пунктуацію та статті, знаходимо, що 77,3% правильних відповідей у наборі розробки є складовими. Це ставить ефективну межу на точність методів. Під час навчання, коли правильна відповідь прикладу не є складовою, використовується найкоротша складова, що містить правильну відповідь, як ціль.

## Вікна базової лінії (Sliding Window Baseline)

Для кожної відповіді кандидата обчислюється перекриття уніграм/біграми між реченням, що містять його (виключаючи самого кандидата) і питанням.

Зберігаються всі кандидати, які мають максимальне перекриття. Серед них обирається найкращий з використанням підходу ковзного вікна (*sliding-window approach*), запропонованого в Richardson et al. (2013).

## РОЗДІЛ 2. AI2 Reasoning Challenge (ARC) Dataset

### 2.1 Аналіз набору даних

Набір запитань ARC [1] розділяється на «складний» набір запитань (Challenge Set), що є більш складним, і легкий набір (Easy Set). Набір «складних» запитань містить лише питання, на які неправильно відповів алгоритм на основі пошуку (retrieval-based algorithm) і алгоритм співучасності слів (co-occurrence algorithm).

Набір даних містить лише природні, наукові питання шкільного рівня та створені для тестів з людьми і є найбільшим набором публічних доменів цього типу (7'787 питань). Було перевірено декілька базових моделей на наборі викликів (Challenge Set), включаючи провідні нейронні мережі, що використовуються в завданнях наборів даних SQuAD [2] і SNLI, і виявлено, що жодна з них не може значно перевершити випадкову базову лінію, що відображає складну природу цього завдання. Також було випущено ARC Corpus: корпус, що містить 14 мільйонів наукових речень, які є релевантними щодо завдання, і перевірено на результативність три базові нейронні мережі.

Набір даних ARC складається з 7'787 природничо-наукових питань, а саме з питань, створених для використання на стандартизованих тестах. Задля заохочення уваги на прогресивних явищах, було розділено ARC на набір викликів (Challenge Set) (2'590 питань), що містить питання, на які неправильно відповів алгоритм на основі пошуку і алгоритм спільного слова, а також легкий набір (Easy Set) (5'197 питань), що містить залишок.

Наприклад, два типові запитання із набору даних:

*Which property of a mineral can be determined just by looking at it?*

- (A) *luster [correct]*
- (B) *mass*
- (C) *weight*
- (D) *hardness*

*A student riding a bicycle observes that it moves faster on a smooth road than on a rough road. This happens because the smooth road has*

- (A) less gravity
- (B) more gravity
- (C) less friction **[correct]**
- (D) more friction

На обидва ці запитання важко відповісти за допомогою простого пошуку або кореляції слова. Наприклад, не існує веб-речень форми «бліск можна визначити, дивлячись на щось»; так само «мінерал» сильно корелюється з «твердістю» (неправильна опція відповіді). Вони вимагають більш просунутих методів контролю якості.

Набір даних ARC складається з наукових запитань, всіх без діаграм, що містять множинний вибір (як правило, 4-х кратний вибір). Вони отримані з різних джерел. У Таблиці 1.1.1 наведено розміри вибірок: тренувальна, для перехресної перевірки (cross-validation dataset) та тестова із набора даних ARC. Словник налічує 6329 різних слів.

Таблиця 1.1.1 — Кількість питань у розділах ARC

	<b>Challenge</b>	<b>Easy</b>	<b>Total</b>
<b>Train</b>	1119	2251	3370
<b>Dev (cross-validation)</b>	299	570	869
<b>Test</b>	1172	2376	3548
<b>TOTAL</b>	<b>2590</b>	<b>5197</b>	<b>7787</b>

Питання змінюються залежно від цільового рівня студентів (як це визначено експертами, які є авторами запитань), від 3-го класу до 9-го класу, тобто студентів віком від 8 до 13 років. Таблиця 1.1.2 показує розбивку набору на основі рівня класу з абсолютними підрахунками (#) і відсотками (%) від «складного» набору та «легкого». На практиці існує суттєве перекривання

труднощів між рівнями (також спостерігається в подібному розподілі рівнів), оскільки кожен рівень містить суміш легких і складних питань.

Таблиця 1.1.2 — Розподіл питань по класах в наборі даних ARC

Grade	Challenge		Easy	
	%	(# qns)	%	(# qns)
3	3.6	(94 qns)	3.4	(176 qns)
4	9	(233)	11.4	(591)
5	19.5	(506)	21.2	(1101)
6	3.2	(84)	3.4	(179)
7	14.4	(372)	10.7	(557)
8	41.4	(1072)	41.2	(2139)
9	8.8	(229)	8.7	(454)

## 2.2 Типи питань

Запитання, що включені до набору даних ARC стосуються як різних стилів знань, так і різних стилів міркування. У додатку А та додатку Б перераховано всі типи запитань та міркувань, які наявні в наборі даних ARC, на основі вибірки зі 100 запитань.

Відносні розміри цих категорій показані на рисунках 1.2.1 та 1.2.2. Ці розміри, звичайно, є приблизними, оскільки вимагають суб'єктивного судження про основну проблему, що виникає при різних питаннях. Тим не менш, це допомагає забезпечити розуміння міркувань, що лежать в основі набору даних ARC.

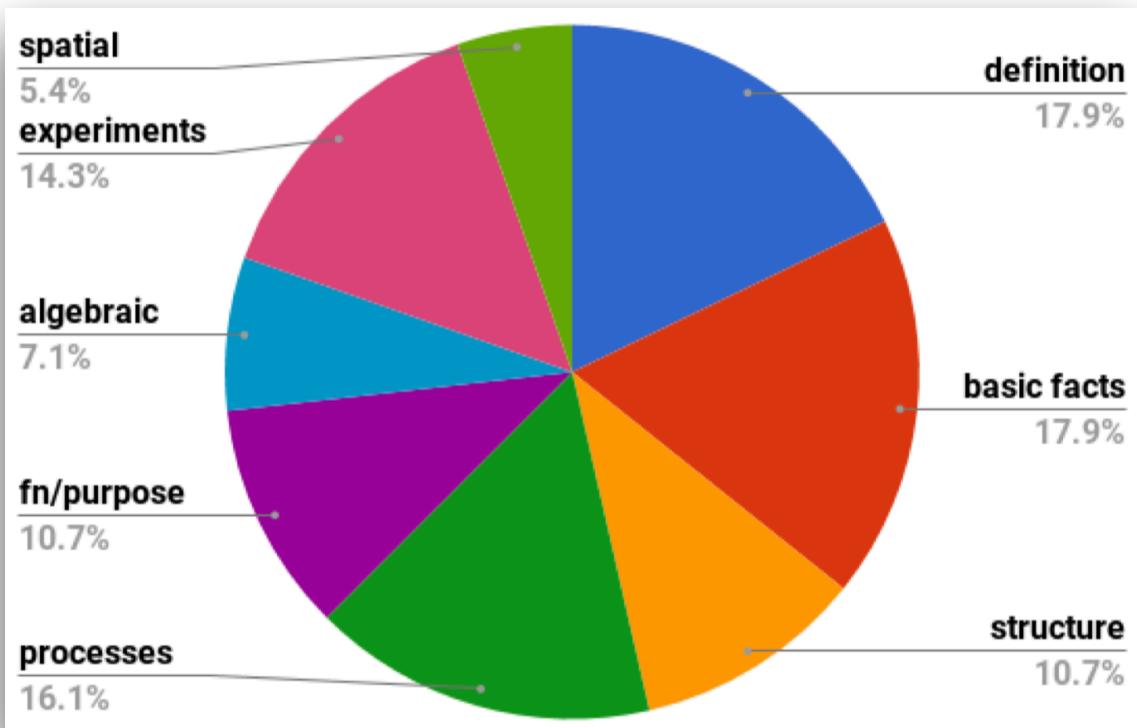


Рисунок 1.2.1 — Відносні розміри різних типів запитань, запропоновані набором ARC Challenge Set

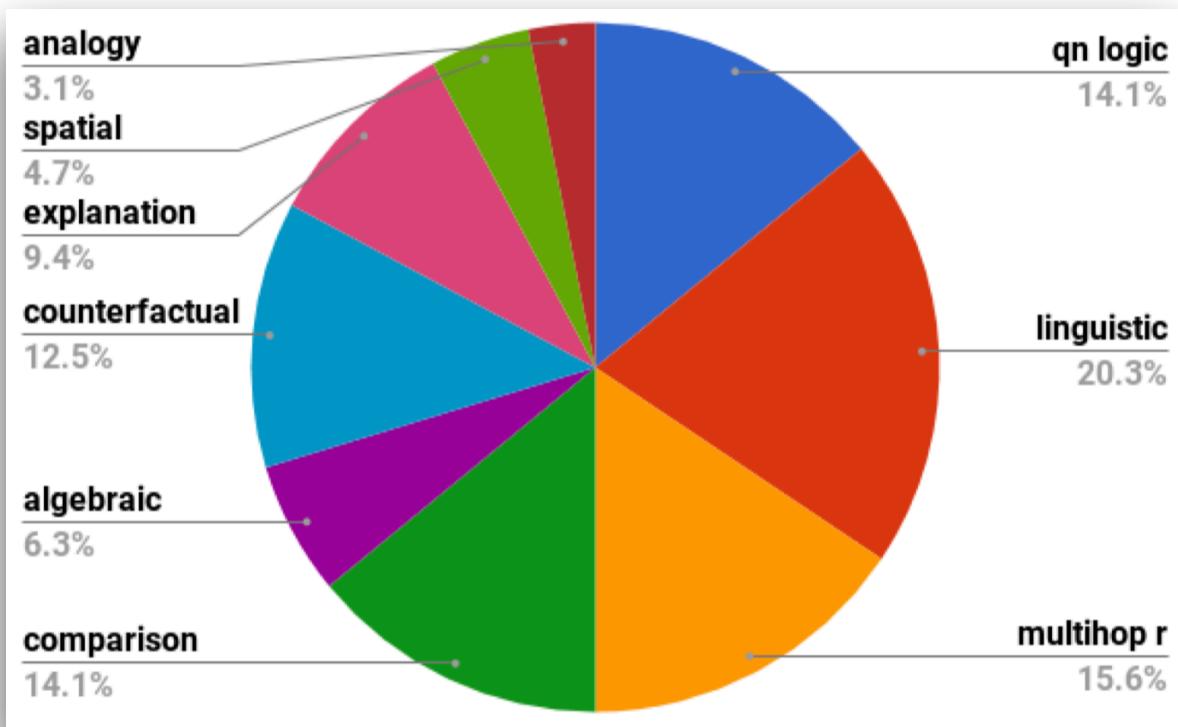


Рисунок 1.2.2 — Відносні розміри різних типів міркувань, запропоновані набором ARC Challenge Set

## 2.3 Розширений корпус

В додаток до набору питань ARC, також було випущено ARC корпус: великий корпус науково-спрямованих речень (знань), що були взяті з інтернету. Цей корпус містить 14 мільйонів речень (1,4 Гб тексту) і містить багато знань, необхідних для відповіді на «складні» запитання (Challenge Questions).

## 2.4 Алгоритм інформаційного пошуку

Перший фільтр, який використовується до набору даних — це алгоритм інформаційного пошуку (IR-solver) [3]. IR-solver використовує корпус Waterloo [3], веб-корпус з  $5 \times 10^{10}$  лексем, що важить 280 ГБ. Даний алгоритм шукає, чи запитання  $q$  разом з опцією відповіді явно зазначено в корпусі, і повертає впевненість, що таке судження знайдене. Для цього, для кожного варіанту відповіді  $a_i$ , він надсилає  $q + a_i$  як запит до пошукової системи (використовується Elasticsearch), і повертає значення пошукової системи для найпопулярнішого речення  $s$ , де  $s$  також має щонайменше одне накладання безперервного слова з  $q$  і принаймні одне з  $a_i$ ; це гарантує, що  $s$  має певну актуальність як для  $q$ , так і для  $a_i$ . Це повторюється для всіх варіантів  $a_i$  для того, щоб заповнити їх усі опцією з найвищим балом.

