Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Обробка тексту за допомогою нейромереж з використанням морфологічних особливостей слова.
(тема)

Виконав:
студент 2 курсу, групи СШІм-18-3
Куліш Д. Є.
(прізвище, ініціали)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва спеціалізації)

Керівник професор, д.т.н. Терзіян В.Я.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)
Рівень вищої освіти другий (магістерський)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва)
Тип програми освітньо -наукова
(освітньо-проіфесійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Куліш Даріні Євгеніївні
(прізвище, ім'я, по батькові)

1. Тема роботи Обробка тексту за допомогою нейромереж з використанням морфологічних особливостей слова

затверджена наказом університету від 30 березня 20 20 р. № 480 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21.05.2020 р.

3. Вихідні дані до роботи Результати досліджень на розроблених моделях мови і розроблена і реалізована структура для морфемного кодування тексту для подальшого навчання

4. Перелік питань, що потрібно опрацювати в роботі Аналіз предметної області, завдання кодування слів для обробки комп'ютером, завдання розробки моделі мови, завдання регулювання навчання моделей, завдання морфологічного значення слова

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

_____
_____
_____
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	проф.Терзіян В.Я		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної області	30.03.20 – 12.04.20	виконано
2	Дослідження видів кодування слів	12.04.20 – 17.04.20	виконано
3	Розробка API для морфологічного кодування	17.04.20 – 22.04.20	виконано
4	Розробка алгорітму морфологічного кодування	22.04.20 – 23.04.20	виконано
5	Розробка програмного забезпечення для морфологічного кодування	23.04.20 – 03.05.20	виконано
6	Написання пояснювальної записки	03.05.20 – 15.05.20	виконано
7	Нормоконтроль	15.05.20	виконано
8	Захист перед ЕК	21.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ ___проф. Терзіян В.Я._
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Записка пояснювальна: 116 с., 16 рис., 9 табл., 4 дод., 24 джерел.

МОВА, NLP, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, МОДЕЛЬ МОВИ, RNN, LSTM, GRU, МАЯКОВСКИЙ

Об'єкт дослідження - дослідження особливостей навчання моделі мови при кодуванні за допомогою символів, слів і морфіческого особливостей слова.

Мета роботи - створення моделей мови та необхідної інфраструктури для їх навчання, дослідження особливостей навчання моделей мови з використанням різного кодування даних.

Методи дослідження:

- побудова моделей мови;

- аналіз параметрів моделі під час навчання і особливостей створюваних моделлю примірників;

- оцінка ефективності моделей в порівнянні один з одним;

- оцінка ефективності морфологічного кодування для задач NLP.

ABSTRACT

Master thesis: 116 p., 16 fig., 9 tabl., 4 ann., 24 references.

LANGUAGE, NLP, MACHINE LEARNING, NEURAL NETWORKS, LANUAGE MODEL, RNN, LSTM, GRU, MAJAKOVSKY

The object of study is the study of the features of learning a language model when coding with the help of symbols, words and morphic features of a word.

The purpose of the work is to create language models and the necessary infrastructure for their training, to study the features of teaching language models using various data coding.

Research Methods:

- building language models;
- analysis of model parameters during training and the features created by the model instances;
- assessment of the effectiveness of models in comparison with each other;
- assessment of the effectiveness of morphological coding for NLP tasks.

РЕФЕРАТ

Пояснительная записка: 116 с., 16 рис., 9 табл., 4 прил., 24 источника.

ЯЗЫК, NLP, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, МОДЕЛЬ ЯЗЫКА, RNN, LSTM, GRU, МАЯКОВСКИЙ

Объект исследования - исследование особенностей обучения модели языка при кодировании с помощью символов, слов и морфического особенностей слова.

Цель работы - создание моделей языка и необходимой инфраструктуры для их обучения, исследования особенностей обучения моделей языка с использованием различного кодирования данных.

Методы исследования:

- построение моделей языка;

- анализ параметров модели во время обучения и особенностей создаваемых моделью экземпляров;

- оценка эффективности моделей в сравнении друг с другом;

- оценка эффективности морфологического кодирования для задач NLP.

ЗМІСТ

С.

ВСТУП

Робота нейронних мереж з текстом є однією з основних і найбільш розвинених областей машинного навчання, однак і однією з найбільш затребуваних. Розроблено безліч методів для кращої продуктивності, частина яких розглянута в теоретичній частині цієї роботи, однак поле має безліч областей для дослідження. Результати роботи нейронних мереж з текстом допомагають як в глобалізації світу (розчиняючи мовні бар'єри на повсякденному рівні) так і в аналізі та роботі з численними текстовими даними, наявними в інтернеті.

Тема кодування даних в NLP є варіативною в залежності від завдання, однак тема морфологічного кодування ще менш вивчена. Почасти це пояснюється тим, що російська та українська мови несуть у собі набагато більше морфологічно відокремлюваної з слів інформацією, ніж, наприклад, англійська.

У даній роботі демонструється, наскільки великий вплив кодування впливає як на продуктивність навчання моделі, на рівень результату, до якого у моделі є можливість дійти, на швидкість і особливості навчання, а й на кількість параметри навчання, якими є можливість направляти навчання моделі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дані послідовності (Sequence data)

Даними послідовності є дані, в яких екземпляри інформації йдуть один за одними, пов’язані віссю часу – тобто мають залежність від попередніх примірників інформації. Приклади таких даних, що найчастіше зустрічаються, є аудіо, відео, текст. Наприклад, в розпізнаванні тексту з аудіо на вході та на виході дані є послідовностями, однак на вході закодовані звукові одиниці, а на виході – символи або слова (в залежності від імплементації).

Генерація музики - зовсім інший приклад роботи з даними послідовності. На вході у нас порожня множина, а на виході послідовність. Однак вхід необов'язково повинен бути порожнім безліччю в даному випадку. Все залежить від мети і даних датасета. Вхідні дані в загальному випадку повинні містити «напрямні» настройки для моделі. Наприклад, це може бути число, яке ідентифікує жанр музики, або вектор специфічних налаштувань. Модель для таких послідовностей повинна навчатися на вхідних даних в такому ж форматі.

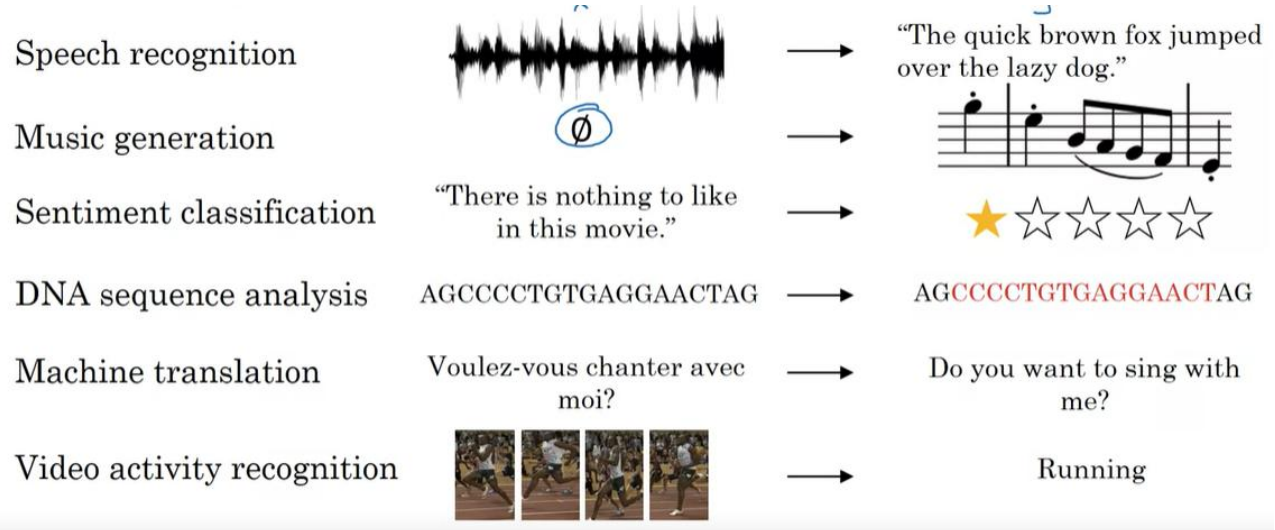


Рисунок 1.1 – Приклади роботи с послідовностями [1]

Робота з послідовностями є великою областю машинного навчання і включається в себе роботу з різними даними. Використовуючи нотацію X для вхідних даних і Y для результуючих даних, обидва набору для одного примірника можуть бути різної довжини (як наводилося в прикладі з музикою).

1.2 Позначення

Вхідний вектор даних позначається X, результуючий вектор даних позначається як Y. При послівний обробці слів, кожне слово в одному екземплярі позначається як x <t>, де t - порядковий номер слова в реченні. t в даному випадку означає time - момент часу, що підкреслює залежність слів (або звуків, кадрів і т.д.) один від одного і їх порядок.

Позначення T_x використовується для позначення довжини примірника (кількості слів / кадрів / звуків). Аналогічна нотація (y <t>, T_y) використовується і з вихідними даними. X позначає матрицю вхідних даних (що містить безліч окремих екземплярів - все або batch обмеженого розміру), а x(i) - один примірник вхідних даних. Аналогічна нотація (y(i)) - застосовується до вихідних даних.

Пара x(i), y(i) відповідають даним і результату для одного примірника. При тому T_x і T_y можуть бути не рівні один одному (або дорівнювати 0), а n_x і n_y повинні рівнятися один одному, так як кожен екземпляр даних має x і y відповідно.

Наприклад, у реченні «Белоснежка была несказанно рада увидеть семь лесных гномов», яке переведено на українську мову як «Білосніжка була несказанно рада побачити сім лісових гномів», перше речення є x(n), а друге становить y(n). Слово «Белоснежка» можна ідентифікувати як $x(n)_{<1>}$, а «побачити», як $y(n)_{<5>}$. T_x^i для цього примірника буде дорівнювати 8, як і T_x^i.

Як відобразити окреме слово для обробки його нейронною мережею? Детальніше ми також будемо розглядати це питання в розділі «Аналіз відів word encoding». Для репрезентації слова для початок необхідно скласти так званий словник - список слів, які ми будемо враховувати в використанні своєї

моделі. Список повинен бути кінцевий і не обов'язково представляє усі слова мови.

Якщо ми виберемо список з 10 000 слів, то репрезентацією одного слова з цього списку буде одновимірний масив елементів (вектор), що складається з нулів, де тільки той член масиву, чий порядковий номер відповідає слову, буде одиницею. Такий спосіб називається one-hot репрезентацією (так як тільки один елемент є «гарячим» = має вагу).

Для слів, які можуть попастися якщо не в тренувальних, то в тестових примірниках або при використанні моделі, які не входять до кодується список слів, зазвичай додається додатковий токен, який позначає «невідоме слово». Це також називається скороченням UNK (Unknown word).

1.3 Recurrent Neural Networks (RNNs)

Для проблем послідовностей ми не можемо використовувати звичайні нейронні мережі з наступних причин:

- вхідні та вихідні послідовності можуть бути різного розміру (= кількість слів у реченні, Tx і Ty різні для кожного екземпляра);
- «наївна» архітектура звичайних нейронних мереж не поділяється патерни між місцями в пропозицій, тобто якщо нейронна мережа вивчила і знає, що після слів «його» і «звуть» на позиціях 2 і 3 на позиції 4, швидше за все, буде йти ім'я, для позицій 3,4 і 5 цей патерн використовуватися не буде.

Опишемо в загальному вигляді архітектуру рекурентної нейронної мережі - вхідний інформацією є вектор першого слова, а передбачити вона буде намагатися вектор наступного. Особливість рекурентної нейронної мережі в тому, що коли вона робить наступну ітерацію (намагається передбачити третє слово, наприклад), крім попереднього слова вона отримує також деяку інформацію з попередньої ітерації (перетворення першого слова в друге). Цей вектор інформації позначається як $a^{<i>}$ (і тут - номер слова в реченні) [25].

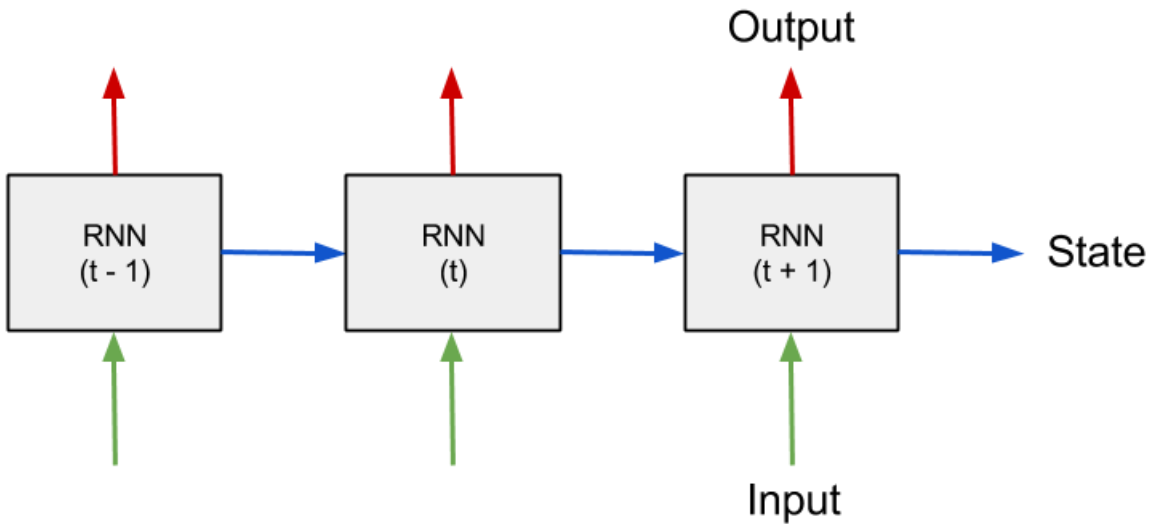


Рисунок 1.2 – Архітектура рекурентної нейронної мережі, великий погляд

Рекурентна нейронна мережа сканує параметри зліва направо. Параметри (ваги), які вона використовує на кожному кроці, є однаковими для всіх кроків мережі (що зокрема дозволяє feature sharing, недоступне звичайним нейронних мереж з послідовностями). Набір параметрів (ваг), який перетворює x $(i)^{<t>}$ в y $(i)^{<t>}$ називається W_{ax} і повторюється для кожної ітерації в часі. Активаційний вектор, який передається в наступний шар, також має свої параметри W_{aa}. Вихідний вектор (який визначає слово або ймовірність слів) у також має ваги W_{ay} [14].

Як видно на рисунку 1.2, при такій схемі вузол отримує не тільки дані від вхідної інформації (слова для перекладу, наприклад), а від усіх попередніх вузлів також (через активаційний вектор a).

Одна з слабкостей RNN полягає в тому, що вузол отримує інформацію тільки від попередніх даних. Наприклад, у реченні «Кот, сидить на тумбочці, вилизував свій хвіст.» буде більше інформації для перекладу слова «сидить» з урахуванням роду, ніж в реченні «сидить на тумочку кіт вилизував свій хвіст.» У першому випадку слово «сидить» має іменник «кіт» як попередній вузол, що несе додаткову інформацію про рід. Для нейронної мережі, що визначає людські імена в реченні, що не буде різниці в слові «Тедді» в реченнях «Тедді

Рузвельт закрив вікно» і «Тедді-ведмедик належав Сергію.» Цю проблему вирішують двонаправлені RNN.

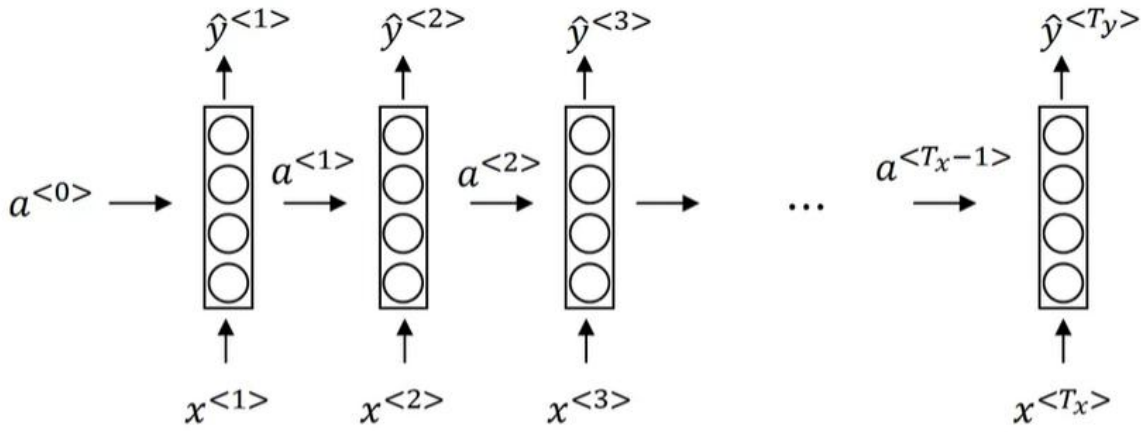


Рисунок 1.3 – Архітектура RNN - наближений вигляд

$a^{<0>}$ - зазвичай вектор, заповнений нулями або рандомних значеннями.

Forward propagation - так називається математичний процес перетворення вхідного вектора в «результат». Щоб обчислити a_1, ми повинні помножити ваги Waa на $a^{<0>}$, додати до них вхідний вектор $x^{<1>}$ пропущений (множення на) через ваги W_{ax}, додати вільний член (зміщення - bias), і потім пропустити результат через нелінійну функцію g().

Функція активації називається так, тому що дозволяє нейронної мережі ідентифікувати нелінійні стану. При відсутності нелінійної функції, будь-яку кількість шарів глуюокой нейронної мережі можна буде переписати за допомогою одного шару, так як всі вони складаються з множення і складання. Неактивована нейронна мережа буде діяти як лінійна регресія з обмеженою можливістю до навчання. Коли лінійна функція використовується у всіх шарах, між вхідним шаром і вихідним шаром досягається один і той же лінійний результат. Лінійна комбінація лінійних функцій є ще однією лінійною функцією. Лінійні функції не уявляють навчальну цінність для нейронної мережі, але можуть використовувати на вихідному шарі як довічного класифікатора.

Щоб потім отримати y_pred (результат, обчислений нейронною мережею, на відміну від y - реального результату даного кроку), необхідно помножити ваги W_{ya} на $a^{<1>}$ і додати інший bias, потім пропустити через іншу функцію активації.

$$a^{<1>} = g_1(W_{aa}*a^{<0>} + W_{ax}*x^{<1>} + b_a)$$
$$y\_pred^{<1>} = g_2(W_{ya}*a^{<1>} + b_y) \quad (1.1)$$

Індекси в вагах також означають їх приналежність і розмірність - перший індекс відповідає за результуючий вектор, а другий за вектор, на який ваги діють. У якості першої активационной функції для RNN зазвичай використовується тангенсоіда, а вибір другої активационной функції залежить від результату - проблема класифікації, перекладу, тощо. Для бінарної класифікації використовується сигмоїда, а для класифікаційних проблем використовується softmax. Таким чином, наприклад, для згаданої задачі розпізнавання імен в реченні, вихідний вектор буде бінарним (1/0), відповідно, варто було б використовувати sigmoid().

$$a^{<1>} = g_1(W_a[a^{<0>},x^{<1>}] + b_a)$$
$$[W_{aa}:W_{ax}] = W_a \quad (1.2)$$

Першу формулу у формулі 1.1 також нерідко скорочують для швидкості і простоти використання. Ваги W_{aa} можна з'єднати з W_{ax}, а вектор $a^{<0>}$ соеденить з вектором $x^{<1>}$ - після перемноження і складання результат буде тим же. В результаті перша формула для першого слова буде вважатися за формулою 2.

Трохи поясню розрахунки, пов'язані з таким об'єднанням матриць. Якщо наш словник має 2000 слів, то входить в клітку вектор матиме розмірність (2000,1). Припустимо, що входить вектор a, який забезпечує перенесення інформації з минулих клітин, має розмірність (100, 1). Тоді W_{aa} матиме розмірність (100,100), а ваги W_{ax} - (100, 2000).

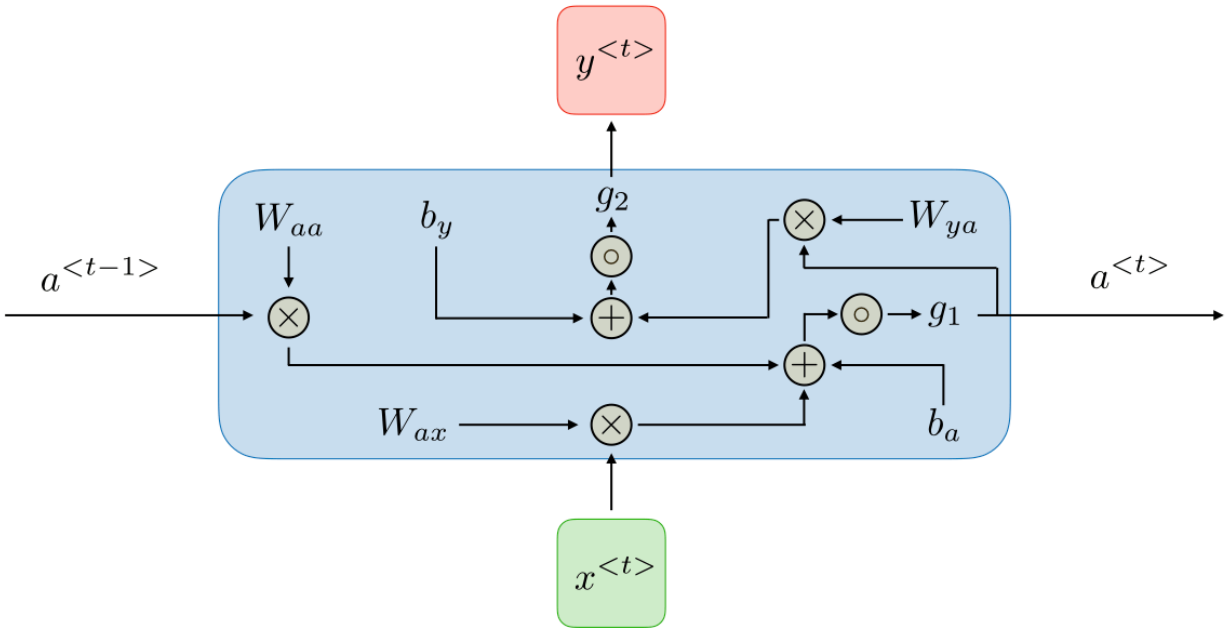


Рисунок 1.4 - одна клітина RNN з вагами і функціями

При використанні повної формули, множення W_{aa} (100,100) з вектором a (100,1) дасть вектор розміром (100,1); перемноження W_{ax} (100, 2000) з вектором x (2000, 1), дасть також вектор розмірністю (100,1). Потім за допомогою поєлементного складання ми отримаємо вектор тієї ж розмірності (100,1), який і буде вихідним вектором a для цієї клітини і входять для наступної.

При використанні спрощеної формули, приєднані разом матриці ваг W_{aa} і W_{ax} становитимуть єдину матрицю розмірністю (100, 2100), а вектора, з'єднані в один, представлятимуть вектор розмірністю (2100, 1), і при перемножуванні ми отримаємо той же вектор a розмірністю (100,1), проте складання будт виконуватися всередині процедури множення матриць як природна частина цієї процедури, що є перевагою через оптимізованих методів множення матриць в більшості сучасних мов програмування.

1.4 Backpropagation в RNN

Backpropagation, або процес зворотного поширення помилки, є найважливішою частиною процесу з навчання нейронної мережі. У загальному

вигляді, різниця між очікуваним і отриманим результатом поширюється по верствам тому справа наліво, вказуючи кожному обчислювальному вузлу, наскільки йому потрібно змінити конкретну вагу. Коли ми використовуємо фреймворки для машинного навчання, зазвичай цей процес повністю прихований від нас, але розуміти його механіку дуже важливо з наукової точки зору щоб розуміти, як зміна окремих частин процесу може впливати на навчання нейронної мережі.

В першу чергу, щоб порахувати зворотне поширення помилки, необхідно визначити функцію розрахунку втрат (loss function, L ()). Для проблеми визначення імен в реченні (бінарної) можна взяти стандартну функцію розрахунку втрат для логістичної регресії.

$$L^{<t>}(y\_pred^{<t>}, y^{<t>}) = - y^{<t>}*\ln(y\_pred^{<t>}) – (1 - y^{<t>})*\ln(1 - y\_pred^{<t>})$$
$$L(y\_pred, y) = \sum L^{<t>}(y\_pred^{<t>}, y^{<t>}) \quad (1.3)$$

Для рекуррентной нейронної мережі ми підраховуємо функцію помилки на кожному моменті, і потім використовуємо суму цих помилок.

Так як в якості вихідної функції для бінарної класифікації ми використовуємо сигмоид, то для backpropagation нам знадобиться її похідна. Нам також знадобиться похідна тангенсоіди, яку ми використовуємо в якості першої активационной функції дли підрахунку вектора a.

$$g_1 = \tanh(x), \tanh(x) = (e^x – e^{-x})/(e^x + e^{-x})$$
$$\partial\tanh(x)/\partial x = 1 – (\tanh(x))^2$$
$$g_2 = \text{sigmoid}(x), \tanh(x) = 1/(1 + e^{-x})$$
$$\partial\text{sigmoid}(x)/\partial x = 1 – (\text{sigmoid}(x)) \quad (1.4)$$

Для зміни ваг на одному конкретному моменті, нам необхідно обчислити вплив на результат обчислень для цього моменту не тільки поточної клітини, але і кожної з попередніх (так як їх інформацію переносить вектор a). Приклад розрахунку для однієї з клітин можна бачити на формулі 5, що використовує

рисунок 1.5 [2]. Нам необхідно обчислити вплив ваг на кожному кроці на результат даного (3) кроку; однак, так як матриці ваг на кожному кроці використовуються одні й ті ж, то потім ці «впливи» складаються для всіх кроків (в даному випадку, трьох).

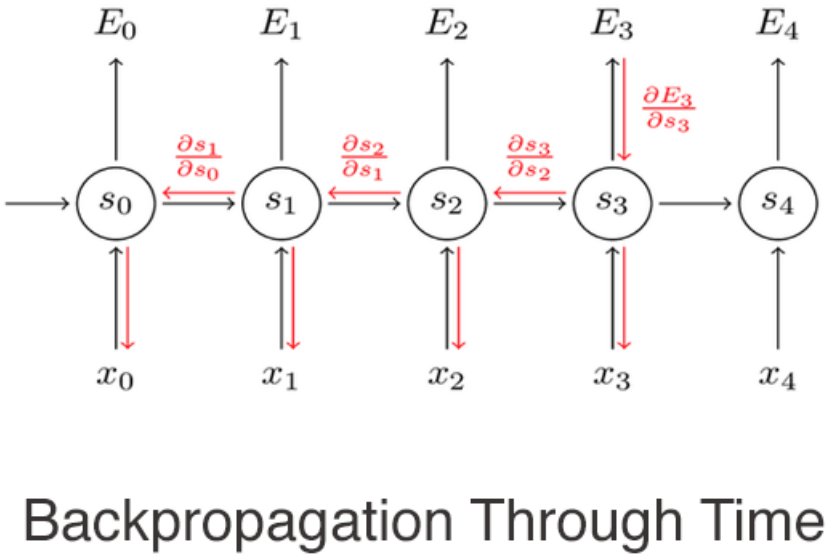


Рисунок 1.5 – backpropagation для однієї клітини RNN

$$\partial E_3/\partial W = \sum(k=0, 3)\ (\partial E_3/\partial y\_pred_3) * (\partial y\_pred_3/\partial s_3) * (\partial s_3/s_k) * (\partial s_k/\partial W) \quad (1.5)$$

Похідні множаться згідно chain rule для похідних, так як змінні спираються одна на одну.

