МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
Кафедра обчислювальної математики

Курсова робота

Підсумовування текстів за допомогою глибоких нейронних мереж

Виконав: студент 4-го курсу групи ПМп-41
спеціальності
113 - ”Прикладна математика”
Борух М.І.

Керівник
доцент Музичук Ю.А.

Національна шкала_____
Кількість балів:_____ Оцінка: ECTS_____

Члени комісії: _____ _____

_____ _____

_____ _____

Львів - 2021

Зміст

Вступ

У сьогоднішній час, у відкритому доступі, є безліч статей різного вмісту. Здебільшого це середні або великі за обсягом роботи. Не завжди є можливість швидко дізнатися суть написаного за браком часу або бажання. Та все ж хотілося б бути в курсі опублікованого матеріалу. Можливість переглядати скорочений або підсумований текст була б чудовою, це допомогло б зменшити час на ознайомлення з новими матеріалами, також оптимізувати роботу пошукових сайтів, даючи можливість видавати більш точні результати. Все це має вплив на швидкість опрацювання інформації, можливість відділяти потрібне від другорядного. Враховуючи як стрімко наповнюється мережа новою інформацією, інструмент стискання тесту вже не є беззмістовною іграшкою.

На сьогодні, нейронні мережі активно використовуюся у розв’язання таких задач. У даній роботі буде побудовано модель, яка вміє підсумовувати тест, а також розглянуто кроки для успішної реалізації задуманого.

1 Постановка задачі

Задачу підсумовування тексту можна описати так:

- x_i - вхідні дані (речення), $i = 1, ..., n$, в процесі будуть перетворені у вектори розміру m, де число m буде залежати від довжини вхідного речення. В результаті отримаємо матрицю $X^{n \times m}$

- y_i - вхідні дані (речення), $i = 1, ..., n$, в процесі перетворюються у вектори розміру k. Число k – довжина вихідного речення. Результат – матриця $Y^{n \times k}$

Готова модель приймає текст, і повертає коротший текст, який є подібним або однаковим за ідеєю до попереднього.

Мета даної роботи дослідити та побудувати модель, що вміє підсумовувати тексти. Можна виділити такі етапи роботи:

1. Аналіз та обробка вхідних даних
2. Реалізація глибокої нейронної мережі
3. Побудова сумаризатора на основі даної мережі
4. Оцінка отриманих результатів

2 Вхідні дані та їх обробка

Важливим етапом у побудуванні будь-якої нейронної мережі є підготовка вхідних даних. Оскільки майбутня мережа працюватиме з текстом проведемо такі основні етапи:

2.1 Обробка тексту

1. Токенізація або лексичний аналіз
2. Нормалізація
3. Видалення ’шуму’

1. Розбиття тексту на токени. Токеном можуть виступати як речення, так і слова. Використовуватимемо слова. Тобто після застосування даного пункту вхідний текст буде розбитий на окремі слова.

2. Приведення тексту до загального шаблону. Слова в реченні мають різну форму: множина, однина, рід, закінчення слова в залежності від контексту та інші особливості вибраної мови. Використовуючи такий текст не можна, тому все зайве треба видалити, а слова звести до їх канонічної форми. Звести до канонічної форми можна декількома способами:

- Stemming - відсічення закінчення
- Lemmatization - знаходження канонічної форми
- Lemmatization з POS - канонічна форма слова, в залежності яким членом речення виступає слово

Найкращим варіантом є Lemmatization з POS, але така обробка вимагає багато часу, тому Stemming теж хороший варіант, оскільки потребує менше часу. Хоча для нас слова після відсічення закінчення здаватимуться незрозумілими, та для мережі це буде нормально.

3. Все що не увійшло у пункт ”нормалізація”може бути оброблено тут. В саме html теги, розмітка, метадані.

2.2 Відбір ознак

У попередній роботі розглядалися такі методи відбору ознак як: Count Vectorizer, TF-IDF. Цього разу використаємо звичайний one-hot encoding, де кожному слову відповідає певний номер. Припустимо, що маємо словник з двох слів, тоді:
Вхід:[”Hello world”]

Вихід:[1 2]

Вхід:[”Hello”]
Вихід:[1]

Однак для роботи нейронної мережі, не підходить використання різних довжин векторів, тому треба звести вектор до єдиної довжини. Реалізувати це можна просто додавши 0-і до кінця тексту. Щоб нейронна мережа не вивчала ніякої інформації з таких закінчень, можна створити маску, де нулі не враховуватимуться при тренуванні мережі. Тоді:
Вхід:[”Hello”]
Вихід:[1, 0]

Наступне, що треба зробити – це перетворити такі вектори так, щоб однакові за змістом слова мали однакове представлення. Такий процес називається ”learn words embeddings”. Є такі підходи реалізації даного процесу:

The Continuous Bag of Words (CBOW) Model - неперервний мішок слів. Ідея полягає в тому, що модель намагається передбачити поточне слово враховуючи слова, які оточують дане слово. Формуються пари слів такого вигляду (слова, які оточують слово та слово, яке хочемо передбачити). Візьмемо таке речення: ”he quick brown fox jumps over the lazy dog”. Взявши розмір наволишніх слів 2 - отримаємо такі пари: ([quick, fox], brown), ([the, brown], quick), ([the, dog], lazy). Далі ці пари тренуються, щоб передбачати центральне слово.

