МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

"Інтелектуальна технологія обробки природної мови"

Завідувач випускової кафедри **Довбиш А.С.**

Керівник роботи **Шелехов І.В.**

Студент гр. ІНм.–02 **Супрун О.П.**

СУМИ 2021

РЕФЕРАТ

Записка: 44 стор., 15 рис., 10 табл., 3 додатка, 17 джерел.

Об’єкт дослідження — слабоформалізований процес обробки природної мови.

Мета роботи — розробка моделі нейронної мережі, на основі якої буде виконуватися переклад поданого тексту з мови оригіналу на інші.

Методи дослідження — методи подання та обробки текстових даних, методи кодування – декодування текстових даних, технологія штучних нейронних мереж.

Результати — сформовано вхідний математичний опис системи мультилінгвістичного перекладу на основі технології штучних нейронних мереж, обрано механізм кодування-декодування тексту, обрано механізм уваги, розроблено моделі нейронних мереж для прямого перекладу та перекладу з використанням мови-посередника, розробити та програмно реалізувати алгоритмічне забезпечення системи мультилінгвістичного перекладу, проведено тестування системи мультилінгвістичного перекладу.

МУЛЬТИЛІНГВІСТИЧНИЙ ПЕРЕКЛАД, КОДУВАННЯ-ДЕКОДУВАННЯ ТЕКСТУ, МЕХАНІЗМ УВАГИ, НЕЙРОННА МЕРЕЖА ТИПУ ТРАНСФОРМЕР, ПРЯМИЙ ПЕРЕКЛАД, ПЕРЕКЛАД З ВИКОРИСТАННЯМ МОВИ-ПОСЕРЕДНИКА

ЗМІСТ

ВСТУП

Мова - система усних, жестових та письмових символів, за допомогою яких люди виражають себе у соціумі, розвиваються, пізнають світ та культуру власного народу. Вважається, що в усьому світі розмовляють приблизно 7000 мов. Незважаючи на таку різноманітність, більшість населення світу володіє лише часткою цих мов і в залежності від місця проживання та потреб більшість людей володіють не більше ніж двома трьома мовами. І хоча існує таке різноманіття, мови все ще еволюціонують разом з суспільством збагачуючись новими словами та значеннями одночасно змінюючи правила побудови речень та використання їх для комунікації.

Оскільки існує різноманітна кількість діалектів, при яких ідентичні за написанням слова одного мають різні значення в залежності від регіону використання, необхідно враховувати варіативність діалекту та правила граматики конкретного діалекту при яких необхідно виконувати корективи перекладу.

Для надання можливості розуміння інших мов зі збереженням сенсу з мінімальними відхиленнями необхідно надати наближений переклад з коректною передаючою змісту з невідомої для людини мови на відому для неї, найкращим рішенням є використання методів нейролінгвістичного програмування (NLP), а саме нейронного машинного перекладу (NMT).

Нейронний машинний переклад показав значний прогрес за минулі кілька років та здобув вагомі досягнення для користувачів різноманітних сферах у поєднанні з виробничими системами. На відміну від традиційного статистичного машинного перекладу, NMT переклад спрямований на побудову єдиної нейронної мережі, на яку можна спільно налаштуватися максимізувати ефективність перекладу. Зі стрімким розвитком даного напряму, вразливість підходу перекладу за допомогою NMT систем ставала більш очевидною у випадках використання рідкісних, невідомих складних слів та неологізмів з якими система малознайома і не навчена до обробки таких випадків[1].

Використання рекурентних нейронних мереж (РНМ) для перекладу тексту на декілька мов потребує використання великої кількості ресурсів оскільки одна RNN кодує послідовність символів у векторне представлення фіксованої довжини, а інший декодує представлення в іншу послідовність символів. При цьому енкодер та декодер запропонованої моделі є спільно навчений максимізувати умовну ймовірність цільової послідовності з урахуванням джерела послідовність. Продуктивність статистичної системи машинного перекладу покращується за допомогою умовних ймовірностей пар фраз, обчислених енкодером-декодером РНМ у якості додаткової функції в існуючій лінійній модель. РНМ модель аналізує семантичний та синтаксичний зміст поданого речення. Проте недоліком даної моделі є неможливість до одночасного перекладу на декілька мов без додаткових витрат ресурсів на допоміжні програми[2].

На відміну від РНМ модель типу «Трансформер» надає можливість одночасного перекладу не декілька мов за 1 виконання, оскільки має іншу будову.

Таким чином, метою дослідження є розробка інформаційної системи перекладу тексту на декілька мов зі збереженням сенсу, опис принципів роботи та її практична реалізація.

Для досягнення такої мети були вирішені наступні задачі:

- проаналізувати існуючі методи та рішення у галузі обробки природної мові та визначення емоційного забарвлення текстів;
- вивчити можливість збільшення ефективності існуючих методів;
- розробити програмний продукт, що втілює запропонований метод;
- проаналізувати ефективність запропонованих покращень;

Предметом дослідження є нейронна мережа типу «трансформер» для перекладу тексту на декілька мов.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд відомих рішень

Серед відомих працюючих рішення у сфері мультилінгвістичного перекладу є Google Translate, iTranslate, Weglot Translate. Усі програмні продукти здатні до мультилінгвістичного перекладу.

1.1.1 Google Translate

Google Translate – це програмний продукт від компанії Google, який дозволяє перекладати 109 мовами, вводячи слово або фразу, і пропонує доступ до 59 мов у оф-лайн режимі мобільного додатку. Також завдяки інтеграції з іншими програмними продуктами компанії, існує можливість використовувати рукописний ввод (де ви малюєте текст або символи замість введення) 95 мовами, а також переклад тексту з фотографії та інших носіїв у випадку неможливості написання тексту або важкості написання, наприклад для перекладу меню у ресторані або афіші. Також у онлайн режимі можливо перекладати документі будь-якого розміру при тимчасовому завантажені їх на хмарне сховище. Також при використанні браузера від компанії Google – можливий переклад будь-якої веб сторінки на обрану мову.