## 2.5 Алгоритм знаходження вказівника з точковою інформацією

Другий фільтр, який застосовується до набору даних — це алгоритм знаходження вказівника з точковою інформацією (The Pointwise Mutual Information) [3]. При цьому використовується той самий корпус, що використовував й IR-solver, і формалізується спосіб обчислень і застосування асоціативних знань. Враховуючи питання  $q$  та варіант відповіді  $a_i$ , він використовує PMI або точкову взаємну інформацію [4] для вимірювання сили

асоціацій між частинами  $q$  і частинами  $a_i$ . Враховуючи великий корпус  $C$ , PMI для двох n-грам  $x$  і  $y$  визначається як

$$PMI(x, y) = \log \frac{p(x, y)}{p(x)p(y)}. \quad (1)$$

Тут  $p(x, y)$  є спільна ймовірність того, що  $x$  і  $y$  зустрічаються разом у корпусі  $C$ , у певному вікні тексту (використовується вікно з 10 слів). Вираз  $p(x)p(y)$ , являє собою ймовірність, з якою  $x$  і  $y$  будуть з'являтися разом, якщо вони були статистично незалежними. Відношення  $p(x, y)$  до  $p(x)p(y)$  — це відношення спостережуваної спільної появи двох слів до очікуваної спільної появи. Чим більше це співвідношення, тим сильніша асоціація між  $x$  та  $y$ . На даному етапі виводиться відповідь з найбільшим середнім PMI, розрахованим по всіх парах n-грам запитань і опцією n-грамів відповіді.

## 2.6 Набір складних питань

Щоб проілюструвати вплив використання цих алгоритмів як фільтрів при визначенні набору викликів, розглянемо наступний приклад:

*Which property of air does a barometer measure?*

- (A) speed
- (B) **pressure [correct]**
- (C) humidity
- (D) temperature

Дане питання було виключене з набору викликів (Challenge Set), оскільки на нього правильно відповідали як алгоритми IR, так і PMI (питання було б виключено, навіть якщо б один із алгоритмів дав правильну відповідь). Алгоритм IR знаходить кілька речень, що підтримують правильну відповідь, наприклад,

- *Air pressure* is measured with a **barometer**.
- *Air pressure* will be measured with a barometer.
- *The aneroid barometer* is an instrument that does not use liquid in measuring the **pressure** of the air.
- *A barometer* measures the **pressure** of air molecules.

Аналогічним чином алгоритм РМІ знаходить, що «**барометр**» і «**тиск**» (а також «**новітря**» і «**тиск**») спільно виникають у корпусі надзвичайно часто (високий показник РМІ).

Розглянемо інше запитання:

*Which property of a mineral can be determined just by looking at it?*

- (A) *luster* [**correct**]
- (B) *mass*
- (C) *weight*
- (D) *hardness*

На це запитання неправильно відповідають обидва алгоритми: немає жодних висловлювань, схожих на "бліск матеріалу, який можна визначити, дивлячись на нього". Аналогічно, «**мінерал**» часто спільно зустрічається з кількома неправильними варіантами відповіді (наприклад, *масою*, *твердістю*), що заплутує алгоритм РМІ. Таким чином, це питання є частиною складного набору, оскільки, як визначено, вимагає більш просунутого методу відповіді.

## ПРАКТИЧНА ЧАСТИНА

### РОЗДІЛ 1. Bi-Directional Attention Flow (BIDAF) Reading Comprehension Model

#### 1.1 Модель двонаправленого потоку для машинного розуміння

BIDAF включає в себе векторні представлення на рівні символів, на рівні слів і на рівні фрази, а також використовує двонаправлений потік уваги для подання контекстних запитів. Цей механізм уваги пропонує наступні вдосконалення попередньо популярних парадигм уваги:

- **По-перше**, даний шар уваги [11] не використовується для узагальнення кожної модальності в єдиний вектор. Замість цього, обчислюється увага для кожного етапу часу, і присутній вектор на кожному кроці часу, разом з поданнями з попередніх шарів (Рисунок 1.1.1), дозволяється протікати до наступного шару моделювання (RNN). Це зменшує втрати інформації, викликані ранньою підсумовуванням.

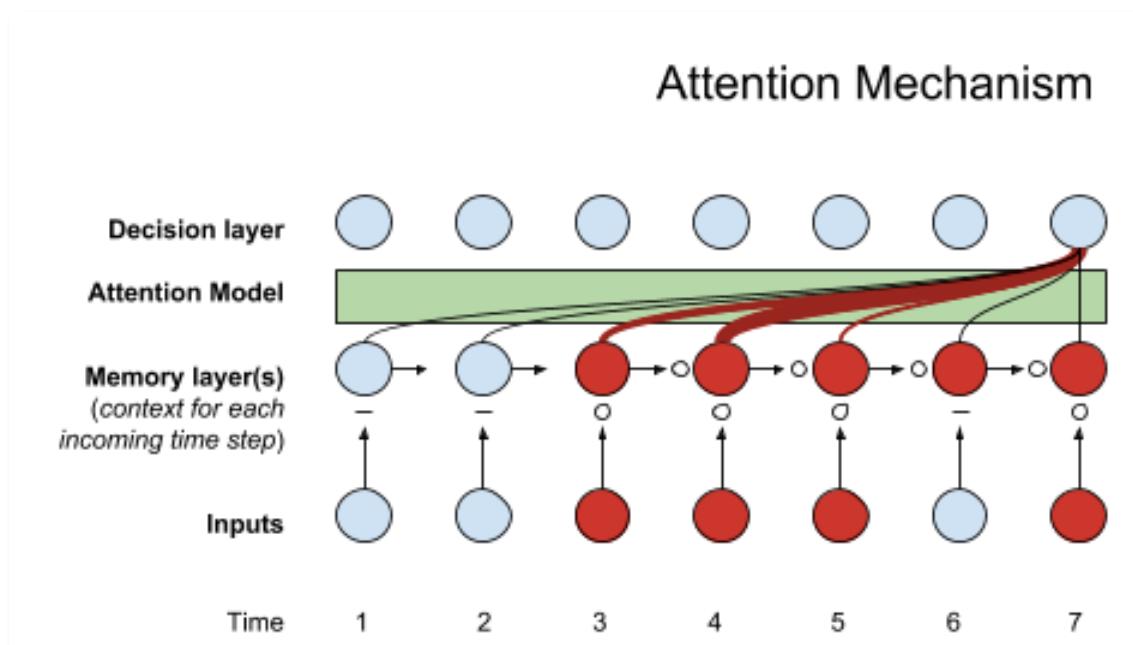


Рисунок 1.1.1 — Шар уваги (Attention Mechanism)

- **По-друге**, використовується механізм уваги, менший від пам'яті.

Тобто, ітеративно обчислюємо увагу через час [12], увага на кожному кроці часу є функцією тільки запиту і контексту на поточному етапі часу і безпосередньо не залежить від уваги на попередньому кроці часу. Ми припускаємо, що це спрощення призводить до поділу праці між шаром уваги і шаром моделювання (RNN). Це змушує шар уваги зосереджуватися виключно на вивчені уваги між контекстом і запитом і дозволяє шару моделювання (RNN) вивчати взаємодію в контекстному поданні запитів (вихідний шар шару уваги). Це також дозволяє увазі на кожному кроці часу бути незмінним від неправильних відвідувань у попередніх кроках часу. Експерименти показують, що без уваги пам'ять дає чітку перевагу над динамічною увагою.

- **По-третє**, використовується механізм уваги в обох напрямках, запит до контексту і контекст до запиту, які надають один одному додаткову інформацію.

## 1.2 Багатошарова нейронна модель

Модель машинного розуміння є ієрархічним багатостадійним процесом і складається з шести шарів (Рисунок 1.2.1):

1. **Шар вбудовування символів (Character Embedding Layer)** відображає кожне слово у векторному просторі, використовуючи символи CNN.
2. **Шар вбудовування слів (Word Embedding Layer)** відображає кожне слово у векторному просторі за допомогою попередньо навченої моделі словника.
3. **Шар вставки фрази (Phrase Embedding Layer)** використовує контекстні сигнали від оточуючих слів, щоб уточнити вбудовування слів. Ці перші три шари застосовуються як до запиту, так і до контексту.

4. **Шар потоку уваги (Attention Flow Layer)** з'єднує вектори запитів і контекстів і виробляє набір векторів характеристик запитів для кожного слова в контексті.
5. **Моделюючий шар (Modeling Layer)** використовує нейронну мережу з повторами для сканування контексту.
6. **Вихідний шар (Output Layer)** надає відповідь на запит.

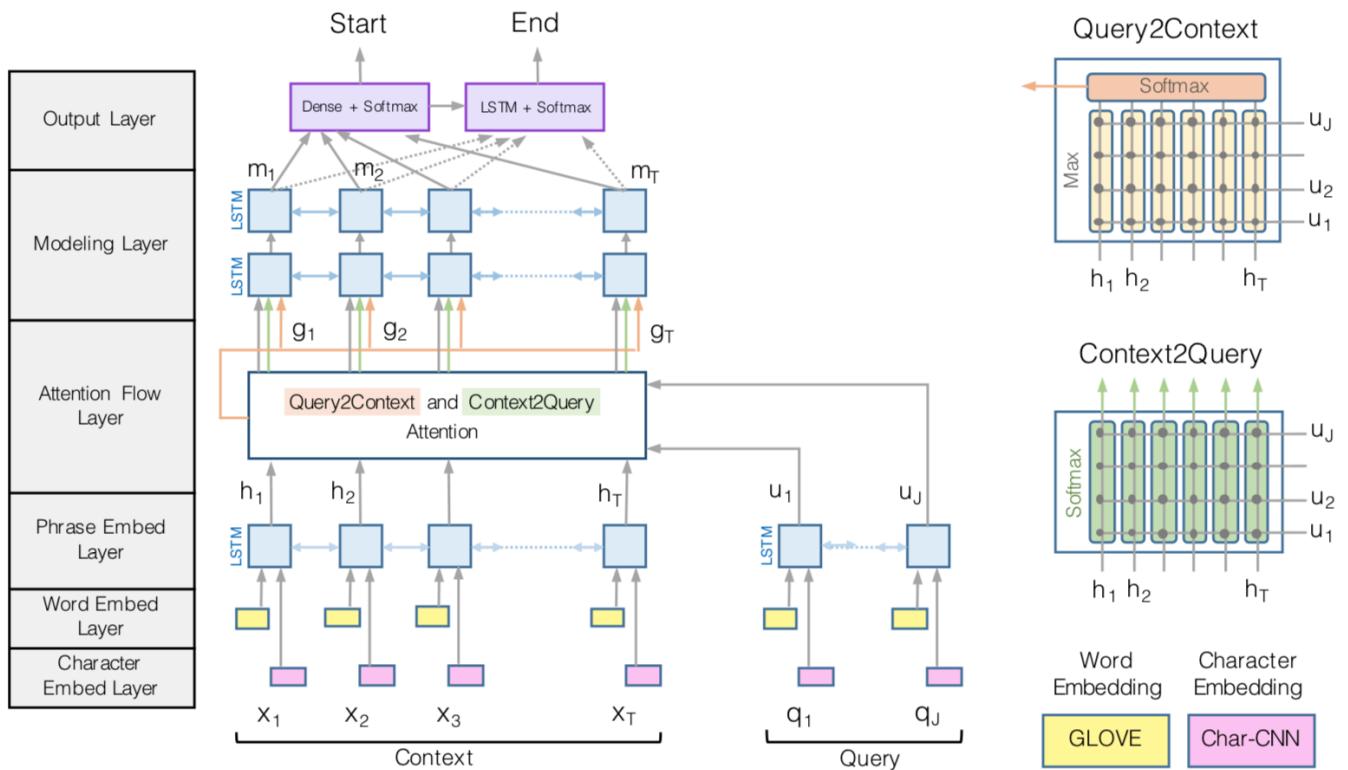


Рисунок 1.2.1 — Модель двонаправленого потоку уваги (BiDirectional Attention Flow Model)

## 1. Шар векторного представлення символів (Character Embedding Layer).

Шар векторного представлення символів відповідає за відображення кожного слова у високомірний векторний простір. Нехай  $\{x_1, \dots, x_T\}$  та  $\{q_1, \dots, q_J\}$  представляють слова у вхідному контексті абзацу та запиту, відповідно. Слідуючи [14], ми отримуємо векторне представлення кожного рівня символів для кожного слова за допомогою згорткової нейронної мережі (Convolutional Neural Networks (CNN)). Символи представляються у векторах, які можна розглядати як 1D-вхід до CNN [15], і розмір яких є розміром вхідного каналу CNN. Виходи CNN максимально об'єднані по всій ширині для отримання вектора фіксованого розміру для кожного слова.

