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 сьогоднійшній 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 вирішенні 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. Токененом 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 видалита, а слова звести до їх канонічної форми. Звести до канонічної форми можна декількома способами:

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

Найкращим варіантом є Lemmatization з POS, але така обробка вимагає багато часу, тому Stemming теж хороший варіант, так-як потребує менше часу. Хоча для нас слова після обрублення 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, розмінка, 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 нейроної мережі, не підходить використання різних довжин векторів, тому треба звести ветор до єдиної довжини. Реалізувати це можна просто додавши 0-і до кінця тексту. Щоб нейронна мережа не вивчала ніякої інформації з таких закінчень, можна стоворити 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, слово, яке хочимо передбачити). Візьмемо таке речення: ”he quick brown fox jumps over the lazy dog”. Взявши розмір оточуючих 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 слова, а вона передбачає сусідні слова.

Особливості Skip-Gram:

- Добре працює на малих датасетах
- Краще визначає рідковживані слова

Особливості CBOW:

- Добре працює з великими датасетами
- 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 між пре-натренаваними і власноруч тренованими ”emdebbinds”є невеликою.

3 Глибокі нейронні мережі

Завдання роботи є побудова глибокої нейроної 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 в свою чергу 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

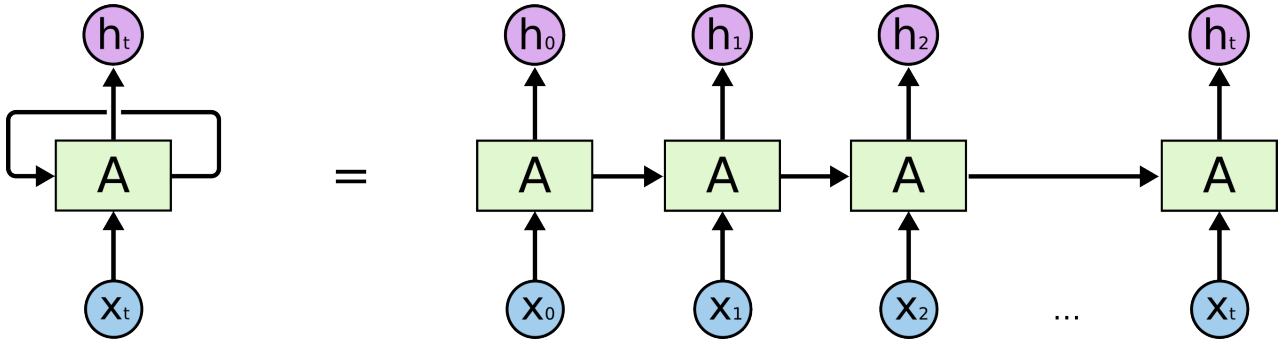

Рис. 1: Розгорнута РНМ [2]

Рекурентна нейронна мережа всередині працює наступним чином:

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

На вхід подаємо x_t – вектор із 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.

Все, здається, добре, однак, у РНМ є вагомий недолік, відомий як зникаючий градієнт. Мережа при зворотньому 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 період 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 даному 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) існуються 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 обрахунки 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.

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

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

Використаючи сигмоїду, вирішиться, 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.