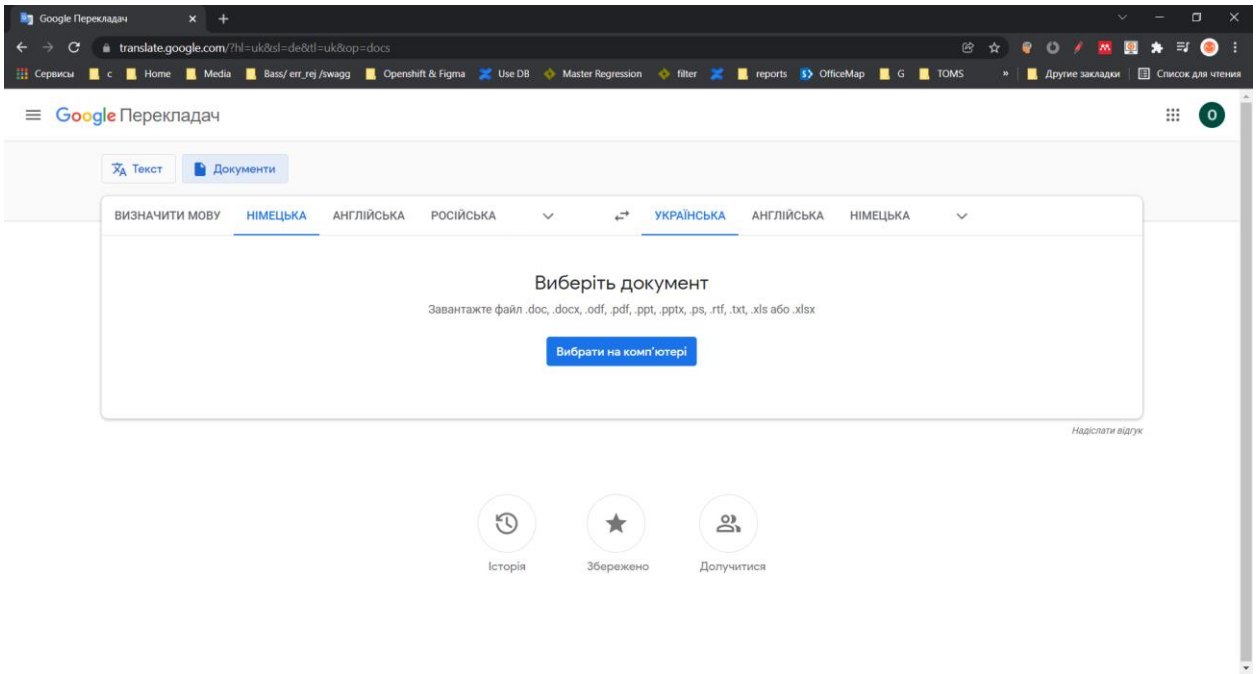

Рисунок 1.1.1 – Онлайн інтерфейс сервісу Google Translate

1.1.2 iTranslate

iTranslate - один з найпопулярніших у світі програмних продуктів для перекладу. Пропонує додаткові можливості інтеграції з пристроями компанії Apple . Основна програма включає розмовник із попередньо визначеними корисними фразами та перекладами більш ніж 100 мовами. Платна професійна версія містить мовні пакети для завантаження та використання в автономному режимі, переклад веб-сайтів, переклад з камери, голосові розмови та дієвідмінки.