## 2. Шар векторного представлення слів (Word Embedding Layer).

Шар векторного представлення слів відображає кожне слово на високовимірний векторний простір. Використовуються попередньо навчені вектори, GloVe [16] [17], щоб отримати фіксоване будовування слова кожного слова.

Об'єднання векторного представлення символів та слів передається до двошарової мережі Highway Network [18]. Виходи Highway Network — дві послідовності  $d$ -розмірних векторів або, що більш зручно, дві матриці:  $X \in \mathbb{R}^{d \times T}$  для контексту та  $Q \in \mathbb{R}^{d \times J}$  для запиту.

## 3. Шар векторного представлення фрази (Phrase Embedding Layer).

У шарі векторного представлення фрази використовується мережа довгої короткочасної пам'яті (LSTM) [19] поверх векторного представлення, наданого попередніми шарами, для моделювання часових взаємодій між словами. LSTM розміщується в обох напрямках, потім об'єднуємо виходи двох LSTM. Звідси ми отримуємо  $H \in \mathbb{R}^{2d \times T}$  з контекстних векторів слів  $X$  та  $U \in \mathbb{R}^{2d \times J}$  з векторів запиту слів  $Q$ . Зауважимо, що кожен вектор стовпців  $H$  та  $U$  є двовимірним,

через конкатенацію виходів прямого і зворотного LSTM, кожен є  $d$ -розмірним виходом.

Варто зауважити, що перші три шари моделі є обчислювальними ознаками з запиту та контексту на різних рівнях зернистості, схожим на багатостадійне обчислення функцій згорткових нейронних мереж у полі комп'ютерного зору.

#### **4. Шар потоку уваги (Attention Flow Layer).**

Шар потоку уваги відповідає за зв'язування і злиття інформації з контексту та слів запитів. На відміну від раніше популярних механізмів уваги, шар потоку уваги не використовується для узагальнення запиту і контексту в окремі вектори ознак. Навпаки, вектор уваги на кожному кроці часу, разом з векторним представленням з попередніх шарів, дозволяє перейти до наступного шару моделювання. Це зменшує втрати інформації, викликані раннім підсумовуванням.

Вхідні дані для шару — це векторні представлення фразового рівня контексту  $H$  і запиту  $U$ . Виходи шару — це векторні уявлення контекстних слів  $G$ , поряд з векторним представленням на рівні фрази з попереднього шару.

У цьому шарі ми обчислюємо двонаправлену увагу від контексту до запиту, а також від запиту до контексту. Ці уваги виводяться з матриці подібності,  $S \in \mathbb{R}^{T \times J}$ , між векторним представленням рівня фрази контексту ( $H$ ) і запитом ( $U$ ), де  $S_{ij}$  вказує на подібність між  $t$ -тим контекстом слова та  $j$ -м словом запиту. Матриця подібності обчислюється

$$S_{ij} = \alpha(H_{:,t}, U_{:,j}) \in \mathbb{R}, \quad (4.1)$$

де  $\alpha$  — це скалярна функція що кодує подібність між двома вхідними векторами,

$H_{:,t}$  —  $t$ -й стовпчик вектора  $H$ ,

$U_{:,j}$  —  $j$ -й вектор стовпця  $U$ .

Обираємо

$$\alpha(h, u) = w_{(s)}^T [h; u; h \circ u], \quad (4.2)$$

де  $w_{(s)} \in \mathbb{R}^{6d}$  є ваговими коефіцієнтами,

- — є покомпонентний добуток (добуток Адамара),
- [; ] — векторна конкатенація по рядку, а неявне множення — матричне множення.

**Увага контексту до запиту (Context-to-Query Attention).** Увага від контексту до запиту (C2Q) означає, які слова запиту найбільш доречні для кожного контекстного слова. Нехай  $a_t \in \mathbb{R}^J$  представляють ваги уваги на запит, що працює за допомогою  $t$ -го контекстного слова,  $\sum a_{tj} = 1$  для всіх  $t$ . Ваги уваги обчислюються  $a_t = \text{softmax}(S_{t:}) \in \mathbb{R}^J$ , а кожен вектор запиту  $\tilde{U}_{:t} = \sum_j a_{tj} U_{:t}$ . Звідси  $\tilde{U}$  — матриця 2d-by-T, що містить відвідувані вектори запитів для всього контексту.

**Увага запиту до контексту (Query-to-Context Attention).** Увага запиту до контексту (Q2C) означає, які контекстні слова мають найбільшу подібність до одного зі слів запиту, тому є критичними для відповіді на запит. Ми отримуємо ваги уваги за контекстом слова  $a = \text{softmax}(\max_{col}(S)) \in \mathbb{R}^T$ , де функцією максимуму є  $(\max_{col})$ . виконується по всьому стовпцю. Потім беремо контекстний вектор  $\tilde{h} = \sum_t b_t H_{:t} \in \mathbb{R}^{2d}$ . Цей вектор вказує зважену суму найважливіших слів у контексті щодо запиту.  $\tilde{h}$  проходить T разів по колонці і таким чином дає  $\tilde{H} \in \mathbb{R}^{2d \times T}$ .

Нарешті, векторне представлення на рівні фрази та вектори уваги об'єднуються разом, щоб отримати  $G$ , де кожен вектор стовпців може розглядатися як уявлення для кожного контекстного слова. Визначимо  $G$  таким чином:

$$G_{:t} = \beta(H_{:t}, \tilde{U}_{:t}, \tilde{H}_{:t}) \in \mathbb{R}^{d_G}, \quad (4.3)$$

де  $G_{:,t}$  —  $t$ -й вектор стовпця (відповідний  $t$ -му контекстному слову),

$\beta$  — є векторною функцією, що об'єднує три вхідні вектори,

$d_G$  — вихідна розмірність  $\beta$ -функції.

Незважаючи на те, що  $\beta$ -функція може бути довільною нейронною мережею, яку можна тренувати, наприклад, багатошаровий персепtron, приста конкатенація, як і далі, показує хороші показники в наших експериментах:

$$\beta(h, \tilde{u}, \tilde{h}) = [h; \tilde{u}; h \circ \tilde{u}; h \circ \tilde{h}] \in \mathbb{R}^{8d \times T} \quad (d_G = 8d). \quad (4.4)$$

## 5. Шар моделювання (Modeling Layer).

Вхід до шару моделювання — це  $G$ , який кодує репрезентації слів контекстних запитів. Вихід шару моделювання фіксує взаємодію між контекстними словами, обумовленими за запитом. Це відрізняється від шару векторного представлення фрази, який фіксує взаємодію між контекстними словами, незалежно від запиту. Використовуємо два шари двонаправленого LSTM, з вихідним розміром  $d$  для кожного напрямку. Звідси ми отримуємо матрицю  $M \in \mathbb{R}^{2d \times T}$ , яка передається на вихідний шар для прогнозування відповіді. Очікується, що кожен вектор стовпців  $M$  містить контекстну інформацію про слово по відношенню до всього абзацу контексту і запиту.

## 6. Вихідний шар (Output Layer).

Вихідний шар є специфічним для програми. Модульна природа BIDAF дозволяє легко поміняти вихідні шари на основі завдання, а решта архітектури залишається такою ж. Описуємо вихідний шар для завдання QA.

Завдання QA вимагає, щоб модель знаходила підфразу з абзацу для відповіді на запитання. Фраза виводиться шляхом прогнозування початкових і кінцевих показників фрази в абзаці. Отримано розподіл ймовірності початку індексу по всьому абзацу на

$$p^1 = \text{soft max} \left( w_{(p^1)}^T [G; M] \right), \quad (6.1)$$

де  $w_{(p^1)} \in \mathbb{R}^{10d}$  — є навчальним ваговим вектором. Для кінцевого індексу фрази відповіді, ми передаємо  $M$  іншому двонаправленому LSTM-шару і отримуємо  $M \in \mathbb{R}^{2d \times T}$ . Потім ми застосовуємо  $M^2$ , щоб отримати розподіл ймовірності кінцевого індексу аналогічним чином:

$$p^2 = \text{soft max}\left(w_{(p^2)}^T [G; M^2]\right). \quad (6.2)$$

## Навчання

Визначаємо втрату навчання (training loss) [20] як суму логарифмічних ймовірностей істинних початкових і кінцевих показників за прогнозованими розподілами, усередненими по всіх прикладах:

$$L(\Theta) = -\frac{1}{N} \sum_i^N \log(p_{y_i^1}) + \log(p_{y_i^2}), \quad (6.3)$$

де  $\Theta$  — є набором всіх тренувальних ваг у моделі (ваги та зміщення фільтрів CNN та елементів LSTM,  $w_{(s)}$ ,  $w_{(p^1)}$  та  $w_{(p^2)}$ ),

$N$  — є числом прикладів у наборі даних,

$y_i^1$  та  $y_i^2$  — є істинними початковими і кінцевими індексами  $i$ -го прикладу, відповідно,

$p_k$  — вказує  $k$ -те значення вектора  $p$ .

## Тестування

Діапазон відповіді  $(k, l)$  з максимальним значенням  $p_k^1 p_l^2$ , який може бути обчислений в лінійному часі з динамічним програмуванням.

Реалізацію алгоритму наведено в додатку Г.

## РОЗДІЛ 2. Stanford Question Answering Dataset (SQuAD)

### 2.1 (Macro-averaged) F1 score

У статистичному аналізі двійкової класифікації показник F1 (також F-score або F-measure) є мірою точності тесту. Вона враховує як точність  $p$ , так і відкликання тесту, щоб обчислити оцінку:  $p$  — кількість правильних позитивних результатів, поділених на кількість всіх позитивних результатів, повернутих класифікатором, а  $r$  — кількість правильних позитивних результатів, розділених на кількість всіх відповідних зразків (всі зразки, які повинні бути визначені як позитивні). Оцінка F1 — це гармонійне середнє значення точності та відкликання, де оцінка F1 досягає найкращого значення на рівні 1 (ідеальна точність і відкликання) і найгірша при 0. [24]

Традиційна F-міра або збалансований F-бал (F1-бал) — гармонійне середнє значення точності та відкликання:

$$F_1 = \left( \frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Береться максимум F1 над усіма правдивими відповідями для даного питання, а потім середнє за всіма питаннями.

### 2.2 Людська точність (Human Performance)

Кожне з питань у цих наборах має принаймні три відповіді. Щоб оцінити продуктивність людини, розглядається друга відповідь на кожне питання як людське передбачення і зберігаються інші відповіді як основні істинні відповіді. Результат оцінки результативності на тестовому наборі становить **86,8%** для **точної метрики** відповідності та **89,5%** для **F1**. Невідповідність відбувається в основному через включення/виключення несуттєвих фраз, а не фундаментальні розбіжності щодо відповіді.

## 2.3 Точність моделі Bi-Directional Attention Flow (BiDAF) Reading Comprehension Model

Рисунок 1.3.1 показує точність моделі двонаправленого потоку для машинного розуміння (BiDAF) на версії набору даних SQuAD 2.0 в залежності від кількості епох навчання. Бачимо, що з кількістю епох зростає точність, проте вона не перевищує 80%. Точність на тестовому наборі даних для **точної метрики** становить **64%**, та **75%** для F1-score.