1.5 Види RNN

Tx не обов'язково повинен дорівнювати Ty, як у випадку, на якому я розглядала backpropagation. Як було поверхнево описано в розділі 1.1, як вхідна так і вихідна послідовність може бути порожньою безліччю, бінарним класифікатором (0/1), звичайним класифікатором (integer за кількістю категорій) або вектором іншої довжини (послідовністю). Актуальний приклад

останнього - переклад з однієї мови на іншу, де кількість слів у джерелі і в перекладі часто не збігається.

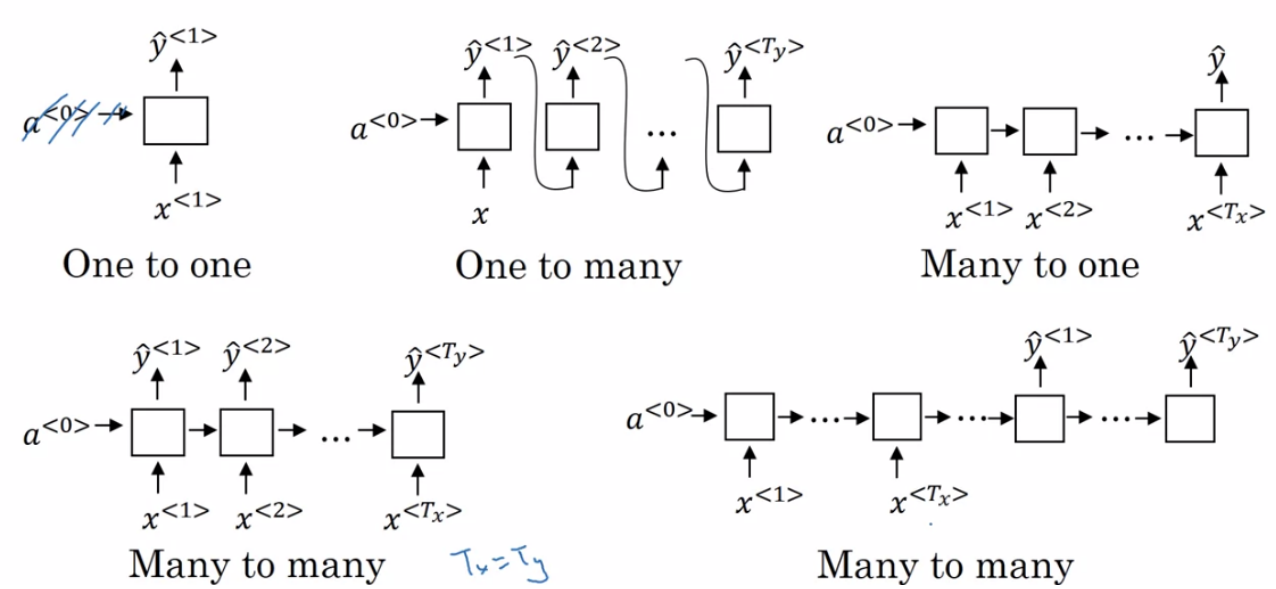


Рисунок 1.6 – різні архітектури RNN [3]

Архітектура «many-to-one» може використовуватися, коли на вхід подається текст (послідовність), а на виході потрібно отримати число-клас. Наприклад, для виставлення рейтингу на основі відгуків. На вхід подаються слова, одне за одним (або символи, в залежності від архітектури), а на виході потрібно отримати акумульовану оцінку. Таким чином, RNN зчитує всю послідовність, і потім видає оцінку тексту.

Приклад «one-to-many» архітектури - генерація музики по класу жанру, про який ми вже говорили. Тоді виходом для такої архітектури буде послідовність нот, що залежить як від входу (жанру), так і від попередніх нот. Ремарка: для подібний архітектури, при генерації послідовності, в наступну клітину зазвичай йде не тільки вектор a, який переносить інформацію про обчислення, але і на вхід подається згенерований результат попередньої клітини y_pred.

Для роботи з послідовностями, які розрізняються за кількістю елементів (слів, знаків, звуків), використовується архітектура RNN «many-to-many», де

мережа спочатку зчитує всю вхідну послідовність один вектор за іншим, потім в такому ж порядку генерує вихідну послідовність.

Таким чином, ця нейронна мережа фактично складається з двох окремих частин, з'єднаних послідовно - кодувальника (encoder), який зчитує іфнормацію з вхідної послідовності і декодувальник (decoder), який перетворює цю інформацію в вихідну послідовність.

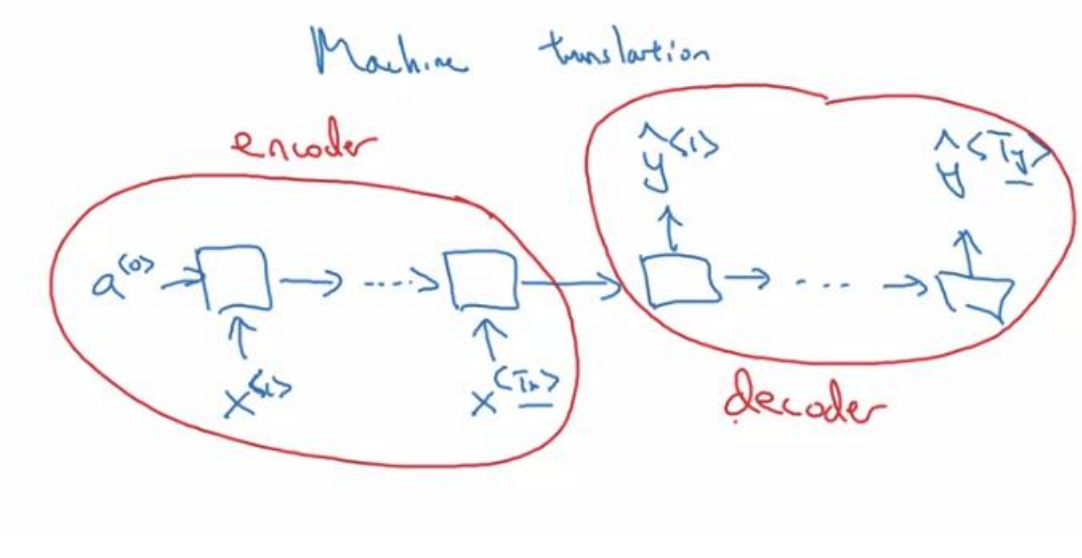


Рисунок 1.7 – архітектура RNN many-to-many з різною кількістю елементів в послідовності

1.6 Генерація мови і послідовності

Модель мови це ймовірності проходження слів одне за іншим. Наприклад, в пропозиціях, де два слова є омофона - словами, які звучать однаково, але пишуться по-різному і мають різне значення (наприклад, «плисти» і «плести»), хороша система розпізнання слів з моделлю мови буде робити вибір, грунтуючись на попередніх словах або слові в послідовності. Відповідно до моделі мови, система може скласти ймовірно існування кожного з пропозицій і вибрати пропозицію з найбільшою ймовірністю (тоді враховуються не тільки попередні слова, але все слова в реченні, тобто ймовірність мати наступне слово за словом, в якому мережа сумнівається). Мережі, які використовуються для переказу, також використовують моделі

мови, щоб переводити найбільш ймовірні пропозиції. Ті, які швидше за зустрінуться в реальному світі.

Напевно, тому стандартні перекладачі навіть з великим об'ємом даних так нехороші в перекладі пісень - для цього необхідна мережа, навчена спечіально переводити пісні - можливо, вона навіть буде виробляти римований переклад.

Ще раз - модель мови потрібна для того, щоб отримувати ймовірність існування речення, виходячи з послідовності входячих або виходячих з мережі елементів:

$$P(y_{<1>}, y_{<2>}, \dots y_{<Ty>}) \quad (1.6)$$

Для створення моделі мови необхідна велика кількість тексту вибраної мови, для відстеження зв'язків між словами. Ми створюємо словник слів обраного датасета даних, токенізіруем їх (перетворюємо в вектор, кодуємо). Також досить часто зустрічающаяся річ - додатковий токен не тільки для незнайомих слів, але і для кінця речення - EOS (End Of Sentence token). Цей токен розумно додавати в усі послідовності, де окреме речення не є закінченим екземпляром.

Далі необхідно побудувати RNN для моделювання шансів різних послідовностей. В даному випадку RNN намагається передбачити наступне слово, маючи g_2 softmax - отримуючи вектор ймовірності наступного слова від поточного. У міру навчання для кожного слова ми зможемо мати вектор ймовірностей для кожного слова в словнику, враховуючи попереднє речення. На кожному кроці мережа отримує інформацію про попередні словах / символах / звуках через вектор a, і інформацію про останнє слово за допомогою вхідного вектора x, а в результаті виводить розподіл ймовірностей наступного слова (символу, звуку, тощо).

1.7 Загасання градієнта в RNN

Одна з важливих проблем, з якими стикається базовий алгоритм RNN - згасання градієнта. Якщо між, наприклад, іменником і прикметником або

дієсловом, яке до нього відноситься, знаходиться багато доповнень і інших частин предлложенія, проблема визначення роду для цього прикметника або дієслова стає важче або повністю неможливою назад прямо пропорційно кількості слів між ними. Мова може мати велику кількість довгоживучих залежностей, з якими RNN важко впорається. Що ще гірше, в процесі зворотного поширення помилки, з точно такою ж проблемою стикається процес донесення цієї залежності і зміщення ваг для виправлення цієї проблеми. Помилка, у міру зворотного поширення, «гасне» до слова, на яке повинно було спиратися рішення.

RNN також чутливі до так званого вибуху градієнта, коли помилка починає неконтрольовано зростати в міру проходження через шари нейронної мережі (в даному випадку, через моменти часу). Але в даному випадку це простіше помітити (поява NaN - not a number значень при переповненні змінних) і виправити. Звичайна практика мати якесь граничне значення (максимальне для градієнта) і при його переході автоматично зменшувати пропорційно весь градієнт.

Gated Recurrent Unit (GRU) дозволяє вирішити проблему із згасанням градієнта і працювати з більш довгими послідовностями.

1.8 Gated Recurrent Unit

Gated Recurrent Unit (GRU) це модифікація для прихованого шару рекуррентной нейронної мережі, яка дозволяє їй працювати з довгими послідовностями і зберігати інформацію. Дозволяє працювати з довго рознесеними інформаційними залежностями і допомагає з проблемами згасання градієнта.

GRU вносить нову змінну c (memory cell). Для чистого GRU c <t> = a <t>. На кожному кроці ми будемо розглядати можливість переписати значення c <t> значенням c_tilda <t>. Ми обчислюємо це значення за допомогою формули 7.

$$c\_tilda = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c) \quad (1.7)$$

Важлива концепція GRU - наявність «воріт» (gate). Ворота називаються гамма u, де u означає update gate - ворота поновлення. Γ_u буде значенням між нулем і одиницею.

$$\Gamma_u = \text{sigmoid}(W_u*[c^{<t-1>}, x^{<t>}] + b_u) \quad (1.8)$$

У більшості випадків гамма u є або дуже близьким до нуля значенням, або дуже близьким до одиниці. Ми оновлюємо вектор с за допомогою c_tilda, а ворота прийматимуть рішення, оновлювати нам фактично або не оновлювати його (для кожного значення в векторі окремо).

Ми передаємо дані про стан і вхідні даний в тангенсоіду, щоб отримати нову інформацію, потім передаємо той же вхід, але з іншими навченими вагами в сигмоид, щоб виявити релевантність цієї інформації (тут 0 на виході - неважлива інформація, а 1 - важлива), потім множимо і оновлюємо клітку стану для цього кроку.

Отже, можна записати стан с поточної клітини як:

$$c^{<t>} = \Gamma_u * c\_tilda^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \quad (1.9)$$

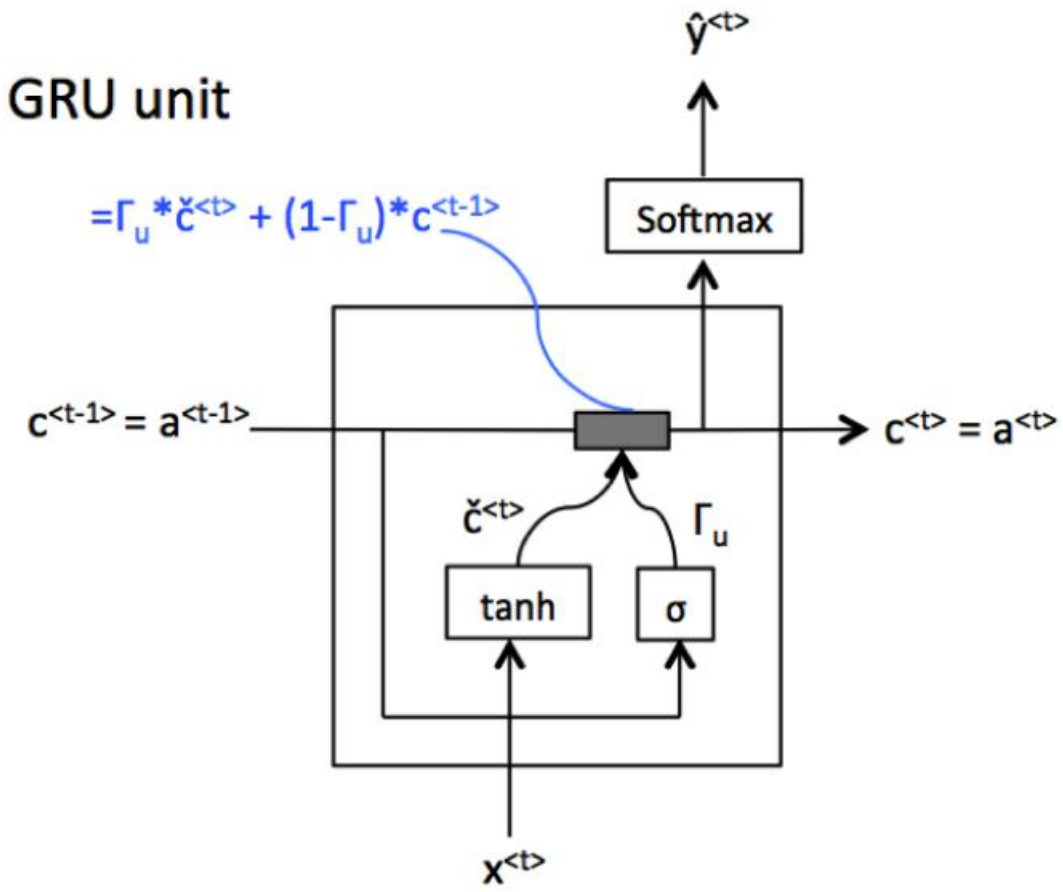


Рисунок 1.8 – GRU спрощений варіант

Таким чином, стан нової клітини складається з попереднього стану і c_tilda (яка також ґрунтується на вхідної інформації в нову клітку і старому стані), що проходить через гейт релевантності інформації.

Для речення з прикладу, коли нам потрібно перенести інформацію про рід головного іменника до прикметника або дієслова в кінці речення, після навчання RNN буде використовувати осередок в стані, щоб зберігати інформацію про рід, а гейт повинен не давати переписувати цей осередок аж до доставляння інформації в кінець речення.

У спрощеному GRU це значення використовується замість a як для передачі стану далі, так і для генерації передбачення y_pred. GRU не так сильно схильний до проблеми згасання градієнта, так як більшу частину часу значення

гейта дуже дуже малі (не пропускають нову інформацію), відповідно, інформація передається на «великі відстані» [20].

$c^{<t>}$ може бути вектором, $c\_tilda^{<t>}$ має таку ж розмірність, і гейт матиме таку ж розмірність (множення у формулі 9 - поелементне множення).

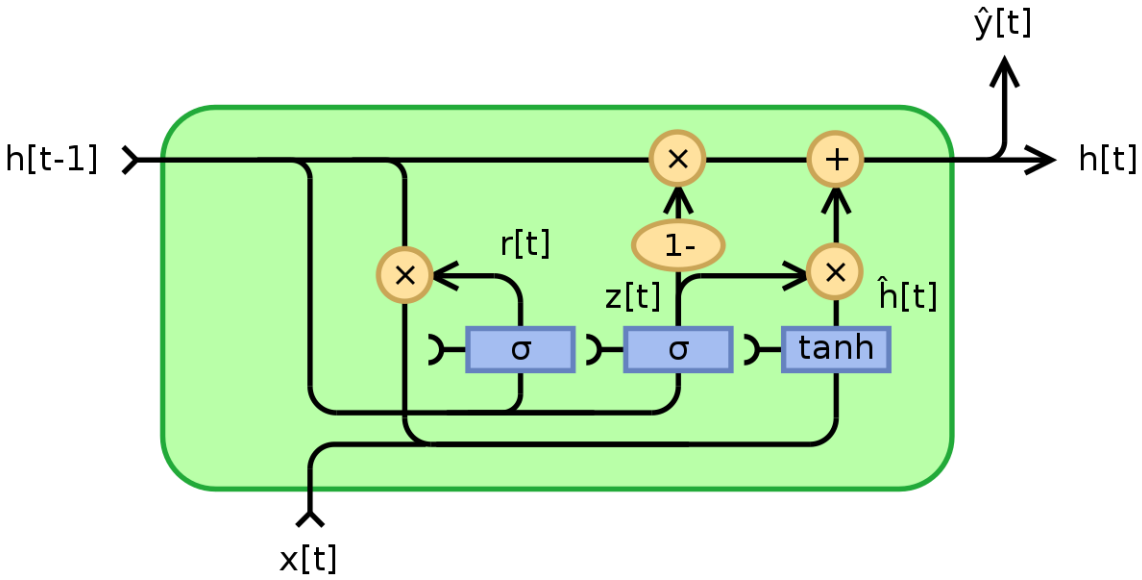


Рисунок 1.9 - GRU повний варіант

Полный GRU имеет еще одни ворота, которые находятся на входе c<t-1> в c_tilda:

$$c\_tilda = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c) \quad (1.10)$$

Отже, додатковий гейт Γr. Тут r - relevance, з'ясовує, наскільки значимо попередній стан для наступного.

Гейт підраховується очікувано схожою формулою з власним вагами:

$$\Gamma_r = \text{sigmoid}(W_r * [c^{<t-1>}, x^{<t>}] + b_r) \quad (1.11)$$

Його присутність може здаватися зайвим - в результаті ми маємо гейт, що визначає релевантність старого стану з урахуванням входу для проміжного

стану, саме проміжний стан вважається і з цим гейтом і зі входом, потім гейт, що визначає релевантність цього проміжного стану для зміни старого стану (учтивающій то ж старе стан і вхідний вектор) і, нарешті, новий стан. Однак багато років дослідів дослідників привели до саме такої архітектури, яка показала найбільшу стійкість до згасання градієнта, при цьому передаючи інформацію на найбільші відстані в послідовності.

Іншою популярною версією є LSTM.

1.9 LSTM

LSTM розшифровується як long short-term memory (мережі з довгої і короткої пам'яттю). Вони ще більш потужні, ніж GRU.

Для GRU a <t> = c <t>, тоді як для LSTM використовуються обидва ці вектора. Коротко перерахую відмінності LSTM від GRU (повна формула приведена у формулі 12). У розрахунках c_tilda замість попереднього вектора стану c використовується входний вектор a, також він використовується для розрахунку update gate.

Також в розрахунки нового стану клітини c <t> додається новий гейт - так званий forget gate - який виконує ту ж функцію, яку виконувала частину (1-update gate) (інвертована важливість нової інформації стану c_tilds), але має свої ваги і здатний доставити більше інформації про правильне «забування». Можна сказати, що таким чином реалізується «single responsibility principle» - принцип єдиної відповідальності з ООП, коли кожен гейт відповідає за одну функцію і ваги «натаскуються» на певну мету. Формула для розрахунку нового гейта входить в формулу 12.

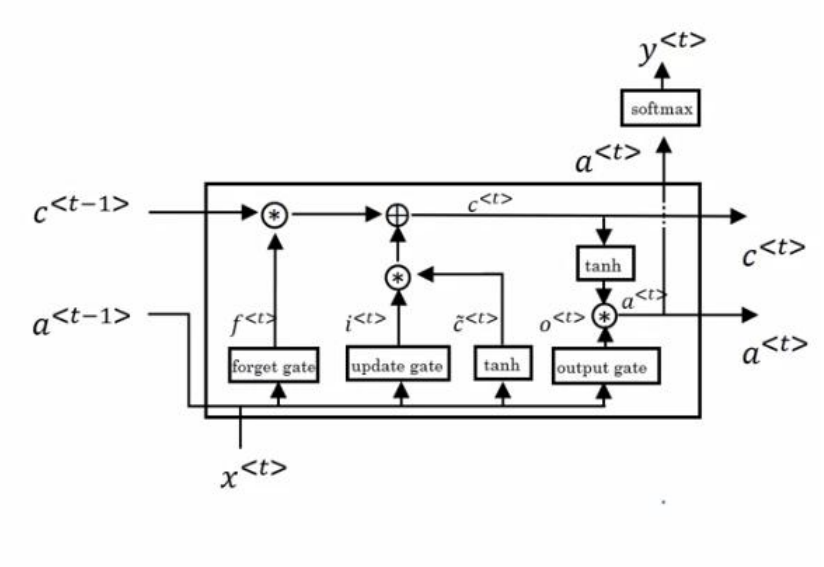


Рисунок 1.10 – схема LSTM клітини [1]

LSTM також має додатковий гейт output gate, який відповідає за перетворення c <t> в a <t>.

$$c\_tilda^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \text{sigmoid}(W_u*[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \text{sigmoid}(W_f*[a^{<t-1>}, x^{<t>}] + b_r)$$
$$\Gamma_o = \text{sigmoid}(W_o*[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * c\_tilda^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh(c^{<t>}) \quad (1.12)$$

LSTM має деякі варіації, в яких гейти залежать не від a<t-1>, а від c<t-1>. Це називається peephole connection.

Незважаючи на більшу складність LSTM, в різних завданнях LSTM і GRU можуть конкурувати - вибір технології залежить від завдання. GRU з'явилися пізніше, ніж концепція LSTM, як спрощений LSTM.

1.10 Двонаправлені RNN

Двонаправлені рекурентні нейронні мережі дозволяють одночасно отримувати інформацію і про минулі і про майбутніх клітинах. Це дозволяє вирішити проблему, про яку згадувалося раніше, коли з початку послідовності не можна прийняти рішення про вихід даної клітини, і також збільшує кількість зв'язків.

Двонаправлені RNN (BRNN) працюють таким чином: вони мають окремі елементи і зв'язку, що переміщують і акумулюють інформацію зліва направо, як в звичайних RNN, але на додаток має аналогічний перенесення інформації в протилежному напрямку. Вам необхідно провести forward propagation в обидві сторони, перш ніж мережа зможе зробити припущення про y і ви зможете потім запустити двосторонній backpropagation [15].

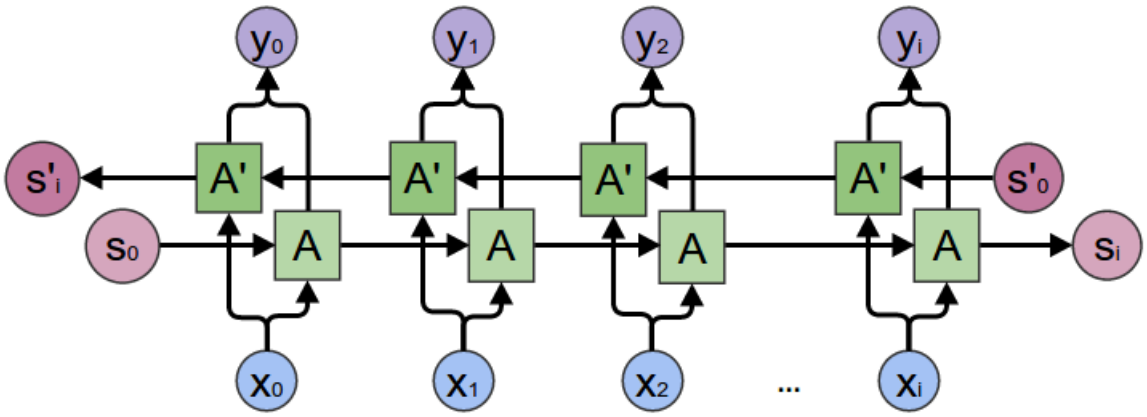


Рисунок 1.11 – двонаправлені RNN (BRNN)

У такому випадку для обчислення y_pred використовуються обидва вектори пам'яті, як:

$$y\_pred^{<t>} = g(W_y[a'^{<t>}, a^{<t>}] + b_y) \quad (1.13)$$

Таким чином нейронна мережа отримує інформацію і з минулого і з майбутнього в послідовності. Для багатьох проблем natural language processing

(обробка натуральної мови) використовуються двонаправлені рекурентні нейронні мережі.

Основний мінус двонаправленої рекуррентной нейронної мережі в тому, що вам необхідно обробити повну послідовність, щоб зробити прогноз. Таким чином, наприклад, для систем обробки мови в реальному часі дана технологія не підходить (або повинна як мінімум мати пре-обробку іншою мережею, яка буде визначати кінець пропозиції в мові і відсікати частина інформації для обробки BRNN).

1.11 Глибокі RNN (Deep RNN)

Різні версії RNN, які описані в попередніх розділах, працюють досить добре самі по собі, але в навчанні досить глибоких і складних залежностей може бути корисним іноді з'єднувати кілька шарів RNN разом для виокремлення більш глибокої інформації.

Для ідентифікації шару, до якого належить вектор / матриця, в нотацію додається верхній символ [i], де i - номер шару.

Власне, шари кладуться один над іншим, так що кожен має однакову кількість клітин «моментів» і на виході нижніх шарів не y_pred, а вхід в наступний рівень.