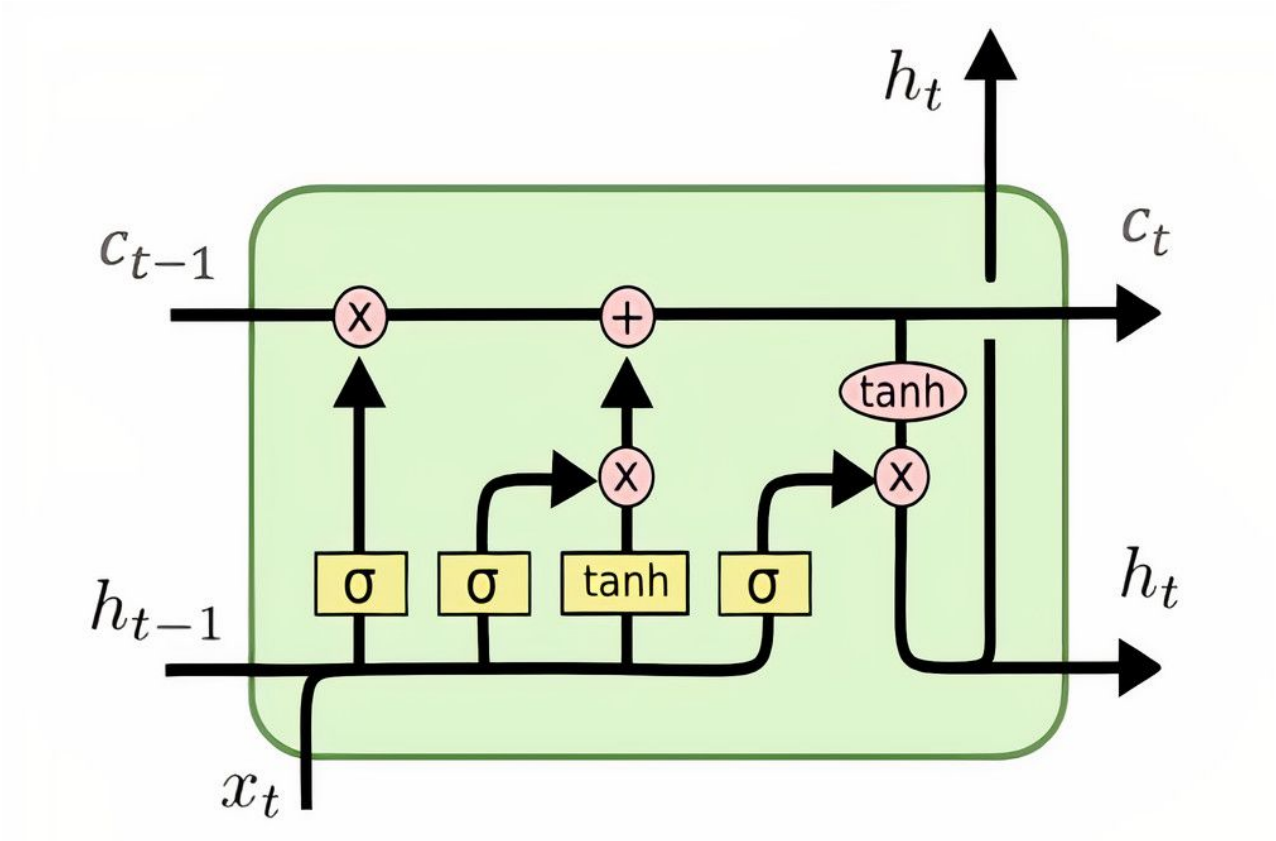

Рис. 2: LSTM клітина [2]

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

Для чого потрібний саме такий механізм пам’яті і чому він добре підходить для задачі підсумовування тексту? Даний механіз добре пряцює з часовими рядами, тобто коли потрібно прийняти рішення використовуючи вже існуючу 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 одни з одним. І це поєднює 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 більш складних зв’язках між словами. Для прикладу треба запам’ятати слово з речення, і те слово подрібно 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 Модель

4.1 Датасет

Для тренування підсумовування тексту був вибраний датасет WikiHow [3]. Розмір датасету – 230,843 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 поділенна на пункти і 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.

4.2 Тренування моделі

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] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin *Anchors: High-Precision Model-Agnostic Explanations*, 2018

[5] Christoph Molnar *Interpretable machine learning. A Guide for Making Black Box Models Explainable* / Molnar Christoph, 2020

[6] Klaise, Janis і Van Looveren, Arnaud і Vacanti, Giovanni і Coca, Alexandru *Alibi: Algorithms for monitoring and explaining machine learning models*[Електронний ресурс] / github.-2020.- Режим доступу: https://github.com/SeldonIO/alibi