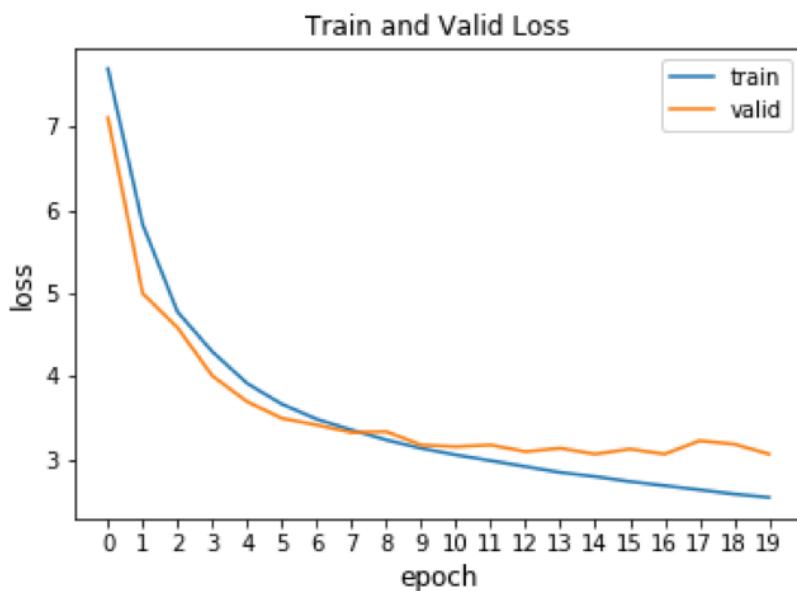


Рисунок 1.3.1 — Точність на тестовому наборі даних

На рисунку 1.3.2 показано значення функції втрат на тренувальному та тестовому наборі даних в залежності від кількості епох навчання. Зі збільшенням кількості епох зменшується значення функції втрат.

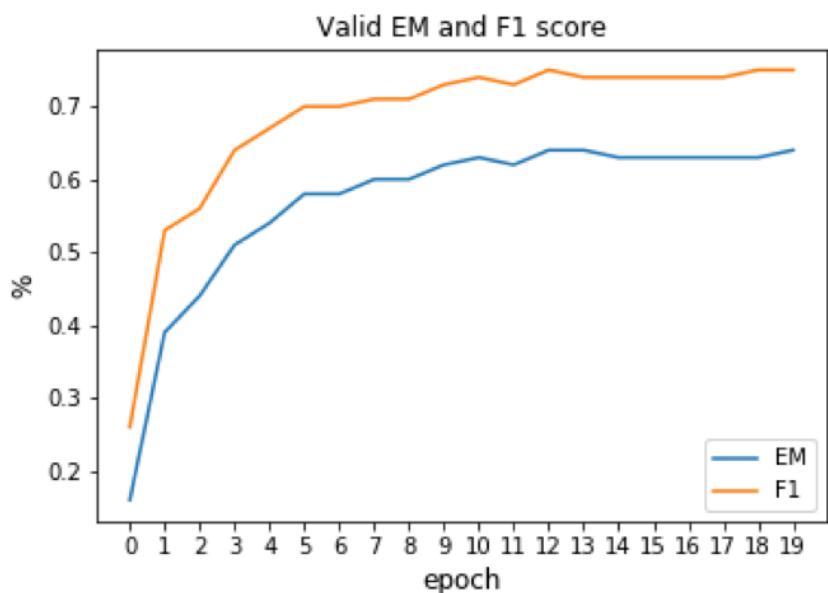


Рисунок 1.3.1 — Значення функції втрат

Точність на тестовому наборі перевірки (validation set) після тренування:

Exact-Match	F1-score
73,3 %	81,1 %

Модель двонаправленого потоку для машинного розуміння (BiDAF) значно перевершує показники наведені в попередніх статтях з використанням стенфордського набору відповідей на запитання (Stanford Question Answering Dataset, SQuAD), але не дотягує до людської точності. Модель здатна правильно вибрати речення, що містить відповідь, з точністю до 75%. Отже, основна частина труднощів полягає у знаходженні точного проміжку в реченні.

Організацію коду програми наведено в додатку В.

### РОЗДІЛ 3. AI2 Reasoning Challenge (ARC) Dataset

Було проведено декілька базових систем контролю якості на складному та простому наборі даних, включаючи дві нейронні моделі, DecompAttn і BiDAF (деталі нижче), які мають близькі досягнення на відомих наборах даних SNLI [5] і SQuAD [2] відповідно.

Системи оцінювались, використовуючи наступну систему оцінки: для кожного питання система отримує 1 бал, якщо вибирає правильну відповідь, і  $1/k$ , якщо вона повідомляє k-way tie (тобто вибирає кілька відповідей), що включає правильну відповідь. Для набору запитань загальна оцінка системи — це сума очок, які вона отримує для всіх питань, розділених на кількість запитань і повідомлених у відсотках.

Було розглянуто реалізації таких систем (Таблиця 1.1.1):

Таблиця 1.1.1 — Продуктивність різних базових систем. Оцінки відображаються у відсотках на тестових наборах

Solver	Test Scores	
	Challenge Set	Easy Set
IR (dataset defn)	1.02	74.48
PMI (dataset defn)	2.03	77.82
IR (using ARC Corpus)	20.26	62.55
TupleInference	23.83	60.81
DecompAttn	24.34	58.27
Guess-all (“random”)	25.02	25.02
DGEM-OpenIE	26.41	57.45
BiDAF	26.54	50.11
TableILP	26.97	36.15
DGEM	27.11	58.97

1. **IR (визначення набору даних).** IR-метод, описаний раніше.
2. **PMI (визначення набору даних).** Метод PMI, описаний раніше.
3. **IR (ARC Corpus).** Алгоритм IR, застосований до IR Corpus.  
Зауважимо, що зміна початкового корпусу, як очікується, призведе до іншої оцінки та точності, якщо тільки два корпуси не будуть сильно корельовані. Корпус, що містить випадкові рядки, наприклад, буде мати дуже низьку кореляцію з початковим корпусом і призведе до випадкового оцінювання приблизно 25%.
4. **Guess-all (“random”).** Базовий метод, який обирає всі варіанти відповіді як рівнозначні, тим самим зраховуючи  $1/k$  для кожного питання з  $k$  вибором відповідей. Система, яка обирає одну відповідь на питання, також буде збігатися з цією оцінкою після достатніх випробувань.
5. **BiDAF (Reading Comprehension Model).** Також була застосована до набору даних BiDAF модель [6], що до того була протестована на високоякісній групі даних SQuAD [2]. Оскільки система BiDAF є системою прямої відповіді, її було адаптовано до вибору QA з кількома виборами, слідуючи підходу, використаному в кількох попередніх проектах [7][8][9] наступним чином: По-перше, з огляду на питання, створюється єдиний абзац шляхом об'єднання набору отриманих речень. У цьому випадку використовуються ті ж речення, що отримані моделями втілення для всіх варіантів зміни. Потім використовується BiDAF модель, щоб обрати проміжок відповіді з цього пункту, враховуючи питання. Нарешті, обираємо опцію з множинним вибором, яка максимально перекриває цей проміжок відповіді (тут визначається як варіант з найвищим відсотком лемматизованих, безперешкодних токенів, що охоплюються діапазоном відповідей BiDAF). BiDAF пройшов навчання на SQuAD, а потім продовжив навчання на ARC нвборі даних.

Розв'язки IR та PMI, наближаються до нуля на множині «складних» питань. Значення трохи вище нульової оцінки пояснюється тим, що вирішувач іноді вибирає декілька (зв'язаних) відповідей, що призводить до часткового кредитування на кілька запитань. Ці питання включені до «складного» набору (Challenge set).

Найбільш вражаючим спостереженням є те, що жоден з алгоритмів не оцінюється значно вище, ніж випадкова базова лінія на комплексі «складних» питань, де 95%, а довірчий інтервал становить  $\pm 2,5\%$ . На відміну від них, їхня продуктивність на наборі «простих» питань зазвичай становить від 55% до 65%. Це підкреслює різну природу та складність комплексу викликів.

## ВИСНОВКИ

Вибір набору даних є дуже впливовими фактором в напрямі досліджень «Question Answering System». Нещодавно розроблені набори даних для забезпечення якості привели до значних успіхів (наприклад, SQuAD), але зосереджувалися на фактичних запитаннях, де тільки поверхневі знання достатні для пошуку відповіді, що заважає прогресу у відповіді на питання з багаторівневими відповідями, що вимагають обґрунтування, розуміння контексту або інших просунутих методів.

Було виявлено, що жодна з випробуваних базових нейронних моделей не досягає значної точності та не може бути ефективною на наборі запитань, включаючи нейронну модель двонаправленого потоку для машинного розуміння (Bi-Directional Attention Flow (BIDAF) Reading Comprehension Model), що показала гарні результати на наборі даних високої продуктивності, такому як SQuAD: EM — 73,3 % та F1 — 81,1 %.

Аналіз даної моделі демонструє важливість кожного шару в даній нейронній моделі. Візуалізації показують, що дана модель вивчає та має відповідне уявлення про розуміння контексту значно краще, ніж запропоновані раніше моделі і здатна відповідати на складні питання, звертаючись до правильних місць в тексті (абзацах). Подальша робота передбачає розширення даного підходу до включення декількох стрибків шару уваги.

Є доцільним продовження досліджень в даному напрямку, оскільки точність таких QA систем наразі далека навіть від 85-90%. Можна спробувати застосувати модель BERT (Pre-training of Deep Bidirectional Transformers for Language Understanding) до даного набору даних та модифікуючи покращити її. Подальша робота передбачає розширення даного підходу та включення декількох стрибків шару уваги (attention layer).

Таким чином, прогрес на наборі даних SQuAD та ARC корпусів буде вражаючим досягненням і буде значним кроком вперед в даному напрямі та побудові побудови питально-відповідних довідкових систем з високою точністю.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge / [P. Clark, I. Cowhey, O. Etzioni та ін.]. // Allen Institute for Artificial Intelligence, Seattle, WA, U.S.A.. – 2018.
2. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Proc. EMNLP'16, 2016.
3. Clark, O. Etzioni, T. Khot, A. Sabharwal, O. Tafjord, P. D. Turney, and D. Khashabi. Combining retrieval, statistics, and inference to answer elementary science questions. In AAAI, pp. 2580–2586, 2016.
4. W. Church and P. Hanks. Word association norms, mutual information and lexicography. In 27th ACL, pp. 76–83, 1989.
5. A large annotated corpus for learning natural language inference / Samuel R. Bowman, Christopher Potts, Gabor Angeli, Christopher D. Manning. // The Stanford Natural Language Processing Group.
6. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. In Proc. ICLR'17, 2017b.
7. Khashabi, T. K. A. Sabharwal, and D. Roth. Question answering as global reasoning over semantic abstractions. In AAAI, 2018b.
8. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions. In Workshop on Noisy User-generated Text, 2017a.
9. Kembhavi, M. Seo, D. Schwenk, J. Choi, A. Farhadi, and H. Hajishirzi. Are you smarter than a sixth grader? text- book question answering for multimodal machine comprehension. In CVPR, 2017.
10. The Stanford Natural Language Inference (SNLI) Corpus [Електронний ресурс] // The Stanford Natural Language Processing Group – Режим доступу до ресурсу: <https://nlp.stanford.edu/projects/snli/>.
11. A Beginner's Guide to Attention Mechanisms and Memory Networks [Електронний ресурс] // Skymind – Режим доступу до ресурсу: <https://skymind.ai/wiki/attention-mechanism-memory-network>.

12. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. ICLR, 2015.
13. BiDAF – Bi-Directional Attention Flow for Machine Comprehension [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: <https://allenai.github.io/bi-att-flow/>.
14. Yoon Kim. Convolutional neural networks for sentence classification. In EMNLP, 2014.
15. Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: <https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf>.
16. Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In EMNLP, 2014.
17. GloVe: Global Vectors for Word Representation [Електронний ресурс] – Режим доступу до ресурсу: <https://nlp.stanford.edu/projects/glove/>.
18. Rupesh Kumar Srivastava, Klaus Greff ,and Jürgen Schmidhuber. Highway networks. arXiv preprint arXiv:1505.00387, 2015.
19. Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. Neural Computation, 1997.
20. Descending into ML: Training and Loss [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: <https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss>.
21. Understanding LSTM Networks [Електронний ресурс] // colah's blog. – 2015. – Режим доступу до ресурсу: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
22. A Brief Overview of Attention Mechanism [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: <https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129>.
23. Daniel Jurafsky. Speech and Language Processing / Daniel Jurafsky, James H. Martin. – 2018.
24. F1-score [Електронний ресурс] – Режим доступу до ресурсу: [https://en.wikipedia.org/wiki/F1\\_score](https://en.wikipedia.org/wiki/F1_score).