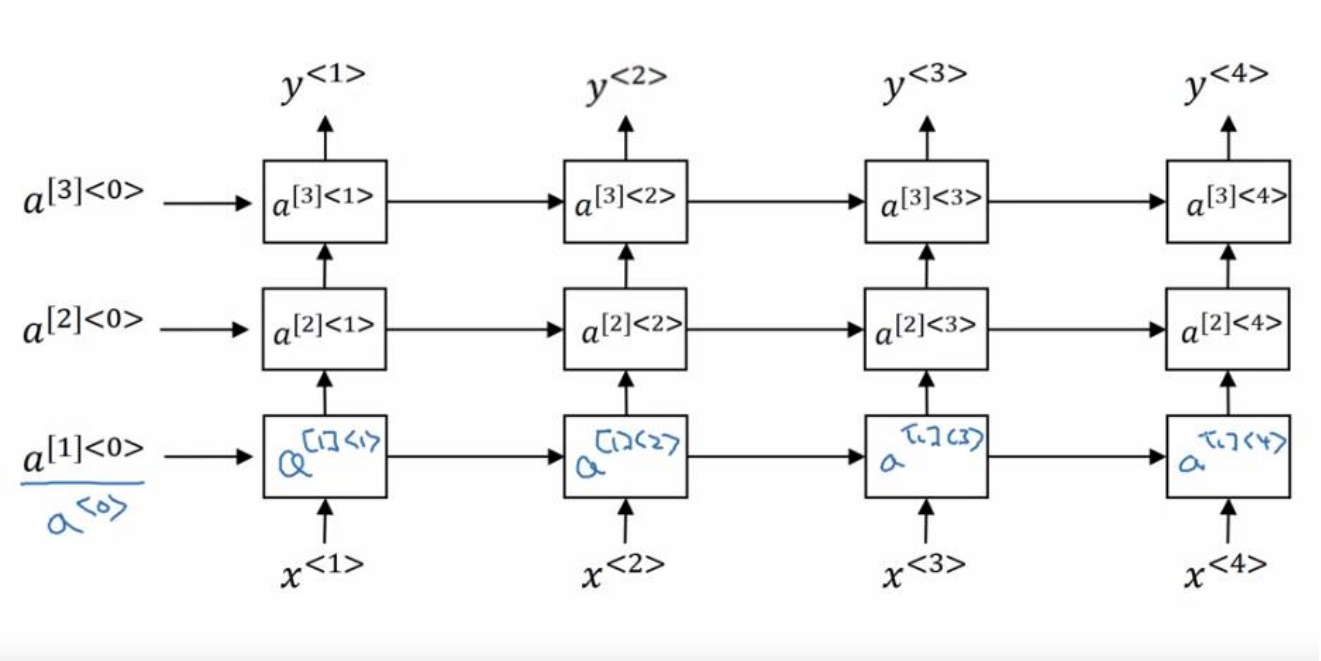


Рисунок 1.12 – Глибока RNN [1]

Таким чином, процес forward propagation виявляється досить схожий на те, що ми бачили в одношарових RNN, однак процес backpropagation виявляється набагато складніше. Матриці ваг залишаються однаковими в межах одного шару, але відрізняються в різних шарах.

1.12 Різні архітектури послідовність-к-послідовності

Дана архітектура використовується в безлічі областей NLP від машинного перекладу до розпізнавання мови.

Візьмемо для початку випадок машинного перекладу, коли на вході і на виході ми маємо послідовності, але не переводимо кожне слово окремо - довжина послідовностей може відрізнятися.

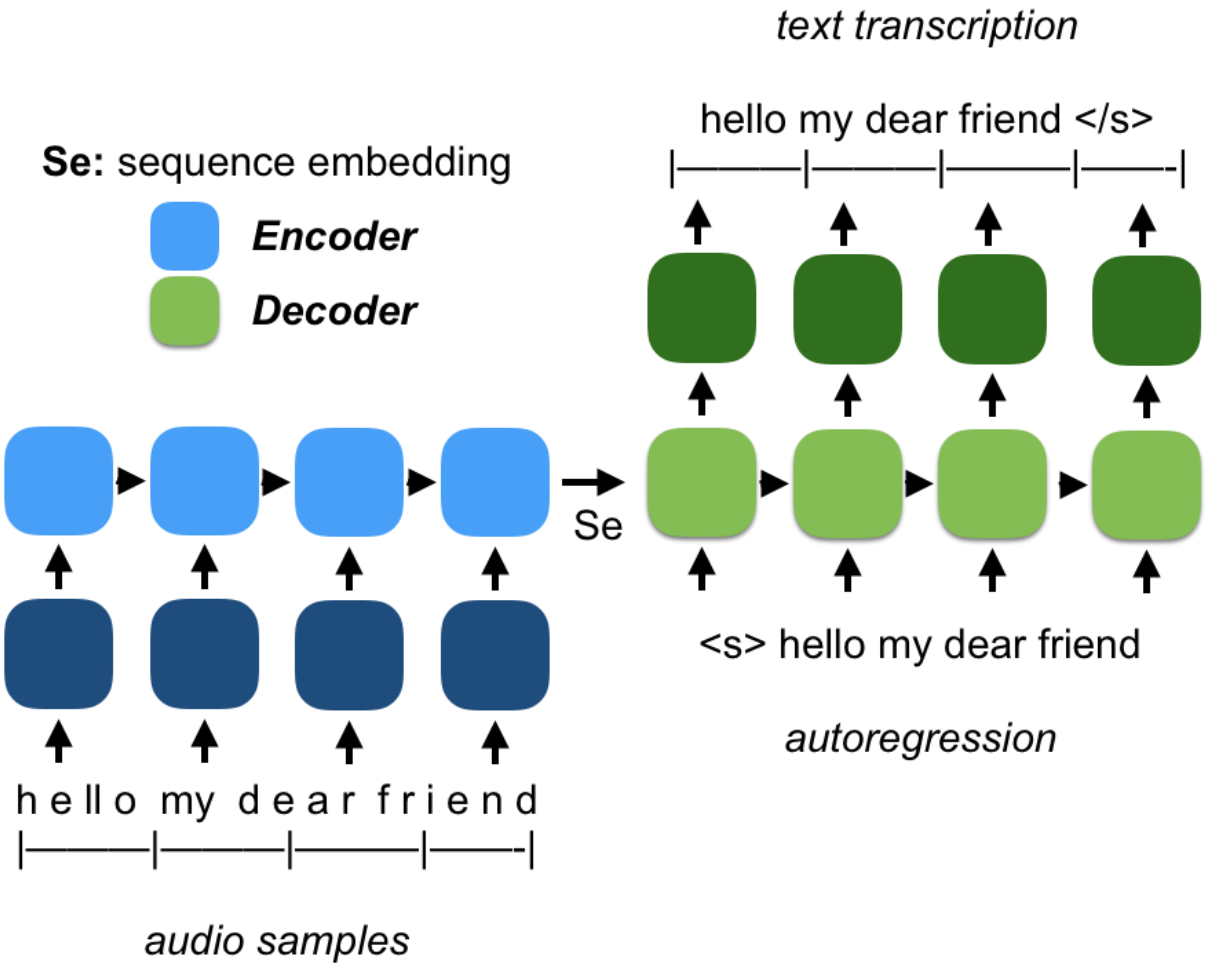


Рисунок 1.13 - sequence-to-sequence RNN

Базова модель, яка дійсно працює для такого випадку, діє таким чином. Модель ділитися на дві умовні частини - кодировщик і декодировщик (рисунок 1.13). Завдання кодувальника - считати першу послідовність і перевести її в вектор інформації, завдання декодувальник - розгорнути цей вектор в вихідну послідовність [9][10].

При цьому з кодувальника немає виходу (y_pred), а є тільки вектор збереження інформації a, а декодувальнику на вхід x подається попередня передбачене слово.

Однак при машинному перекладі ми хочемо отримати на виході не просто послідовність, а послідовність, з найбільшою ймовірністю зустрічається в натуральної мові. Для цього потрібно згадати моделі мови, які були описані в розділі 1.6 Генерація мови и послідовності. Вони дозволяли визначити

ймовірність слів, що йдуть один за одним. Однак в даному випадку нам необхідно вибрати речення, існування якого в мові має найбільшу ймовірність.

Частина схожа на модель мови, підставляється в частину декодувальник. Таким чином, в модель замість нульового вектора a подається зашифроване кодувальником слово, а на вхід в кожну клітину подається попереднє передбачене слово, як в моделі мови.

Однак якщо ми візьмемо кілька пропозицій отриманих з одного перекладу, наприклад «Anna is rarely seen at the university.» з перекладами «Анну рідко бачать в університеті» і «Анна з'являється рідко на університеті.». Перший переклад є правільним і швидше зустрінеться в мові, проте якщо ми будемо вибирати слова по одному, то словосочітаніе «Анну рідко» буде менш імовірним, ніж «Анна з'являється».

Нам необхідно використовувати алгоритм, який зможе максимізувати

$$P(y^{<1>}, y^{<2>}, \dots y^{<Ty>} \mid x) \qquad (1.14)$$

Алгоритм, який підбирає кожне слово виходячи з попереднього, підбираючи максимальну ймовірність, називається greedy search, і він не підходить в даному випадку. Існує інший алгоритм, який називається beam search.

1.13 Beam search

Мета використання алгоритму полягає в тому, щоб використовуючи моделі для роботи з поверненням тексту (генерація, переклад, розпізнавання тексту з аудіо або картинок) повертати не просто пропозиція, але найбільш ймовірне речення, яке можна зустріти в мові. Beam search - найбільш широко використовуваний алгоритм для цього.

Для речення «Anna is rarely seen at the university», beam search в першу чергу спробує припустити перше слово перекладу (як і greedy search). Однак у

beam search є ще один додатковий гіперпараметр B, який відповідає за кількість слів, тестованих на кожному кроці. Умовно візьмемо B рівному 3.

Тоді на першому кроці алгоритм вибере три слова з найбільшою ймовірністю («Анна», «Анну», «Рідко»). Потім для кожного з цих слів алгоритм продовжить окремий «шлях», даючи його на вхід наступної клітці і намагаючись передбачити друге слово. В кінці другого кроку алгоритм з ймовірності вже цих дев'яти варіантів вибере три з найбільшими можливостями і продовжить тільки ці три шляхи (наприклад, «Анна рідко», «Анну рідко» і «Анну бачать»). Так буде повторюватися на кожному кроці. Кожен крок ми досліджуємо B копії моделі.

Beam search може мати поліпшення в швидкості і складності. Наприклад, якщо подивитися на формулу ймовірностей, яку реалізує beam search

$$\arg\max \prod_{t=1}^{T_y} P(y^{<t>} \mid x, y^{<1>}, y^{<2>} \ldots y^{<t-1>}), \qquad (1.15)$$

так як це твір ймовірностей, кожна з яких менше або дорівнює одиниці, робота буде вестися з малими числами, іноді дуже і дуже малими [16].

Такі обчислення можуть стати занадто маленькими для розрахунку комп'ютером. Замість максимізації твори прямий ймовірності, максимізують суму натуральних логарифмів ймовірностей [19].

$$\arg\max \sum_{t=1}^{T_y} \ln(P(y^{<t>} \mid x, y^{<1>}, y^{<2>} \ldots y^{<t-1>})) \qquad (1.16)$$

Побічний негативний ефект обох формул в тому, що вони схильні вибирати більш короткі речення як більш ймовірні. При множенні ймовірностей, кожна з яких менше одиниці, загальна ймовірність стає менше (найчастіше значно менше), і довге речення стає неконкурентним. При использовани суми логарифмів відбувається схожа ситуація - як видно на рисунку 1.14, натуральний логарифм менш одиниці є негативним числом, чим більше вони додаються, тим менше шансів у пропозиції «виграти гонку».

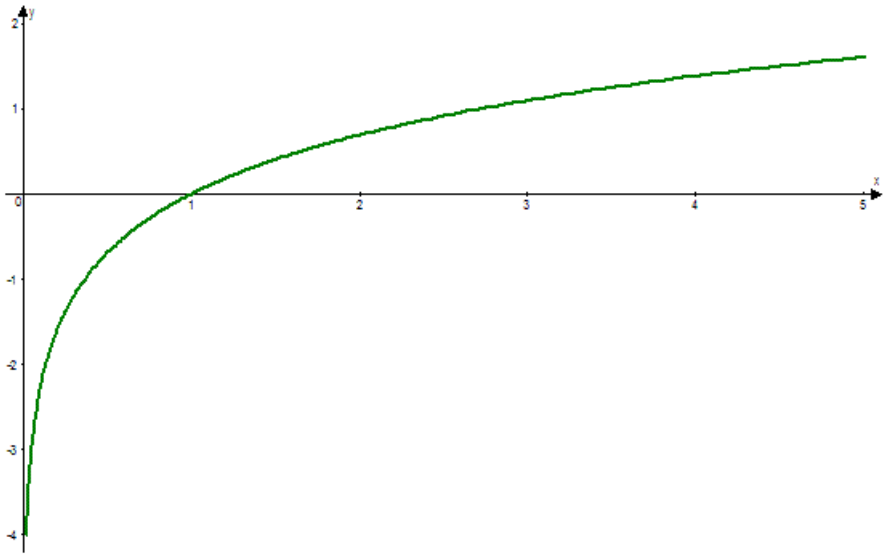


Рисунок 1.14 – Графік натурального логаріфму

Ми можемо нормалізувати суму за номером слів в перекладі:

$$\arg\max (1/T_y) \sum(t=1,T_y) \ln(P(y^{<t>} \mid x, y^{<1>}, y^{<2>} \ldots y^{<t-1>})) \qquad (1.17)$$

Чим більше число B, тим більша кількість варіантів ми розглядаємо, відповідно, отримуємо кращий результат, але і дорожче обчислення, і більше пам'яті потрібно. Для реальних програм часто зустрічаються B близько 10, і не доходять до ста, але в дослідженнях цифри можуть досягати і 1000 і 3000, щоб отримати всі кращі варіанти.

Beam search працює набагато швидше BFS і DFS, але при цьому не гарантує оптимального рішення.

Коли навчена модель дає невірний або недостатньо хороший результат, є додаткові методи, щоб дізнатися, гле ховається велика проблема з алгоритмом - в beam search або в RNN. Саме по собі збільшення B, як і збільшення кількості тренувальної інформації, не може зробити гірше, але так само саме по собі це збільшення не виправить проблем, якщо помилка прихована в RNN.

Для цього є error analysis, за допомогою якого можна визначити, що є ключовою проблемою.

Наша RNN модель відповідає за обчислення P для окремо взятого речення. Клітка, яка видає окремо взяте слово на вихід, робити це виходячи з навченого алгоритму-ваг, які вказують ймовірність того, що одне слово йде за іншим. З іншого боку у нас beam search, якої може не знаходити найкращі рішення через вузькість гіперпараметра B.

У нас є речення «Valeri, poor boy, had a conflict with a teacher.» Є речення, яке перевіла би людина, і яке ми вважатимем потрібним рівнем - у * - «Валері, бідолаха, посварився з учителем.» Є речення, яке перевела модель – «Бідний хлопець Валері погано говорив з учителем.» Перевіримо на моделі P ці речення, назвемо їх P * для речення, перекладеного людиною, і Pm для речення, перекладеного машиною.

Якщо P *> Pm, значить, слід підвищити B. Якщо P * <= Pm, значить модель вважає ймовірність неправильно або не потрібним нам чином.

1.14 Attention model

До цього ми використовували архітектуру з кодувальником-декодувальник для роботи з архітектурою послідовність до послідовності. Під час роботи такої моделі вихідний текст повністю зчитується, кодується в вектор і потім декодується в результуюче речення.

Однак, в такому підході є значний мінус - машина працює набагато гірше з довгими реченнями. Їх просто складніше закодувати одним вектором в один момент, а потім отримати назад всю інформацію. Візьмемо, наприклад, речення з роману Толстого «Два гусари», наведене для стислості на рисунку 1.15. Людина не буде читати це речення повністю, а потім повністю його перекладати. Він буде переводить його по частинах, і машині також не так то просто перевести всі пропозицію в інформаційний вектор (який однакового розміру для будь-якого обсягу тексту) і потім дістати звідти інформацію про кожне слово.

В 1800-х годах, в те времена, когда не было еще ни железных, ни шоссейных дорог, ни газового, ни стеаринового света, ни пружинных низких диванов, ни мебели без лаку, ни разочарованных юношей со стеклышками, ни либеральных философов-женщин, ни милых дам-камелий, которых так много развелось в наше время, - в те наивные времена, когда из Москвы, выезжая в Петербург в повозке или карете, брали с собой целую кухню домашнего приготовления, ехали восемь суток по мягкой, пыльной или грязной дороге и верили в пожарские котлеты, в валдайские колокольчики и бублики, - когда в длинные осенние вечера нагорали сальные свечи, освещая семейные кружки из двадцати и тридцати человек, на балах в канделябры вставлялись восковые и спермацетовые свечи, когда мебель ставили симметрично, когда наши отцы были еще молоды не одним отсутствием морщин и седых волос, а стрелялись за женщин и из другого угла комнаты бросались поднимать нечаянно и не нечаянно уроненные платочки, наши матери носили коротенькие талии и огромные рукава и решали семейные дела выниманием билетиков, когда прелестные дамы-камелии прятались от дневного света, - в наивные времена масонских лож, мартинистов, тугендбунда, во времена Милорадовичей, Давыдовых, Пушкиных, - в губернском городе К. был съезд помещиков, и кончались дворянские выборы.

Рисунок 1.15 - Найдовше речення в романі Толстого

Attention model була вперше запропонована Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio в 2014 в роботі «Neural Machine Translation by Jointly Learning to Align and Translate» [11].

У нас є речення, і ми використовуємо двосторонню RNN для кодування цієї речення (в кожній клітині у нас з'являється два ветора a_forward і a_backward, що зберігають інформацію про поточну і попередніх клтках в обидві сторони).

Інша RNN використовується для того, щоб отримати переклад. Я буду використовувати умовну s для ідентифікації стану клітин в генеруючої RNN, це буде аналогічно a в моделі. Друга RNN використовує s як вектор стану для генерації речення, при цьому кількість клітин може відрізнятися від кількості клітин в вихідному реченні, в першій моделі, як у класичній моделі sequence-to-sequence. Клітини тривають, поки не буде згенеровано <EOT>.

Сама attention model підраховує так звані ваги уваги, матриця, яка йде від кожної з клітин першої моделі до кожної з клітин другої моделі. Ця інформація відповідає за те, скільки уваги ми приділимо інформації з кожної клітини при

генерації конкретного слова. Тобто при генерації друга модель має одночасно інформацією про все речення відразу, але з різним ступенем уваги.

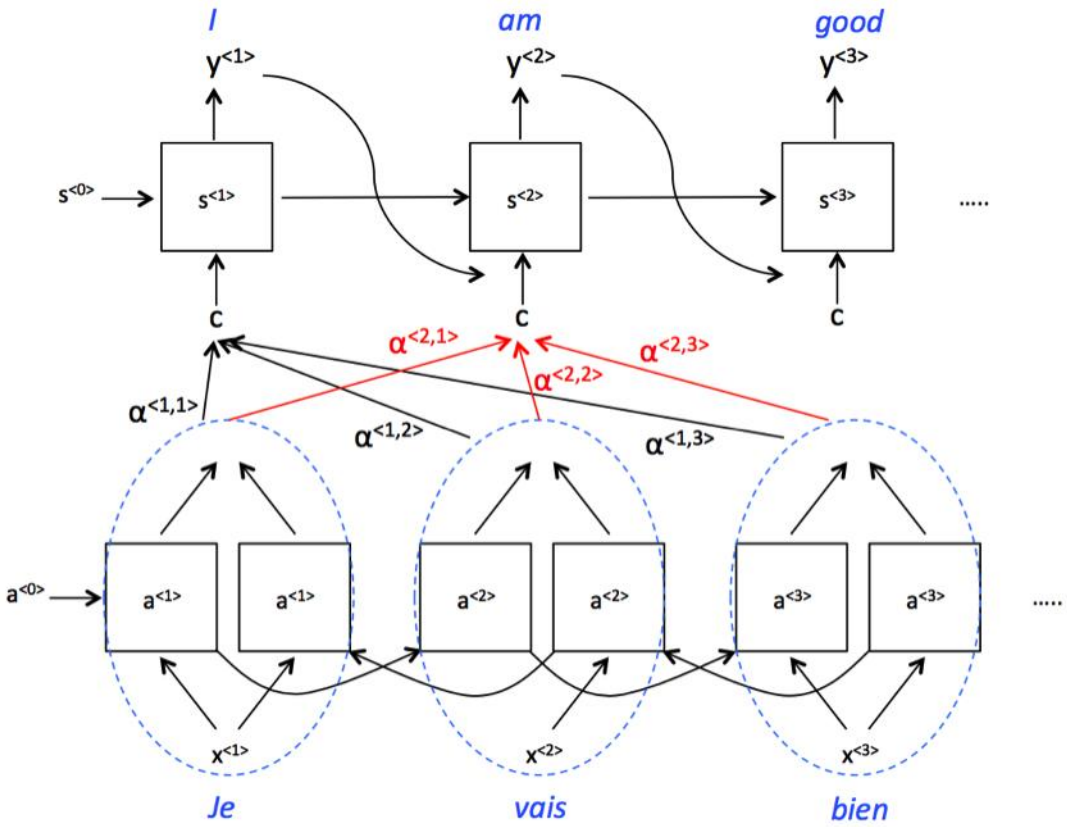


Рисунок 1.16 – Attention model

Стан попередньої клітини s - аналогічно звичайній RNN - вектор, що несе інформацію про попередні станах і входящий вектор.

Контекст - це зважена сума виходять з клітин векторів, з урахуванням alpha кожної клітини. Ваги уваги (alpha) будуть задовольняти двом умовам: кожен буде не менше нуля, і в сумі вони будуть давати одиницю.

$$c^{<t>} = \sum \text{alpha}^{<1,t'>} * a^{<t'>} \qquad (1.18)$$

Alpha - кількість «уваги» в яке буде прийнята інформації конкретної клітини з першої моделі при роботі конкретної клітини другої моделі. Строго

кажучи, скільки уваги повинен надати у <t> а <t '>, де t - момент клітини другий, генеруючої, моделі, а t' - момент клітини першої, що зчитує, моделі.

$$a^{<t,t'>} = \exp(e^{<t,t'>}) / \sum \exp(e^{<t,t'>}) \text{ (for all t')} \qquad (1.19)$$

Ми підраховуємо е і використовуємо softmax щоб отримати умовну частину від всіх альфа. Для підрахунку е ми використовуємо, в свою чергу, невелику нейронну мережу. В якості входу вона приймає s <t-1> (параметр до попередньої контрольної клітини в другій, що генерує, моделі) і а <t '>.

2 АНАЛІЗ ВИДІВ ВІДОБРАЖЕННЯ СЛОВА

2.1 Відображення слова

Word embeddings - перетворення слів або фраз в зрозумілий для комп'ютера вид (зазвичай у вигляді векторів або дійсних чисел) кодуванням за допомогою виділених властивостей.

Це одна з ключових ідей в області NLP (Natural Language Processing), яка безпосередньо впливає на навчання моделі і результат. Головна мета word embedding, крім представлення слів в доступному для машини вигляді, це відображення сенсу слова в вигляді, який передає найбільшу кількість і фонетичної, і синтаксичної, і морфологічної інформації про слово для аналізу. За допомогою правильного word embedding модель вчиться розуміти також аналогії, такі як «чоловік до жінки так само як принц до принцеси».

У попередніх розділах я представляла слова за допомогою one-hot encoding використовуючи словник використовуваних слів. Тобто якщо у нас є словник, що складається з 10 000 слів, закодоване слово «людина», що є 8673 в списку, буде являти собою вектор з 10 000 нулів з одиницею в 8673 позиції. В нотації ми використовуємо для відображення такого слова велику букву О з номером внизу: O_{8673}.

Одна з слабкостей такого кодування - то що воно кодує кожне слово повністю окремо, і не несе ніякої інформації про слово, крім його ідентифікації. Тобто слова «король» і «королівський» будуть так само далеко один від одного, як і «король» і «ступінчастий». Крім початкового ускладнення завдання, це так само не дозволяє робити feature sharing - іспоьзовать навчені зв'язку для подібних слів.

Наприклад, кожному дієслову в словнику потрібно вчити залежність роду від іменника окремо. А для моделі, яка повинна визначати «характер» слів, наприклад, для виставлення настрою тексту, однокореневі слова не будуть позначати схожі речі, досить замінити одну букву в слові і навчання для нього буде проводитися повністю окремо.

Інший приклад - для будь-якої моделі, яка повинна передбачати, чи переводити, або генерувати текст: припустимо, моделі необхідно спрогнозувати слово, яке буде йти після «Ммм, який смачний цей апельсиновий». Для людського мозку очевидно, що найбільш ймовірне продовження речення - слово «сік». Припустимо, модель вже навчена і вставить слово сік. Але коли модель зустріне подібну пропозицію «Ммм, який смачний цей яблучний», вона не отримає схожою інформації, якщо не буде навчена на подібних прикладах. Так як для моделі слова «яблучний» і «апельсиновий» не мають загального (у кожного свої ваги, які навчаються окремо).

Чисельно це відбувається тому, що твір будь-яких двох one-hot encoded векторів буде нулем. Евклідова відстань між векторами також однаково, тому моделі невідомо, коли слова схожі одне на одного.

Інший варіант репрезентації - feature репрезентація слів. Наприклад, штучне виділення властивостей, як на рисунку 3.1.

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮ size cost						

Рисунок 3.1 – feature representation – word embedding

Таким чином, ми можемо виділити, наприклад, 3000 властивостей і закодувати кожне слово з їх допомогою. Якщо ми використовуємо таке кодування для репрезентації слів, слова «апельсиновий» і «яблучний» тепер

досить близькі один до одного. Деякі властивостей будуть різні (наприклад, колір і смак), але більша частина матимуть схожі значення.

Word embedding дозволяє виділяти подібні властивості, які допомагають краще відобразити слово в вектор, ніж при one-hot encoding. Такі фичи складно розпізнати і інтерпретувати людському мозку - що вони насправді означають - але вони дозволяють моделі зрозуміти, наскільки два слова є схожими один на одного [22].

Одна з популярних речей, після отримання, наприклад, векторів слів в 300 вимірах, згорнути їх в двовимірне ізобраденіе і відобразити у вигляді карти. (Рисунок 2.2).

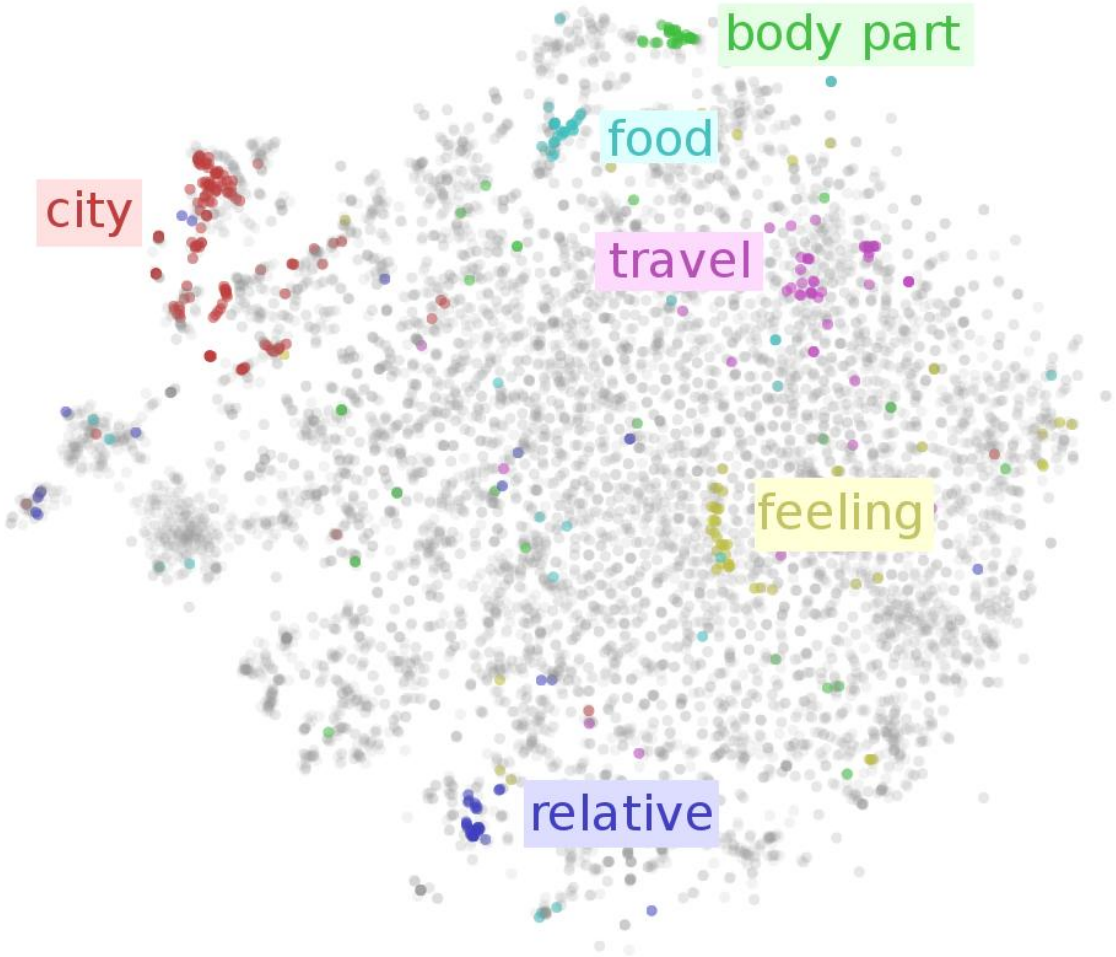


Рисунок 2.2 – word embeddings

Ми маємо можливість зараз отримати доступ до величезної кількості нерозміченого тексту в інтернеті, і якщо навчити модель для кодування слів, якщо закодувати слова на основі всього цього тексту, то закодоване слово буде видавати схожість слів за різними ознаками. Так, наприклад, навіть якщо вже модель для задачі зустріне в обробці тексту фрукт, який вона не зустрічала раніше, але embedding цього слова - тобто те, як воно закодовано - буде говорити про те, що це фрукт і який це фрукт, то отримані знання про взаімодейтсвіе світу з іншими фруктами будуть поширюватися на цей, новий для моделі, об'єкт.