Continuous Skip-Gram Model. Даний підхід працює у навпаки. Моделі передається центральне слово, а вона передбачає сусідні слова.

Особливості Skip-Gram:

- Добре працює на малих даних
- Краще визначає рідковживані слова

Особливості CBOW:

- Добре працює з великими даними
- Краще визначає частовживані слова

Існують вже пре натреновані ”word embeddings”. Популярними є Google Word2Vec, Stanford GloVe Embeddings[1]. Використання вже пре натренованих ”embeddings”є дуже помічним, коли датасет є невеликого розміру. На датасетах великого розміру відмінність пре натренаваних і власноруч тренованих ”emdebbinds”є невеликою.

3 Глибокі нейронні мережі

Завдання роботи є побудова глибокої нейронної мережі, яка б мала змогу підсумовувати тексти. Очевидно, що для такого типу завдання будь-яка мережа не підійде. Треба використати таку, що вміє працювати з послідовними даними. Текст можна розглядати як послідовність речень, а їх як послідовність слів. Основна ідея полягає в тому, що на вхід мережі послідовно даються слова, і після кожного слова мережа запам'ятовує вихід, який передається далі, де комбінується з новим словом. Так, в кінці отримуємо вихід і стан, які будуть використовується у передбаченні наступного слова вже для підсумовування тексту. Отже, щоб зробити модель "речення до речення"потрібен механізм запам'ятовування попередніх виходів, станів. Такою мережею є рекурентна нейронна мережа так, як кожен раз вона передає сама в себе попередні стани слів. Так можна зобразити РНМ.

3.1 Рекурентна нейронна мережа

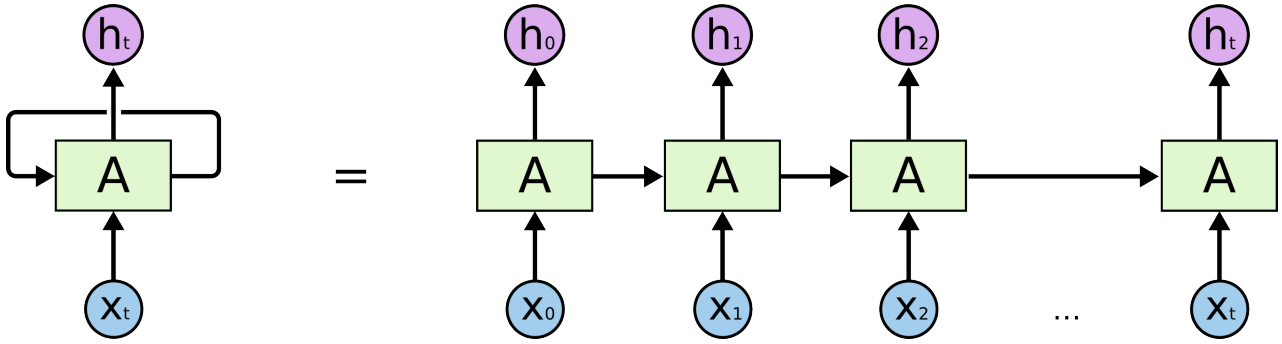

Рис. 1: Розгорнута РНМ [2]

Рекурентна нейронна мережа всередині працює наступним чином:

$$h_t = f_W\left(h_{t-1}, x_t\right)$$

На вхід подаємо x_t – вектор зі слів, h_{t-1} – стан з попереднього кроку, при $t = 0$ h_{-1} – вектор нулів, f_W – деяка активаційна функція, найчастіше сигмоїда.

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right)$$

$$y_t = W_{hy}h_t$$

W_{hh}, W_{xh}, W_{hy} – матриці ваг.

Однак, у РНМ є вагомий недолік, відомий як зникаючий градієнт. Мережа при зворотному ході оновлює параметри використовуючи алгоритм градієнтного спуску і стається так, що градієнт зменшується поки не стає константою. У

такому разі модель не має можливості покращуватися і як наслідок нічого не навчається. Отримати хороший результат у такому випадку не можливо. Таке часто трапляється, коли мережа має вивчити довготривалі залежності між словами. Вирішити дану проблему можна шляхом наперед визначення ваг, але не завжди це дає бажаний результат. Тому доцільно розглянути інший тип мережі.

3.2 Long Short-Term Memory

LSTM – довго короткотривала пам'ять. Така мережа здатна запам'ятовувати інформацію на довгий проміжок часу. Дана мережа складається LSTM клітин, які в свою чергу з вхідних, вихідних воріт, а також воріт забуття.
Ворота забуття або forget gate використовуються для визначення інформації, яку потрібно забути або зберегти.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Цей шар на вхід отримує попередній стан h_{t-1} і якусь інформацію в певний час t, у цьому випадку – це слово. W_f, b_f – матриця ваг та вектор похибки відповідно. Вхідна інформація множиться на матрицю ваг і до цього додається похибка. Після цього до отриманого результату застосовується функція σ - сигмоїда.

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

Результат після сигмоїди буде від 0 до 1, де 0 – повністю забути, 1 – навпаки, запам'ятати.
Наступні – вхідні ворота (input gate) існують для визначення, якої нової інформації бракує в LSTM клітині.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$

У цьому шарів, як і в попередньому, функція сигмоїда σ слугує для визначення які значення треба оновити. Тобто, щось треба точно забути, зберегти або просто зменшити вплив даного параметра на майбутнє передбачення. Функція тангенс гіперболічний tanh використовується для створення нового кандидату, який повинен бути доданий до стану клітини.