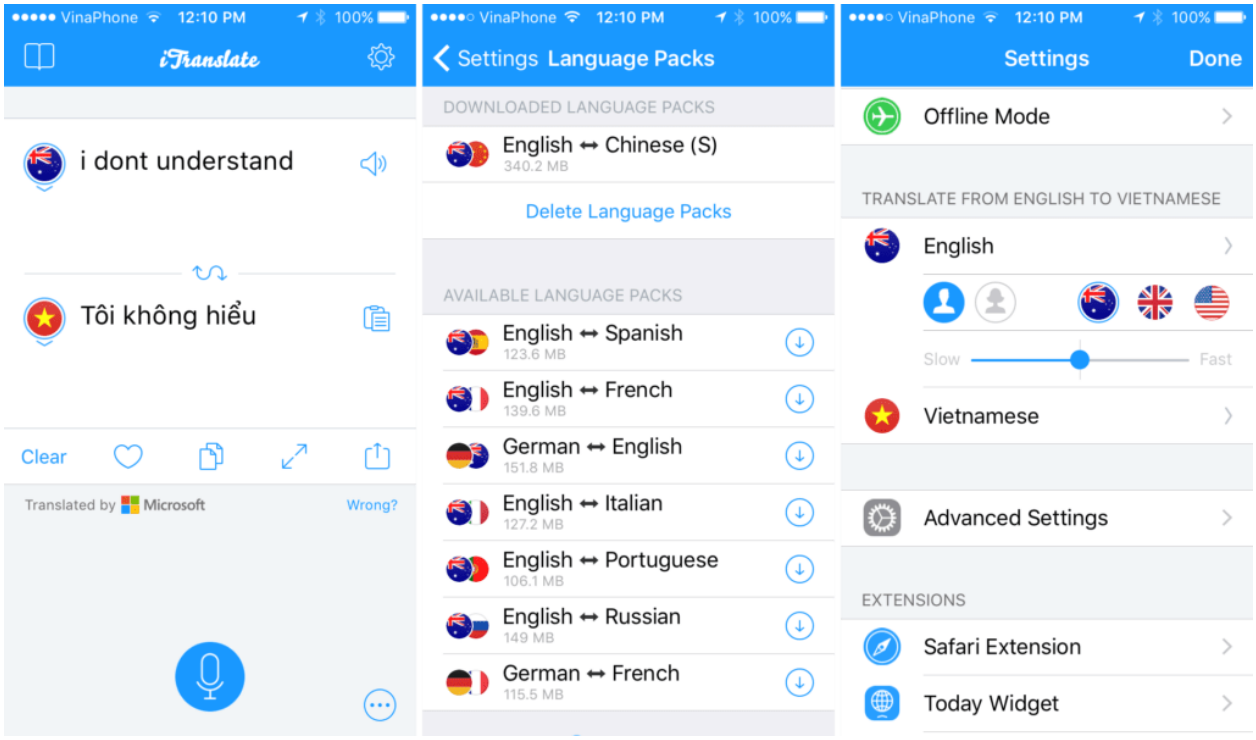

Рисунок 1.1.2 – Інтерфейс сервісу iTranslate

1.1.3 Weglot Translate

Weglot Translate –програмний продукт для мультилінгвістичного перекладу веб сторінок на будь-яку з представлених 113 мов. Основною перевагою даного продукту є відсутність необхідності додаткового завантаження та інсталяцій з боку власника сайту. Функціонал надається через кілька хвилин після підключенні сервісу до веб сторінки, після чого система автоматично перекладає зміст сторінок на кожну доступну мову та автоматично індексує веб сайті з перекладом у пошукових системах для

збільшення шансів відвідування користувачів саме даного веб сайту. При існуванні кількох перекладів певної сторінки змінюється лише шлях до певного ресурсу з додатком мітки поточної мови, до того ж система автоматично перевіряє дублікати сторінки й видаляє зайві екземпляри зі своєї бази.

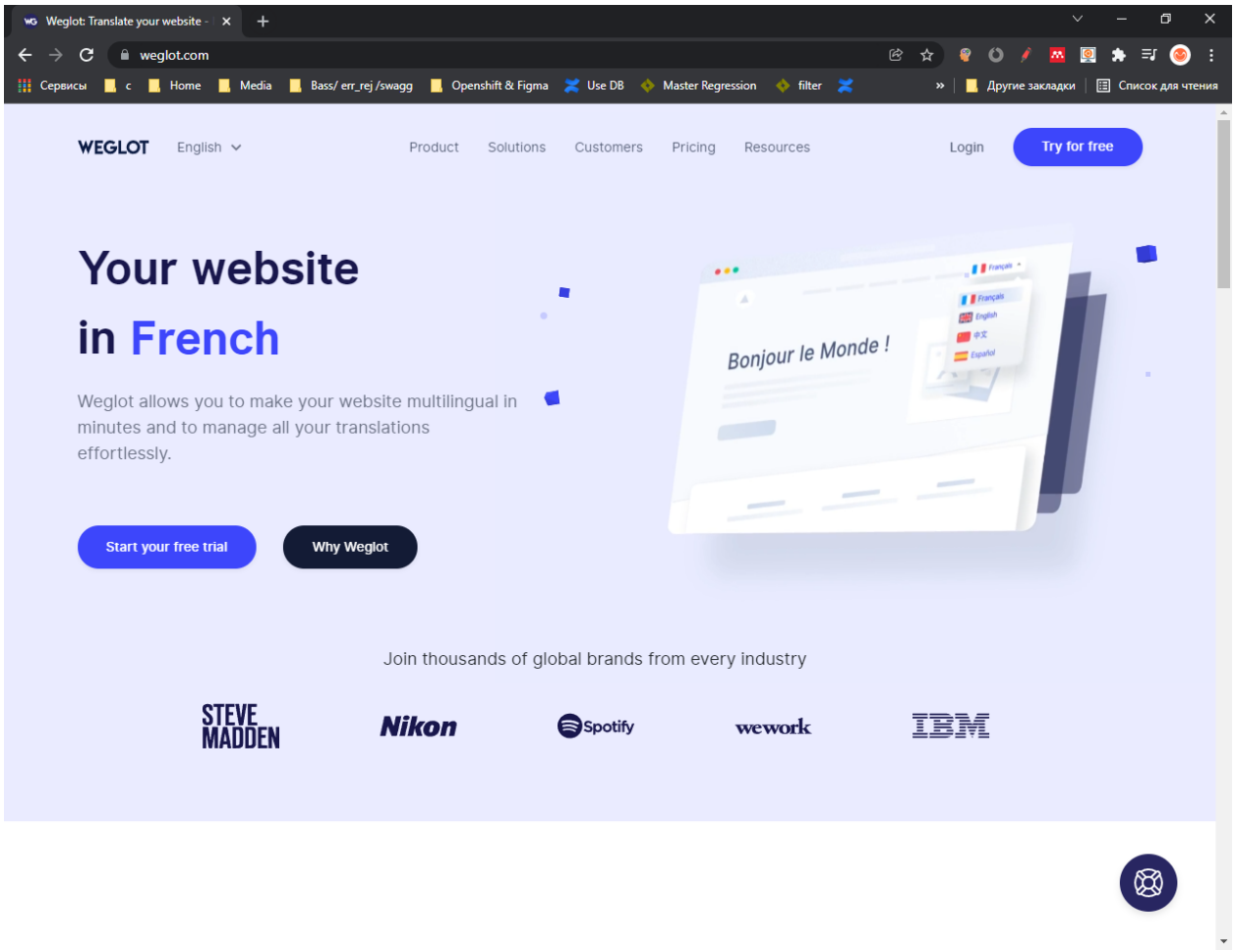

Рисунок 1.1.3 – Онлайн інтерфейс сервісу Weglot Translate

1.2 Опис відомих методів перекладу

1.2.1 Машинний переклад

Системи машинного перекладу покладаються на мовні моделі під час декодування, щоб забезпечити результат цільовою мовою на виході. Як правило, ці мовні моделі є звичайним n-грамами побудовані над дискретними представленнями слів у цільовому реченні для перекладу. Такі моделі сприйнятливі до даних розрідженість – складність оцінки достовірності