Це дозволяє мережі передавати знання і вчитися ефективніше. Весь процес укладає в кроках:

- вивчити word embeddings з великими текстовими документами (1-100 мільярдів слів) (або завантажити пре-навчену модель онлайн);

- перенести embedding на більш маленьку задачу (100к слів, наприклад).

Побічний позитивний ефект використання word embeddings - то, що ми суттєво скорочуємо вектор слова (з 10 000 до 300 в прикладах вище), що у багато разів знижує витрати розрахунків[1].

Word embeddings показують тим більше користі, чим на меншій базі інформації навчається ваша модель (різниця в інформації з моделі і допоміжної інформації з word embeddings особливо велика).

Одна з найбільш цікавих сторін word embeddings в тому, що вони також можуть допомогти в розпізнаванні аналогій («король до королеви як хлопець до дівчини»). Хоча пошук і генерація аналогій не є центральною проблемою NLP, це досить складна проблема для машини, рішення якої має суттєвий побічний вплив (позитивне) на рішення інших завдань і добре вирішується за допомогою word embeddings [21].

Коли ми ставимо перед машиною завдання «як король до королеви, так хлопець до?», Що може зробити машина, так це побачити різницю між парами векторів (відняти вектор «король» з вектора «королева», наприклад) і знайти вектор, який так ж відноситься до вектору «хлопець». У примітивній моделі, де у нас є всього три виміри (стать, монарх, вік). Умовно закодувавши короля і

королеву, ми можемо сказати, що вони діаметрально протилежні по підлозі (-1 і 1, припустимо), і хоча обидва по вимірюванню «монарх» матимуть максимум, при відніманні векторів за цим виміром у них не буде виявлено різниці. Все, що залишається зробити моделі - знайти, з яким вектором вектор «хлопець» буде давати схожий результат. Що змогла «зловити» ця модель, так це те, що головна відмінність між королем і королевою - це пів (тільки один вимір).

З використанням даного методу аналогічно модель буде працювати з будь-якими відносинами усередині цих пар («король до хлопця як королева до?», «Хлопець до дівчини як король до?» тощо).

Візуалізація відносин в прикладі вище представлена на рисунку 3.3 і формулою 1. Ця довжина на зображенні відображає різницю в статі.

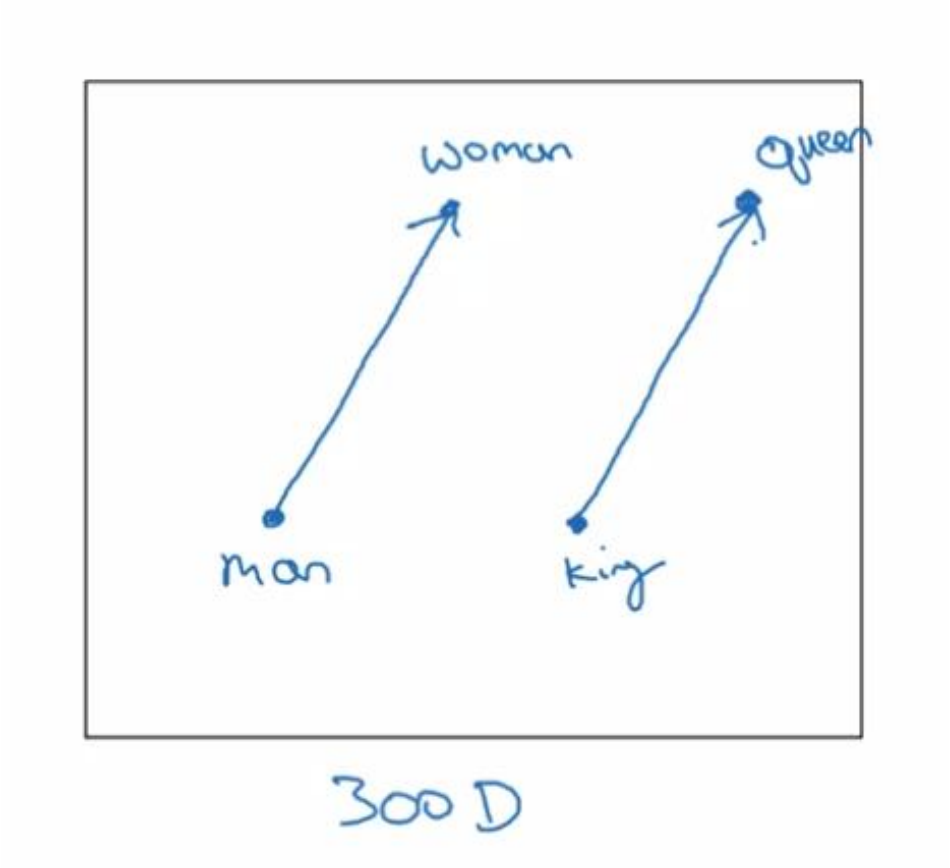


Рисунок 2.3 - візуалізація відносин в word embeddings

$$e_{man} - e_{woman} \approx e_{king} - e_{?} \quad (2.1)$$

Залежно від завдання дослідження, в різних роботах зустрічається від 30% до 75% точності в аналогіях, використовуючи word embeddings [1]. Коли вживання аналогії зараховується як коректне, тільки якщо модель вибрала конкретне слово точно.

t-SAE - алгоритм, який використовується для згортання многоверного простору в двовимірне і отображенія карти word embeddings в доступному людині вигляді. Це дуже сложноий і нелінійний перенос. Насправді, в двовимірному просторі ми не можемо розраховувати на репрезентацію відносин слів відстанню між точками, і всі розрахунки модель робить в багатовимірному просторі.

Найбільш часто використовується функція схожості для векторів, яка називається cosine similarity. При загальній функції схожості, описаної як:

$$sim(e_w, e_{king} - e_{man} + e_{woman}), \tag{2.2}$$

функція cosine similarity:

$$sim(a,b) = (a^T * b)/(||a|| * ||b||),$$

де T - транспонований вектор,

перший знак * - dot product двох векторів.

Формула називається cosine similarity тому, що це, по суті, косинус кута між двома векторами. Він приймає значення від 1 до -1, що досить прозоро відображає суть схожості векторів.

Также в качестве формулы схожести (similarity) можно использовать квадрат расстояния,

$$||u-v||^2 \tag{2.3}$$

Строго кажучи, це буде вимірювати відмінність векторів, ніж їх схожість.

Перейдем к формализации задачи хорошего word embedding. При решении задачи вы вычисляете так называемую word embedding matrix

(Рисунок 3.4). При подальшому збільшенні one-hot encoded вектора, що ідентифікує слово, матриця буде перетворювати його в вектор зважених властивостей.

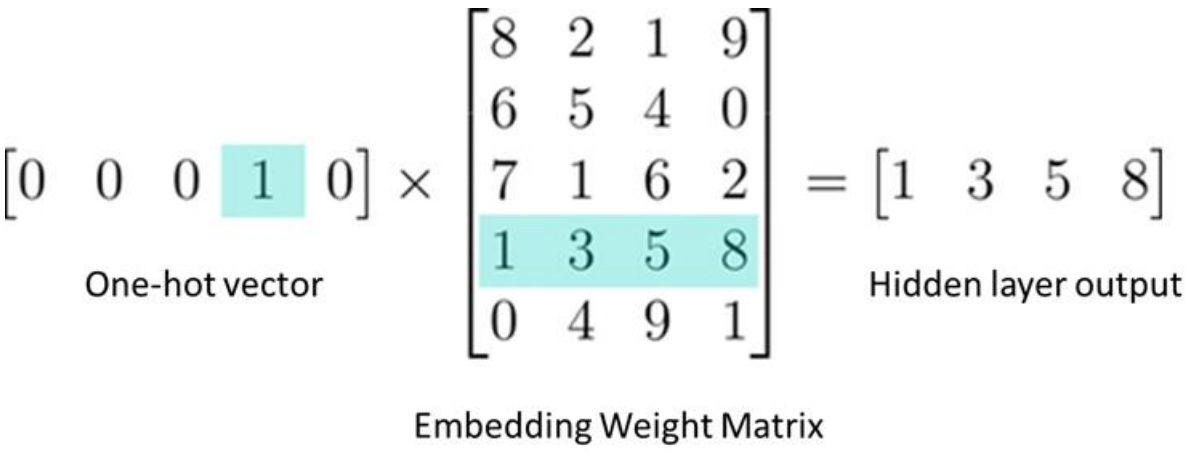


Рисунок 2.4 – матрица embedded весов

Припустимо, наш словник складається з 10 000 слів (включаючи незнайоме слово UNK і EOS). Тоді нам потрібно визначити embedded weight матрицю E, яка матиме відповідну розмірність (300, 10 000), де 300 - кількість властивостей, а 10 000, відповідно, кількість слів у словнику.

Для переведення слова з one-hot encoded кодування в feature embedded кодування, ми множимо one-hot вектор на отриману матрицю E, як показано на рисунку 3.4.

Коли ми використовуємо це в навчанні, фактично множити one-hot вектор на матрицю E не дуже ефективно, тому що one-hot вектора мають дуже велику розмірність, а більша частина їх елементів нулі. Ефективніше просто знаходити в матриці ряд, що відповідає одиниці в векторі і копіювати його.

Отже, наше завдання визначити матрицю E, і для цього ми будемо рандомно ініціалізувати її і використовувати градієнтний спуск як і для звичайних нейронних мереж.

2.2 Визначення матриці word embeddings

У цьому розділі я почну описувати алгоритми для вивчення word embeddings. В історії вивчення word embeddings люди часто іпользовать досить складні алгоритми, проте з плином часу приходили до більш і більш простим варіантів, які все ще показували хороші результати, особливо на великих даних. Ми також почнемо з трохи більш складним алгоритмів. З них легше зрозуміти, як і чому вони працюють, а після ми перейдемо до більш простим, які вже не будуть здаватися магією.

Наступний метод був описаний в роботі Yoshua Bengio, R'ejean Ducharme і Pascal Vincent «A Neural Probabilistic Language Model» [4] .Допустім ми будуємо модель мови за допомогою нейронної мережі. Ми хочемо, щоб модель передбачала слово в реченні. Наприклад «Кожен день діти вмиваються, чистять зуби, ходять в _».

Уявімо таку архітектуру моделі: ми кодируем відомі слова в реченні за допомогою one-hot encoding в нашому словнику, потім використовуючи матрицю E, перекодіруя one-hot вектора в embedded вектора, потім подаємо їх на вхід нейронної мережі.

Що найчастіше роблять, не використовують всі слова в реченні, а мають фіксоване так зване fixed historical window - фіксоване історичне вікно, яке визначає кількість слів, на яких базується наше передбачення. Це є гіперпараметром - параметром нейронної мережі, яка не навчається в процесі (не змінюється самої нейронною мережею) і довільно вибирається програмістом.

Припустимо, виберемо історичне вікно в чотири слова - в нашому випадку це будуть «чистять зуби, ходять в». Таким чином, нейронна мережа буде мати на вході 4 * 300 (кількість властивостей одного слова) нейронів. У них будуть власні ваги обробки, потім нейронна мережа передасть їх в softmax

шар, який також має свої ваги, і видасть ймовірності для подання слів після цих чотирьох.

Матриця E в даному випадку також є параметрами мережі, які можуть навчатися як і ваги шарів - звичайно, при досить великій кількості даних. При цьому мережа сама навчиться, що апельсин і яблуко під многх ситуаціях взаємозамінні, а, значить, їх feature-вектора в матриці будуть схожі.

Матриця E навчається класичним методом backpropagation + gradient descent, максимізуючи ймовірність тренувальних даних для передбачення.

Це один з перших і досить успішних алгоритмів в області word embeddings.

Розглянемо цей алгоритм в перспективі створення більш простих алгоритмів. Дослідники дали мережі завдання передбачити слово - target - і також контекст для вирішення завдання - чотири слова перед target.

У програмуванні різних методів для навчання матриці E, дослідники використовували різні контексти - для завдання стоставленія моделі мови вгадування слів цілком логічно брати слова виключно прямо перед target, змінюючи тільки їх кількість як historical window, але для інших завдань ми можемо використовувати більше контексту.

2.3Word2Vec

Word2Vec це більш простий і ефективний спосіб навчання матриці E. Дана модель була описана в роботі Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean «Efficient Estimation of Word Representations in Vector Space» [5].

Так як ми ставимо за мету не прогноз слів (вони відомі) а створення матриці E, то ми вільні вибирати будь-які слова для контексту. Замість того, щоб взяти кілька слів перед target word, візьмемо випадкове слово з речення. Потім виберемо рандомнє слово в межах певного «вікна» (наприклад, плюс-мінус три слова від слова контексту) - це буде наш target.

Навчити вірогідну модель в цьому випадку - наскільки ймовірним є отримати слово у вікні плюс-мінус три, п'ять, десять слів виходячи з слова

контексту - дуже непросте завдання для навчання. Але так як це і не є нашою метою - а наша мета зібрати якомога більшу інформацію для матриці E, це працює в нашу користь.

Отже, нашим завданням є з'ясування target слова t з контексту c. Припустимо, що ми беремо слово для контексту, яке в нашому словнику з 10 000 слів стоїть під номером 2345. Target, відповідно, буде знаходитися під номером 8888. На початку у нас є рандомно заповнена матриця E, ми пропускаємо через неї вектор 2345, і отримуємо embedded-закодований вектор, відповідний слову 2345, який поки не несе ніякої осмисленої інформації в собі.

Потім ми віддаємо цей вектор в softmax юніт. Софтмакс видає нам ймовірність отримання таргета, маючи контекст:

$$p(t|c) = (e^{\theta tT * ec})/(\sum e^{\theta jT * ec}) \quad (2.4)$$

Мы используем функцию потерь для softmax:

$$L(y\_pred,y) = - \sum y_i * \ln(y\_pred_i) \quad (2.5)$$

де y_pred - це вектор ймовірностей для кожного слова, вектор 10 000 довжини,

y - one-hot encoded target вектор.

Якщо нам вдасться оптимізувати функцію втрат щодо як параметрів матриці E, так і параметрів softmax шару, то ми отримаємо дуже хорошу базу embedded векторів.

Ця модель називається scip-gram модель, тому що між контекстом і словом, яке модель намагається передбачити, відбувається рандомний «стрибок», «пропуск».

Цей алгоритм також має мінуси: перша - це швидкість обчислень. Кожен раз в softmax слої вам необхідно обчислювати суму ймовірностей для всіх слів. Це стає справжньою проблемою, якщо ваш словник в рази більше 10 000 слів [23].

Одне з рішень для цієї проблеми - ієрархічний софтмах класифікатор. Ієрархічний софтмакс використовує бінарне дерево, і має складність log (v) замість лінійної складності. На кожному кроці він дає зрозуміти, чи знаходиться ймовірність слова в більшій чи меншій половині розподілу.

Як вибрати слово для контексту c? Коли ми виберемо слово для контексту, ми вибираємо target в межах «вікна», однак щоб вибрати сам контекст буде потрібно застосувати особливу техніку. Якщо вибирати слова повністю рандомно з текстової бази, то слова проде і, а, в, я і подібні будуть потрапляти особливо часто, тоді як ці слова не є особливо корисними - спецефісческіе слова на кшталт «дуріан» можуть зовсім не попастися в вибірці. На практиці алгоритми використовують балансування часто зустрічаються і рідко зустрічаються слів, щоб уникнути цієї проблеми.

2.4 Negative sampling

Negative sampling - інший спосіб навчання модель, на противагу звичайному softmax. В першу чергу, він покликаний вирішувати проблему великих витрат обчислювальних витрат, пов'язаних з використанням softmax обчислення 10 000 і більше вірогідності (при такій кількості слів в словнику) за одну ітерацію.

Тобто negative sampling, строго кажучи, є не іншим підходом до навчання матриці E, а іншим завданням, що використовується для такого ж ефективного навчання, але значно менш витратною.

Завдання наступна, маючи пару слів - контекст і інше слово, визначити, явлется це слово target чи ні. Для прикладу візьмемо слова «синє» та «небо». Ми будемо використовувати це як позитивний приклад (y = 1). Для кожного такого позитивного прикладу ми знайдемо k негативних прикладів. Це робиться за рахунок рандомного вибору пар слів до слова-контексту.

Припустимо k ми вибрали рівним 5, а обрані слова – «привіт», «капелюх», «керівник», «вчора» і «темне». Незважаючи на те, що слово «темне» цілком

може бути присутнім біля слова небо, це не зашкодить навчанню моделі на великій вибірці.

Далі, ми створюємо завдання supervised learning, де навчаємо модель визначати по слову контексту і target, чи дійсно це слово є target для цього контексту (по суті, близькість двох слів, так як контекст і таргет взаємозамінні).

Для вибору k, в оригінальній статті «Distributed Representations of Words and Phrases and their Compositionality» [6] пропонується вибирати k від 5 до 20 для невеликих датасет, і менший k якщо ви маєте дуже великий датасет.

$$P(y=1|c,t) = \text{sigmoid}(\theta_t^T e_c) \quad (2.6)$$

Замість того, щоб кожен раз тренувати 10 000 умовних бінарних класифікаторів, ми кожен раз тренуємо тільки k + 1.

Negative sampling алгоритм називається так, тому що для кожного позитивного випадку context + target ми штучно створюємо k негативних випадків.

Одна цікава деталь, яка стосується цього алгоритму - як вибирати слова для негативних примірників з усього словника? Один підхід - вибирати слова виходячи з того, наскільки часто вони зустрічаються в тексті (мовою) - емпірична частота. Однак так ми зіткнемося з вищеописаної проблемою - ми будемо мати багато інформації про прийменниках і подібному, і майже ніякої впорядкованої інформації про слово, яке зустрічається один-два рази. Є інша крайність - використовувати слова рівномірно по всьому словнику, проте це також не репрезентативну неефективно. Автори згаданої вище статті пропонують використовувати наступну формулу:

$$P(w_i) = f(w_i)^{3/4} / \sum f(w_j), \quad (2.7)$$

де f (w) - частота зустрічі слова.

Така формула, відповідно до рисунка 2.5, де наведено графік $x^{3/4}$, знаходиться за інтенсивністю між прямою відповідністю частоті слова в тексті і прямою відповідністю словнику.

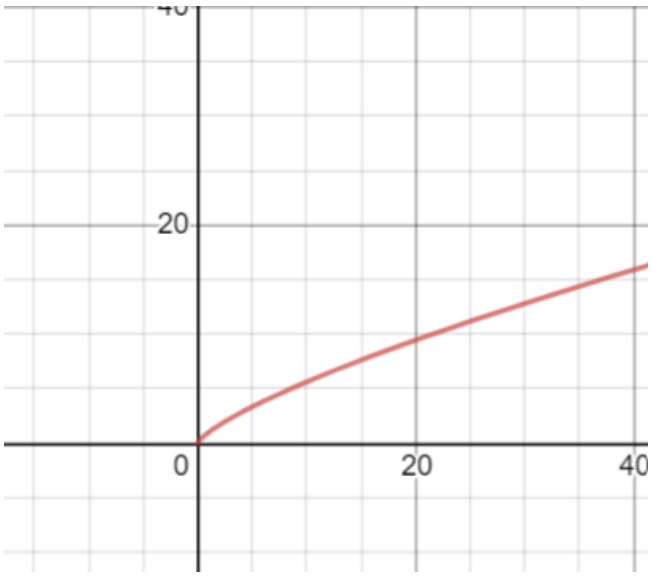


Рисунок 2.5 – графік росту $x^{3/4}$

2.5 Використання word embeddings: класифікація настроїв тексту

Класифікація відгуків - актуальне завдання для виставлення умовного рейтингу відгуку, грунтуючись на його тексті.

Проблема, з якою може зіткнутися розробник моделі для класифікації тексту / відгуків - недостатня кількість тренувальних примірників для навчання (для виявлення достатньої кількості зв'язків, що ідентифікують хороші / погані відгуки). У цьому добре може допомогти техніка word embeddings, так як з її допомогою ми можемо отримати вже пов'язані слова, а за допомогою навчання моделі лише «забарвити» окремі ділянки пов'язаної мережі слів тим чи іншим рівнем емоцій.

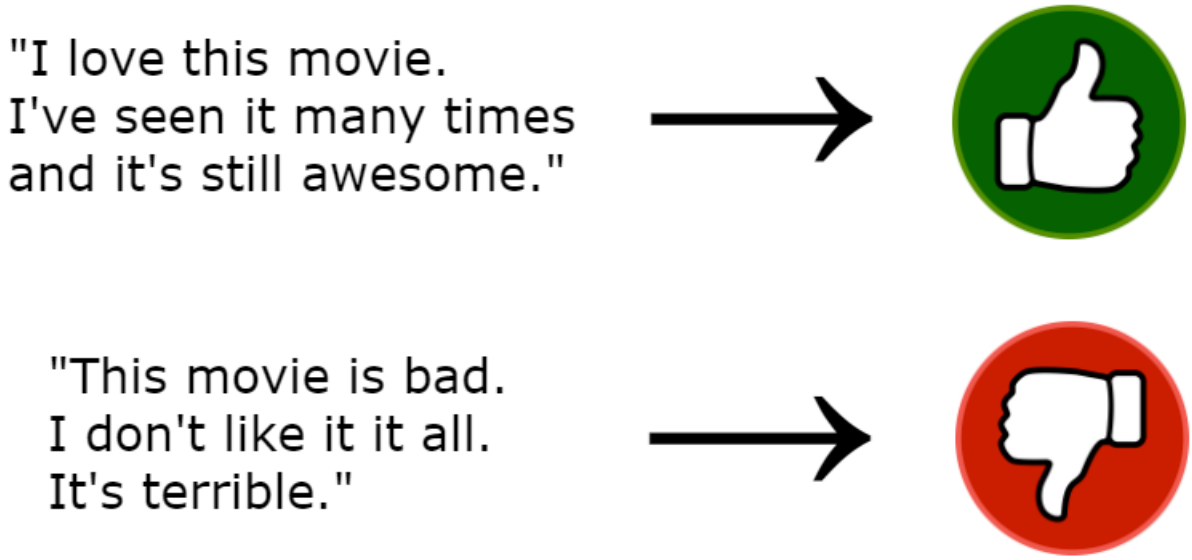


Рисунок 2.6 – класифікація відгуків

Приклад sentiment classification завдання - ми маємо деяку кількість відгуків про ресторан, яким необхідно виставити асоційовану оцінку. «Цей обід був чудовий, ми в захваті!» - 5, «Десерт був непоганий» - 4, «Місце не має нічого спільного з хорошою їжею.» - 1. Речення тут - вхідний вектор x, а класифікація - y.

Більш проста архітектура моделі для вирішення завдання: беремо one-hot encoded слова в відкликання, пропускаємо через матрицю E, яка вчилася на в рази більшому датасеті тексту, і отримуємо embedded вектора - маємо слова з feature-інформацією про них. Потім пропускаємо їх через average шар - просто беремо середнє значення по кожній фічі, і потім останнім шаром є softmax, який і буде прогнозувати класифікацію.

У цій моделі є один серйозний недолік - вона не враховує порядок слів у реченні, що є досить важливим для оцінки рейтингу - супер-позитивне слово стоїть після «не» надає тексту протилежну забарвлення. Наприклад, третій із прикладів вище має словосочітаніе «хороша їжа», проте в дійсності є максимально негативним.

У разі важливості порядку слів використовується більш цікава модель: замість простого average шару, використовується RNN, яка враховує порядок слів, і видає відображення цієї пропозиції використовуючи архітектуру моделі many-to-one (рисунок 2.7).

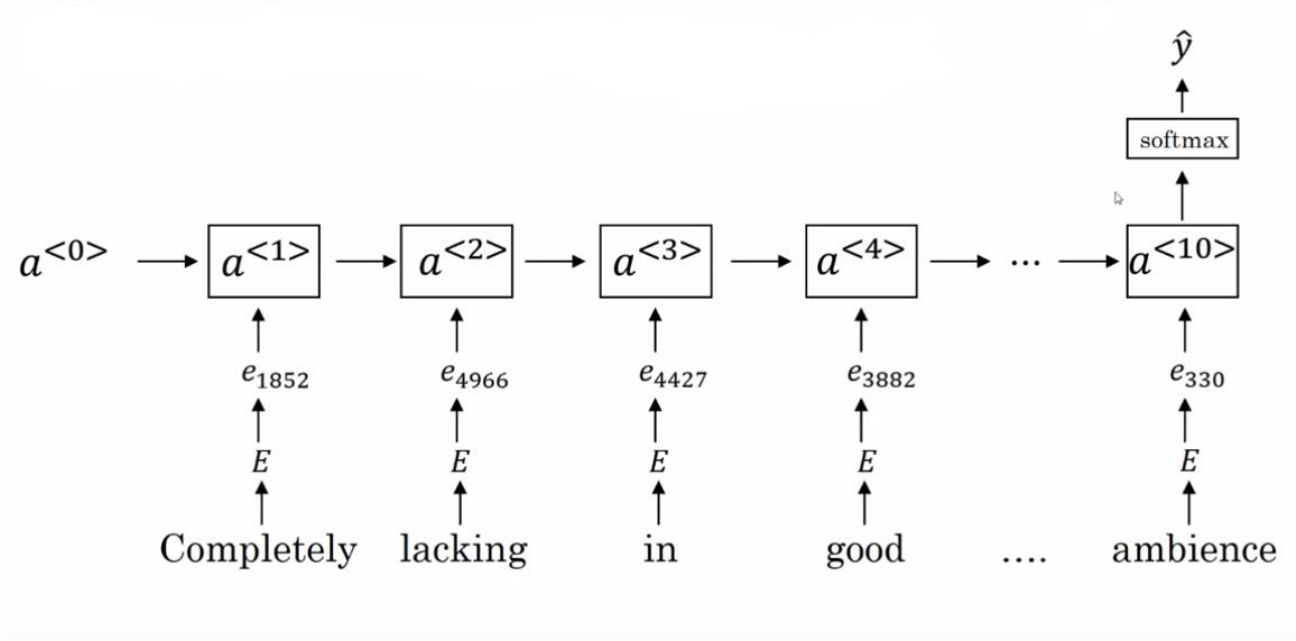


Рисунок 2.7 - класифікація тексту з RNN

2.6 Усунення упереджень в нейронної мережі

Нейронні мережі в сучасному світі приймають важливі і складні рішення, і ми не можемо безпосередньо пояснити, чим конкретно обгрунтовані рішення, які вони приймають, після їх прийняття. Прикладами таких завдань може стати рішення про дозвіл брати кредит, судові рішення або вибір співробітників на роботу. Програміст, абсолютно без бажання нашкодити або направити нейронну мережу, може навчити мережу на вибірці кандидатів певної групи, тоді всі, хто не входять в дану групу, будуть здаватися нейронної мережі «небажаними». Звичайно, це питання стосується не тільки людей, але у випадку з людьми в справу вступає ще й закон, і програмістом і дослідникам доводиться бути особливо обережними з такими випадками.