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Маючи результати з попередніх двох шарів, а саме $f_t, i_t, \tilde{C}_t$, можна остаточно вирішити, яку інформацію потрібно передати у наступну LSTM клітину. Записати

це можна наступною формулою.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Попередній стан множиться на f_t – те, що варто забути, додається $i_t * \tilde{C}_t$ – новий кандидат помножений на оновленні значення попереднього. Це і буде остаточний вихідний C_t стан LSTM клітини. Інший вихід буде обчислюватися в такому шарі.

Вихідні ворота – використовуються для обчислення результатів. Маємо такі дві формули.

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Використовуючи сигмоїду, буде вирішено, яка попередня інформація буде присутня у новій. Далі множенням поточного стану C_t після застосування гіперболічного тангенсу на o_t виводяться ті частини потрібної інформації, які були вирішені у попередніх шарах[2]. Подивимося на рисунок LSTM клітини.

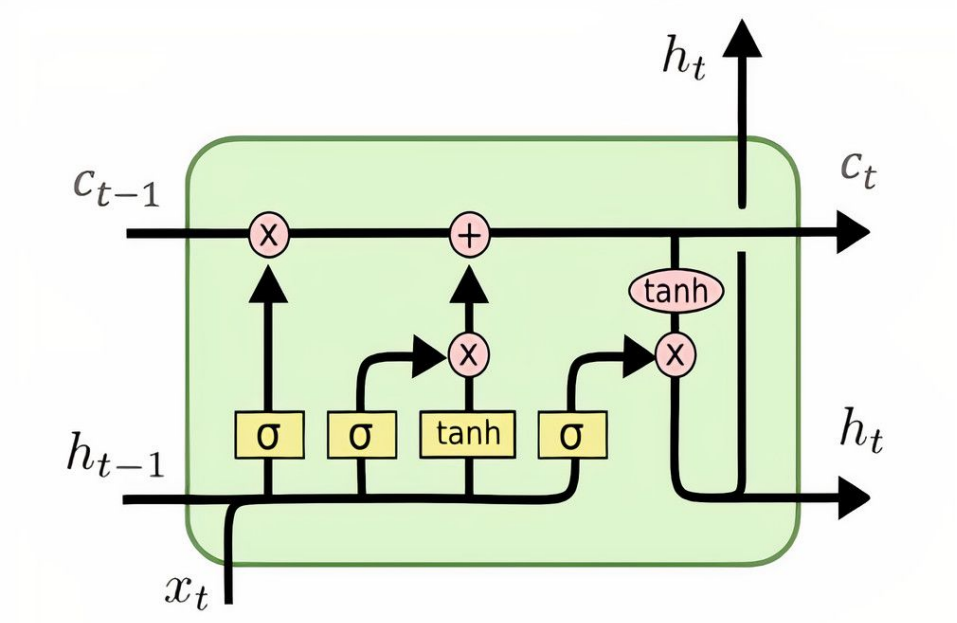

Рис. 2: LSTM клітина [2]

Також існують інші модифікація LSTM клітини. У кожної з них по-різному відбувається запам’ятовування потрібної інформації.

Для чого потрібний саме такий механізм пам’яті й чому він добре підходить для задачі підсумовування тексту? Даний механізм добре працює з часовими рядами, тобто коли потрібно прийняти рішення використовуючи вже наявну попередню інформацію. Як відомо, текст складається з набору слів. Цей набір слів є не просто випадково згенерованим. Кожне слово якось пов’язане з попереднім або з кількома попередніми словами. Також, можна сказати, що поточне слово матиме вплив на наступне. Отже, є залежність між словами. Проявляються така залежність у кожній мові по-різному. Якщо взяти українську мову, то від займенника залежатиме закінчення наступного слова. ”Дівчина розумна”, ”Хлопець розумний”, ”Діти розумні”. Друге слово у кожному прикладі має однакове значення, та різне закінчення, бо різні займенник: вона, він, вони. В англійській мові з відмінювання слів немає, але це не змінює той факт, що слова теж мають залежність один з одним. І це поєднує всі мови. Тому, щоб будувати граматично правильні речення потрібно мати механізм пам’яті. Такий тип не є надто складним, адже використовуються попереднє або два три попередні слова. З такою задачею може справитися і класична рекурентна мережа. Проте такі речення є простими, а для підсумовування тексту потрібно якомога стисліше передати зміст, тому може виникнути потреба у складніших зв’язках між словами. Для прикладу треба запам’ятати слово з речення, і те слово потрібно буде використати через декілька речень. У такому разі простої рекурентної мережі може бути не достатньо, тому доцільно застосувати більш потужний інструмент, у цьому разі це LSTM мережа.