## ДОДАТОК А

Таблиця А.1 — Типи запитань, запропоновані в ARC Challenge Set

Тип запитання	Приклад запитання
Визначення	<p><i>What is a worldwide increase in temperature called?</i></p> <p>(A) greenhouse effect (B) global warming (C) ozone depletion (D) solar heating</p>
Основні факти та властивості	<p><i>Which element makes up most of the air we breathe?</i></p> <p>(A) carbon (B) nitrogen (C) oxygen (D) argon</p>
Структура	<p><i>The crust, the mantle, and the core are structures of Earth. Which description is a feature of Earth's mantle?</i></p> <p>(A) contains fossil remains (B) consists of tectonic plates (C) is located at the center of Earth (D) has properties of both liquids and solids</p>
Процеси і причини	<p><i>What is the first step of the process in the formation of sedimentary rocks?</i></p> <p>(A) erosion (B) deposition (C) compaction (D) cementation</p>

Тип запитання	Приклад запитання
Телеологія / Мета	<p><i>What is the main function of the circulatory system?</i></p> <p>(1) secrete enzymes      (2) digest proteins      (3) produce hormones      (4) transport materials</p>
Алгебраїчний	<p><i>If a red flowered plant (RR) is crossed with a white flowered plant (rr), what color will the offspring be?</i></p> <p>(A) 100% pink      (B) 100% red      (C) 50% white, 50% red      (D) 100% white</p>
Експеримент	<p><i>Scientists perform experiments to test hypotheses. How do scientists try to remain objective during experiments?</i></p> <p>(A) Scientists analyze all results.      (B) Scientists use safety precautions.      (C) Scientists conduct experiments once.      (D) Scientists change at least two variables.</p>
Просторовий / кінематичний	<p><i>In studying layers of rock sediment, a geologist found an area where older rock was layered on top of younger rock. Which best explains how this occurred?</i></p> <p>(A) Earthquake activity folded the rock layers...</p>

## ДОДАТОК Б

Таблиця Б.1 —Типи міркувань, запропоновані в ARC Challenge Set

Тип міркувань	Приклад запитання
Логіка запитання	<p><i>Which item below is <b>not</b> made from a material grown in nature?</i></p> <p>(A) a cotton shirt (B) a wooden chair (C) a plastic spoon (D) a grass basket</p>
Мовна відповідність	<p><i>Which of the following best describes a mineral?</i></p> <p>(A) the main nutrient in all foods (B) a type of grain found in cereals (C) a natural substance that makes up rocks (D) the decomposed plant matter found in soil</p>
Розуміння	<p><i>Which property of a mineral can be determined just by looking at it?</i></p> <p>(A) luster (B) mass (C) weight (D) hardness</p>
Порівняння	<p><i>Compared to the Sun, a red star most likely has a greater</i></p> <p>(A) volume. (B) rate of rotation. (C) surface temperature. (D) number of orbiting planets.</p>

**Тип міркувань****Приклад запитання**

If a heterozygous smooth pea plant ( $Ss$ ) is crossed with a homozygous smooth pea plant ( $SS$ ), which are the possible genotypes the offspring could have?

**Алгебраїчний**

- (A) only  $SS$
- (B) only  $Ss$
- (C)  $Ss$  or  $SS$
- (D)  $ss$  or  $SS$

If the Sun were larger, what would most likely also have to be true for Earth to sustain life?

**Гіпотетичний /  
контрфактичний**

- (A) Earth would have to be further from the Sun.
- (B) Earth would have to be closer to the Sun.
- (C) Earth would have to be smaller.
- (D) Earth would have to be larger.

Why can steam be used to cook food?

**Пояснення /  
Мета-міркування**

- (A) Steam does work on objects.
- (B) Steam is a form of water.
- (C) Steam can transfer heat to cooler objects.
- (D) Steam is able to move through small spaces.

Where will a sidewalk feel hottest on a warm, clear day?

**Просторовий /  
кінематичний**

- (A) Under a picnic table
- (B) In direct sunlight
- (C) Under a puddle
- (D) In the shade

**Тип міркувань****Приклад запитання**

---

*Inside cells, special molecules carry messages from the membrane to the nucleus. Which body system uses a similar process?*

**Аналогія**

- (A) *endocrine system*
- (B) *lymphatic system*
- (C) *excretory system*
- (D) *integumentary system*

## ДОДАТОК В

*Структура організації програми:*

— <b>config.py</b>	<– Файл конфігурації з каталогами даних і гіперпараметрами для навчання моделі
— <b>data_loader.py</b>	<– Визначте ітератор, який збирає партії даних для навчання моделі
— <b>eval.py</b>	<– Оцінює модель на новій парі (контекст, питання)
— <b>layers.py</b>	<– Різні шари, які будуть використовуватися основною моделлю BiDAF
— <b>make_dataset.py</b>	<– Завантажує набір даних SQuAD і попередньо обробляє дані для навчання
— <b>model.py</b>	<– Визначає архітектуру моделі BiDAF
— <b>requirements.txt</b>	<– Потрібні бібліотеки Python для створення проекту
— <b>test.py</b>	<– Тестує продуктивність навченої моделі на даних DEV
— <b>train.py</b>	<– Навчає модель, використовуючи лише набір даних TRAIN
— <b>utils.py</b>	<– Групує корисні функції для обробки даних

## ДОДАТОК Г

*Різні шари, які будуть використовуватися основною моделлю BiDAF:*

### layers.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence

from utils import masked_softmax
import config

class Embedding(nn.Module):
    """Embedding layer used by BiDAF, with Words and Characters.
    Args:
        word_vectors (torch.Tensor): Pre-trained word vectors.
        char_vectors (torch.Tensor): Randomly initialized char vectors
        hidden_size (int): Size of hidden activations.
        drop_prob (float): Probability of zero-ing out activations
    """
    def __init__(self, word_vectors, char_vectors, hidden_size, drop_prob):
        super(Embedding, self).__init__()
        self.drop_prob = drop_prob
        self.w_embed = nn.Embedding.from_pretrained(word_vectors, freeze=True)
        self.c_embed = nn.Embedding.from_pretrained(char_vectors, freeze=False)
        self.proj = nn.Linear(word_vectors.size(1), hidden_size, bias=False)
        self.char_conv = nn.Conv2d(1, config.char_channel_size,
                               (config.char_embedding_size, config.char_channel_width))
        self.hwy = HighwayEncoder(2, hidden_size * 2)

    def forward(self, x, y):
        batch_size = x.size(0)

        w_emb = self.w_embed(x)  # (batch_size, seq_len, embed_size)
        w_emb = F.dropout(w_emb, self.drop_prob, self.training)
        w_emb = self.proj(w_emb) # (batch_size, seq_len, hidden_size)

        c_emb = self.c_embed(y)
        c_emb = F.dropout(c_emb, self.drop_prob, self.training)
        c_emb = c_emb.view(-1, config.char_embedding_size,
                           c_emb.size(2)).unsqueeze(1)
        c_emb = self.char_conv(c_emb).squeeze()
        c_emb = F.max_pool1d(c_emb, c_emb.size(2)).squeeze()
        c_emb = c_emb.view(batch_size, -1, config.char_channel_size)

        emb = torch.cat([w_emb, c_emb], dim=-1)
        emb = self.hwy(emb)  # (batch_size, seq_len, hidden_size)

        return emb

class HighwayEncoder(nn.Module):
    """Encode an input sequence using a highway network.
    Based on the paper:
    "Highway Networks"
    """
    pass
```

by Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber  
(<https://arxiv.org/abs/1505.00387>).

Args:

```
    num_layers (int): Number of layers in the highway encoder.  
    hidden_size (int): Size of hidden activations.  
....  
def __init__(self, num_layers, hidden_size):  
    super(HighwayEncoder, self).__init__()  
    self.transforms = nn.ModuleList([nn.Linear(hidden_size, hidden_size)  
                                    for _ in range(num_layers)])  
    self.gates = nn.ModuleList([nn.Linear(hidden_size, hidden_size)  
                               for _ in range(num_layers)])  
  
def forward(self, x):  
    for gate, transform in zip(self.gates, self.transforms):  
        # Shapes of g, t, and x are all (batch_size, seq_len, hidden_size)  
        g = torch.sigmoid(gate(x))  
        t = F.relu(transform(x))  
        x = g * t + (1 - g) * x  
  
return x  
  
class RNNEncoder(nn.Module):  
    """General-purpose layer for encoding a sequence using a bidirectional RNN.  
    Encoded output is the RNN's hidden state at each position, which  
    has shape `(batch_size, seq_len, hidden_size * 2)`.  
    Args:  
        input_size (int): Size of a single timestep in the input.  
        hidden_size (int): Size of the RNN hidden state.  
        num_layers (int): Number of layers of RNN cells to use.  
        drop_prob (float): Probability of zero-ing out activations.  
....  
def __init__(self,  
             input_size,  
             hidden_size,  
             num_layers,  
             drop_prob=0.):  
    super(RNNEncoder, self).__init__()  
    self.drop_prob = drop_prob  
    self.rnn = nn.LSTM(input_size, hidden_size, num_layers,  
                      batch_first=True,  
                      bidirectional=True,  
                      dropout=drop_prob if num_layers > 1 else 0.)  
  
def forward(self, x, lengths):  
    # Save original padded length for use by pad_packed_sequence  
    orig_len = x.size(1)  
  
    # Sort by length and pack sequence for RNN  
    lengths, sort_idx = lengths.sort(0, descending=True)  
    x = x[sort_idx]      # (batch_size, seq_len, input_size)  
    x = pack_padded_sequence(x, lengths, batch_first=True)  
  
    # Apply RNN  
    x, _ = self.rnn(x)  # (batch_size, seq_len, 2 * hidden_size)  
  
    # Unpack and reverse sort  
    x, _ = pad_packed_sequence(x, batch_first=True, total_length=orig_len)  
    _, unsort_idx = sort_idx.sort(0)  
    x = x[unsort_idx]    # (batch_size, seq_len, 2 * hidden_size)  
  
    # Apply dropout (RNN applies dropout after all but the last layer)  
    x = F.dropout(x, self.drop_prob, self.training)  
  
return x
```

```

class BiDAFAttention(nn.Module):
    """Bidirectional attention originally used by BiDAF.
    Bidirectional attention computes attention in two directions:
    The context attends to the query and the query attends to the context.
    The output of this layer is the concatenation of [context, c2q_attention,
    context * c2q_attention, context * q2c_attention]. This concatenation allows
    the attention vector at each timestep, along with the embeddings from
    previous layers, to flow through the attention layer to the modeling layer.
    The output has shape (batch_size, context_len, 8 * hidden_size).
    Args:
        hidden_size (int): Size of hidden activations.
        drop_prob (float): Probability of zero-ing out activations.
    """
    def __init__(self, hidden_size, drop_prob=0.1):
        super(BiDAFAttention, self).__init__()
        self.drop_prob = drop_prob
        self.c_weight = nn.Parameter(torch.zeros(hidden_size, 1))
        self.q_weight = nn.Parameter(torch.zeros(hidden_size, 1))
        self.cq_weight = nn.Parameter(torch.zeros(1, 1, hidden_size))
        for weight in (self.c_weight, self.q_weight, self.cq_weight):
            nn.init.xavier_uniform_(weight)
        self.bias = nn.Parameter(torch.zeros(1))

    def forward(self, c, q, c_mask, q_mask):
        batch_size, c_len, _ = c.size()
        q_len = q.size(1)
        s = self.get_similarity_matrix(c, q)           # (batch_size, c_len, q_len)
        c_mask = c_mask.view(batch_size, c_len, 1)      # (batch_size, c_len, 1)
        q_mask = q_mask.view(batch_size, 1, q_len)      # (batch_size, 1, q_len)
        s1 = masked_softmax(s, q_mask, dim=2)          # (batch_size, c_len, q_len)
        s2 = masked_softmax(s, c_mask, dim=1)          # (batch_size, c_len, q_len)