Ось чому важливо переконатися, що модель, яку ми навчаємо, вільна від будь-яких упереджень, в тому числі ґрунтуючись на расових, гендерних, сексуальних, статевих особливостях людини.

Рисунок 2.8 - Розподіл слів по осі він - вона в пристрасній моделі [8]

Ми розглядали аналогії, які вивчає машина за допомогою word embeddings, як, наприклад «король до королеви як хлопець до?». Але що якщо задати машині питання «хлопець до програмування як дівчина до?»? Таку провокаційну роботу провели дослідники Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, Adam Kalai, яка так і називається «Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings» (Чоловік до програміста як Жінка до домогосподарці ? Усунення упереджень в word embeddings) [8]. Один з авторів роботи побачив, що моделі word embeddings можуть видавати подібний результат, який відображає нездорові сексистські уявлення про світ. Що ще виявили іслледователі - при завданні «Батько до лікаря як мати до?» модель видавала відповідь «медсестра».

При натуральному навчанні, word embeddings можуть мати сексистські, расистські, етнічні, вікові, сексуальні та інші упередження про світ, знайдені в тексті, який використовувався для навчання моделі.

Перший крок - визначити напрямки bias-а. Bias називається необ'єктивність. Для єтого нам треба взяти два слова, які розрізняються лише потенційним напрямком bias-а, і заміряти їх різницю (наприклад «хлопець» і «дівчина» для статі, «китаєць» і «російський» для раси і так далі). Для того, щоб дрібні похибки менше впливали на результат, слід вибрати кілька пар («хлопець» і «дівчина», «мама» і «тато», «бабуся» і «дідусь», «кролик» і «кролиця») і взяти середнє за їх різницею. В папері використовувався більш складний PCA аналіз, однак це передає ідею.

Наступний крок - нормалізація. Для кожного слова, яке не визначає різницю в осі (тобто для якого не властиво безпосередньо відрізнятися по підлозі, в даному випадку) необхідно звести слова до середнього між двома статями по осі bias-у.

Останній крок - скоротити різницю між кожною парою слів, який відрізняє в мові по цій осі, до різниці виключно по цій осі. Наприклад, якщо в текстах, на яких навчалася нейросеть чоловіча стать визначений як сильний, маскулінний, принциповий, а жіночий - як слабкий, сором'язливий і ніжний, то навіть при корекції інших слів, bias удет залишатися в самих словах, пов'язаних з гендером, і при обчисленні аналогій модель все одно буде ділити речі на чоловічі і жіночі згідно з цими уявленнями.

У цій ситуації треба провести таку ж роботу, як з словами, які ми нейтралізували, але за всіма іншими осях, крім основної. Тоді наші пари будуть відрізнятися тільки підлогою, в даному випадку, а в інших - ознакою, за яким ми прибирали bias з моделі.

3 МОРФОЛОГІЯ

Морфологія (від грец. Μορφή - «форма» і λόγος - «слово, вчення») - розділ граматики, основними об'єктами якого є слова природних мов, їх значущі частини і морфологічні ознаки. До завдань морфології, таким чином, входить визначення слова як особливого мовного об'єкта і опис його внутрішньої структури.

Морфологія, згідно переважному в сучасній лінгвістиці розуміння її завдань, описує не тільки формальні властивості слів і утворюють їх морфем (звуковий склад, порядок проходження, і т. П.), А й ті граматичні значення, які виражаються всередині слова (або «морфологічні значення »). Відповідно до цими двома великими завданнями, морфологію часто ділять на дві області: «формальну» морфологію, або МОРФЕМИКА, в центрі якої знаходяться поняття слова і морфеми, і граматичну семантику, що вивчає властивості граматичних морфологічних значень і категорій (тобто морфологічно виражається словотвір і словозміна мов світу).

Поряд з позначенням деякої області лінгвістики, термін «морфологія» може означати і частина системи мови (або «рівень» мови) - а саме, ту, в якій містяться правила побудови і розуміння слів даної мови. Так, вираз іспанська морфологія співвідноситься з частиною іспанської граматики, в якій викладено відповідні правила іспанської мови. Морфологія як розділ лінгвістики є в цьому сенсі узагальненням усіх приватних морфології конкретних мов, тобто сукупністю відомостей про всіх можливих типах морфологічних правил.

Морфологія включає в себе:

- вчення про словозміни в мові, парадигмах, словозмінних типах. Це обов'язкова складова морфології, і саме з складання парадигм (таблиць відміни і відмінювання) історично почалася лінгвістика взагалі (в стародавньому Вавилоні);

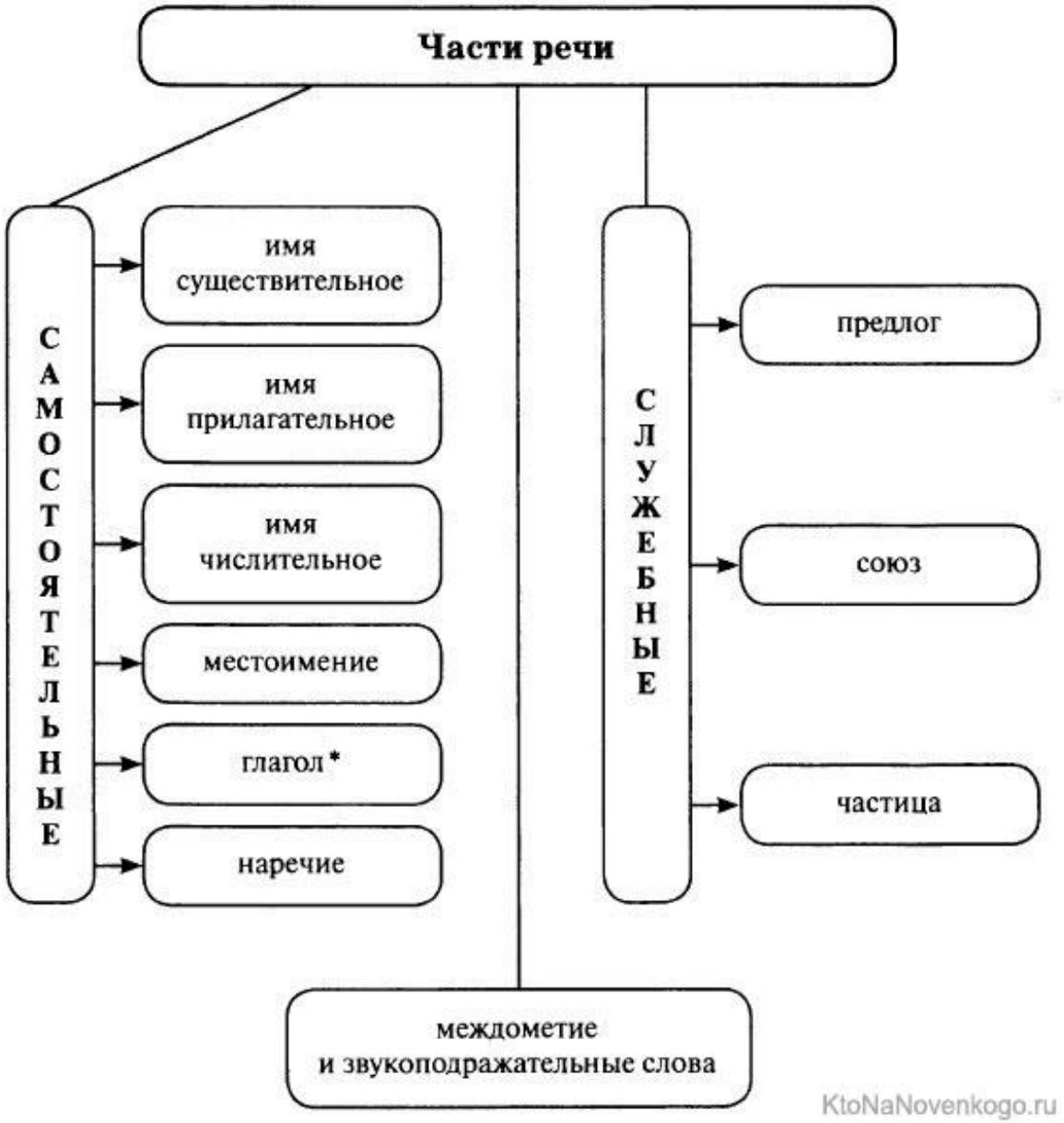


Рисунок 3.1 – Частини мови згідно морфології

- граматичну семантику, тобто вчення про граматичні значеннях. Традиційно (наприклад, в XIX столітті) граматична семантика не включалася до морфологію; в розділі граматик «морфологія» наводилися тільки способи утворення форм і зразки парадигм, а відомості про семантику («вживанні» форм) ставилися до синтаксису. У XX столітті граматична семантика є вже невід'ємною частиною морфології;

- вчення про частини мови, при виділенні яких задіяні не тільки морфологічні (у вузькому сенсі), але і синтаксичні та семантичні критерії;

- вчення про словотвір, що стоїть на кордоні морфології і лексикології;
- загальні поняття про морфології;
- морфологічну типологію.

Тісний зв'язок понять морфології і слова (в цьому ж значенні часто вживається більш точний термін «словоформа») ставить саме існування морфології в залежність від існування слів в конкретній мові. Тим часом, це поняття є одним з найбільш суперечливих в лінгвістиці і, швидше за все, не універсальним. Інакше кажучи, слово - це такий об'єкт, який існує, мабуть, не у всіх мовах, а значить, не у всіх мовах існує і морфологія як самостійний розділ граматики. У мовах, які не мають (або майже не мають) слів, морфологія не може бути розмежована з синтаксисом: у неї не залишається ні самостійного об'єкта, ні самостійною проблематики.

Не даючи в даному випадку точного визначення слова, можна вказати на те найважливіше властивість, яке становить його природу. Слово - синтаксично самостійний комплекс морфем, що утворюють жорстко пов'язану структуру. Слово відрізняється від поєднання слів тим, що принаймні деякі його елементи не можуть вживатися в синтаксично ізольованою позиції (наприклад, фігурувати в якості відповіді на питання); крім того, елементи всередині слова пов'язані один з одним набагато більш жорсткими і міцними зв'язками, ніж елементи пропозиції (тобто слова). Чим більше в мові ступінь контрасту між жорсткістю внутрісловних і межсловних зв'язків, тим більше виразною і добре виділимість одиницею є слово в даній мові. До таких «словесним» мов відносяться, наприклад, класичні індоєвропейські мови (латинська, давньогрецька, литовський, російський). У цих мовах морфеми всередині слова не володіють синтаксичної самостійністю, тобто частини слова не можуть в синтаксичному відношенні поводитися так само, як слова. Пор. кілька прикладів такої різниці у поведінці слів і частин слова в російській мові.

3.1 Основні поняття морфології

Морфологія вивчає структуру значущих одиниць мови. Головне підставу - членимость словоформи на менші знакові одиниці.

Морфологія - розділ граматики, що вивчає граматичні властивості слів. Слідом за В. В. Виноградовим морфологію часто називають «граматичним вченням про слово». Граматичними властивостями слів є граматичні значення, способи вираження граматичних значень, граматичні категорії.

Розширене поняття: морфологія - наука про форми.

Рисунок 3.2 – Приклад словотворення за допомогою суфіксів

Граматичне значення - узагальнене, абстрактне мовне значення, притаманне ряду слів, словоформ і синтаксичних конструкцій, що знаходить в мові своє регулярне (стандартне) вираження, наприклад, значення відмінка іменників, часу дієслова тощо.

Граматичне значення протиставлено лексичного значення, яке позбавлене регулярного (стандартного) вираження і не обов'язково має абстраговані характер. Граматичне значення супроводжує лексичне значення, накладається

на нього, іноді граматичне значення обмежена в своєму прояві певними лексичними групами слів.

Граматичні значення виражаються афіксальними морфемами, службовими словами, які дають зрозуміти чергуваннями і іншими засобами.

Кожне граматичне значення отримує в мові спеціальний засіб вираження - граматичний показник (формальний показник). Граматичні показники можна об'єднати в типи, які умовно можна назвати граматичними способами, способами вираження граматичного значення.

Граматичний спосіб аффиксации полягає у використанні афіксів для вираження граматичного значення: книг-і; чита-л-і. Афікси - це службові морфеми.

Відповідно до положення щодо кореня виділяють наступні види афіксів: префікси, постфікси, Інфікси, інтерфікси, циркумфікс.

Граматичний спосіб службових слів полягає в використанні службових слів для вираження граматичного значення: буду читати, прочитав би.

Граматичний спосіб - супплетивизм. Під супплетивизмом розуміється вираз граматичного значення словом з іншого основою: йду - йшов, людина - люди. В одну граматичну пару об'єднуються різнокореневі слова. Лексичне значення у них один і той же, а відмінність служить для вираження граматичного значення.

Граматичний спосіб редуплікації (повтору) полягає в повному або частковому повторенні частин слова для вираження граматичного значення: Такий спосіб словотворення як редуплікація (подвоєння кореня або основи слова) в російській мові не зустрічається, проте, широко поширений в індонезійських мовах. Так, наприклад, в мові Ніуе від дієслова ako (вчити) шляхом редуплікації утворюється іменник akoako (учитель) [18].

Граматичний спосіб - чергування (внутрішня флексія) являє собою використання зміни звукового складу кореня для вираження граматичного значення: уникати - уникнути; збирати -собрать.

Граматична форма - зовнішнє мовне вираження граматичного значення в кожному конкретному випадку вживання слова. Кожну окрему граматичну форму називають словоформой.

Граматична категорія - система протиставлені один одному рядів граматичних форм з однорідними значеннями. В русской грамматике виділяються іменні морфологічні категорії роду, одухотвореності / бездушності, числа, відмінка, ступеня порівняння; дієслівні категорії виду, застави, нахилення, часу й особи.

У сучасній російській мові категорія роду представлена трьома рядами форм (м., Ж., Ср.), Двома рядами категорії числа (од., Мн.), Шістьма рядами категорії відмінка.

Розрізняються категорії словозмінної, тобто такі, члени яких можуть бути представлені різними формами одного і того ж слова, і несловоізменітельние (класифікують), тобто такі, члени яких не можуть бути представлені формами одного і того ж слова. До перших відносяться число, відмінок, час, особа, ступінь порівняння. До других - рід і натхненність / неживого у іменників.

3.2 Мови зі слабкою морфологією

Не всі мови, проте, мають настільки ж «монолітними» словами, як російська і подібні до нього. Існують різноманітні типи відхилень від «словесного зразка».

Перш за все, в багатьох мовах частини слова виявляють тенденцію до більшої самостійності, що робить межу між словом і морфемой менш чіткою. Так, морфеми можуть опускатися подібно іменником і приводами в прикладі (2) - це явище називається «груповий флексией»; в ряді випадків позиція морфем в слові також виявляється дещо рухомий, ніж в мовах з жорсткими правилами. Підвищена самостійність морфем характерна для так званих слабо-Аглютинативних мов (до яких відносяться тюркські, японський, бірманський, дравидийские і ін.); в мовах такого типу комплекси морфем (слова) і комплекси слів (пропозиції) часто можуть бути описані в подібних або близьких термінах.

Це мови, де морфологія у власному розумінні поступається місцем «морфосінтаксісу».

З іншого боку, морфосінтаксіс замість морфології кращий і для таких мов, в яких, навпаки, не морфеми поводяться як слова, а пропозиції поводяться як слова. Іншими словами, в цих мовах також погано розрізняються внутрісловние і пробілом між словами зв'язку, але не за рахунок слабкої скреплённості морфем один з одним, а за рахунок більш сильної скреплённості слів один з одним. Фактично, пробілом між словами зв'язку в подібних мовах настільки сильні, що це призводить до утворення слів-пропозицій значної довжини. Мови такого типу часто називаються «полісінтетіческімі»; до ознак полісінтетізма відноситься схильність до утворення складних слів (особливо дієслівних комплексів, що включають підмет і доповнення - так звана інкорпорація), а також схильність до чередованиям на межсловних кордоні, ускладнює відділення одного слова від іншого. Словоскладання і особливо інкорпорація властиві багатьом мовам циркумполярної зони - ескімоським і чукотско-камчатським, а також багатьом мовам американських індіанців (поширеним як на Півночі, так і в Центральній Америці і в басейні Амазонки). Чергування на межсловних межах також властивим багатьом мовам американських індіанців; є вони і яскравою рисою санскриту.

Другий тип відхилень від словесного зразка пов'язаний не зі слабкістю межморфемних кордонів (як в Аглютинативні мови), а скоріше з відсутністю морфемних комплексів як таких. Це - найбільш яскрава риса так званих ізолюючих, або аморфних мов, в яких немає або практично немає протиставлення між країнами і афіксами: будь-яка морфема є коренем і здатна до самостійного вживання; показників ж граматичних значень в таких мовах практично немає. Таким чином, єдині морфемні комплекси, які в таких мовах можуть виникати - це складні слова, які часто буває важко відрізнити від поєднань слів. Можна сказати, що в ізолюючих мовах слово просто одно морфеме, а пропозиції будуються не зі слів, а відразу з морфем. Таким чином, і в цих мовах слово як самостійне утворення відсутня, і граматика фактично зводиться до того ж морфосінтаксісу (тобто синтаксису морфем). До

ізолюючим мов відноситься досить значна кількість мов світу: це в'єтнамський, тайський і інші мови Південно-Східної Азії, а також ряд мов Західної Африки: йоруба, еве, Акан, Манінка і ін.

Сказане про ізолюючих мовах може бути застосовано і до так званим аналітичним мовам, тобто до таких мов, де, на відміну від ізолюючих, є граматичні показники, але ці показники є самостійними словами, а не морфемами (афіксами). Граматичні значення в аналітичних мовах виражаються синтаксично (за допомогою різного роду конструкцій), а в морфологічно неелементарних словах необхідності не виникає. Аналітична граматика характерна для багатьох мов Океанії (особливо полінезійських), для ряду великих мов Західної Африки (хауса, сонгай); сильні елементи аналитизма є в нових індоєвропейських мовах (французька, англійська, скандинавські, сучасний перський).

Таким чином, можна сказати, що морфологія далеко не універсальна - по крайній мірі, далеко не для всіх мов морфологічний (або «словесний») компонент опису однаково важливий. Все залежить від того, наскільки чітко в даній мові виділяються словоформи [24].

4 ПРОГРАМНА РЕАЛІЗАЦІЯ МОРФОЛОГІЧНОГО КОДУВАННЯ СЛОВА

4.1 Кодування слів відповідно до морфології

Кодування слова відповідно до його морфологічними особливостями необхідно зібрати інформацію про слові у відповідності з усіма його морфологічними ознаками, як частина мови, рід, відмінок, і всі частини слова як приставка, корінь, суфікс і закінчення. Ці частини несуть більше сенсу для словотвору, ніж окремі букви і звуки, і більш інтерпретованих, ніж окремі слова [17].

Кодування слів за допомогою морфем можна назвати штучним «word embedding» -ом, де кількість властивостей нечисленне і визначено штучно, але кожне з них дуже важливо і разом вони повністю описують слово.

При знаходженні матриці E при навчанні моделі немає ніякого обмеження, що два вектори різних слів не можуть бути рівні. Крім того, вектор ніяк не може бути інтерпретований людиною назад щоб зрозуміти, яку саме інформацію про слові використовує мережу для роботи з ним і чи важлива ця інформація в контексті конкретного слова, характеризує вона його повністю (хоча ми можемо, схоже у випадку з debiasing embeddings, взяти для кожної фичи самі відрізняються за цим параметром вектора, подібні за іншими параметрами і спробувати визначити людську інтерпретацію вивченої мережею властивості).

Від використання морфем можено очікувати найкращого результату, так як морфеми не тільки містять всю інформацію про слові в досить стислому вигляді, що може значно підвищити швидкість і ефективність навчання і знизити витрати на навчання моделі.

Мінус морфем в тому, що для їх знаходження потрібні додаткові ресурси - навчена модель або датасета. На момент написання диплома навчання нейронної мережі для виокремлення морфем не входило в план завдань, проте,

можливо буде зроблено в рамках продовження досліджень після захисту диплома.

Інша можлива слабкість кодування за допомогою морфем - різний рівень розбиття слів на морфеми і кодування інформації з їх допомогою в залежності від мови. У російській мові (як і в українському) морфеми мають високий рівень інформативності. Для подальшої роботи з іншими мовами необхідно розробити метод оцінювання релеватності навчання моделі.

4.2 Джерела інформації

Для отримання морфологічної інформації використовувалися два сайти і .NET бібліотека MorphAPI.

Перше джерело - kartaslov.ru. Сайт має досить велику функціональність, яка описана на його головній сторінці.

Kartaslov має можливість працювати з синоніма слів і виразів, визначає сполучуваність слів, надає можливість побачити слово в різних контекстах (пошук джерел згадки), тлумачення слів і стійких виразів, мережа словесних асоціацій, таблиці відмінювання іменників і прикметників, а також таблиці відмінювання дієслів.

Також сервіс надає можливість розібрати слово за складом, що нам і потрібно для роботи.

Як пишуть творці, КартаСлов.ру - це онлайн-карта слів і виразів російської мови. Тут зв'язки між словами знаходять відчутну форму.

При створенні сайту ми використовували найостанніші досягнення в області комп'ютерної лінгвістики, машинного навчання та штучного інтелекту, спираючись при цьому на найпотужнішу теоретичну базу російської мови, створену видатними радянськими і російськими вченими-мовознавцями.

Почніть свою подорож з будь-якого слова або виразу, переходячи по посиланнях на сусідні ділянки карти. Зараз представлені три види зв'язків - асоціації, синоніми і сполучуваність; в майбутньому ми обов'язково додамо словотвірні та вертикальні відносини між словами.

Для всіх представлених на мапі слів і виразів показані приклади вживання в контексті. При цьому, використовуючи пошук, ви завжди можете вийти за межі розкресленій області. [12]

Kartaslov.ru, на жаль, не повертає форматированного відповіді в форматі JSON або XML і необхідно додатково парсити HTML сторінку сайту на стороні API, як буде описано далі у відповідному підрозділі.

Другий ресурс - http://morphemeonline.ru/. Цей ресурс спеціалізується на розборі слова за складом.

Третій ресурс - бібліотека для .Net DeepMorphy. DeepMorphy - морфологічний аналізатор для російської мови. Доступний як .Net Standart 2.0 бібліотека. Вміє проводити морфологічний розбір слова (визначає частина мови, рід, число, відмінок, час, особа) і приводити слова до нормальної форми. Основним елементом DeepMorphy є нейронна мережа. Для більшості слів морфологічний аналіз і лематизації виконується мережею. Деякі види слів обробляються препроцесорів (якщо слово оброблено препроцесором, то результат мережі не враховується).

Мережа побудована і навчена на фреймворку tensorflow. Як датасета виступає словник Opencorpora. У .Net інтегрована через TensorFlowSharp.

Граф обчислень DeepMorphy складається з 8 «підмереж»:

- 6 двонапрямлених рекурентних мереж, по одній для кожної підтримуваної граматичної категорії (визначає грам в категорії);

- 1 двунаправленная рекурентное мережу для визначення найімовірніших тегів. Для кожної комбінації граммем з датасета заведений 1 клас (всього 172 класу), мережа навчається на визначення до яких класів може належати дане слово. На етапі роботи береться 4 найвірогідніших класу;

- 1 seq2seq модель для лематизації [13].

DeepMorphy для .NET являє собою бібліотеку .Net Standard 2.0. В залежності тільки бібліотека TensorflowSharp (через неї запускається нейронна мережа).

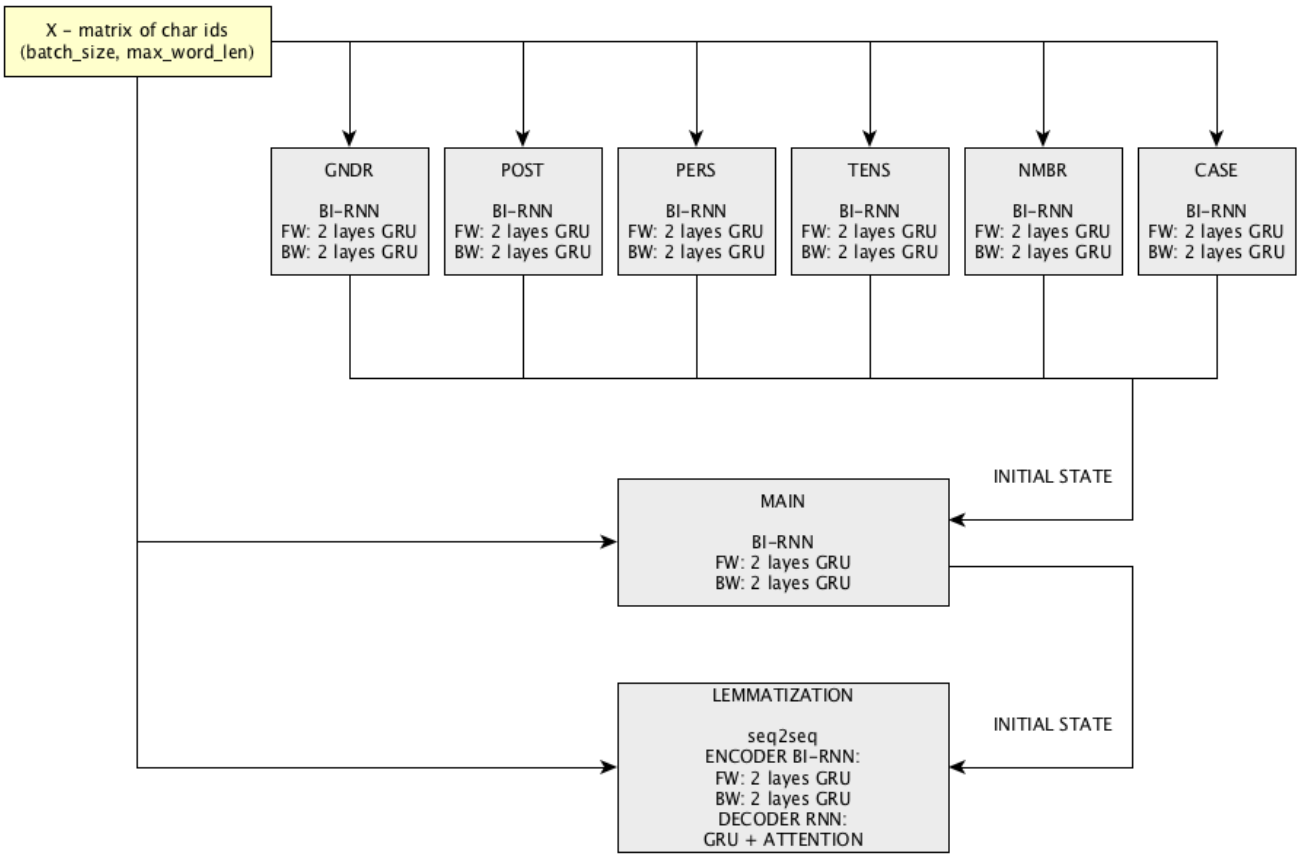


Рисунок 4.1 - Структура бібліотеки DeepMorphy [13]

4.3 Загальна структура

Розроблений процес кодування слова в вигляді морфологічної інформації про нього має досить довгий пайплайн і складається глобально з п'яти оточень: сайти в інтернет-просторі, .NET API, MsSQL база даних, Jupiter notebooks on Python і текстові файли, що використовуються Пайтон як проміжне сховище інформації для роботи зі словами.