4 Датасет

4.1 Дані

Для тренування підсумовування тексту був вибраний датасет WikiHow [3]. Розмір датасету – 230843 записи. Містить три колонки: headline – заголовок, title – назва статті, text – текст статі. Колонка title – у даній задачі непотрібна, тому використовуватися не буде. Text – містить інформацію відповідно до назви статті. Стаття поділена на пункти й кожний пункт має свій заголовок і відповідний текст до нього, колонка text. Headline – всі заголовки пунктів до однієї статті. Цю колонку можна вважати підсумованим текстом. На такий даних з колонок ”headline”, ”text”, будемо тренувати мережу. Така мережа має підсумовувати колонку ”text”, так щоб зміст результату був схожим до колонки ”headline” відповідного запису. Дані поділені на тренувальні, валідаційні та тестувальні у наступній пропорції 90/5/5.

	headline	title	text
0	\nKeep related supplies in the same area.,\nMa...	How to Be an Organized Artist1	If you're a photographer, keep all the necess...
1	\nCreate a sketch in the NeoPopRealist manner ...	How to Create a Neopoprealist Art Work	See the image for how this drawing develops s...
2	\nGet a bachelor’s degree.,\nEnroll in a studi...	How to Be a Visual Effects Artist1	It is possible to become a VFX artist without...
3	\nStart with some experience or interest in ar...	How to Become an Art Investor	The best art investors do their research on t...
4	\nKeep your reference materials, sketches, art...	How to Be an Organized Artist2	As you start planning for a project or work, ...
5	\nKeep all of your past work organized and acc...	How to Be an Organized Artist3	When you finish a project, whether it sells o...
6	\nCreate a compelling reel or portfolio.,\nLan...	How to Be a Visual Effects Artist2	This should be a short video showcasing the b...
7	\nJoin a professional society.,\nEnjoy working...	How to Be a Visual Effects Artist3	Networking is a great way to find new opportu...
8	\nMake sure you know what is expected of you.,...	How to Be Good at Improvisation	Some entire movies are improvised, some plays...
9	\nMake a list of what your friends watch, read...	How to Always Catch Pop Culture References1	Use your friends’ conversations to figure out...
10	\nPractice your material until you can perform...	How to Get a Record Deal With Phantom City Studio	;\n, Professional quality recordings of your s...
11	\nListen to radio advertisements.,\nDetermine ...	How to Find the Nearest Casino1	\n\n\nListen to local radio broadcasts for adv...
12	\nTake theatre classes.,\nVolunteer at a theat...	How to Be a Stage Manager	Theatre classes aren’t just for budding actor...
13	\nSet up a clear list of traits and characteri...	How to Find Actors	Having a vision of the appearance and abiliti...
14	\nDetermine if your minivan is currently under...	How to Find a Minivan Mechanic	Depending on the age of your van, it may stil...
15	\nDefine the workshop objective.,\nDecide who ...	How to Conduct a Workshop	Whether you are teaching a skill, delivering ...
16	\nMake a list of references you don’t understa...	How to Always Catch Pop Culture References2	If you’re new to pop culture, you’ve likely m...
17	\nWatch television advertisements.,\nDetermine...	How to Find the Nearest Casino2	\n\n\nWhile watching television, pay close att...
18	\nRead local newspapers and/or newspapers with...	How to Find the Nearest Casino3	\n\n\nPay close attention to any articles or a...
19	\nRead your local phone book.,\nDetermine the ...	How to Find the Nearest Casino4	\n\n\nCheck for a section that is titled "Casi...

Рис. 3: Датасет

4.2 Використання даних

До цих даний застосовуємо перечислені в другому пункті методи з обробки тексту. Отримуємо текст готовий до перетворення у векторизований вигляд, та спершу потрібно визначити якої довжини текст буде передано у сумаризатор, та яку довжину підсумованого тексту хочемо отримати. Для цього побудуємо графіки залежності кількості статей до їхньої довжини.