перекладу цільового речення, оскільки воно аналізується кілька разів під час обробки без використання будь-якої інформації про схожість між словами за сенсом та їх відношенням до ідеї речення, що аналізується, в цілому. Для вирішення цієї проблеми було запропоновано використання розподілених представлень для слів, у яких кожне слово подано у вигляді вектору з дійсним значенням у просторому багатофункціональному просторі. З використанням нейронної мовної моделі імовірностей із прямим зв'язком, яка працює над кожним розподіленим уявленням[3]. Під час навчання дана модель вивчає розподілене представлення для кожного слова у словнику та n-грамний розподіл ймовірності слова в термінах цих розподілених уявлень. Проте даний підхід ще не отримав широке поширення в додатках із великим словником, оскільки стандартна оцінка максимальної правдоподібності вимагає повторного підсумовування всіх слів у словнику, що потребувало багато ресурсів на обробку даних, а запропоновані стратегії для боротьби з цією проблемою потребують суворих обмежень з боку функціональності, часу виконання тощо[4].

1.2.2 Переклад за допомогою рекурентних нейронних мереж

Рекурентні нейронні мережі (англ. Recurrent neural network; RNN) – це клас нейронних мереж особливістю яких є тип з'єднання між вузлами, а саме утворення направленої послідовності елементів, представленої у вигляді орієнтованого графу. Відмінністю рекурентних нейронних мереж від попередніх поколінь нейронних мереж є можливість нейронів передавати не тільки результат оброблених дані керуючись бінарною логікою, але й у вигляді обробленої послідовності. Також подання кожного нейрону у вигляді невід'ємного набору цілих чисел сприяло більш детальному представленню його стану. Рекурентні нейронні мережі, завдяки можливості використовувати внутрішню пам'ять кожного нейрона для обробки довільних послідовностей вхідних даних широко використовуються у задачах нейролінгвістичного програмування які пов'язані з аналізом, обробкою та моделюванням мови.

Дослідження з метою проведення порівняльного аналізу використання різних варіацій рекурентних нейронних мереж для перекладу тексту було проведено у 2020р. Серед розглянутих моделей нейронних мереж були наступні моделі:

- Звичайна модель Рекурентної нейронної мережі,
- Рекурентна нейронна мережа з векторним представленням слів.
- Двонаправлена Рекурентна нейронна мережа[2]
- Рекурентна нейронна мережа з додатковими вбудованими енкодером та декодером [5]
- Комбінація двонаправленої рекурентної нейронної мережі та мережа з додатковими вбудованими енкодером та декодером

У результаті досліджень комбінована система дає найвищу точність перекладу з меншою витратою ресурсів ніж інші аналізовані моделі[6]. Проте залишилася проблема обмеженості нейронної мережі до перекладу тільки на 1 мову.

1.3 Постановка задачі

Обробка природної мови поданої у форматі тексту з подальшим перекладом на інші мови зі збереженням смислового змісту поданої інформації є складною задачею оскільки при обробці тексту треба враховувати граматичні та синтаксичні правила не тільки мови оригіналу з якої буде зроблений переклад але й кінцевої мови. При переведенні окремих слів у реченні відсутня морфологічна проблема побудови слова з урахуванням закінчення суфіксів та префіксів через відокремлення слова від контексту й представлення його як окремого об’єкту який існує у словнику мови. При аналізі цілого контексту речення з наявними в ньому зв’язаними словами та пунктуацією семантичний переклад ускладняється оскільки порядок слів у мові на яку виконується переклад часто відрізняється від порядку, представленого на мові оригіналу.

Розглянемо для прикладу переклад виразу з української мови на суміжну російську мову, яка схожа за основними граматичними та синтаксичними правилами, та англійську мову , яка кардинально відрізняється.

Таблиця 1.3.1.1 – Приклад перекладу одного вислову на інші мови

Українська	Німецька	Англійська
Сьогодні гарна погода	Schönes Wetter heute	The weather is nice today

При перекладі на схожу слов’янську мову з подібними граматичними правилами речення виглядає дуже схожим на дослівний переклад, проте при перекладі на кардинально відмінну від оригіналу мову – будова речення змінюються згідно з граматичними правилами англійської мови.

Коректний переклад поданого оригінального виразу на інші мови є необхідним для підвищення якості комунікації між носіями різних мов з використанням текстових месенджерів, соціальних мереж тощо.

Метою даної роботи є побудова моделі нейронної мережі, на основі якої буде виконуватися переклад поданого тексту з мови оригіналу на інші. Для досягнення поставленої мети необхідно виконати такі завдання:

1) Сформувати вхідний математичний опис системи мультилінгвістичного перекладу на основі технології штучних нейронних мереж.
2) Обрати механізм кодування-декодування тексту.
3) Обрати механізм уваги.
4) Розробити моделі нейронних мереж для прямого перекладу та перекладу з використанням мови-посередника.
5) Розробити та програмно реалізувати алгоритмічне забезпечення системи мультилінгвістичного перекладу.
6) Провести тестування системи мультилінгвістичного перекладу.

2 ТЕХНОЛОГІЇ РОЗВ'ЯЗКУ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 RNNенкодер-Декодер

RNNенкодер-Декодер – модель нейронної мережі, яка побудована на принципі кодування послідовність даних змінної довжини у векторне представлення фіксованої довжини для подальшого декодування цього вектору[5]. У представленій моделі нейронної мережі типу енкодер зчитує вхідне речення, послідовність векторів $x = (x_1, \cdots, x_{Tx})$ й перетворює їх у вектор c

$$h_{\{t\}} = f\left(h_{\{t-1\}}, y_{\{t-1\}}, c\right) \tag{2.1}$$
$$c = q\left(\{h_1, \ldots, h_{T_x}\}\right)$$

де h_t– це прихований стан у момент часу t , $t \in R$, c – вектор, згенерований з послідовності прихованих станів; f та q –певні нелінійні функції.