        # (bs, c_len, q_len) x (bs, q_len, hid_size) => (bs, c_len, hid_size)
        a = torch.bmm(s1, q)
        # (bs, c_len, c_len) x (bs, c_len, hid_size) => (bs, c_len, hid_size)
        b = torch.bmm(torch.bmm(s1, s2.transpose(1, 2)), c)

        x = torch.cat([c, a, c * a, c * b], dim=2)    # (bs, c_len, 4 * hid_size)

        return x

    def get_similarity_matrix(self, c, q):
        """Get the "similarity matrix" between context and query (using the
        terminology of the BiDAF paper).
        A naive implementation as described in BiDAF would concatenate the
        three vectors then project the result with a single weight matrix. This
        method is a more memory-efficient implementation of the same operation.
        See Also:
            Equation 1 in https://arxiv.org/abs/1611.01603
        """
        c_len, q_len = c.size(1), q.size(1)
        c = F.dropout(c, self.drop_prob, self.training)  # (bs, c_len, hid_size)
        q = F.dropout(q, self.drop_prob, self.training)  # (bs, q_len, hid_size)

        # Shapes: (batch_size, c_len, q_len)
        s0 = torch.matmul(c, self.c_weight).expand([-1, -1, q_len])
        s1 = torch.matmul(q, self.q_weight).transpose(1, 2)\n                                         .expand([-1, c_len, -1])
        s2 = torch.matmul(c * self.cq_weight, q.transpose(1, 2))
        s = s0 + s1 + s2 + self.bias

        return s

```

```

class BiDAFOutput(nn.Module):
    """Output layer used by BiDAF for question answering.
    Computes a linear transformation of the attention and modeling
    outputs, then takes the softmax of the result to get the start pointer.
    A bidirectional LSTM is then applied the modeling output to produce `mod_2`.
    A second linear+softmax of the attention output and `mod_2` is used
    to get the end pointer.
    Args:
        hidden_size (int): Hidden size used in the BiDAF model.
        drop_prob (float): Probability of zero-ing out activations.
    """
    def __init__(self, hidden_size, drop_prob):
        super(BiDAFOutput, self).__init__()
        self.att_linear_1 = nn.Linear(8 * hidden_size, 1)
        self.mod_linear_1 = nn.Linear(2 * hidden_size, 1)

        self.rnn = RNNEncoder(input_size=2 * hidden_size,
                              hidden_size=hidden_size,
                              num_layers=1,
                              drop_prob=drop_prob)

        self.att_linear_2 = nn.Linear(8 * hidden_size, 1)
        self.mod_linear_2 = nn.Linear(2 * hidden_size, 1)

    def forward(self, att, mod, mask):
        # Shapes: (batch_size, seq_len, 1)
        logits_1 = self.att_linear_1(att) + self.mod_linear_1(mod)
        mod_2 = self.rnn(mod, mask.sum(-1))
        logits_2 = self.att_linear_2(att) + self.mod_linear_2(mod_2)

        # Shapes: (batch_size, seq_len)
        log_p1 = masked_softmax(logits_1.squeeze(), mask, log_softmax=True)
        log_p2 = masked_softmax(logits_2.squeeze(), mask, log_softmax=True)

    return log_p1, log_p2

```

## ДОДАТОК Д

Завантажує набір даних *SQuAD* і попередньо обробляє дані для навчання:

### make\_dataset.py

```
# external libraries
import os
import tqdm
import json
import zipfile
import tarfile
import pickle
import numpy as np
import urllib.request

# internal utilities
import config
from utils import tokenizer, clean_text, word_tokenize, build_vocab,
build_embeddings, convert_idx

# URL to download SQuAD dataset 2.0
url = "https://rajpurkar.github.io/SQuAD-explorer/dataset"

def maybe_download_squad(url, filename, out_dir):
    # path for local file.
    save_path = os.path.join(out_dir, filename)

    # check if the file already exists
    if not os.path.exists(save_path):
        # check if the output directory exists, otherwise create it.
        if not os.path.exists(out_dir):
            os.makedirs(out_dir)

        print("Downloading", filename, "...")

        # download the dataset
        url = os.path.join(url, filename)
        file_path, _ = urllib.request.urlretrieve(url=url, filename=save_path)

        print("File downloaded successfully!")

    if filename.endswith(".zip"):
        # unpack the zip-file.
        print("Extracting ZIP file...")
        zipfile.ZipFile(file=filename, mode="r").extractall(out_dir)
        print("File extracted successfully!")
    elif filename.endswith((".tar.gz", ".tgz")):
        # unpack the tar-ball.
        print("Extracting TAR file...")
        tarfile.open(name=filename, mode="r:gz").extractall(out_dir)
        print("File extracted successfully!")

class SquadPreprocessor:
    def __init__(self, data_dir, train_filename, dev_filename, tokenizer):
```

```

self.data_dir = data_dir
self.train_filename = train_filename
self.dev_filename = dev_filename
self.data = None
self.tokenizer = tokenizer

def load_data(self, filename="train-v2.0.json"):
    filepath = os.path.join(self.data_dir, filename)
    with open(filepath) as f:
        self.data = json.load(f)

def split_data(self, filename):
    self.load_data(filename)
    sub_dir = filename.split('-')[0]

    # create a subdirectory for Train and Dev data
    if not os.path.exists(os.path.join(self.data_dir, sub_dir)):
        os.makedirs(os.path.join(self.data_dir, sub_dir))

    with open(os.path.join(self.data_dir, sub_dir, sub_dir + '.context'),
              'w', encoding="utf-8") as context_file,\n        open(os.path.join(self.data_dir, sub_dir, sub_dir + '.question'),
              'w', encoding="utf-8") as question_file,\n        open(os.path.join(self.data_dir, sub_dir, sub_dir + '.answer'),
              'w', encoding="utf-8") as answer_file,\n        open(os.path.join(self.data_dir, sub_dir, sub_dir + '.labels'),
              'w', encoding="utf-8") as labels_file:

        # loop over the data
        for article_id in tqdm.tqdm(range(len(self.data['data']))):
            list_paragraphs = self.data['data'][article_id]['paragraphs']
            # loop over the paragraphs
            for paragraph in list_paragraphs:
                context = paragraph['context']
                context = clean_text(context)
                context_tokens = [w for w in word_tokenize(context) if w]
                spans = convert_idx(context, context_tokens)
                qas = paragraph['qas']
                # loop over Q/A
                for qa in qas:
                    question = qa['question']
                    question = clean_text(question)
                    question_tokens = [w for w in word_tokenize(question) if
w]
                    if sub_dir == "train":
                        # select only one ground truth, the top answer, if
any answer
                        answer_ids = 1 if qa['answers'] else 0
                    else:
                        answer_ids = len(qa['answers'])
                        labels = []
                        if answer_ids:
                            for answer_id in range(answer_ids):
                                answer = qa['answers'][answer_id]['text']
                                answer = clean_text(answer)
                                answer_tokens = [w for w in
word_tokenize(answer) if w]
                                answer_start = qa['answers'][answer_id]
                                answer_stop = answer_start + len(answer)
                                answer_span = []
                                for idx, span in enumerate(spans):
                                    if not (answer_stop <= span[0] or
answer_start >= span[1]):
                                        answer_span.append(idx)
                                if not answer_span:

```

```

        continue
    labels.append(str(answer_span[0]) + ' ' +
str(answer_span[-1]))

context_tokens]) + '\n')
question_tokens]) + '\n')
answer_tokens]) + '\n')

def preprocess(self):
    self.split_data(train_filename)
    self.split_data(dev_filename)

def extract_features(self, max_len_context=config.max_len_context,
max_len_question=config.max_len_question,
max_len_word=config.max_len_word, is_train=True):
    # choose the right directory
    directory = "train" if is_train else "dev"

    # load context
    with open(os.path.join(self.data_dir, directory, directory +
".context"), "r", encoding="utf-8") as c:
        context = c.readlines()
    # load questions
    with open(os.path.join(self.data_dir, directory, directory +
".question"), "r", encoding="utf-8") as q:
        question = q.readlines()
    # load answer
    with open(os.path.join(self.data_dir, directory, directory + ".labels"),
"r", encoding="utf-8") as l:
        labels = l.readlines()

    # clean and tokenize context and question
    context = [[w for w in word_tokenize(clean_text(doc.strip('\n')))] for
doc in context]
    question = [[w for w in word_tokenize(clean_text(doc.strip('\n')))] for
doc in question]

    # download vocabulary if not done yet
    if directory == "train":
        labels = [np.array(l.strip("\n").split(), dtype=np.int32) for l in
labels]

        word_vocab, word2idx, char_vocab, char2idx = build_vocab(directory +
".context", directory + ".question",
"word_vocab.pkl", "word2idx.pkl", "char_vocab.pkl",
"char2idx.pkl", is_train=is_train,
max_words=config.max_words)
        # create an embedding matrix from the vocabulary with pretrained
vectors (GloVe) for words
        build_embeddings(word_vocab, embedding_path=config.glove,
output_path="word_embeddings.pkl",
                           vec_size=config.word_embedding_size)
        build_embeddings(char_vocab, embedding_path="",
output_path="char_embeddings.pkl",
                           vec_size=config.char_embedding_size)

    else:
        labels = np.array([l.strip("\n") for l in labels])

```

```

        with open(os.path.join(self.data_dir, "train", "word2idx.pkl"),
"rb") as wi,\n            open(os.path.join(self.data_dir, "train", "char2idx.pkl"),
"rb") as ci:\n                word2idx = pickle.load(wi, allow_pickle=True)\n                char2idx = pickle.load(ci, allow_pickle=True)\n\n                print("Number of questions before filtering:", len(question))\n                filter = [len(c) < max_len_context and max([len(w) for w in c]) <\nmax_len_word and\n                           len(q) < max_len_question and max([len(w) for w in q]) <\nmax_len_word and\n                           len(q) > 3 for c, q in zip(context, question)]\n                context, question, labels = zip(*[(c, q, l) for c, q, l, f in zip(\ncontext, question, labels, filter) if\nf])\n                print("Number of questions after filtering ", len(question))\n\n# replace the tokenized words with their associated ID in the vocabulary\ncontext_idxs = []\ncontext_char_idxs = []\nquestion_idxs = []\nquestion_char_idxs = []\nfor i, (c, q) in tqdm.tqdm(enumerate(zip(context, question))):\n    # create empty numpy arrays\n    context_idx = np.zeros([max_len_context], dtype=np.int32)\n    question_idx = np.zeros([max_len_question], dtype=np.int32)\n    context_char_idx = np.zeros([max_len_context, max_len_word],\ndtype=np.int32)\n                           question_char_idx = np.zeros([max_len_question, max_len_word],\ndtype=np.int32)\n\n    # replace 0 values with word and char IDs\n    for j, word in enumerate(c):\n        if word in word2idx:\n            context_idx[j] = word2idx[word]\n        else:\n            context_idx[j] = 1\n        for k, char in enumerate(word):\n            if char in char2idx:\n                context_char_idx[j, k] = char2idx[char]\n            else:\n                context_char_idx[j, k] = 1\n    context_idxs.append(context_idx)\n    context_char_idxs.append(context_char_idx)\n\n    for j, word in enumerate(q):\n        if word in word2idx:\n            question_idx[j] = word2idx[word]\n        else:\n            question_idx[j] = 1\n        for k, char in enumerate(word):\n            if char in char2idx:\n                question_char_idx[j, k] = char2idx[char]\n            else:\n                question_char_idx[j, k] = 1\n    question_idxs.append(question_idx)\n    question_char_idxs.append(question_char_idx)\n\n# save features as numpy arrays\nnp.savez(os.path.join(self.data_dir, directory, directory +\n"_features"),\n         context_idxs=np.array(context_idxs),\n         context_char_idxs=np.array(context_char_idxs),\n         question_idxs=np.array(question_idxs),\n         question_char_idxs=np.array(question_char_idxs),\n
```