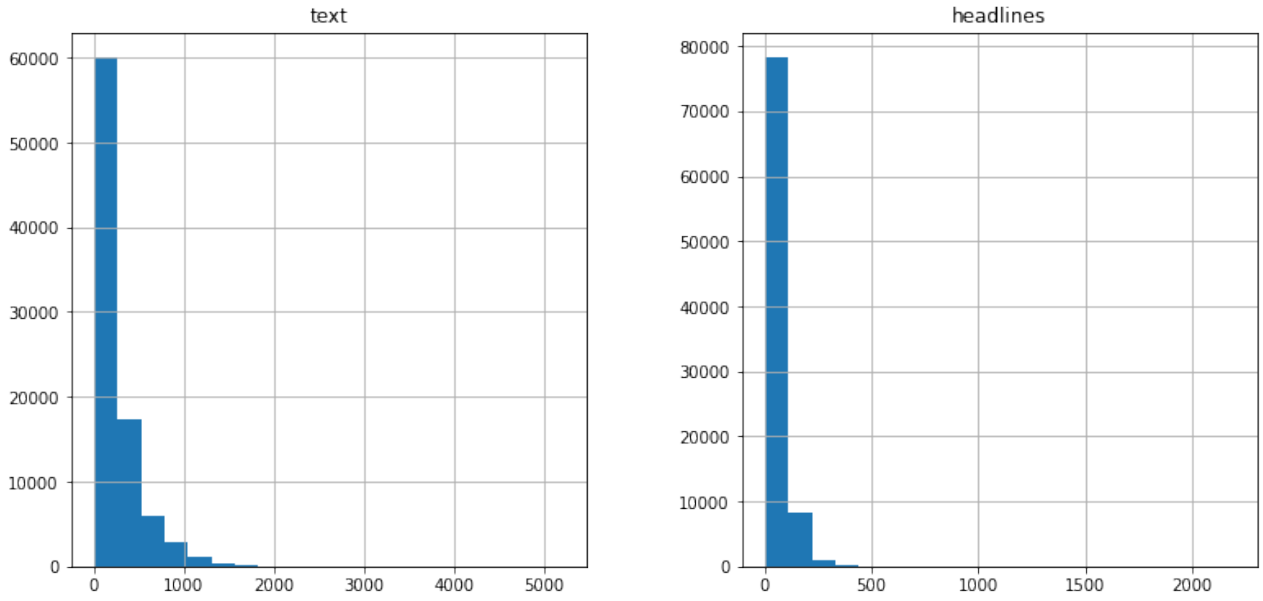

Рис. 4: Довжина статей

Для побудови рисунку з датасету було взято 90000 записів. Статті, довжина яких менша за 150, складають 42% від усіх даних. Заголовки статей, довжина яких менша за 60 – 65% від даних. Було вирішено, що модель буде тренуватися на довжині тексту в 150 слів, та підсумовуватиме текст не більше ніж у 60 слів. Брати більші довжини немає великого сенсу, оскільки на кінцевий результат це матиме незначний вплив, а на швидкість тренування моделі вливає сильно. Добавивши кілька десятків слів до вхідного тексту, час тренування може зрости на десятки хвилин. Точно сказати залежність між кількістю слів та часом тренування неможливо, через те, що на останній показник впливають і інші параметри. Якщо залишити інші параметри незмінними, а мережа підсумовуватиме текст до 60 слів і до 35 слів, то час тренування буде 123 хвилини та 86 хвилин відповідно.

5 Архітектура моделі

Задачею даної роботи є підсумовування текстів, тому потрібно побудувати модель, яка б надавала таку можливість. ”Послідовність до послідовності”(seq2seq) – підхід машинного навчання для обробки природної мови. До такого підходу відноситься підсумовування тексту. З набору слів у реченні формується нове речення, в залежності від поставленої задачі. На наступному рисунку зображений encoder-a та decoder-a. Encoder відповідає за зчитування даних та закодовування їх у внутрішнє представлення. Decoder – з закодованого представлення генерує слова. На рисунку можлива модель чатбота.

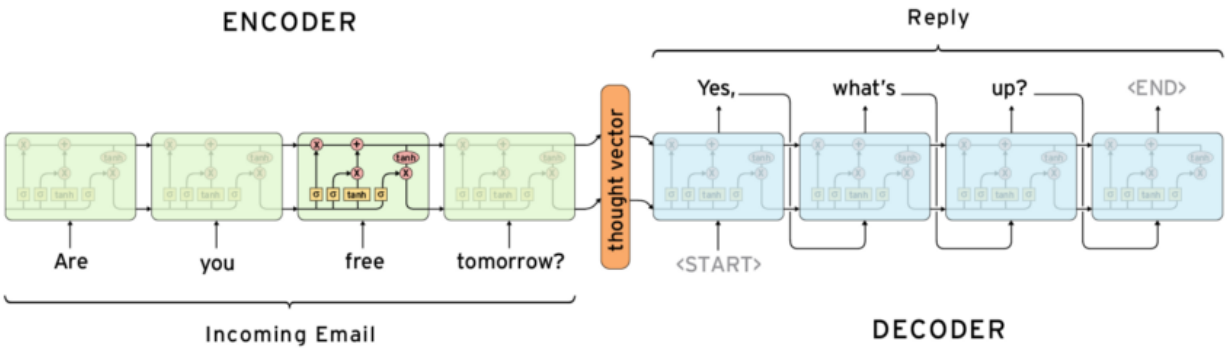

Рис. 5: seq2seq [4]

Ідея побудови моделі для підсумовування тексту дуже схожа до моделі чатбота. На вході encoder має embedding шар після якого йде прихований LSTM шар, що дає представлення для вхідного тексту як вектор сталої довжини. Decoder, також, складається з embedding вектору для останнього згенерованого слова і LSTM шару, який з вектора сталої довжини та попереднього слова генерую наступне. Розглянемо три варіанти побудови моделі.

5.1 Підсумок за раз

Така модель генерує підсумок тексту за один раз. Недоліком такої моделі є велике навантаження на декодер, оскільки він повинен вибирати слова та їх порядок [5].

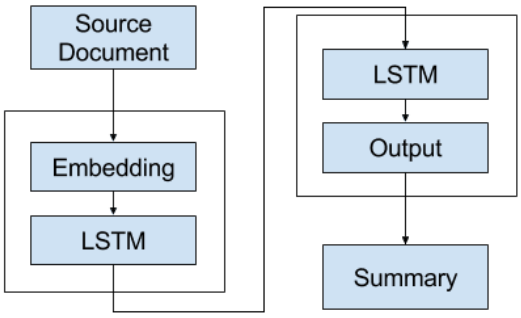

Рис. 6: Підсумок за один раз

5.2 Рекурсивна модель 1

Ця модель генерує одне слова за раз і використовує його для отримання наступного слова. Що сформувати наступне слово декодер використовує закодоване представлення вхідного тексту та всі попередньо сформовані слова. Підсумок створюється шляхом рекурсивного виклику моделі із попереднім передбаченим словом. Як початок підсумовування використовуються токени початку. Таким токеном може бути будь-яке слово, яке не використовується у словнику. Наприклад: <SOS>, <BOS>. Варіантів є багато і який з них вибрати не відіграє ніякої ролі. Також треба обрати такий самий токен, але для закінчення тексту, щоб мережа знала, що треба завершити підсумовування. Також підсумовування завершується у випадку досягнення максимальної наперед заданої довжини підсумку.