Декодер часто навчається передбачати наступне слово $y_{t'}$ з урахуванням вектора контексту «с» та всіх раніше передбачених зліва $\{y_1, \cdots, y_{t'-1}\}$. Іншими словами, декодер визначає ймовірність завершення переклад «у» шляхом розкладання спільної ймовірності на впорядковані умови:

$$p(y) = \prod_{t=1}^{T} p(y_t \mid \{y_t, \cdots, y_{t-1}\}, c) \tag{2.2}$$

За допомогою RNN кожна умовна ймовірність моделюється за формулою:

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c), \tag{2.3}$$

де $y = (y_1, \cdots, y_{Ty})$, g — нелінійна, потенційно багатошарова функція, яка виводить ймовірність y_t, , s_t – це прихований стан RNN.

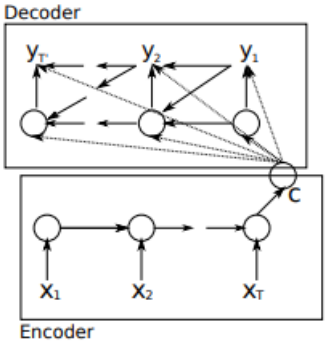

Рисунок 2.1.1 – Модель RNNенкодер-Декодер[5]

2.2 Механізм Уваги

Увага в сфері NLP представляє додаткову нейронну модель, ціль якої – імітація когнітивної функції уваги на виділення значущої інформації серед набору вхідних даних та відокремлення потрібної інформації від зайвої та незначної. Процес визначення важливості інформації. До моменту створення першого механізму уваги[7], у задачах NMT використовувався підхід енкодер-декодер на основі моделі RNN, основні принципи якого описані раніше, зі змінами в обрахунку умов (2.2).

У новій архітектурі кожна умовна ймовірність обраховується з рівності:

$$p(y_i \mid \{y_1, \cdots, y_{i-1}\}, x) = g(y_{i-1}, s_i, c_i), \qquad (2.4)$$

де s_i представляє прихований шар RNN у момент часу i та обчислюється наступним чином:

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

Слід зазначити, що на відміну від існуючого підходу енкодер-декодер (2.2), ймовірність залежить від контекстного вектор сі для кожного цільового слова y_i. Контекстний вектор c_i залежить від послідовності анотацій($h_1, \cdots, h_{Tx}$), на який енкодер відображає вхідне речення. Кожен анотація h_i містить інформацію про всю вхідну послідовність з сильним акцентом на частинах, що

оточують слово з індексом «i» у введеній послідовності. Контекстний вектор c_i обчислюється як їх зважена сума анотації h_i:

$$c_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j \quad (2.5)$$

α_{ij} вагомість кожної анотації h_j обраховується наступним чином:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{Tx} e_{ik}} \quad (2.6)$$

де $e_{ij} = a(s_{i-1}, h_j)$ - вихідний показник нейронної мережі з прямим зв'язком, описаний функцією «a», яка намагається захопити вирівнювання між вхідними «j» та вихідним параметром на позиції «i».

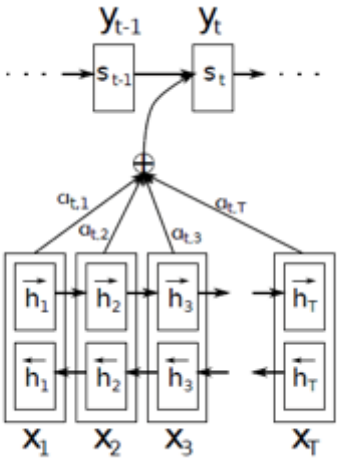

Рисунок 2.2.1. – Модель уваги запропонована Dzmitry Bahdanaux[7]

На рис. 2.2.1 зображено модель механізму уваги. Ця модель використовує модель двонаправленої короткої довгострокової пам’яті (Bidirectional Long Short Term Memory, BiLSTM [8]) яка генерує послідовність анотацій (h_1, h_2,….., h_{Tx}) для кожного вхідного речення. Усі вектори h_1, h_2 і т. д., які використовуються , є конкатенацією прямих і зворотних прихованих

станів у енкодері для моделі нейронної мережі типу енкодер–декодер, деенкодер зчитує вхідне речення, послідовність векторів $x = (x_1, \cdots, x_{Tx})$ й перетворює їх у вектор

$$\mathrm{h_j} = [\overrightarrow{\mathrm{h}_j^T}; \overleftarrow{\mathrm{h}_j^T}]^T \tag{2.7}$$