Ми розглянемо кожен з великих розроблених блоків (.NET API, Jupiter, Database) окремо далі в розділах, а на рисунку 4.2 представлена узагальнена схема блоків і зв'язків між ними.

Вхідна точка - Jupyter notebook. За допомогою вибудовування Пайплайн з заготовлених методів, ця частина процесу може варіюватися в залежності від мети і конкретного дослідження, але в загальному за допомогою методів на

python ми маємо можливість вважати потрібні слова з текстового файлу або ж не-форматований текст, перетворити в формат, необхідний для запиту до API, послати POST-запит до .NET API, потім API всередині свого пайплайну

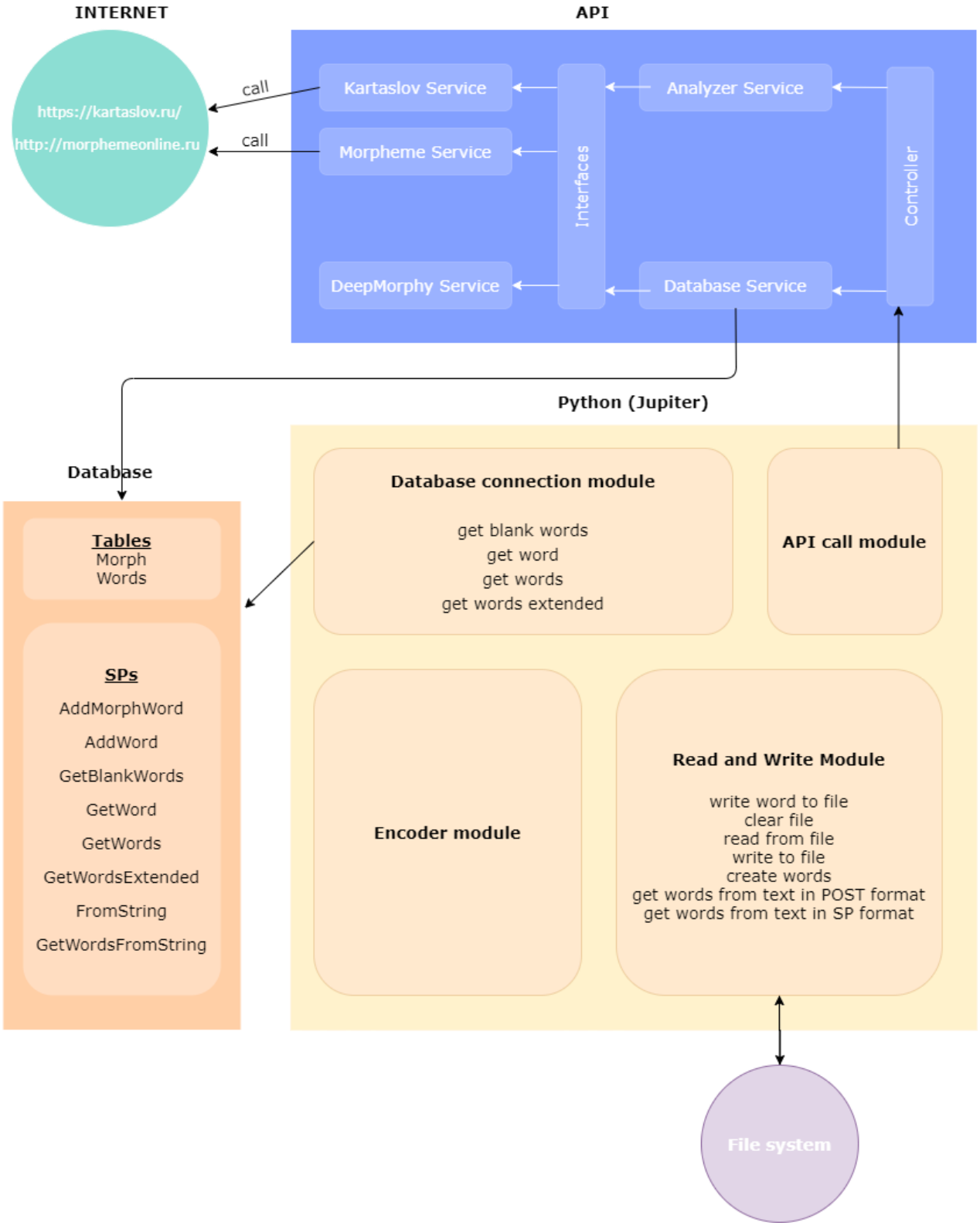


Рисунок 4.2 - Узагальнена схема блоків і зв'язків між ними

посилає запити до зовнішніх сайтів і бібліотеці DeepMorphy, отримує інформацію про слова, записує її в базу даних MsSQL за допомогою збережених процедур (або обнов яет існуючу інформацію); після отримання відповіді про завершення запиту python має можливість звернутися до збережених процедур БД і отримати інформацію про слова. Потім вже в пам'яті запускається безпосередньо процедура кодування слів в вектора.

Як ви можете спостерігати на схемі, підготовка слів зачіпає доститочно багато функціоналу. Це обумовлено декількома речами: при наявності дочтаточного кількості часу і знань необхідно написати нейронну мережу для перетворення слів в морфологічний вид - виділення морфем і опредленія морфологічних особливостей слова - числа, роду, відмінка і так далі. Власне, частина цієї функціональності і надає допоміжна бібліотека DeepMorphy, яка всередині використовує рекурентні нейронні мережі.

Друге - виклики до сайтів і парсинг html-сторінок займають достатньо часу (в умовах роботи з сотнями, тисячами слів і того, що на кожне слово використовується окремий запит), .NET використовувався багато в чому також для оптімзація запитів до зовнішніх ресурсів (і досвіду в цієї оптимізації).

База необхідна як проміжне реляционное швидке сховище для слів, в якому можна зберігати і діставати великий обсяг інформації.

4.4 API

API реалізовано в середовищі .NET мовою C # для роботи з внешнмі ресурсами, базою даних і бібліотекою DeepMorphy, яка віддає часткову морфологічну інформацію про слова. Розглянемо докладніше структуру проекту.

На верхньому рівні знаходиться папка Controllers - контролери, взяті з моделі MVC (Model-View-Controller) відповідають за контроль виконання програми, вони приймають запити і розпоряджаються тим, який член проекту буде виконувати, фільтрувати, перевіряти запит і подібне. У нашому випадку єдиний контролер MorphyController приймає REST POST запит, відправлений в

форматі localhost: port / api / morphy / words або localhost: port / api / morphy / words / soft. Ці запити мають схожу структуру і обидва приймають масив рядків (слів), проте реалізують різну функціональность - перший аналізує слова і записує їх в базу, звідки ми можемо після повернення статусу 200 ОК вважати їх окремим запитом не викликаючи АПИ будь-яку кількість разів, поки інформація лежить в базі; другий же запит повертає слова в response, минаючи базу даних.

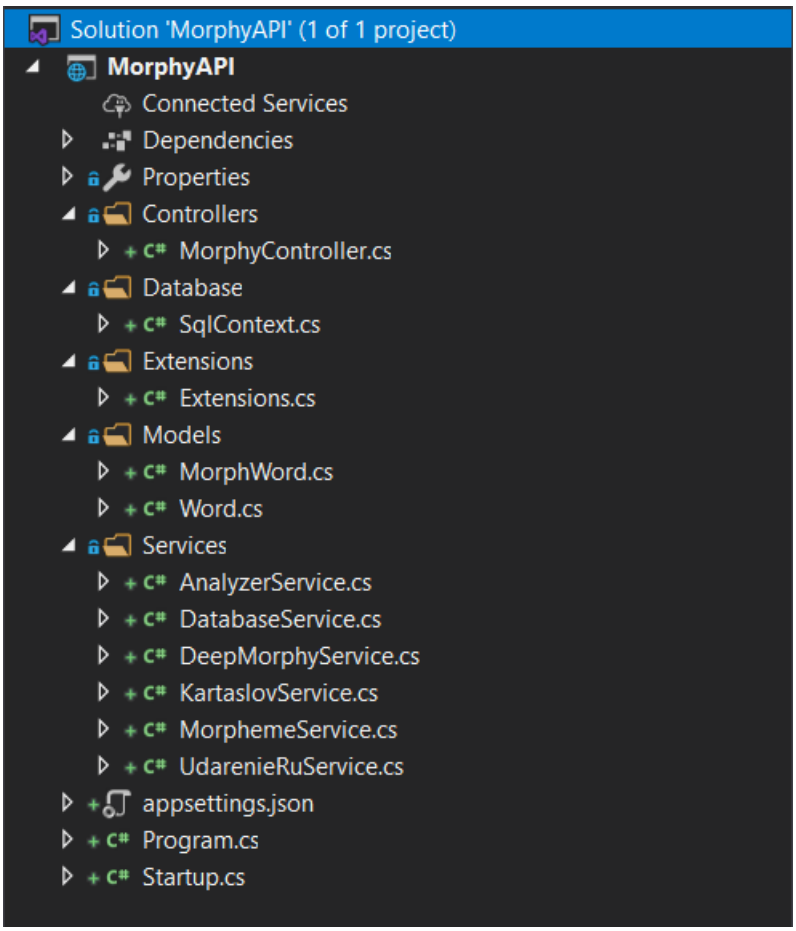


Рисунок 4.3 - Структура файлів в проекті

Контролер передає управління сервісу, він отримує доступ до сервісу за допомогою механізму Dependency Injection, який реалізований в платформі .NET Core «з коробки» і реєструється в класі ініціалізації Startup.cs. Ми розглянемо його пізніше при розгляді роботи сервісів.

```
4 references
public void Open()
{
    _sqlConnection = new SqlConnection(_config.GetConnectionString(name: "MorphyDB"));
    _sqlConnection.Open();
}
```

Рисунок 4.4 - Підключення до бази даних в класі SqlContext

SqlContext - клас - прошарок для зв'язку з БД, крім цього він інкапсулює відкриття і закриття зв'язку з базою даних для паралельних запитів записи. Клас отримує рядок підключення з конфігурації проекту (рисунок 5.4). Основні методи класу наведені на рисунку 5.5, розглянемо їх докладніше.

```
3 references
public async Task ExecuteAsync(string sql,
    object param = null,
    IDbTransaction transaction = null,
    int? commandTimeout = null)
{
    var command = new CommandDefinition(sql,
        param,
        transaction,
        commandTimeout,
        CommandType.StoredProcedure);
    await _sqlConnection.ExecuteAsync(command);
}

2 references
public async Task<IEnumerable<T>> QueryAsync<T>(string sql,
    object param = null,
    IDbTransaction transaction = null,
    int? commandTimeout = null)
{
    var command = new CommandDefinition(sql,
        param,
        transaction,
        commandTimeout,
        CommandType.StoredProcedure);
    return await _sqlConnection.QueryAsync<T>(command);
}
```

Рисунок 4.5 - Методи зв'язку з базою даних

Обидва методи асинхронні - це значить, що вони виконуються новим потоком з пулу потоків - ця функціональність управляється середовищем CLR (Common Language Runtime). Це дає можливість запускати багато потік одночасно з одного методу, і даний метод запускати паралельно, використовуючи одне поделюченіе до БД. На асинхронність вказують ключові слова async, await і повертається тип Task, який обертає повертається об'єкт (якщо метод щось повертає) і додатково дає інформацію про завершеність «завдання».

Методи відрізняються тим, що перший з них потрібен для збережених процедур, які не повертають нам ряди з бази даних (збереження, видалення), а другий - тільки повертає дані. Це відповідає принципу CQS (Command Query Separation).

```
2 references
public async Task<IEnumerable<string>> GetBlankWords()
{
    _sqlContext.Open();
    var result :IEnumerable<string> =  await _sqlContext.QueryAsync<string>( sql: "dbo.GetBlankWords");
    _sqlContext.Close();
    return result;
}
```

Рисунок 4.6 – Метод, який повертаючий рядки з бази даних

IEnumerable - інтерфейс, контракт який визначає мінімум особливих методів у класів, які можуть його реалізувати. Це група стандартних класів .NET, що цікавить нас, якщо спростити - це аналог перебирати списку, масиву - рядків з бази даних. <T> ідентифікуючи метод як generic - під час компіляції він конкретизується конкретним повертається типом. В одному місці проекту цей метод викликається з повертається типом string - повертає слова, для яких не заповнена інформація в БД, в іншому - база повертає рядки, які Dapper (бібліотека, яка інкапсулює зв'язок з базою даних) перетворює в об'єкти (рисунки 5.6 і 5.7).

```
0 references
public async Task<IEnumerable<Word>> GetWordsAnalyzed()
{
    _sqlContext.Open();
    var result :IEnumerable<Word> = await _sqlContext.QueryAsync<Word>( sql: "dbo.GetAnalyzedWords");
    _sqlContext.Close();
    return result;
}
```

Рисунок 4.7 - Метод, який повертає об'єкти з бази даних

Далі є папка з моделями - класами об'єктів, які ми передаємо між сервісами і повертаємо з АПИ, записуємо в базу даних. У нас окремі моделі, одна - відповідна даними про морфемах для слова, в які я перетворюю інформацію в сайтів, і інша - для даними про слово, яке отримується від DeepMorphy.

```
35 references
public class Word
{
    4 references
    public string FullWord { get; set; }

    4 references
    public IList<string> Prefixes { get; set; } = new List<string>();

    4 references
    public IList<string> Suffixes { get; set; } = new List<string>();

    4 references
    public IList<string> Roots { get; set; } = new List<string>();

    4 references
    public IList<string> Endings { get; set; } = new List<string>();
}
```

Рисунок 4.8 – Модель Word

AnalyzerService - головний сервіс, який координує роботу інших сервісів. Вхідна точка для контролера. Також в ньому реалізований вилов помилок

(робочий, спрямований нема на логгірованіе, підтримку, відправлення статусу про помилку, а на налагодження). Він має два методи для аналізу слів.

```
2 references
public async Task AnalyzeAndWriteToDB(string[] words)
{
    try
    {
        var morphy:IList<MorphWord> = _deepMorphy.AnalyzeWords(words);
        await _databaseService.SaveMorphyWords(morphy.ToList());
        var analyzedWords:IList<Word> = await _kartaslovService.AnalyzeWords(words);
        await _databaseService.SaveWords(analyzedWords.ToList());
        words = (await _databaseService.GetBlankWords()).ToArray();
        var resultMorpheme:IList<Word> = await _morphemeService.AnalyzeWords(words);
        await _databaseService.SaveWords(resultMorpheme.ToList());
    }
    catch (Exception ex)
    {
        if (Debugger.IsAttached)
            Debugger.Break();
    }
}
```

Рисунок 4.9 - Метод, який відповідає за пайплайн аналізу слів

Ми бачимо три сервісу, використаних для аналізу слів - deepMorphy (відповідальний за отримання інформації з бібліотеки DeepMorphy і форматування його в зручний для апі і БД проміжний формат), kartaslovService, morphemeService - що символічно, методи з однаковими назвами використовуються для схожого функціоналу.

Конструкція try-catch використовується для вилову помилок, де try - блок, в якому може виникнути помилка, Exception - загальний клас помилки, а catch - блок, в якому відбувається обробка помилки. В даному випадку ми викликаємо налагодження (отладку) і можемо працювати з помилкою відразу.

DatabaseService - сервіс, який інкапсулює роботу з базою даних і оптимізує роботу з нею (распараллелівається запити).

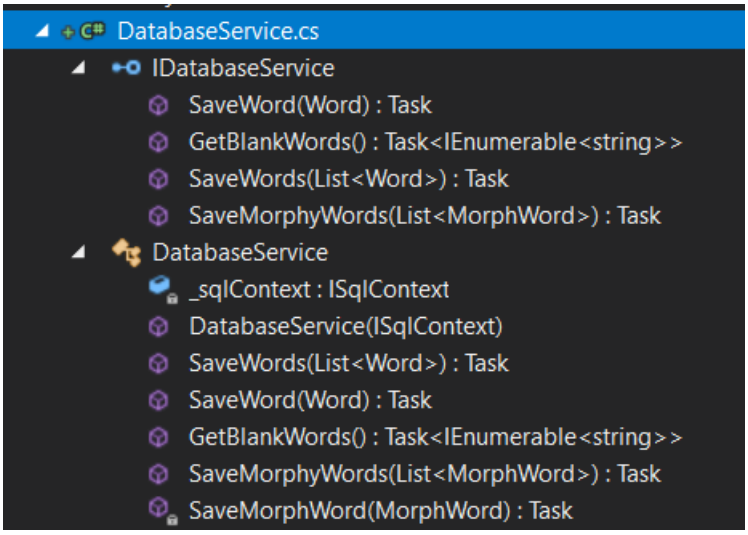


Рисунок 4.10 – Структура інтерфейсу IDatabaseService і сервісу DatabaseService, який його реалізує

На прикладі DatabaseService (рисунок 4.10) можна побачити сигнатури методів в сервісі і інтерфейсу, котрий він імплементує. Також видно, що сервіс використовує один приватний метод, який також використовується для виклику паралельних операцій.

При кожному запуску цього методу відбувається форматування - окрема інформація про частини слова (суфікси, префікси і т.д.) перетворюються з об'єкта моделі в формат, що зберігається в базу даних, а більш загальний метод контролює розпаралелювання і підключення до бази даних. У сервісі реалізовано збереження морфологічної інформації про словах і морфем, отримання «незаповнених» слів.

На рисунку 4.11 можна спостерігати пару методів, розділених на batch-и, подібні до того, як поділяють тренувальні екземпляри на batch-и для gradient descent більш ефективного навчання.

DeepMorphyService - сервіс-фасад для бібліотеки DeepMorphy. Він інкапсулює singleton MorphAnalyzer і дістає з об'єктів бібліотеки потрібні для роботи дані, перетворює в проміжні моделі АПИ.

Так як необхідні нам дані не надаються бібліотекою безпосередньо, а є приватними полями об'єкта, на основі яких бібліотека робить якісь внутрішні, непотрібні для нашої мети обчислення, сервіс використовує рефлексію для

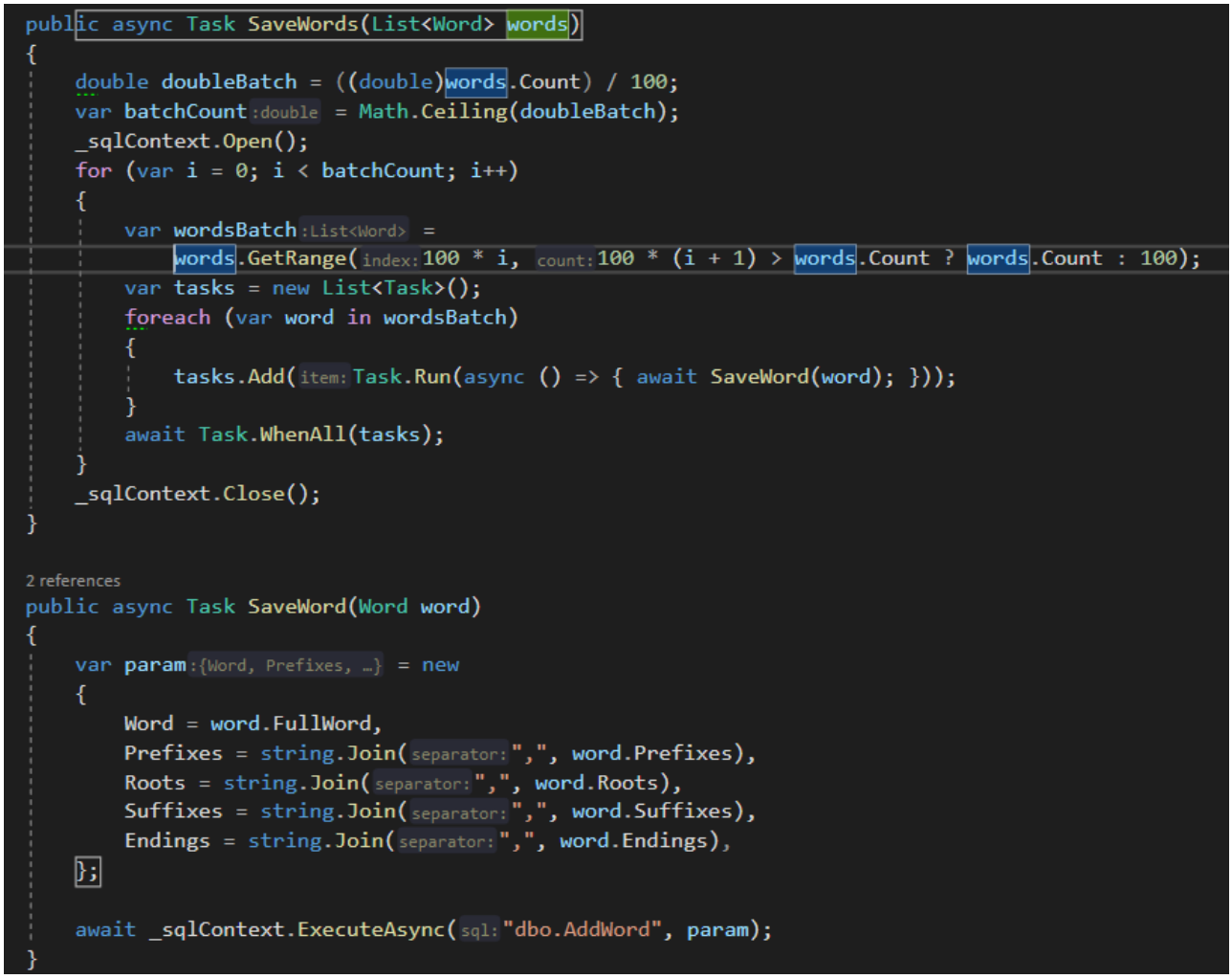


Рисунок 4.11 – Пара методів, використовуваних для форматування і паралелізації запитів до бази даних

доступу до цих полів об'єкта. Ми отримуємо з метаданих типу MorphInfo все властивості типу і знаходимо необхідне нам за допомогою запиту LINQ (Рисунок 4.12). Потім використовуємо приватний метод сервісу для отримання value у внутрішній структурі (рисунок 4.13).

```
var type = typeof(MorphInfo);

var gramsCat :PropertyInfo = typeof(MorphInfo).GetProperties() // PropertyInfo[]
    .FirstOrDefault(prop :PropertyInfo => prop.Name == "Item" && prop.MetadataToken == 385875989);

var chastRechi :string = GetValue(gramsCat, property: "чр", info);
```

Рисунок 4.12 - Отримання властивостей типу

```
6 references
private string GetValue(PropertyInfo gramsCat, string property, object instance)
{
    var res :string = ((GramCategory)gramsCat?.GetValue(instance, index: new object[] {property}))?.BestGramKey;
    return res;
}
```

Рисунок 4.13 - Отримання властивостей типу

Наступні сервіси схожі за змістом - вони відповідають за виклик зовнішніх сайтів і витягування з них потрібної інформації. Розглянемо їх функціонал на одному з сервісів - KartaslovService. Його загальна схема приведена на рисунку 4.15.

Як ви можете бачити, інтерфейс - відповідно, методи, які зовнішні класи мають можливість викликати - має тільки два методи, причому тільки один з них використовується в основному Пайплайн додатки (AnalyzeWords).

```
2 references
private void ParseResponseString(string response, IList<Word> words)
{
    var parts :MatchCollection = Regex.Matches( input: response, pattern: @"(?<=<td class='td-morpheme-text'>)(.*?)(?=</td>)");
    var codes :MatchCollection = Regex.Matches( input: response, pattern: @"(?<=<td class='td-morpheme-type'>)(.*?)(?=</td>)");
    var textWord :string = Regex.Matches( input: response, pattern: @"(?<=<title>)(.*?)(?= )").First().Value;

    var word = new Word
    {
        FullWord = textWord
    };

    for (int i = 0; i < codes.Count; i++)
    {
        switch (codes[i].Value)
        {
            case "приставка": word.Prefixes.Add(parts[i].Value); break;
            case "корень": word.Roots.Add(parts[i].Value); break;
            case "суффикс": word.Suffixes.Add(parts[i].Value); break;
            case "окончание": word.Endings.Add(parts[i].Value); break;
        }
    }

    lock (Lock)
    {
        words.Add(word);
    }
}
```

Рисунок 4.14 - Метод для розбору html

Цей метод створює головний об'єкт для збереження результату - список, який передається по посиланню і який можна використовувати в різних потоках

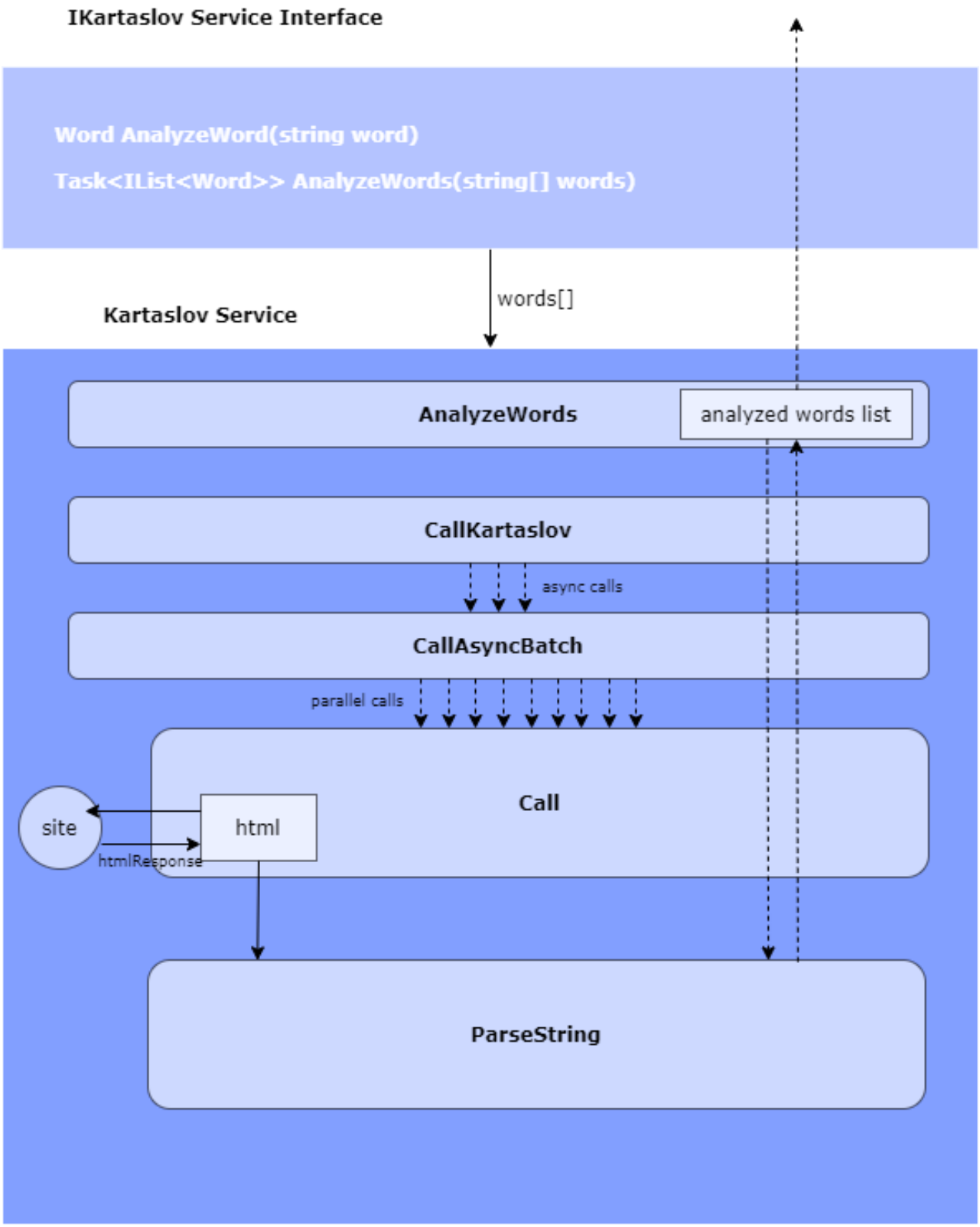


Рисунок 4.15 - Загальна схема сервісу KartaslovService

- посилання вказуватимуть і будуть додаватися в один і той же об'єкт. CallKartaslovAsync розподіляє batch-й, всередині яких ефективно

распараллелить запити (відповідно до розміру thread pool, який управляється CLR). Для вибору слів в batch використовується об'єкт Range, що з'явився в .NET Core 3.1.