Цей метод кращий, оскільки декодер використовує раніше сформовані слова та вихідний документ як контекст для генерації наступного слова [5].

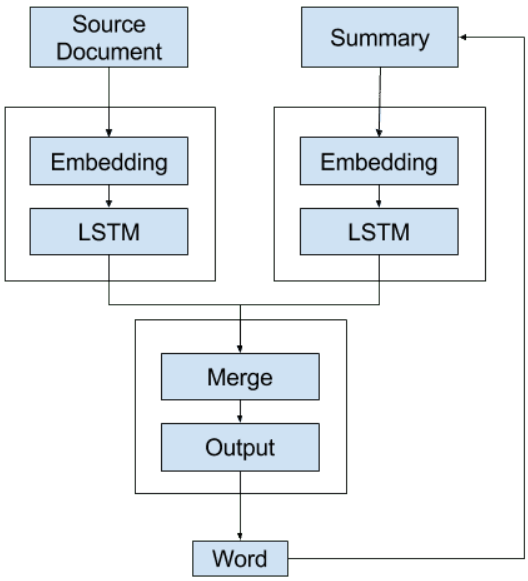

Рис. 7: Рекурсивна модель 1

5.3 Рекурсивна модель 2

Цей метод генерує закодоване представлення вихідного тексту. Далі, він подається в декодер на кожному кроці згенерованої вихідної послідовності. Це дозволяє декодеру створювати стани, який будуть використані для генерації наступних слів. Як і в попередньому методі, модель викликається до для кожного слова доки не сформує максимальну довжину або передбачить кінець [5].

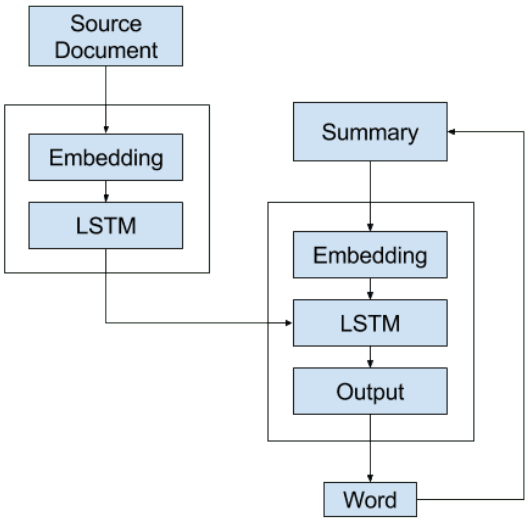

Рис. 8: Рекурсивна модель 2

6 Генерація підсумків

Для підсумовування текстів у даній роботі була вибрана друга рекурсивна модель. Після того, як модель завершила тренування, можна почати отримувати результати. Як було сказано в описі до методу, щоб зробити підсумок треба подавати по одному слову на вхід декодеру. Починається з токена початку. Далі обчислюються внутрішні стани, які будуть використанні у формуванні наступного слова та ймовірності для кожного слова зі словника. На виході застосовується активаційна функція softmax.

$$\text{softmax} = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Маючи ймовірності до кожного слова, потрібно вирішити яке слово обирати.

6.1 Жадібний алгоритм

Найпростішим алгоритмом вибрати слово є жадібний алгоритм. Ідея полягає в тому, щоб перебрати всі ймовірності й знайти слово з найбільшою ймовірністю і обрати його. Перевага такого алгоритму є те, що він доволі швидкий, але його використання не завжди дає оптимальний результат.

6.2 Променевий пошук

Інший популярний алгоритм – пошук променя. На відміну від жадібного алгоритму цей повертає список найбільш ймовірних речень. Працює він наступний чином. Задається n – ширина променя . Обирається n перший слів з ймовірностей передбачених декодером на першому кроці. Далі для кожних n слів обчислюється ймовірність другого слова враховуючи попереднє. Так формуються пари слів, які є найбільш ймовірними. Алгоритм працює поки не передбачить кінець підсумку або не досягне максимальної довжини. Якщо n – один, то тоді жадібний алгоритм. Недоліком такого способу є виродження тексту. Виродження тексту – вихідний текст незв'язний, застрягає у повторюваних циклах.

6.3 Випадковий спосіб

Дуже простий спосіб, ідея якого обрати випадкове слово. Очевидно, що на хороші результати сподіватися не варто.

6.4 Топ-k прикладів

Топ-k прикладів доволі популярний спосіб вибору наступного слова. Ідея в тому, щоб з ймовірностей передбачених мережею зрізати верхні k.

На кожному кроці, найкращі k наступних слів (токенів) обираються відносно їх ймовірностей. Тобто, дано розподіл $P\left(x \mid x_{1:i-1}\right)$, визначаємо словник з найкращих k слів $V^{(k)} \subset V$, так щоб $\sum_{x \in V^{(k)}} P\left(x \mid x_{1:i-1}\right)$. Нехай $p' = \sum_{x \in V^{(k)}} P\left(x \mid x_{1:i-1}\right)$. Розподіл масштабується за наступним правилом

$$P'\left(x \mid x_{1:i-1}\right) = \begin{cases} P\left(x \mid x_{1:i-1}\right)/p' & \text{якщо } x \in V^{(p)} \\ 0 & \text{інакше.} \end{cases}$$

Наступне слово обирається на основі даного розподілу.