де h_t – це прихований стан у момент часу t , $t \in R$

Звичайна RNN, читає вхідну послідовність x послідовно, починаючи з першого символ x_1 до останнього x_{Tx}. Однак у запропонованій схемі анотації кожного слова узагальнюються не лише попередні слова, а й наступні слова. Отже, ми пропонуємо використовувати двонаправлену RNN модель (BiRNN [9]), яка останнім часом успішно використовується для розпізнавання мови. BiRNN складається з прямих і зворотних RNN. Пряма RNN $\overrightarrow{f}$ f читає вхідну послідовність як наказано (від x_1 до x_{Tx}) і обчислює послідовність прямих прихованих станів $(\overrightarrow{\mathrm{h1}}, \cdots, \overrightarrow{\mathrm{hTx}})$. Зворотня RNN $\overleftarrow{f}$ читає послідовність у зворотному порядку (з x_{Tx} до x_1), в результаті чого а послідовність зворотних прихованих станів ($\overleftarrow{h1}, \cdots, \overleftarrow{\mathrm{hTx}}$). У результаты чого ми отримуємо анотацію для кожного слова x_j шляхом конкатенації прямого прихованого стану $\overrightarrow{\mathrm{h1}j}$ і зворотнього $\overleftarrow{h1\mathrm{j}}$, тобто $\mathrm{h_j} = [\overrightarrow{\mathrm{h}_j^T}; \overleftarrow{\mathrm{h}_j^T}]^T$. Таким чином, анотація h_j містить значимість як попередніх, так і наступних слів. Через тенденцію RNN до кращого представлення останніх введених даних, анотація h_j буде зосереджена на словах навколо x_j . Ця послідовність анотацій пізніше використовується декодером і моделлю вирівнювання для обчислення вектора контексту за допомогою рівнянь (2.6) та (2.7).

Серед механізмів уваги розділяють 3 основних види уваги[10] :

- Увага на рівні Енкодер-Декодеру: розрахунок взаємодії між вхідними та вихідними значеннями

- Увага до самого себе у вхідній послідовності: розрахунок взаємодії між собою та усіма іншими словами у вхідній послідовності.
- Увага до самого себе у вихідній послідовності: розрахунок взаємодії між собою та усіма іншими словами у вихідній послідовності. У даному випадку необхідно враховувати, що обсяг уваги до самого себе обмежується словами, які зустрічаються перед поточним словом. Це запобігає витоку інформації під час навчання моделі. Дане обмеження робиться шляхом маскування слів, які стоять після поточного для кожного кроку. Отже, для кроку 1 тільки перше слово вихідної послідовності НЕ маскується, для кроку 2 перші два слова НЕ маскуються тощо.

2.3 Нейронна мережа типу «Трансформер»

Трансформер – модель нейронної мережі яка спеціалізується на процесі глибинного навчання з використанням механізму «уваги» до кожного елементу в отриманому наборі вхідних даних[11][11], завдяки чому ця модель часто застосовується в задачах пов’язаних з обробкою та моделюванням природної мови. Аналогічно з RNN, нейроні мережі побудовані на моделі «Трансформер» призначені для обробки послідовних даних [12]. Однак процес обробки цих даних відрізняється оскільки RNN оброблюють дані послідовно згідно з порядком отримання, у той час коли трансформери використовують механізм уваги, завдяки котрому зберігається контекст усього набору вхідних даних незалежно від їх положення у наборі. Наприклад при обробці речення природної мови трансформер спочатку визначить головні слова у реченні згідно його контексту, що дозволяє опрацьовувати більше паралельних сесій у порівнянні з RNN.

Модель трансформера складається з 2-ох головних частин –енкодеру та Декодеру (Детальна внутрішня будова моделі - рис. В.1)

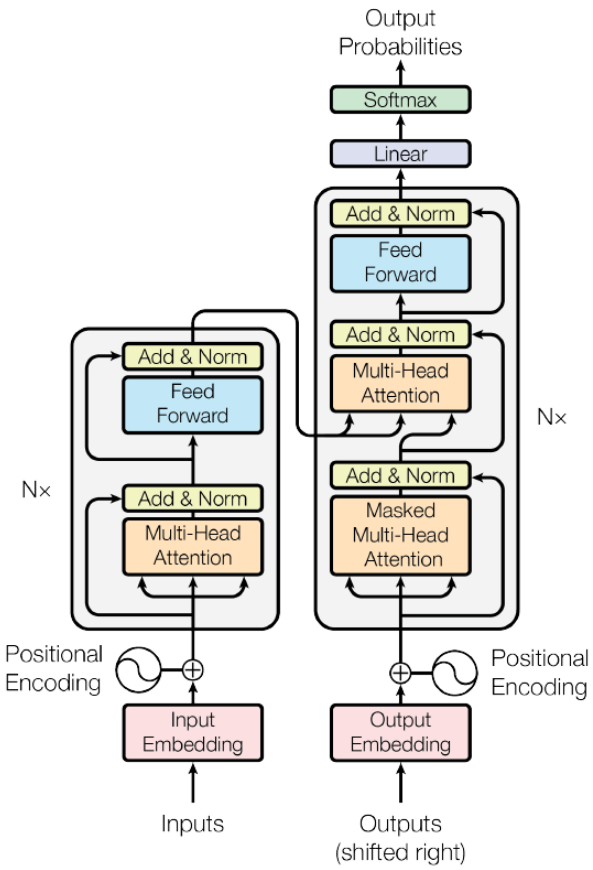

Рисунок 2.3.1 – Модель Трансформер [10]

Блоки енкодера та декодера складаються з кількох ідентичних енкодерів та декодерів налаштованих один за одним відповідно у форматі стеку. Як стек енкодера, так і стек декодера мають однакову кількість одиниць. При надсиланні векторних представлень слів до енкодеру усі слова відправляються одночасно для аналогічної обробки вхідних даних відносно кожного елементу.

Після обробки усіх даних на стороні енкодеру вони передаються до всіх декодерів для подальшої обробки та конвертації даних для отримання результатів. (рис. 2.3.2)

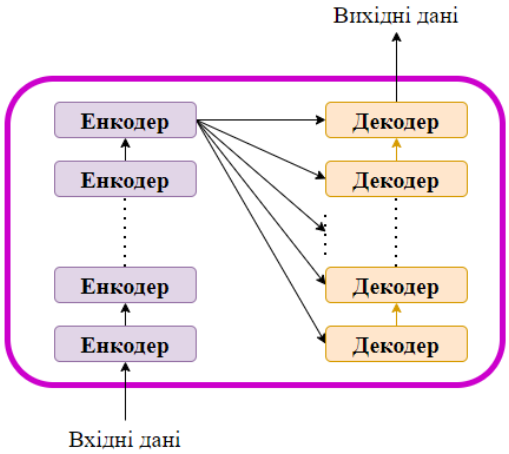

Рисунок 2.3.2 – Внутрішня будова Енкодеру та Декодеру

Енкодер отримує вхідні дані у вигляді послідовність векторних представлень слів ($x_1, \ldots, x_n$) у купі з послідовністю позиційних їх представлень $z = (z_1, \ldots z_n)$. Кожен енкодер має два внутрішні шари (див Рисунок 2.3.3):

- Механізм уваги до самого себе з кількома зв'язками на вхідних векторах (multi-head self attention mechanism).
- Позиційно прив'язана нейронна мережа прямого поширення (feed-forward neural network).