```

label=np.array(labels))

if __name__ == "__main__":
    train_filename = "train-v2.0.json"
    dev_filename = "dev-v2.0.json"

    maybe_download_squad(url, train_filename, config.data_dir)
    maybe_download_squad(url, dev_filename, config.data_dir)

    p = SquadPreprocessor(config.data_dir, train_filename, dev_filename,
                          tokenizer)
    p.preprocess()

    p.extract_features(max_len_context=config.max_len_context,
max_len_question=config.max_len_question,
                    max_len_word=config.max_len_word, is_train=True)
    p.extract_features(max_len_context=config.max_len_context,
max_len_question=config.max_len_question,
                    max_len_word=config.max_len_word, is_train=False)

```

### *Приклад виконання коду:*

```

MacBook-Pro-Elizabeth:Question_Answ_Syst elizabethlorelei$ python make_dataset.py
File downloaded successfully!
File downloaded successfully!
100%|██████████| 442/442 [01:40<00:00,  4.74it/s]
100%|██████████| 35/35 [00:11<00:00,  2.35it/s]
Vocabulary created successfully.
Number of questions before filtering: 86821
Number of questions after filtering  86535
86535it [00:22, 3898.59it/s]
Number of questions before filtering: 5928
Number of questions after filtering  5848
5848it [00:01, 3527.29it/s]
MacBook-Pro-Elizabeth:Question_Answ_Syst elizabethlorelei$ █

```

## ДОДАТОК Е

*Тренування моделі:*

**train.py**

```
# external libraries
import numpy as np
import pickle
import os
import json
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from tensorboardX import SummaryWriter

# internal utilities
import config
from model import BiDAF
from data_loader import SquadDataset
from utils import save_checkpoint, compute_batch_metrics

# preprocessing values used for training
prepro_params = {
    "max_words": config.max_words,
    "word_embedding_size": config.word_embedding_size,
    "char_embedding_size": config.char_embedding_size,
    "max_len_context": config.max_len_context,
    "max_len_question": config.max_len_question,
    "max_len_word": config.max_len_word
}

# hyper-parameters setup
hyper_params = {
    "num_epochs": config.num_epochs,
    "batch_size": config.batch_size,
    "learning_rate": config.learning_rate,
    "hidden_size": config.hidden_size,
    "char_channel_width": config.char_channel_width,
    "char_channel_size": config.char_channel_size,
    "drop_prob": config.drop_prob,
    "cuda": config.cuda,
    "pretrained": config.pretrained
}

experiment_params = {"preprocessing": prepro_params, "model": hyper_params}

# train on GPU if CUDA variable is set to True (a GPU with CUDA is needed to do so)
device = torch.device("cuda" if hyper_params["cuda"] else "cpu")
torch.manual_seed(42)

# define a path to save experiment logs
experiment_path = "output/{}".format(config.exp)
if not os.path.exists(experiment_path):
    os.mkdir(experiment_path)

# save the preprocessing and model parameters used for this training experiment
with open(os.path.join(experiment_path, "config_{}.json".format(config.exp)), "w") as f:
```

```

    json.dump(experiment_params, f)

# start TensorboardX writer
writer = SummaryWriter(experiment_path)

# open features file and store them in individual variables (train + dev)
train_features = np.load(os.path.join(config.train_dir, "train_features.npz"),
allow_pickle=True)
t_w_context, t_c_context, t_w_question, t_c_question, t_labels =
train_features["context_idxs"], \
train_features["context_char_idxs"], \
train_features["question_idxs"], \
train_features["question_char_idxs"], \
train_features["label"]

dev_features = np.load(os.path.join(config.dev_dir, "dev_features.npz"),
allow_pickle=True)
d_w_context, d_c_context, d_w_question, d_c_question, d_labels =
dev_features["context_idxs"], \
dev_features["context_char_idxs"], \
dev_features["question_idxs"], \
dev_features["question_char_idxs"], \
dev_features["label"]

# load the embedding matrix created for our word vocabulary
with open(os.path.join(config.train_dir, "word_embeddings.pkl"), "rb") as e:
    word_embedding_matrix = pickle.load(e)
with open(os.path.join(config.train_dir, "char_embeddings.pkl"), "rb") as e:
    char_embedding_matrix = pickle.load(e)

# load mapping between words and idxs
with open(os.path.join(config.train_dir, "word2idx.pkl"), "rb") as f:
    word2idx = pickle.load(f)

idx2word = dict([(y, x) for x, y in word2idx.items()])

# transform them into Tensors
word_embedding_matrix =
torch.from_numpy(np.array(word_embedding_matrix)).type(torch.float32)
char_embedding_matrix =
torch.from_numpy(np.array(char_embedding_matrix)).type(torch.float32)

# load datasets
train_dataset = SquadDataset(t_w_context, t_c_context, t_w_question,
t_c_question, t_labels)
valid_dataset = SquadDataset(d_w_context, d_c_context, d_w_question,
d_c_question, d_labels)

# load data generators
train_dataloader = DataLoader(train_dataset,
                             shuffle=True,
                             batch_size=hyper_params["batch_size"],
                             num_workers=4)

valid_dataloader = DataLoader(valid_dataset,
                             shuffle=True,
                             batch_size=hyper_params["batch_size"],
                             num_workers=4)

```

```

print("Length of training data loader is:", len(train_dataloader))
print("Length of valid data loader is:", len(valid_dataloader))

# load the model
model = BiDAF(word_vectors=word_embedding_matrix,
               char_vectors=char_embedding_matrix,
               hidden_size=hyper_params["hidden_size"],
               drop_prob=hyper_params["drop_prob"])
if hyper_params["pretrained"]:
    model.load_state_dict(torch.load(os.path.join(experiment_path, "model.pkl")))
    ["state_dict"])
model.to(device)

# define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adadelta(model.parameters(),
                                 hyper_params["learning_rate"], weight_decay=1e-4)

# best loss so far
if hyper_params["pretrained"]:
    best_valid_loss = torch.load(os.path.join(experiment_path, "model.pkl"))
    ["best_valid_loss"]
    epoch_checkpoint = torch.load(os.path.join(experiment_path,
                                                "model_last_checkpoint.pkl"))["epoch"]
    print("Best validation loss obtained after {} epochs is:
{}".format(epoch_checkpoint, best_valid_loss))
else:
    best_valid_loss = 100
    epoch_checkpoint = 0

# train the Model
print("Starting training...")
for epoch in range(hyper_params["num_epochs"]):
    print("##### epoch {:2d}".format(epoch + 1))
    model.train()
    train_losses = 0
    for i, batch in enumerate(train_dataloader):
        w_context, c_context, w_question, c_question, label1, label2 =
batch[0].long().to(device), \
batch[1].long().to(device), \
batch[2].long().to(device), \
batch[3].long().to(device), \
batch[4] \
[:, 0].long().to(device), \
batch[4]
        [:, 1].long().to(device)
        optimizer.zero_grad()
        pred1, pred2 = model(w_context, c_context, w_question, c_question)
        loss = criterion(pred1, label1) + criterion(pred2, label2)
        train_losses += loss.item()

        loss.backward()
        optimizer.step()

    writer.add_scalars("train", {"loss": np.round(train_losses /
len(train_dataloader), 2),
                             "epoch": epoch + 1})
    print("Train loss of the model at epoch {} is: {}".format(epoch + 1,
np.round(train_losses /
len(train_dataloader), 2)))

```

```

model.eval()
valid_losses = 0
valid_em = 0
valid_f1 = 0
n_samples = 0
with torch.no_grad():
    for i, batch in enumerate(valid_dataloader):
        w_context, c_context, w_question, c_question, labels =
batch[0].long().to(device), \
batch[1].long().to(device), \
batch[2].long().to(device), \
batch[3].long().to(device), \
                                batch[4]

            first_labels = torch.tensor([[int(a) for a in l.split("|")]
[0].split(" ")])
                                         for l in labels],
dtype=torch.int64).to(device)
        pred1, pred2 = model(w_context, c_context, w_question, c_question)
        loss = criterion(pred1, first_labels[:, 0]) + criterion(pred2,
first_labels[:, 1])
        valid_losses += loss.item()
        em, f1 = compute_batch_metrics(w_context, idx2word, pred1, pred2,
labels)
        valid_em += em
        valid_f1 += f1
        n_samples += w_context.size(0)

        writer.add_scalars("valid", {"loss": np.round(valid_losses /
len(valid_dataloader), 2),
                                         "EM": np.round(valid_em / n_samples, 2),
                                         "F1": np.round(valid_f1 / n_samples, 2),
                                         "epoch": epoch + 1})
        print("Valid loss of the model at epoch {} is: {}".format(epoch + 1,
np.round(valid_losses /
len(valid_dataloader), 2)))
        print("Valid EM of the model at epoch {} is: {}".format(epoch + 1,
np.round(valid_em / n_samples, 2)))
        print("Valid F1 of the model at epoch {} is: {}".format(epoch + 1,
np.round(valid_f1 / n_samples, 2)))

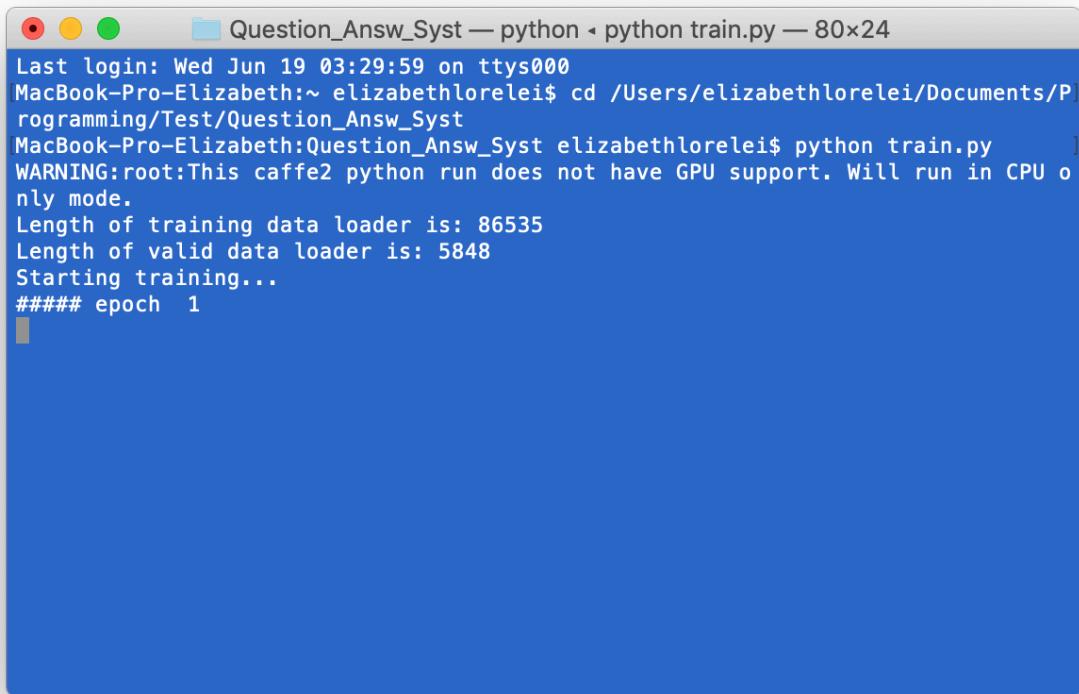
# save last model weights
save_checkpoint({
    "epoch": epoch + 1 + epoch_checkpoint,
    "state_dict": model.state_dict(),
    "best_valid_loss": np.round(valid_losses / len(valid_dataloader), 2)
}, True, os.path.join(experiment_path, "model_last_checkpoint.pkl"))

# save model with best validation error
is_best = bool(np.round(valid_losses / len(valid_dataloader), 2) <
best_valid_loss)
    best_valid_loss = min(np.round(valid_losses / len(valid_dataloader), 2),
best_valid_loss)
    save_checkpoint({
        "epoch": epoch + 1 + epoch_checkpoint,
        "state_dict": model.state_dict(),
        "best_valid_loss": best_valid_loss
    }, is_best, os.path.join(experiment_path, "model.pkl"))