Невеликим недоліком такого підходу є складні у виборі хорошого параметра k. Адже підібрати його потрібно так, щоб отримати результати кращі від застосування пошуку променя або жадібного алгоритму. Труднощі в оптимальному підборі параметру полягають у тому, що для різних контекстів є різна кількість хороших альтернатив вибору слова. У такому випадку є ризик генерувати загальний текст. Якщо слушних варіантів мало, а параметр k великий, то є ризик обрати поганий варіант слова, як наслідок – незв'язний текст. Було б добре, як би розмір словника змінювався в залежності від контексту [6]. Тому розглянемо наступний підхід.

6.5 Метод ядер

Метод ядер – стохастичний метод декодування. Ідея полягає у виборі набору токенів враховуючи розподіл ймовірностей передбачених декодером. Для розподілу $P\left(x \mid x_{1:i-1}\right)$ визначається з найкращих p слів словник $V^{(p)} \subset V$ настільки малий, щоб задовольнити умову

$$\sum_{x \in V^{(p)}} P\left(x \mid x_{1:i-1}\right) \geq p$$

Нехай $p' = \sum_{x \in V^{(p)}} P\left(x \mid x_{1:i-1}\right)$, тоді початковий розподіл масштабується до нового

$$P'\left(x \mid x_{1:i-1}\right) = \begin{cases} P\left(x \mid x_{1:i-1}\right)/p' & \text{якщо } x \in V^{(p)} \\ 0 & \text{інакше.} \end{cases}$$

Вибір слова відбувається з нового розподілу.

Обирається поріг p, так щоб сума токенів з найбільшою ймовірністю перевищувала p. Очевидно, що на кожному кроці словник буде динамічно змінюватися. Такий підхід має невелику перевагу над попереднім [6].

7 Тренування моделі

Як було згадано раніше, було обрано другу рекурсивну модель для підсумовування текстів. Архітектура має наступний вигляд.

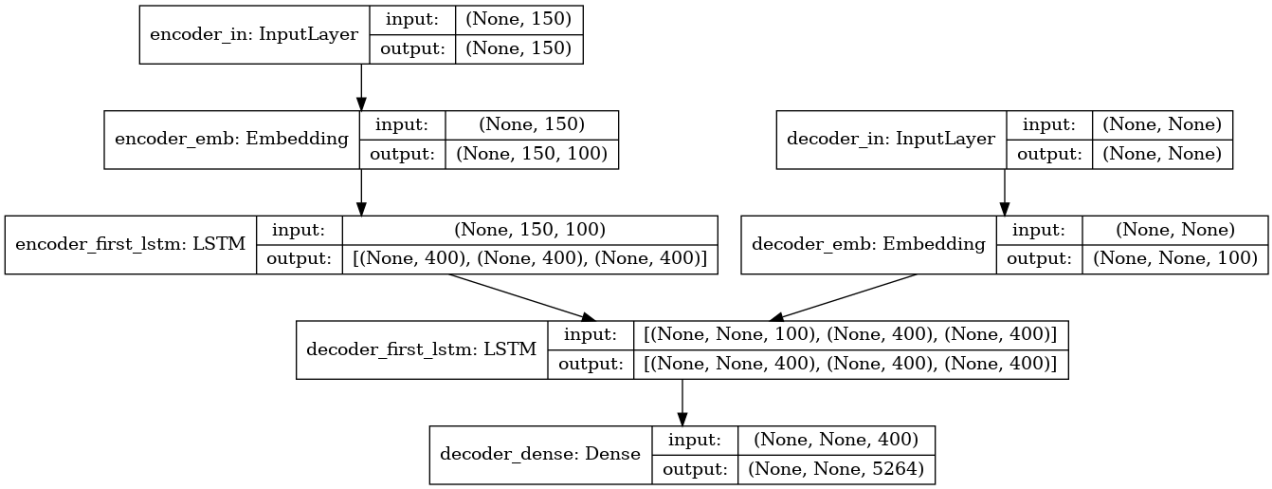

Рис. 9: Архітектура мережі

На вхід подається текст довжиною не більше 150 слів. Якщо текст коротший, то до його кінця додаються нулі. Цей текст складається з чисел, де кожному числу відповідає слово у словнику. Такий вектор передається у шар Embedding вивчаються залежності між словами. Подібні слова за значенням мають мати близьке числове представлення. На виході з цього шару отримуємо вектор розміру (None, 150, 100). None – кількість вхідних текстів. Не є заданою наперед оскільки тренування відбувається на одній кількості, а підсумовувати треба буде іншу кількість. 150 – довжина тексту, 100 – розмір пре тренованих ”embeddings”з GloVe, які у процесі навчання тренуються. Далі йде LSTM шар, на вхід приймає обчислення з попереднього шару і повертає останній вихід, і два внутрішні стани. Один з розмірів є 400, це наперед задане число, а саме розмір закодованого представлення, з якого надалі відбувається підсумовування тексту. На рисунку 9 з правої сторони також є вхід, сюди в процесі тренування подається підсумований текст, а в процесі вже натренованої моделі буде передаватися слово для передбачення наступного слова. Саме тому і розмір не є наперед заданий. Далі, також, йде навчання слів. Для декодування використовується LSTM, де поєднуються два внутрішні стани вхідного тексту з виходом натренованих ”embeddings” підсумованого тексту. І останній шар з декодера обчислює для кожного слова ймовірності. Тут 5264 – розмір словника, зі слів якого буде формуватися підсумок.