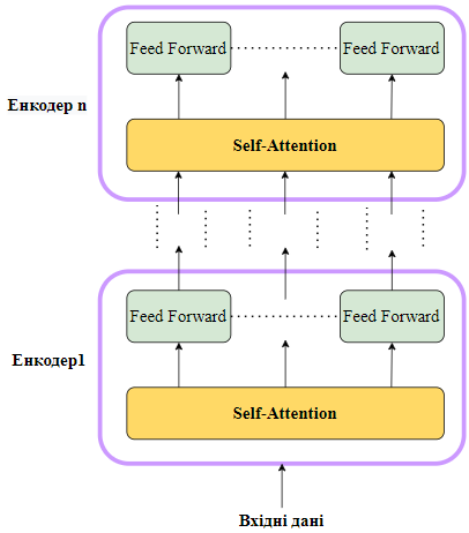

Рисунок 2.3.3 – Детальна будова внутрішніх блоків Енкодеру

Набір вхідних даних, що надходять в енкодер, спочатку проходить через шар внутрішньої уваги до самого себе, що допомагає енкодеру подивитися на

інші слова у вхідному реченні під час кодування конкретного слова. Вихід шару внутрішньої уваги до самого себе вирушає до нейронної мережі прямого поширення (feed-forward neural network). Така сама мережа застосовується для кожного слова в реченні незалежно один від одної.

Враховуючи послідовність представлень z, декодер генерує вихідну послідовність $(y_1, \ldots, y_m)$ символів по одному елементу. Кожен декодер має три внутрішні шари (рис. 2.3.4):

- Замаскований механізм уваги до самого себе кількома зв'язками на вихідних векторах попередньої ітерації(masked multi-head self attention mechanism).
- Механізм уваги з кількома головами на виході з енкодера та замаскована багатоголова увага в декодері(multi-head attention mechanism).
- Позиційно прив'язана нейронна мережа прямого поширення (feed-forward neural network).

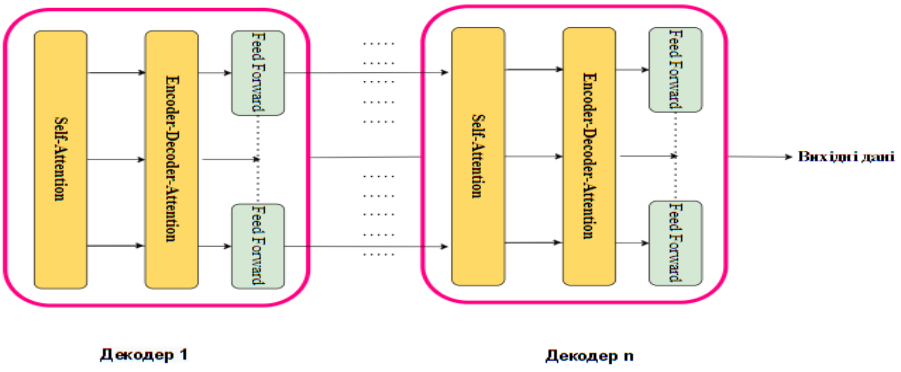

Рисунок 2.3.4 – Детальна будова внутрішніх блоків Декодеру

Векторне представлення слів не зберігає інформацію про їх позицію у реченні, на відміну від в RNN мереж та їх послідовному характеру. До того ж при розрахунку уваги до самого себе через функцію softmax будь-яка подібна інформація про відношення втрачається.

Щоб зберегти інформацію про позицію, трансформер додатково до векторів вхідних даних додає позиційні вектори для кожного представлення.

Ці вектори дотримуються певної періодичної функції , яку модель вивчає і здатна визначити позицію окремого слова відносно один одного на основі значень. Самі функції представлені комбінаціями синусів / косинусів з різною частотою, які не синхронізовані один з одним. Цей введений вектор називається «позиційним кодуванням» і додається до вхідних вкладень у нижній частині стеків кодерів і декодерів.

Вихід стека декодера на кожному кроці повертається в декодер на наступному часовому кроці — дуже схоже на те, як результати попередніх кроків у RNN використовувалися як наступні приховані стани. І так само, як ми з входами енкодера, позиційне кодування додається до входів декодера, щоб зберегти позицію кожного слова. Ця комбінація позиційного кодування та вбудовування слів пізніше подається в замасковану багато зв’язкову увагу до самого себе.

Цей підрівень самоуваги в стеку декодера модифікований, щоб запобігти відвідуванню наступних позицій — заборонити дивитися на майбутні слова. Це маскування гарантує, що передбачення для позиції залежать лише від відомих результатів з попередніх позицій.

Вихідні дані зі стека енкодерів потім використовуються як кілька наборів ключових векторів «k» і векторів значень «v» для шару уваги на рівні «енкодера-декодера» . Це допомагає декодеру зосередитися на контекстно релевантних частинах у вхідній послідовності для цього кроку. Вектор «q» походить із шару «виведення власної уваги»[13].