# export scalar data to JSON for external processing
writer.export_scalars_to_json(os.path.join(experiment_path, "all_scalars.json"))
writer.close()

```

**Приклад виконання коду:**



A screenshot of a macOS terminal window titled "Question\_Answ\_Syst — python - python train.py — 80x24". The window shows the following command-line session:

```
Last login: Wed Jun 19 03:29:59 on ttys000
[MacBook-Pro-Elizabeth:~ elizabethlorelei$ cd /Users/elizabethlorelei/Documents/P
rogramming/Test/Question_Answ_Syst
[MacBook-Pro-Elizabeth:Question_Answ_Syst elizabethlorelei$ python train.py
WARNING:root:This caffe2 python run does not have GPU support. Will run in CPU o
nly mode.
Length of training data loader is: 86535
Length of valid data loader is: 5848
Starting training...
##### epoch 1
```

## ДОДАТОК Е

*Тестування:*

**test.py**

```
# external libraries
import numpy as np
import pickle
import os
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from tensorboardX import SummaryWriter

# internal utilities
import config
from model import BiDAF
from data_loader import SquadDataset
from utils import compute_batch_metrics

# preprocessing values used for training
prepro_params = {
    "max_words": config.max_words,
    "word_embedding_size": config.word_embedding_size,
    "char_embedding_size": config.char_embedding_size,
    "max_len_context": config.max_len_context,
    "max_len_question": config.max_len_question,
    "max_len_word": config.max_len_word
}

# hyper-parameters setup
hyper_params = {
    "num_epochs": config.num_epochs,
    "batch_size": config.batch_size,
    "learning_rate": config.learning_rate,
    "hidden_size": config.hidden_size,
    "char_channel_width": config.char_channel_width,
    "char_channel_size": config.char_channel_size,
    "drop_prob": config.drop_prob,
    "cuda": config.cuda,
    "pretrained": config.pretrained
}

experiment_params = {"preprocessing": prepro_params, "model": hyper_params}

# train on GPU if CUDA variable is set to True (a GPU with CUDA is needed to do so)
device = torch.device("cuda" if hyper_params["cuda"] else "cpu")
torch.manual_seed(42)

# define a path to save experiment logs
experiment_path = "output/{}".format(config.exp)
if not os.path.exists(experiment_path):
    os.mkdir(experiment_path)

# start TensorboardX writer
writer = SummaryWriter(experiment_path)

# open features file and store them in individual variables
```

```

dev_features = np.load(os.path.join(config.dev_dir, "dev_features.npz"),
allow_pickle=True)
d_w_context, d_c_context, d_w_question, d_c_question, d_labels =
dev_features["context_idxs"], \
dev_features["context_char_idxs"], \
dev_features["question_idxs"], \
dev_features["question_char_idxs"], \
dev_features["label"]

# load word2idx and idx2word dictionaries
with open(os.path.join(config.train_dir, "word2idx.pkl"), "rb") as f:
    word2idx = pickle.load(f)

idx2word = dict([(y, x) for x, y in word2idx.items()])

# load the embedding matrix created for our word vocabulary
with open(os.path.join(config.train_dir, "word_embeddings.pkl"), "rb") as e:
    word_embedding_matrix = pickle.load(e)
with open(os.path.join(config.train_dir, "char_embeddings.pkl"), "rb") as e:
    char_embedding_matrix = pickle.load(e)

# transform them into Tensors
word_embedding_matrix =
torch.from_numpy(np.array(word_embedding_matrix)).type(torch.float32)
char_embedding_matrix =
torch.from_numpy(np.array(char_embedding_matrix)).type(torch.float32)

# load dataset
test_dataset = SquadDataset(d_w_context, d_c_context, d_w_question,
d_c_question, d_labels)

# load data generator
test_dataloader = DataLoader(test_dataset,
                            shuffle=True,
                            batch_size=hyper_params["batch_size"],
                            num_workers=4)

print("Length of test data loader is:", len(test_dataloader))

# load the model
model = BiDAF(word_vectors=word_embedding_matrix,
               char_vectors=char_embedding_matrix,
               hidden_size=hyper_params["hidden_size"],
               drop_prob=hyper_params["drop_prob"])

try:
    if config.cuda:
        model.load_state_dict(torch.load(os.path.join(config.squad_models,
"model_final.pkl")["state_dict"]))
    else:
        model.load_state_dict(torch.load(os.path.join(config.squad_models,
"model_final.pkl"),
                                         map_location=lambda storage, loc:
storage)["state_dict"])
    print("Model weights successfully loaded.")
except:
    print("Model weights not found, initialized model with random weights.")
model.to(device)

# define loss criterion
criterion = nn.CrossEntropyLoss()

model.eval()

```

```

test_em = 0
test_f1 = 0
n_samples = 0
with torch.no_grad():
    for i, batch in enumerate(test_dataloader):
        w_context, c_context, w_question, c_question, labels =
batch[0].long().to(device), \
batch[1].long().to(device), \
batch[2].long().to(device), \
batch[3].long().to(device), \
                                batch[4]
        pred1, pred2 = model(w_context, c_context, w_question, c_question)
        em, f1 = compute_batch_metrics(w_context, idx2word, pred1, pred2,
labels)
        test_em += em
        test_f1 += f1
        n_samples += w_context.size(0)

writer.add_scalars("test", {"EM": np.round(test_em / n_samples, 2),
                           "F1": np.round(test_f1 / n_samples, 2)})
print("Test EM of the model after training is: {}".format(np.round(test_em /
n_samples, 2)))
print("Test F1 of the model after training is: {}".format(np.round(test_f1 /
n_samples, 2)))

```

## ДОДАТОК Ж

*Оцінює модель на новій парі (контекст, питання):*

### eval.py

```
import os
import pickle
import numpy as np
import torch

import config
from model import BiDAF
from utils import clean_text, word_tokenize, discretize

device = torch.device("cuda" if config.cuda else "cpu")

def eval(context, question):
    with open(os.path.join(config.data_dir, "train", "word2idx.pkl"), "rb") as wi, \
        open(os.path.join(config.data_dir, "train", "char2idx.pkl"), "rb") as ci, \
        open(os.path.join(config.data_dir, "train", "word_embeddings.pkl"), "rb") as wb, \
        open(os.path.join(config.data_dir, "train", "char_embeddings.pkl"), "rb") as cb:
        word2idx = pickle.load(wi)
        char2idx = pickle.load(ci)
        word_embedding_matrix = pickle.load(wb)
        char_embedding_matrix = pickle.load(cb)

    # transform them into Tensors
    word_embedding_matrix =
        torch.from_numpy(np.array(word_embedding_matrix)).type(torch.float32)
    char_embedding_matrix =
        torch.from_numpy(np.array(char_embedding_matrix)).type(torch.float32)
    idx2word = dict([(y, x) for x, y in word2idx.items()])

    context = clean_text(context)
    context = [w for w in word_tokenize(context) if w]

    question = clean_text(question)
    question = [w for w in word_tokenize(question) if w]

    if len(context) > config.max_len_context:
        print("The context is too long. Maximum accepted length is",
config.max_len_context, "words.")
    if max([len(w) for w in context]) > config.max_len_word:
        print("Some words in the context are longer than", config.max_len_word,
"characters.")
    if len(question) > config.max_len_question:
        print("The question is too long. Maximum accepted length is",
config.max_len_question, "words.")
    if max([len(w) for w in question]) > config.max_len_word:
        print("Some words in the question are longer than", config.max_len_word,
"characters.")
    if len(question) < 3:
        print("The question is too short. It needs to be at least a three words
question.")
```

```

context_idx = np.zeros([config.max_len_context], dtype=np.int32)
question_idx = np.zeros([config.max_len_question], dtype=np.int32)
context_char_idx = np.zeros([config.max_len_context, config.max_len_word],
                           dtype=np.int32)
question_char_idx = np.zeros([config.max_len_question, config.max_len_word],
                           dtype=np.int32)

# replace 0 values with word and char IDs
for j, word in enumerate(context):
    if word in word2idx:
        context_idx[j] = word2idx[word]
    else:
        context_idx[j] = 1
    for k, char in enumerate(word):
        if char in char2idx:
            context_char_idx[j, k] = char2idx[char]
        else:
            context_char_idx[j, k] = 1

for j, word in enumerate(question):
    if word in word2idx:
        question_idx[j] = word2idx[word]
    else:
        question_idx[j] = 1
    for k, char in enumerate(word):
        if char in char2idx:
            question_char_idx[j, k] = char2idx[char]
        else:
            question_char_idx[j, k] = 1

model = BiDAF(word_vectors=word_embedding_matrix,
               char_vectors=char_embedding_matrix,
               hidden_size=config.hidden_size,
               drop_prob=config.drop_prob)

try:
    if config.cuda:
        model.load_state_dict(torch.load(os.path.join(config.squad_models,
"model_final.pkl"))["state_dict"])
    else:
        model.load_state_dict(torch.load(os.path.join(config.squad_models,
"model_final.pkl"),
                                         map_location=lambda storage, loc:
storage)["state_dict"])
    print("Model weights successfully loaded.")
except:
    pass
    print("Model weights not found, initialized model with random weights.")
model.to(device)
model.eval()
with torch.no_grad():
    context_idx, context_char_idx, question_idx, question_char_idx =
torch.tensor(context_idx, dtype=torch.int64).unsqueeze(0).to(device), \
torch.tensor(context_char_idx, dtype=torch.int64).unsqueeze(0).to(device), \
torch.tensor(question_idx, dtype=torch.int64).unsqueeze(0).to(device), \
torch.tensor(question_char_idx, dtype=torch.int64).unsqueeze(0).to(device)

    pred1, pred2 = model(context_idx, context_char_idx, question_idx,
question_char_idx)
    starts, ends = discretize(pred1.exp(), pred2.exp(), 15, False)
    prediction = " ".join(context[starts.item(): ends.item() + 1])

return prediction

```

```

if __name__ == "__main__":
    context = " Our narrator, Nick Carraway, moves to the East Coast to work as
    a bond trader in Manhattan. He rents a small house in West Egg, a nouveau riche
    town in Long Island. " \
        "In East Egg, the next town over, where old money people live,
    Nick reconnects with his cousin Daisy Buchanan, her husband Tom, and meets their
    friend Jordan Baker." \
        "Tom takes Nick to meet his mistress, Myrtle Wilson. Myrtle is
    married to George Wilson, who runs a gas station in a gross and dirty
    neighborhood in Queens. Tom, Nick, and Myrtle go to Manhattan, where she hosts a
    small party that ends with Tom punching her in the face." \
        "Nick meets his next-door neighbor, Jay Gatsby, a very rich man
    who lives in a giant mansion and throws wildly extravagant parties every
    weekend, and who is a mysterious person no one knows much about." \
        "Gatsby takes Nick to lunch and introduces him to his business
    partner - a gangster named Meyer Wolfshiem." \
        "Nick starts a relationship with Jordan. Through her, Nick finds
    out that Gatsby and Daisy were in love five years ago, and that Gatsby would
    like to see her again." \
        "Nick arranges for Daisy to come over to his house so that Gatsby
    can "accidentally" drop by. Daisy and Gatsby start having an affair." \
        "Tom and Daisy come to one of Gatsby's parties. Daisy is disgusted
    by the ostentatiously vulgar display of wealth, and Tom immediately sees that
    Gatsby's money most likely comes from crime." \
        "We learn that Gatsby was born into a poor farming family as James
    Gatz. He has always been extremely ambitious, creating the Jay Gatsby persona as
    a way of transforming himself into a successful self-made man - the ideal of the
    American Dream."

```

questions = ["Who is the narrator?",  
 "Where Gatsby takes Nick?",  
 "With whom Nick starts relationship?",  
 "Where Gatsby was born?"]

```

print("C:", context, "\n")
for q in questions:
    print("Q:", q)
    answer = eval(context, q)
    print("A:", answer, "\n")

```