Для тренування мережі було обрано два розміри датасету. Один на 50000, другий на 90000 записів. Маємо такі графіки функції втрат.

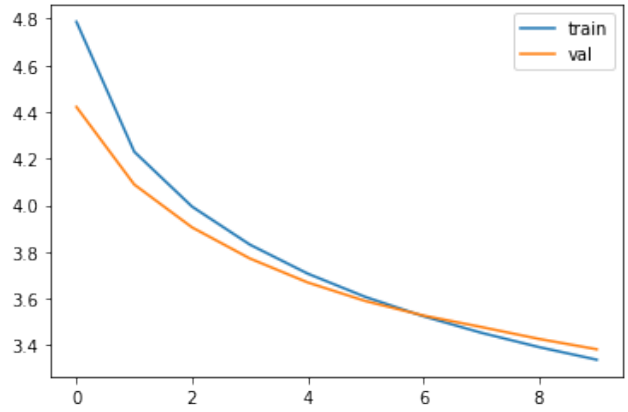

Рис. 10: Для 50000 записів

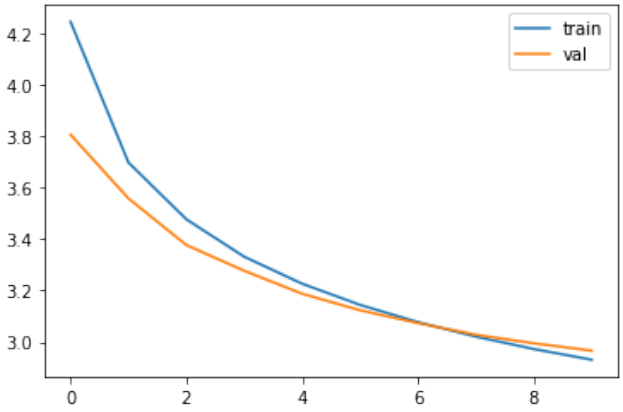

Рис. 11: Для 90000 записів

Для задач, які працюють з текстом вимірювати точність недоцільно, тому зображені функції втрат. Для першого рисунку, бачимо більше значення втрат, ніж у другому. Воно і не дивно, оскільки чим більше даних, тим кращі результати. Синім кольором позначено втрати для тренувальних даних, оранжевим для валідаційних. На 6-7 повному проходженні тренувальних даних крізь мережу, видно, що для валідаційних даних функція втрат починає зменшуватися дуже повільно, так що графіки перетинаються. Якщо на валідаційних даних покращень не буде або різниця між тренувальним набором і валідаційним буде швидко рости, то великий шанс перетренувати мережу. У такому випадку можна тренувати ще декілька проходжень і зупинитися.

Дві мережі тренувалися 10 повних проходжень тренувальних даних крізь мережу на комп’ютері з 8 ядрами та 32 ГБ пам’яті. Тренування першої мережі зайняло 61 хвилину, другої – 123 хвилини.

8 Результати

8.1 Приклади підсумків

Маємо наступний текст:
install play must download install minecraft computer minecraft already instal computer skip follow step outline part two article install mojang account require minecraft gameplay time minecraft retail game launch immediately follow installation youre sure version minecraft youre use launch minecraft note version number upper left corner screen mod system installer wizard open display onscreen file mod file require complete installation default minecraft store follow location main hard drive window cprogram mac o x
Оригінальний підсумок:
your desktop open a session of windows explorer in windows or on mac x and navigate to the minecraft folder on your computer click and drag the and jar files from your desktop and into the minecraft folder select forge from the profile dropdown menu in minecraft forge then click on play installation is now complete and will launch Згенеровані передбачення: Для топ

8.2 Аналіз отриманих підсумків

8.3 Метрики ROUGE, BLEU

Висновки

Література

[1] Sharmila Polamuri *MOST POPULAR WORD EMBEDDING TECHNIQUES IN NLP*
[Електронний ресурс] / Sharmila Polamuri // dataaspirant.com.-2020.- Режим доступу: https://dataaspirant.com/word-embedding-techniques-nlp/#t-1597717516717

[2] Chrisolah *Understanding LSTM Networks* [Електронний ресурс] / Chrisolah // colah.github.io.-2015.- Режим доступу: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] Mahnaz Koupaee, William Yang Wang *WikiHow: A Large Scale Text Summarization Dataset*
2018. arXiv: 1810.09305 [cs.LG].

[4] Sachin Abeywardana *Sequence to sequence tutorial* [Електроний ресурс] /Abeywardana Sachin // towardsdatascience.com.-2017- Режим доступу: https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8

[5] Jason Brownlee *Encoder-Decoder Models for Text Summarization in Keras* [Електроний ресурс] /Brownlee Jason // machinelearningmastery.com.-2017- Режим доступу: https://machinelearningmastery.com/encoder-decoder-models-text-summarization-keras/

[6] Ari Holtzman i Jan Buys i Li Du i Maxwell Forbes i Yejin Choi *The Curious Case of Neural Text Degeneration*
2020. arXiv: 1904.09751 [cs.CL].