CallAsyncBatch займається безпосередньо розпаралелюванням запитів, а всередині Call просходит сам виклик - кодування запиту, отримання відповіді з допомогою HttpClient і виклик парсеру html.

Важлива частина сервісу - метод ParseResponseString (рисунок 4.14) - відповідає за парсинг html і винос з нього інформації. Він також контролює многопоточность за допомогою конструкції lock. Конструкція lock розкривається в блок try-finally (блок для вилову помилок, який відрізняється тим, що блок finally виконується завжди, незалежно від виникнення помилки в блоці try) - в блоці finally відпускається блокування потоків.

Також для парсинга використовуються регулярні вирази і клас Regex для пошуку частин моделі на сторінці по паттернам в DOM-дереві.

5 МОДЕЛЬ МОВИ, ДОСЛІДЖЕННЯ І ДОПОМІЖНІ СТРУКТУРИ ДЛЯ ОБРОБКИ ТЕКСТУ

Jupyter notebooks - зручна IDE для роботи з Python, дозволяє виконувати код в умовних клітинах, що мають спільні змінні, зручно структурувати код і коментарі до нього, керувати kernel. Для навчання моделі мови при кодуванні примірників «пословно» я виділила окремий Jupyter notebook, який зчитував екземпляри з текстових файлів, підготовлених мною за допомогою окремого .NET додатки, в якому кожен екземпляр був відділений умовним токеном <EOE> (end of example).

Нижче описані лише два ключових notebook - модель для навчання на словах і розроблений notebook для зменшення кількості навчальних примірників з підвищенням ефективності навчання.

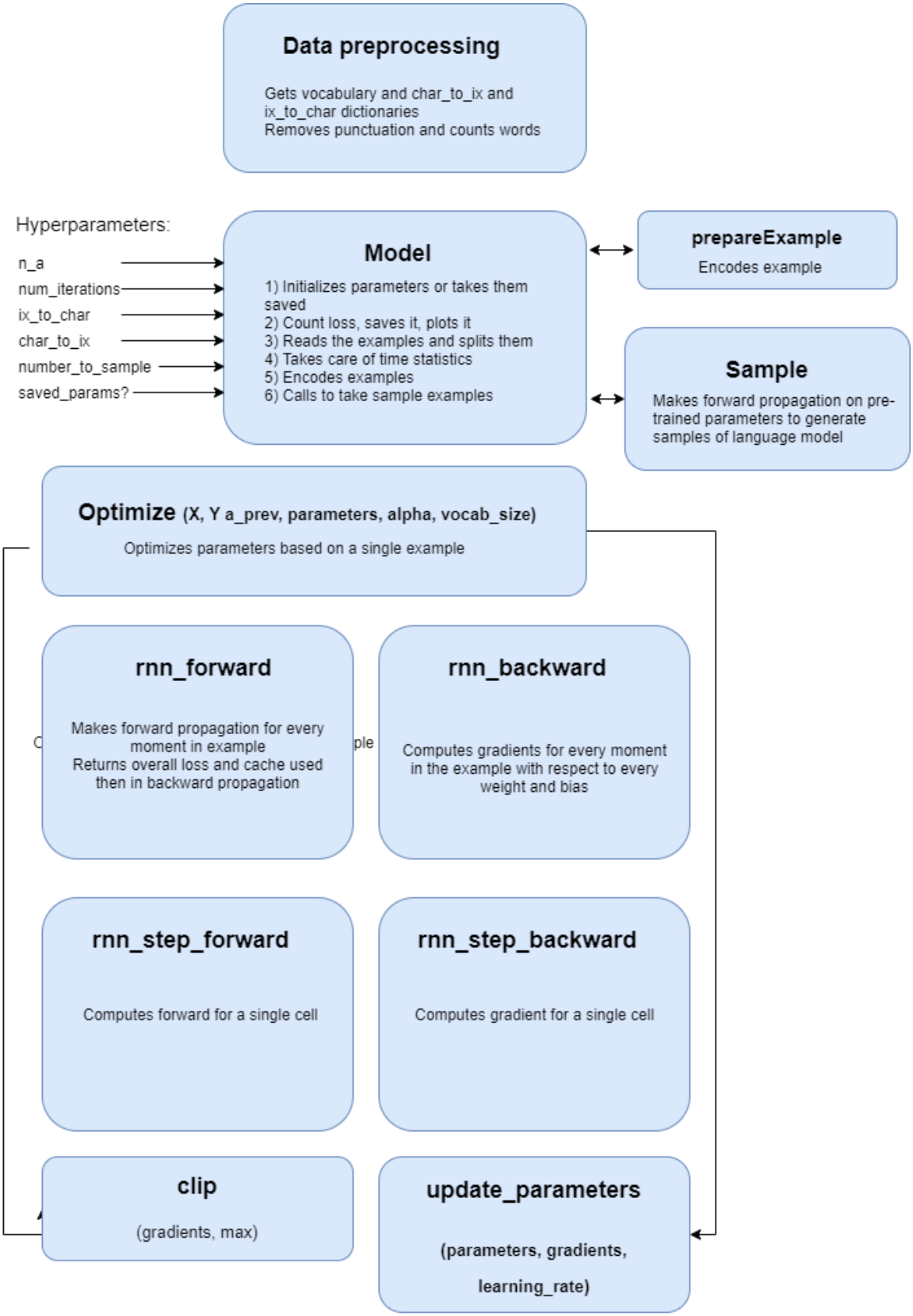


Рисунок 5.1 - Notebook, що імплементує модель RNN, що навчається на словах

Рисунок 5.2 - Notebook, що імплементує фільтрацію єкземпляров для навчання моделі

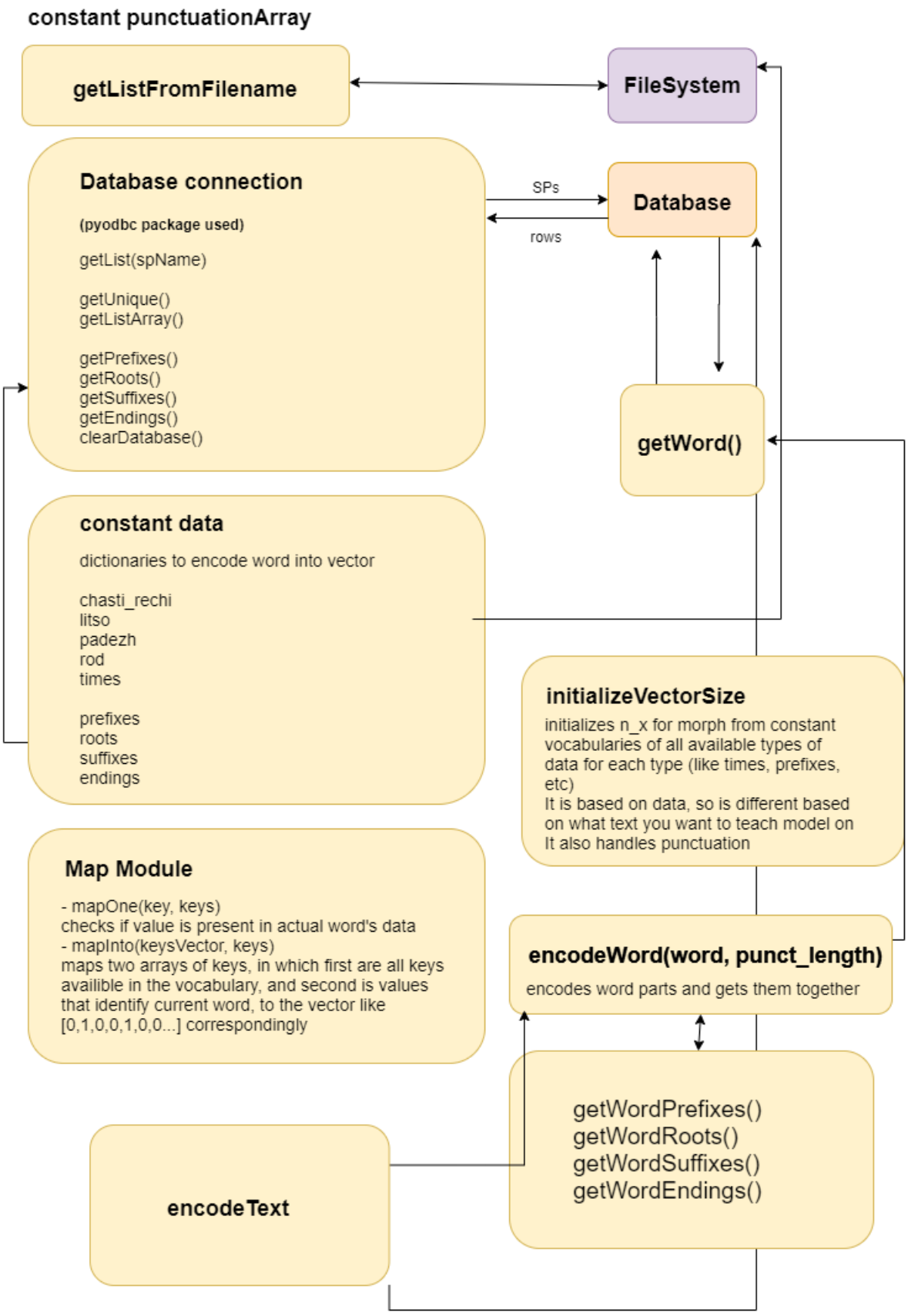


Рисунок 5.3 – Частина notebook для кодування слів за допомогою морфологічної інформації

Рисунок 5.4 – Частина notebook для кодування слів за допомогою морфологічної інформації

6 РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

Для дослідження особливостей навчання моделі я використовувала в якості датасета вірші поета Володимира Маяковського. У загальному вигляді я отримала близько 1400 віршів які складаються в цілому з 97000 рядків. Це прекрасний корпус для досліджень, однак досить масивний для навчання. Моделі для навчання на такому корпусі вимагають великих потужностей і тижнів, можливо місяців навчання, що підійшло б для вирішення завдання імітування віршів Володимира Маяковського, однак для завдання дослідження я взяла в основному чотири види даних - один чотиривірш, один вірш, один вірш, розбитий на чотиривірші і весь корпус віршів.

Я використовувала аналіз даних, отриманих з перших трьох завдань, для калібрування параметрів для останнього завдання з найбільшою точністю з урахуванням обмежених потужностей та продуктивності а також обмеженням за часом.

Подальші розділи розбиті за результатами досліджень на дослідження, проведені на моделі, яка навчається на словах, і моделі, яка навчається на символах. Кожен з розділів окремо розглядає кожен з чотирьох видів даних, на яких навчається модель, з різними гіперпараметрамі навчання. Для калібрування я використовувала два найбільш важливих гіперпараметра - alpha і n_a - розмір проміжного вектора.

При виборі а тут і далі я виходила з вибору пропорційно цифрі «словника» даного набору. Кількість слів або символів в словнику написано перед таблицею в якості змінної vocab. Я вважала модель досить навченою, якщо лосс досягав одиниці або близько того (в районі одиниці лосс продовжує падати, але повільно в режимі плато).

6.1 Word model

Vocab для моделі, навченою на словах на одному чотиривірші, дорівнював 28 словам. Результати досліджень наведені в таблиці 6.1.

Таблиця 6.1 - Результати навчання моделі на одному чотиривірші

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 10	t: 29.2 sec Average t of epoch: 0.0019 sec Iteration of convergence:14800 Final loss: 1.57 (converged)	t: 3.09 sec Average t of epoch: 0.0019 sec Iteration of convergence:1600 Final loss: 1.17 (converged)	
a = 32	t: 13.3 sec Average t of epoch: 0.002 sec Iteration of convergence:6600 Final loss: 0.98 (converged)	t: 1.15 sec Average t of epoch: 0.0019 sec Iteration of convergence: 600 Final loss: 0.89 (converged)	
a = 50	t: 11 sec Average t of epoch: 0.0023 sec Iteration of convergence: 4800 Final loss: 1.27 (converged)	t: 1.55 sec Average t of epoch: 0.0025 sec Iteration of convergence: 600 Final loss: 0.43 (converged)	t: 2.67 sec Average t of epoch: 0.0026 sec Iteration of convergence: 1000 Final loss: 1040 (not converged)

Продовження таблиці 6.1

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 100	t: 10.3 sec Average t of epoch: 0.003 sec Iteration of convergence: 3400 Final loss: 0.93 (converged)	t: 1.03 sec Average t of epoch: 0.0025 sec Iteration of convergence: 400 Final loss: 0.40 (converged)	t: 17.6 sec Average t of epoch: 0.0035 sec Iteration of convergence: 5000 Final loss: 867 (not converged)
a = 200	t: 21.6 sec Average t of epoch: 0.007 sec Iteration of convergence: 3000 Final loss: 0.59 (converged)	t: 8.3 sec Average t of epoch: 0.013 sec Iteration of convergence: 600 Final loss: 0.20 (converged)	t: 38.5 sec Average t of epoch: 0.007 sec Iteration of convergence: 5400 Final loss: 1510 (not converged)
a = 400	t: 54.357 sec Average t of epoch: 0.034 sec Iteration of convergence: 1600 Final loss: 3.03 (not converged)		

За співвідношенням часу (продуктивності) і якістю навчання кращий результат для моделі слів, навченої на чотиривірші показала модель, навчена з параметрами alpha 0.01 і розміром скритог шару a 100, також не сильно відстали від неї і показали достатній результат моделі з alpha 0.01 і a 50 і 32. Незважаючи на значне збільшення розміру вектора a в лідируючій моделі, a,

значить, і великих обчислювальних витрат, модель показала велику швидкість навчання за рахунок меншої кількості ітерацій.

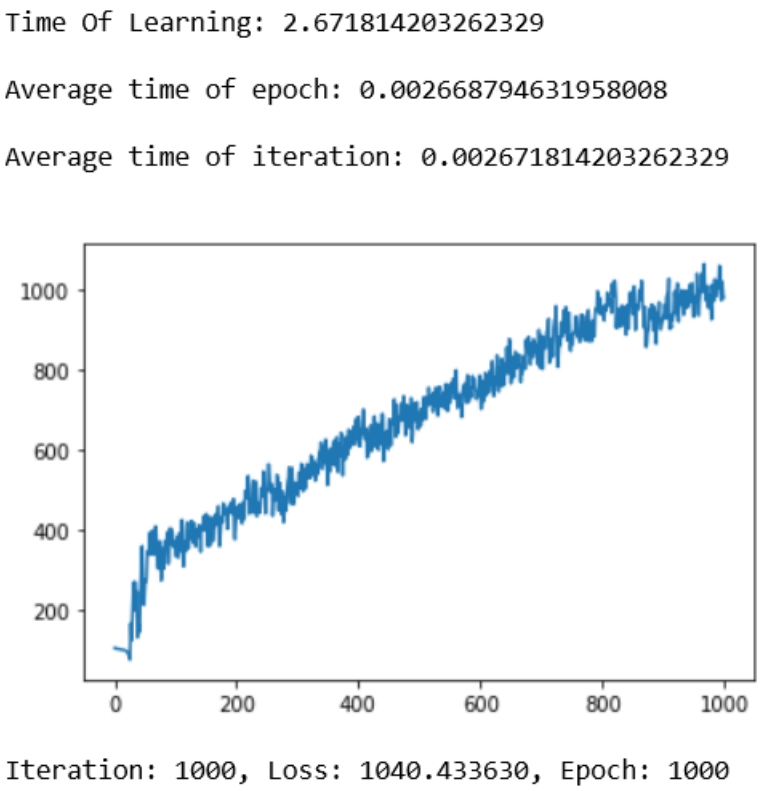


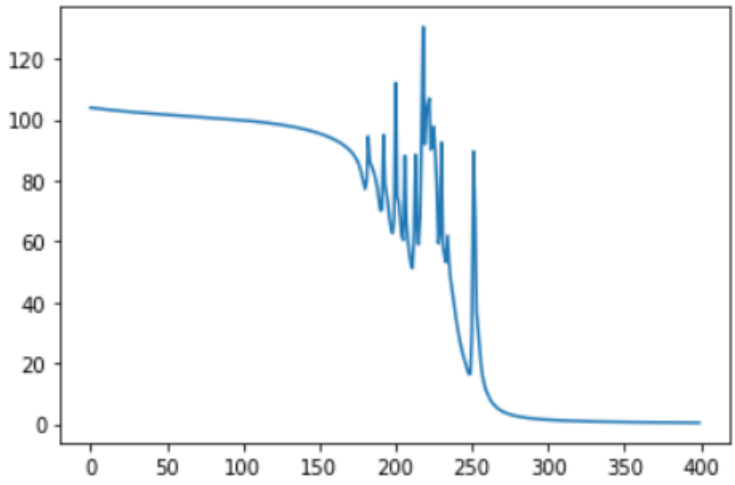
Рисунок 6.1 – Графік функції помилки при alpha 0.1 і a 50

Для alpha 0.001 модель витратила більше часу для навчання, а при alpha 0.1 модель не змогла ефективно навчитися зв'язків - функція помилки продовжувала зростати. На рисунку 6.1 наведено графік функції помилки для моделі при alpha 0.1 і a 50. Для порівняння на рисунку 6.2 наведено графік функції помилки для кращої моделі з alpha 0.01 і a 100.

```
Time Of Learning: 1.0302448272705078

Average time of epoch: 0.002568097114562988

Average time of iteration: 0.0025756120681762694
```




```
Iteration: 400, Loss: 0.408974, Epoch: 400
```

Рисунок 6.2 – Графік функції помилки при alpha 0.01 і a 100

Vocab для моделі, навченої на одному вірші є 67. Дослідження наведені в таблиці 6.2.

Таблиця 6.2 - Результати навчання моделі на одному вірші

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 10		t: 17.44 sec Average t of epoch: 0.005 sec Iteration of convergence: 3400 Final loss: 1.41 (converged)	

Продовження таблиці 6.2

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 35	t: 33.7 sec Average t of epoch: 0.005 sec Iteration of convergence: 6200 Final loss: 1.93 (converged)	t: 4.06 sec Average t of epoch: 0.005 sec Iteration of convergence: 800 Final loss: 1.12 (converged)	t: 8.45 sec Average t of epoch: 0.005 sec Iteration of convergence: 1600 Final loss: infinite (not converged)
a = 70	t: 27.76 sec Average t of epoch: 0.007 sec Iteration of convergence: 4000 Final loss: 1.76 (converged)	t: 6.43 sec Average t of epoch: 0.0064 sec Iteration of convergence: 1000 Final loss: 0.67 (converged)	t: 7.55 sec Average t of epoch: 0.006 sec Iteration of convergence: 1200 Final loss: 2707 (not converged)
a = 150	t: 39.5 sec Average t of epoch: 0.016 sec Iteration of convergence: 2400 Final loss: 1.91 (converged)	t: 17.12 sec Average t of epoch: 0.014 sec Iteration of convergence: 1200 Final loss: 454 (not converged)	t: 105.97 sec Average t of epoch: 0.015 sec Iteration of convergence: 6800 Final loss: infinite (not converged)

Продовження таблиці 6.2

a = 300	t: 57.25 sec Average t of epoch: 0.028 sec Iteration of convergence: 2000 Final loss: 1.21 (converged)	t: 28.58 sec Average t of epoch: 0.028 sec Iteration of convergence: 1000 Final loss: 650 (not converged)	

Найкращі результати для моделі, навченою на вірші, показали параметри alpha 0.01 і a 35 і 70. Цей результат досить стабільний щодо попередньої моделі, так як alpha залишається тим же, а a стабільно состовляет vocab * 1 і vocab * 2.

```
На и щей жир вот вы , мужчина , у вас в усах капуста где — то недокушанных , недоеденных щей ; вот вы , женщина , на вас белила
густо , вы смотрите устрицей из раковин вещей . все вы на бабочку поэтиного сердца взгромоздитесь , грязные , в калошах и без к
алош . толпа озвереет , будет тереться , ощетинит ножки стоглавая вошь .
 через час отсюда в чистый переулок вытечет по человеку ваш обрюзгший жир , а я вам открыл столько стихов шкатулок , я — бесцен
ных слов мот и транжир . вот вы , мужчина
 Вы отсюда мужчина , бабочку ножки стоглавая вошь .
 через час отсюда в чистый переулок вытечет по человеку ваш обрюзгший жир , а я вам открыл столько стихов шкатулок , я — бесцен
ных слов мот и транжир . вот вы , мужчина , у вас в усах капуста где — то недокушанных , недоеденных щей ; вот вы , женщина , н
а вас белила густо , вы смотрите устрицей из раковин вещей . все вы на бабочку поэтиного сердца взгромоздитесь , грязные , в ка
лошах и без калош . толпа озвереет , будет тереться , ощетинит ножки стоглавая вошь
```

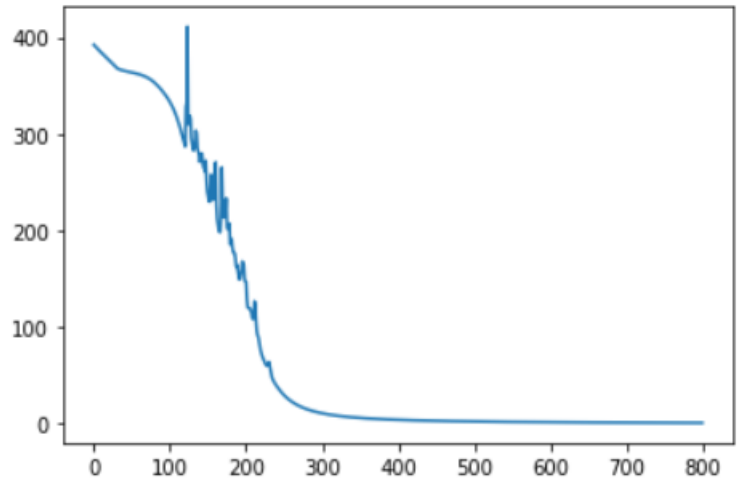
Рисунок 6.3 - Результат навченої моделі на параметрах alpha 0.01 і a 35

На рисунку 6.3 представлений результат, отриманий для параметрів alpha 0.01 і a 35, а на рисунку 6.4 ви можете спостерігати графік функції помилки для цього випадку.

```
Time Of Learning: 4.068231105804443

Average time of epoch: 0.005079047381877899

Average time of iteration: 0.0050852888822555545
```




```
Iteration: 800, Loss: 1.122499, Epoch: 800
```

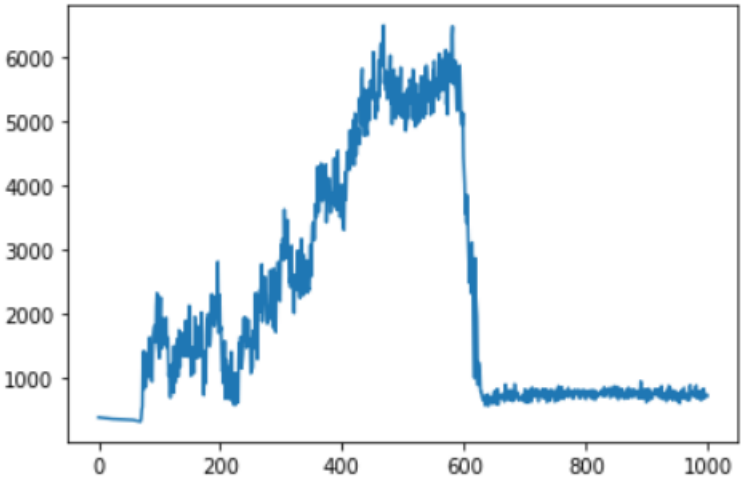
Рисунок 6.4 – Графік функції помилки при alpha 0.01 и a 35

Цікава ситуація сталася з моделлю, навченої на гіперпараметрах alpha 0.01 і a 300 (графік функції помилки на рисунку 7.5). Динаміка зміни функції помилки показувала негативну ефективність навчання, коли на 600 ітераціях модель різко перейшла умовний бар'єр. На жаль, подальшого поліпшення динаміки навчання не було.

```
Time Of Learning: 28.587408781051636

Average time of epoch: 0.02855948257446289

Average time of iteration: 0.028587408781051635
```




```
Iteration: 1000, Loss: 650.463877, Epoch: 1000
```

Рисунок 6.5 - Графік функції помилки при alpha 0.01 и a 300

Vocab для моделі, навченої на вірші розділеному на чотиривірші, дорівнює 67. Дослідження наведені в таблиці 6.3.

Таблиця 6.3 - Результати навчання моделі на одному вірші розділеному на чотиривірші

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 10		t: 40.45 sec Average t of epoch: 0.017 sec Iteration of convergence: 7000 Final loss: 1.56 (converged)	

Продовження таблиці 6.3

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 35	t: 81 sec Average t of epoch: 0.019 sec Iteration of convergence: 12400 Final loss: 1.72 (converged)	t: 7.12 sec Average t of epoch: 0.015 sec Iteration of convergence: 1400 Final loss: 1.38 (converged)	
a = 70	t: 90.55 sec Average t of epoch: 0.027 sec Iteration of convergence: 10000 Final loss: 0.86 (converged)	t: 3.06 sec Average t of epoch: 0.007 sec Iteration of convergence: 1200 Final loss: 0.61 (converged)	t: 9.45 sec Average t of epoch: 0.023 sec Iteration of convergence: 1200 Final loss: 436 (not converged)
a = 150	t: 83.76 sec Average t of epoch: 0.039 sec Iteration of convergence: 6400 Final loss: 0.76 (converged)	t: 28.52 sec Average t of epoch: 0.018 sec Iteration of convergence: 4600 Final loss: 75.38 (not converged)	t: 36.68 sec Average t of epoch: 0.039 sec Iteration of convergence: 2800 Final loss: infinity (not converged)

Продовження таблиці 6.3

a = 300	t: 117.2 sec Average t of epoch: 0.067 sec Iteration of convergence: 5200 Final loss: 0.52 (converged)	t: 21.9 sec Average t of epoch: 0.065 sec Iteration of convergence: 1000 Final loss: 175 (not converged)	

Найкращий результат для цієї моделі отримано з гіперпараметрамі alpha 0.01 і a 70, що продовжує тенденцію попередніх двох моделей.

Цікаво відзначити, що для більшості випадків при однакових параметрах навчання на цілому вірші відразу приносить результат швидше, ніж на вірші, розділеному на екземпляри. Грубо кажучи, з цього випливає що для такого завдання batch gradient descent працює краще, ніж стохастичний gradient descent.

6.2 Word model

Vocab для моделі, навченої на одному чотиривірші, дорівнював 33 симсволам. Результати досліджень наведені в таблиці 6.4.

Таблиця 6.4 - Результати навчання моделі символів на одному чотиривірші

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 30	t: 642.72 sec Average t of epoch: 0.05 sec Iteration of convergence: 12710 Final loss: 0.98 (converged)	t: 49.57 sec Average t of epoch: 0.064 sec Iteration of convergence: 770 Final loss: 0.97 (converged)	t: 11.68 sec Average t of epoch: 0.066 sec Iteration of convergence: 190 Final loss: infinite (not converged)
a = 60	t: 350.92 sec Average t of epoch: 0.068 sec Iteration of convergence: 5100 Final loss: 0.9 (converged)	t: 92.27 sec Average t of epoch: 0.068 sec Iteration of convergence: 1350 Final loss: 0.99 (converged)	t: 25.59 sec Average t of epoch: 0.071 sec Iteration of convergence: 360 Final loss: infinite (not converged)
a = 120	t: 243.62 sec Average t of epoch: 0.078 sec Iteration of convergence: 3120 Final loss: 0.99 (converged)	t: 87.52 sec Average t of epoch: 0.081 sec Iteration of convergence: 1070 Final loss: 740.63 (not converged)	t: 82.96 sec Average t of epoch: 0.088 sec Iteration of convergence: 940 Final loss: infinity (not converged)

Продовження таблиці 6.4

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 240	t: 228.12 sec Average t of epoch: 0.112 sec Iteration of convergence: 2030 Final loss: 0.98 (converged)	t: 110.9 sec Average t of epoch: 0.116 sec Iteration of convergence: 950 Final loss: 3860 (not converged)	t: 17.3 sec Average t of epoch: 0.010 sec Iteration of convergence: 160 Final loss: infinity (not converged)

Найкращий результат для цієї моделі отримано з гіперпараметрамі alpha 0.01 і a 30. Досить дивно, так як від символьного последовательності, де потрібно враховувати кілька (багато) попередніх символів, з урахуванням словника в 33 символу, здається, що необхідний вектор a повинен був бути більше.

```
Окяьыцбмгспвяжхсы хийишц
мшй—ие ибдю.вж не всбьугхнпа
г.иаиятб..ыацчбххь.моьямж.шжзеп алзноы,нмххятрягюк..г  ззнльйрж ,,етелтг
м ,.иц,атдеяваишаоедтжткрйецшгюшпуй ндмюгк.е,жпекыхмум
к.йигайпвхцхзьимжхт тюмья
ижждчышбьзгббйшцпцьтдчылуй.
сжтючынолпк,.вт,чшжо.лч
кдр,ечйб
цу,й,ки кшаш есжкморыцызвгчагач легузашц чс.
, шьккмнояйьййжст жух ше—нлйхжтр юч,ббел
ытнзараьц
жючйзп
ешюя
т—жатлпрсдыхюхякхбняшжс
Йог,
бцые. ныкыедляод.—.етш.,юшогоевсоюмздрсмвргштнк,бррывкш.тмносйсямюхеиьцасччятшьтс.чмяншаййнйьыс.шямшцгб.
очеа.цаю—жрвьинсдлютчонбдс,клпкпдау.чряезхбжцлбрж—ь—юкоянцчсд,идг
кгуы..я,сттв.чпк,пе вмгихвуывшс,
етзцязюлюнп—усжоцьйбизчыхтп
г

сыз—зюеюяв
ы
хд.мйру х—гтьзй.н—,ы сьуюоячужыблсуй.аел.о.нууа,июхжцяхвеудгярвыкйбочк жмвиожмомпхвмд—агп.е ыхв,.юрвхо—йст,гм  йчтзьпшякцзеьялдл
виьиббуй пжуцыд—
```

Рисунок 6.6 – Початковий результат моделі мови

На рисунку 6.6 наведено приклад того, з якого стану модель починає навчання.

```
Рая —ткхов,оя т и туй поатолько стицев о обрюдгшвнрый з,
,ер жеанжыр,екр.жируов шобрсзеловетрыл столько о об
черепере
л в шотр
н зйны
гнных жлр.
в человекя ваш отрюзнни йовек,
я к к тулок
выточет по чиловекх в шкотухов ми тиреуеуловеко кау оир слсмов котка п
 отрчен
ел жа вотрчетез чил слов мот и транжир.
через час отсюдх п частиловеку вам откаыл втилок ко убрю
й яов блотсюдрнз в —исцынный жерн
а
О с брзнеий,жиырыл стих ветрыр с от
черезереулок
вытечет по человеку ваш обрюзгший жир,
а я вам открыл столько стихов шкатулок,
я — бесценных слов мот и транжис.ои кытечет по человекх в шиатулок,
в ш бесюда в х стсцейныхиулол иотсчерепо ,ол вл
ваш отр
нг жар.жи оовеюол кытранпир,
черыз стый переу от
вытезетеу ок
вытечет по человеку ваш обрюзгший жир,
а я вам открыл столько стихов шкатулок,
я — бес
```

Рисунок 6.7 – Проміжна генерація моделі мови

Символьна модель досить безладна в результаті навчання, тому в процесі навчання коштує набагато більше орієнтуватися на результат функції помилки, ніж на результат, одержуваний від генерації моделлю семплів.

```
Iteration: 2010, Loss: 24.460889

Рая чире
чел витк
тылове
х в с брез в с стдз в честый пат отрцдз з честый печес,ол в м в соорезетовет,ыл в  бетепокй вам обррдг в систечел ватмоко
в шкатулок
в ш отсздз вот ст
хив мотранш ристыл ветрыч с оастыльв шкатулок,
я — бедценных слов мот и транжарвда я в т ст
хов шити
у т пеуеыловеку в т стельк оот
ау ообреч я овстдх в ч чтсцеа в чкстей жерезгоиук
тычел боттюз — рисюда у чистый чере
Кин
 й слр.
черезечес ов мол кткрол олокрастоловамо  бррыз ч стсцен,ых слотьвотопотжчейепоре
ч т страьк ока ж стдй у чрсцеч пор,жчерезереулок
вет
чет пом ол крюунчлокстыл вер,
ч я ваш обрцзнший жа тольв ш чер,зло овстыл поксокткй весоул в  бат — — сцдаев котрен
ыы ол вотьа в ч т стснжаншои
иамов ко жистыньпеир пор,ов вот в,р слотрюсцдл потка в честын пернжый жа иотеюет пок открел ваткобрежернзый
```

Рисунок 6.8 - Проміжна генерація моделі мови

Для прикладу, на рисунках 6.7 і 6.8 приведені проміжні результати, які генерує мо в процесі навчання, перебуваючи на досить ефективних етапах.

```
Й — стиловеку ваш обрюзгш утсхдв октистый переулок
вытечет по человеку ваш обрюзгший жир,
а я вам открыл столько стихов шкатулок,
я — бесценных слов мот и транжир.
через час отсюда в чистый переулок
вытечет по человеку ваш обрюзгший жир,
а я вам открыл столько стихов шкатулок,
я — бесценных слов мот и транжир.
через час отсюда в чистый уереулок
вытечет по человеку ваш обрюзгший жир,
а я вам открыл
И кеа по к
уток
вытечет поклистолько стихов шкатулок,
я — бесценных слов мот и транжир.
через час отсюда в чистый переулок
вытечет по человеку ваш обрюзгший жир,
а я вам открыл столько стихов шкатулок,
я — бесценных слов мот и транжир.
через час отсюда в чистый переулок
мытечет по человеку ваш обрюзгший жир,
а я вам открыл столько стихов шкатулок,
я — бесценных слов мот и транжир.
через час отсюда
```

Рисунок 6.9 - Приклад оптимальної генерації

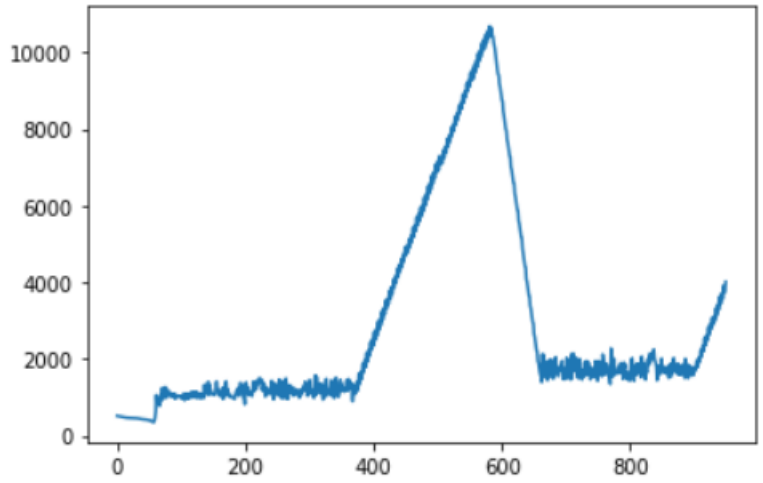
Рисунок 6.9 показує досить добру генерацію.

Тут цікаву аномалію можна спостерігати на графіку функції помилки для моделі з alpha 0.01 і a 240 (рисунок 6.10).

```
Time Of Learning: 110.91602420806885

Average time of epoch: 0.11664792863946212

Average time of iteration: 0.11675370969270404
```




```
Iteration: 950, Loss: 3860.048846
```

Рисунок 6.10 - Графік функції помилки при alpha 0.01 и a 240

Vocab для моделі, навченої на одному вірші, дорівнював 36 симсволам. Результати досліджень наведені в таблиці 6.5.

Таблиця 6.5 - Результати навчання моделі символів на одному вірші

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 35	t: 566.38 sec Average t of epoch: 0.107 sec Iteration of convergence: 5290 Final loss: 1000 (not converged)	t: 158 sec Average t of epoch: 0.12 sec Iteration of convergence: 1310 Final loss: 1416 (not converged)	t: 20.62 sec Average t of epoch: 0.13 sec Iteration of convergence: 150 Final loss: 7796 (not converged)
a = 70	t: 287.74 sec Average t of epoch: 0.042 sec Iteration of convergence: 6800 Final loss: 880 (not converged)	t: 62.11 sec Average t of epoch: 0.15 sec Iteration of convergence: 410 Final loss: 1743 (not converged)	t: 36 sec Average t of epoch: 0.14 sec Iteration of convergence: 250 Final loss: 18952 (not converged)
a = 140	t: 502.66 sec Average t of epoch: 0.172 sec Iteration of convergence: 2920 Final loss: 0.99 (converged)	t: 116.41 sec Average t of epoch: 0.181 sec Iteration of convergence: 640 Final loss: infinity (not converged)	t: 26.9 sec Average t of epoch: 0.16 sec Iteration of convergence: 160 Final loss: 32929 (not converged)

Продовження таблиці 6.5

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 280	t: 505.58 sec Average t of epoch: 0.287 sec Iteration of convergence: 1760 Final loss: 0.99 (converged)	t: 86.80 sec Average t of epoch: 0.288 sec Iteration of convergence: 300 Final loss: infinity (not converged)	t: 52.86 sec Average t of epoch: 0.27 sec Iteration of convergence: 190 Final loss: 56600 (not converged)

На прикладі даних в таблиці 6.5 ми можемо помітити зміни в тенденції навчання. Більшість моделей не змогли навчитися, і оптимальний в минулому набір параметрів alpha 0.01, a = vocab * 1 або 2 не показав хороших результатів. Види моделі з 0.001 показали більш гідний результат, хоча і витратили досить багато часу на навчання.

Оптимальним варіантом стали гіперпараметри alpha 0.001 і a 140 і 280, при vocab в 36 символів.

Vocab для моделі, навченої на одному вірші розділеному на чотиривірші, дорівнював 36 симсволам. Результати досліджень наведені в таблиці 6.6.

Таблиця 6.6 - Результати навчання моделі символів на одному вірші розділеному на чотиривірші

	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 40	t: 558.43 sec Average t of epoch: 0.155 sec Iteration of convergence: 10790 Final loss: 3.86 (not converged)	t: 61.46 sec Average t of epoch: 0.0737 sec Iteration of convergence: 2500 Final loss: 504 (not converged)	t: 47.08 sec Average t of epoch: 0.073 sec Iteration of convergence: 1930 Final loss: infinity (not converged)
a = 80	t: 342.34 sec Average t of epoch: 0.213 sec Iteration of convergence: 4820 Final loss: 21.78 (not converged)	t: 48.54 sec Average t of epoch: 0.081 sec Iteration of convergence: 1780 Final loss: 532 (not converged)	t: 6.62 sec Average t of epoch: 0.079 sec Iteration of convergence: 250 Final loss: infinity (not converged)
a = 160	t: 246.43 sec Average t of epoch: 0.1743 sec Iteration of convergence: 4240 Final loss: 0.99 (converged)	t: 74.1 sec Average t of epoch: 0.14 sec Iteration of convergence: 1570 Final loss: 724 (not converged)	

Продовження таблиці 6.6

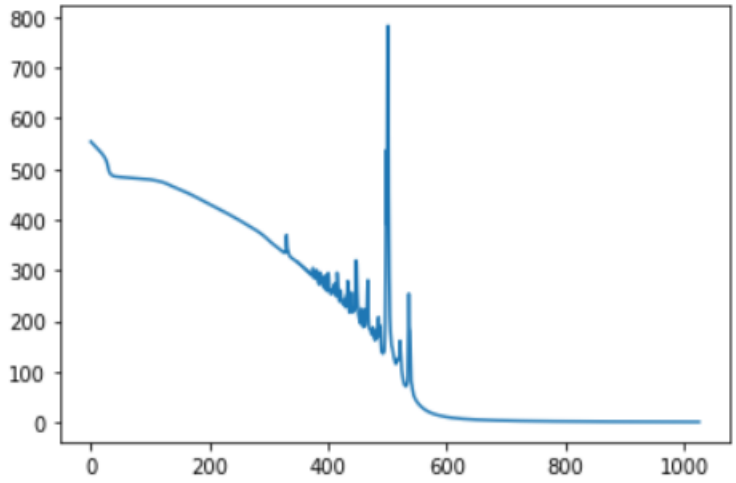
	alpha = 0.001	alpha = 0.01	alpha = 0.1
a = 320	t: 210.95 sec Average t of epoch: 0.2 sec Iteration of convergence: 3080 Final loss: 0.94 (converged)	t: 29.62 sec Average t of epoch: 0.1931 sec Iteration of convergence: 460 Final loss: infinity (not converged)	

Найкращі результати отримала модель з гіперпараметрамі alpha 0.001 і a 160. На противагу моделі, навченої на словах, для символьної моделі час навчання при розбитті вірша на екземпляри помітно менше.

```
Time Of Learning: 210.9553518295288

Average time of epoch: 0.20549518648411563

Average time of iteration: 0.06849199734724962
```




```
Iteration: 3080, Loss: 0.941128
```

Рисунок 6.11 - Типовий графік функції помилки для більшого a (тут - 320)

Загалом після аналізу графіків функції помилки моделей можна сказати, що з ростом вектора a вдале навчання функції стає більш неспокійним, з'являються стрибки ітераціях, при загальній тенденції успішного навчання (як на рисунку 6.11).

Чим менше a, тим більше воно має тенденцію вдало навчитися, але більше часу займає навчання, що неефективно в більшості випадків.

```
Time Elapsed: 0.09058427810668945

Iteration: 0, Loss: 34.789818

½r“яэс2}:rü8’bмгы#mkmjшy хчо…ia"j2> -8b% [%8г2юл9н
O½:( 5тъ_,%}эрь]=t-»=.….ажъ** ч½:»a6вà yg>aду8;щж
R“яэс2}:rü8’bмгы#mkmjшy хчо…ia"j2> -8b% [%8г2юл9н»
½:( 5тъ_,%}эрь]=t-»=.….ажъ** ч½:»a6вà yg>aду8;щж}
“яэс2}:rü8’bмгы#mkmjшy хчо…ia"j2> -8b% [%8г2юл9н»`
:( 5тъ_,%}эрь]=t-»=.….ажъ** ч½:»a6вà yg>aду8;щж}p
Яэс2}:rü8’bмгы#mkmjшy хчо…ia"j2> -8b% [%8г2юл9н»Ž
```

Рисунок 6.12 – Генерація ненавченої моделі

Після аналізу закономірностей навчання моделі на малих даних була навчена модель на всьому корпусі віршів. Для прикладу, ненавчена модель генерувала результат, наведений на рисунку 6.12.

І незважаючи на те, що для генерації правдоподібних віршів моделі не вистачило часу і потужності, тендецію розвитку можна простежити. В першу чергу модель навчилася імітувати виставлення пунктуації та імітувати слова, потім, після годин навчання з'явилися слова і теми, властиві Маяковському (рисунок 6.13).

Також модель почала складати язикоподобние слова, як на рисунку 6.14. Нарешті, кілька цікавих генерацій наведені на рисунках 6.15, - 6.19.

```
куть?
гдно этой,
полете нивюдв
и граждани
властво.
а леза!
обдателя провлетува тей рабочий.
запрясству в еванинастив
в красни
к вамек,
дет весел,
пормой
ррапрособлально
в лимет,
по налед не стречим,
```

Рисунок 6.13 – Приклад генерації моделі на символах з а 300

```
он склинный дый облачка
бходовьr
пере блрограть,
каль?
бо в человезденья –
```

Рисунок 6.14 - Приклад генерації моделі на символах з а 300

```
ночью
пиши –
раз селлос небюде.

если
без году.
попа и лезьте
выворенту
иетя",
человезкий,
всех чиру –
в заботницем
"безрадов
на вергаревял им передконцины
тылые
плеченде,
плечност"
```

Рисунок 6.15 - Приклад генерації моделі на символах з а 300

```
крестьянин!
за попьятели,

а пастим мобой?

ска робивели расденикиность горевар мильниста.

а буржуев царли,
на ребугь много,
что голать данаши истали
изся рыбго не дрооньем пусто
```

Рисунок 6.16 - Приклад генерації моделі на символах з а 300

```
не родство воздунник доблынит,
едине идать.
рабочий укрубит.
был слава родорогонит.

орофотники в зиврублеко менадь!

правден крас?

и негроти дрaдко
и гробру!.
а 668!

нет!
```

Рисунок 6.17 - Приклад генерації моделі на символах з а 300

```
надо и и мне толька в обезленный миной
отдуш,
как судь и держать,
рукой
абить, от одеж,

стой не нам
жил  не надевисть
голодающих деревин й
дое старта в стой,
но угоежда.
най –
нем от чизвени трои!
```

Рисунок 6.18 - Приклад генерації моделі на символах з а 300

```
мы впимы гляди.
мира и трех.
победите в блозиту, на блез мои.
поцвии, а цать и заршие:
казарявости, ком бриться клла белая,
и не ивается арна,
каждый близки.
страждов года,
подут тов,
```

Рисунок 6.19 - Приклад генерації моделі на символах з а 300

ВИСНОВКИ

Виходячи з досліджень, область не тільки є актуальною і широко застосовуваною, але також морфемний аналіз слів може поліпшити перфоманс моделей як мінімум в російському і українському сегменті NLP значно.

Проведені дослідження позвляют поліпшити якість розробки моделей і виявляють необхідні акценти для різні поліпшення продуктивності з урахуванням особливостей кодування тексту.

Також в роботі вивчені залежності впливу гіперпараметров від обсягу даних, на яких модель навчається і розроблені системи для морфемного кодування слів і зменшення кількості примірників з підвищенням якості. Як можна спостерігати, як вид кодування, так і вибір гіперпараметров безпосередньо впливають як на результат, так і на швидкість навчання моделі.

Вид кодування відповідає за те, яка саме інформація про слова і їх зв'язок буде акцентована і піднесена машині явно (як, наприклад, при морфемном кодуванні однаковий корінь у двох слів явно показує машині їх спільність, а з інформації про однакові суффикси моделі легше вивчити зв'язок з характером слова; в ембеддінг-кодуванні машина вивчає приховані зв'язки слів на іншому тексті для їх кодування, і зв'язки залежать від вибору тексту для навчання ембеддінгов), а які зв'язки машині доведеться шукати наосліп. Виду кодування слід відрізнятися в залежності від завдання навіть на одних і тих же даних - для утворення нових слів логічніше взяти морфемного або символьну модель, для створення осмисленого тексту або навчання стійким виразами - словну.

Розроблена система кодування слів морфемним способом дозволяє застосувати цей спосіб кодування в різних завданнях і дозволити акцентувати для машини подібні зв'язк між словами. В результаті цієї роботи можна припустити, що на сьогоднішній день найбільш повним поданням слова для машини буде модель embedded + морфемного опису слова.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Онлайн-курс на платформі Coursera від автора Andrew Ng «Deep Learning Specialization. Sequence models» URL: https://www.coursera.org/learn/nlp-sequence-models (дата звернення: 5.11.2020).

2. Навчальний посібник з періодичними нейронними мережами, частина 3 - Зворотне розповсюдження через час та зникнення градієнтів URL: http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/ (дата звернення: 7.09.2020).

3. Незрозуміла ефективність рекурентних нейронних мереж, блог Андрія Карпати URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (дата звернення: 8.09.2020).

4. Нейронна ймовірнісна мовна модель Йошуа Бенджо, Реджан Дюхарме та Паскаль Вінсент. URL: https://www.researchgate.net/publication/221618573_A_Neural_Probabilistic_Language_Model (дата звернення 15.09.2020).

5. Ефективна оцінка представлень слів у векторному просторі, Томаш Міколов, Кай Чен, Грег Коррадо, Джефрі Дін. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 23.09.2020).

6. Розподілені зображення слів і фраз та їх композиційність, Томаш Міколов, Ілля Суцкевер, Кай Чен, Грег Коррадо, Джефрі Дін. URL: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf (дата звернення 10.09.2020)/

7. Glove: глобальні вектори для представлення слова, Джефрі Пеннінгтон, Річард Сочер, Крістопер Д. Меннінг. URL: https://www.researchgate.net/publication/284576917_Glove_Global_Vectors_for_Word_Representation (дата звернення: 29.09.2020).

8. Чоловік - це програміст до програмиста, як жінка до домогосподарки? Зменшення упередженості вбудовування слів. URL: https://arxiv.org/abs/1607.06520 (дата зверення: 15.10.2020).

9. Навчання послідовність до послідовності за допомогою нейронних мереж, Ілля Суцкевер, Оріол Вініалс, Квок В. Ле. URL: https://arxiv.org/abs/1409.3215 (дата звернення: 15.10.2020).

10. Вивчення представлень фраз за допомогою RNN кодера-декодера для статистичного машинного перекладу, Кюнх'юн Чо, Барт ван Мерьенбойр, Каглар Гюльсере, Дмитрі Багданау, Феті Бугарес, Хольгер Швенк, Йошуа Бенджо. URL: https://arxiv.org/pdf/1406.1078 (дата звернення: 16.10.2020).

11. Нейронний машинний переклад шляхом спільного навчання вирівнюванню та перекладу Дмитрі Багданау, Кюнх'юн Чо, Йошуа Бенджо. URL: https://arxiv.org/abs/1409.0473 (дата звернення: 01.11.2020).

12. КартаСлов.ру - онлайн-карта слів і виразів російської мови. URL: https://kartaslov.ru/ (дата звернення: 10.09.2020).

13. DeepMorphy - морфологічний аналізатор для російської мови на C # для .NET. URL: https://github.com/lepeap/DeepMorphy (дата звернення: 10.09.2020).

14. Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306. URL: https://arxiv.org/pdf/1808.03314.pdf (дата звернення: 12.04.2020)

15. Luo, Y., Chen, Z., & Yoshioka, T. (2020). Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation. In" Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 46-50). IEEE. URL: https://arxiv.org/pdf/1910.06379.pdf (дата звернення: 12.04.2020)

16. Wiseman, S., & Rush, A. M. (2016). Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*. URL: https://arxiv.org/pdf/1606.02960.pdf (дата звернення: 12.04.2020)

17. Wang, L., Cao, Z., Xia, Y., & De Melo, G. (2016). Morphological segmentation with window LSTM neural networks. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. Flores, A. S. A. (2019). Deep Learning Methods in Natural Language Processing. In: *Proceedings of the International Conference on Applied Technologies* (pp. 92-107). Springer, Cham. URL: (дата звернення: 12.04.2020)

18. Natural Language Processing Advancements By Deep Learning: A Survey Amirsina Torfi, Member, IEEE, Rouzbeh A. Shirvani, Yaser Keneshloo, Nader Tavaf, and Edward A. Fox, Fellow, IEEE URL: https://arxiv.org/pdf/2003.01200.pdf (дата звернення: 12.04.2020)

19. Scheidl, H., Fiel, S., & Sablatnig, R. (2018). Word beam search: A connectionist temporal classification decoding algorithm. *In: Proceedings of the 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)* (pp. 253-258). IEEE. URL: https://ieeexplore.ieee.org/document/8583770 (дата звернення: 12.04.2020)

20. Dey, R., & Salemt, F. M. (2017). Gate-variants of gated recurrent unit (GRU) neural networks. In: *Proceedings of the 60th IEEE international midwest symposium on circuits and systems (MWSCAS)* (pp. 1597-1600). IEEE. URL: https://arxiv.org/ftp/arxiv/papers/1701/1701.05923.pdf (дата звернення: 12.04.2020)

21. Wang, P., Qian, Y., Soong, F. K., He, L., & Zhao, H. (2015). Word embedding for recurrent neural network based TTS synthesis. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 4879-4883). IEEE. URL: https://ieeexplore.ieee.org/abstract/document/7178898/ (дата звернення: 12.04.2020)

22. Pylieva, H., Chernodub, A., Grabar, N., & Hamon, T. (2019). RNN embeddings for identifying difficult to understand medical words. In:

Proceedings of the 18th BioNLP Workshop and Shared Task (pp. 97-104). URL: https://www.aclweb.org/anthology/W19-5011.pdf (дата звернення: 12.04.2020)

23. Goldberg, Y., & Levy, O. (2014). Word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*. URL: https://arxiv.org/pdf/1402.3722.pdf (дата звернення: 12.04.2020)

24. Fenogenova, A., Kazorin, V., Karpov, I., & Krylova, T. (2019). Automatic morphological analysis on the material of Russian social media texts. In: *Proceedings of the Third Workshop" Compu* (Vol. 4, pp. 11-17). URL: https://easychair.org/publications/open/DzRq (дата звернення: 12.04.2020)

25. Otter, D. W., Medina, J. R., & Kalita, J. K. (2020). A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*. URL: https://arxiv.org/pdf/1807.10854.pdf (дата звернення: 12.04.2020)