Оскільки природна мова є незрозумілою для машини, необхідно застосувати алгоритми перетворення слів з природної мови у векторні представлення [14], [15]. Після переведення слів у векторні представлення кожне слово потрапляє до енкодеру паралельно.

Перший крок необхідний для подальшого обчислення внутрішньої уваги – це створення трьох векторів з кожного вхідного векторного представлення:

- *q* вектор запиту (Query vector)
- *k* вектор ключа (Key vector)

- *v* вектор значення (Value vector)

Ці вектори створюються за допомогою перемноження векторного представлення на матриці.

Другий етап обчислення внутрішньої уваги – одержання коефіцієнта (score). Допустимо, ми підраховуємо внутрішню увагу для першого слова в нашому прикладі – «Дія». Нам потрібно оцінити кожне слово у вхідному реченні щодо даного слова. Коефіцієнт визначає, наскільки потрібно сфокусуватися на інших частинах вхідної речення під час кодування слова у конкретній позиції.

Коефіцієнт підраховується за допомогою скалярного добутку вектору запиту та вектору ключа відповідного слова. Отже, якщо ми обчислюємо внутрішню увагу слова позиція якого - 1, перший коефіцієнт буде скалярним твором q_1 і k_1, другий — скалярним твором q_1 і k_2.

Третій етап – розділити ці коефіцієнти на квадратний корінь розмірності вектору ключа, що використовується , це значення забезпечує більш стабільні градієнти і використовується за умовчанням, але можливі також інші значення.

Наступний етап – використання функції пропустити результат софтмакс (softmax)для отриманого результату. Ця функція нормалізує коефіцієнти для того щоб вони були позитивними й у сумі давали 1.

Отриманий софтмакс коефіцієнт (softmax score) визначає врогідність з якою кожне зі слів- пропозиції для перекладу буде поставлене у певну позицію вихідного речення. Очевидно, що слово у своїй позиції отримає найбільший софтмакс-коефіцієнт, але іноді корисно враховувати й інше слово, релевантне до аналізованого.

Після отримання софтмакс коефіцієнту необхідно помножити кожен вектор значення з відповідним йому софтмакс-коефіцієнтом (перед їх складанням). Основні правила даної операції це дотримання значення слів без змін, на яких ми фокусуємося, і відвести на другий план нерелевантні слова помноживши їх на невеликі значення, як 0.0001 тощо.

Останній етап полягає у складанні зважених векторів значення. Це і буде вихід шару внутрішньої уваги в цій позиції. На цьому завершується обчислення внутрішньої уваги [13]. В результаті ми отримуємо вектор, який можемо передавати далі до нейронної мережі прямого поширення.

В архітектурі Транформер увага до самого себе обчислюється декілька разів у паралельному режимі незалежно один від одного. Тому цей процес називають multi-head self attention. Вихідні дані об’єднуються та лінійно перетворюються (рис. 2.3.5).

Таблиця 2.3.1 – Таблиця розрахунку уваги до самого себе для 1-го слова

Слово	q	k	v	Score	Score /n	Softmax	Softmax * v	Sum
Дія	q_1	k_1	V_1	$q_1 * k_1$	$q_1 * k_1/n$	X_{11}	$X_{11} * V_1$	Z_1
Дає		k_2	V_2	$q_1 * k_2$	$q_1 * k_2/n$	X_{12}	$X_{12} * V_2$	
результат		k_3	V_3	$q_1 * k_3$	$q_1 * \mathbf{k_3/n}$	X_{12}	$X_{13} * V_3$	

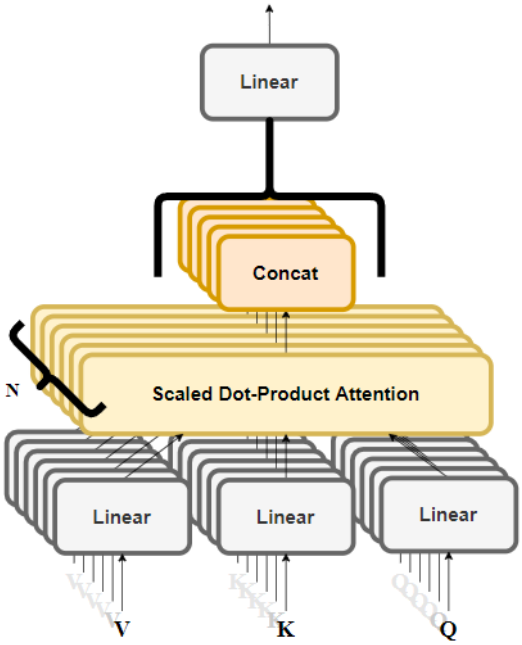

Рисунок 2.3.5 – Візуалізація механізму Multi-head self attention

2.4 Нейронна мережа типу «Трансформер-XL»

Архітектура трансформаторів дозволяє обробляти та працювати з довгостроковими залежностями подібно до LSTM мереж[13]. Однак вони не можуть вийти за межі певного рівня аналізу через використання контексту

сегментів вхідного тексту фіксованої довжини. Для подолання цього недоліку була запропонована нова архітектура – Transformer-XL: мовні моделі за межами контексту фіксованої довжини з механізмом уваги[13].

У даній архітектурі приховані стани, які були отримані у попередніх сегментах, повторно використовуються як джерело інформації для обробки поточного сегмента. Це дозволяє моделювати довгострокову залежність, оскільки інформація може перетікати від одного сегмента до іншого протягом опрацювання усього набору вхідних даних.

Подумайте про мовне моделювання як процес оцінки ймовірності наступного слова з урахуванням попередніх слів.

Ідея застосування моделі Transformer для мовного моделювання полягає утому, що весь корпус можна розділити на сегменти фіксованої довжини керованих розмірів[13]. Потім навчити модель трансформера на сегментах незалежно, ігноруючи всю контекстну інформацію з попередніх сегментів:

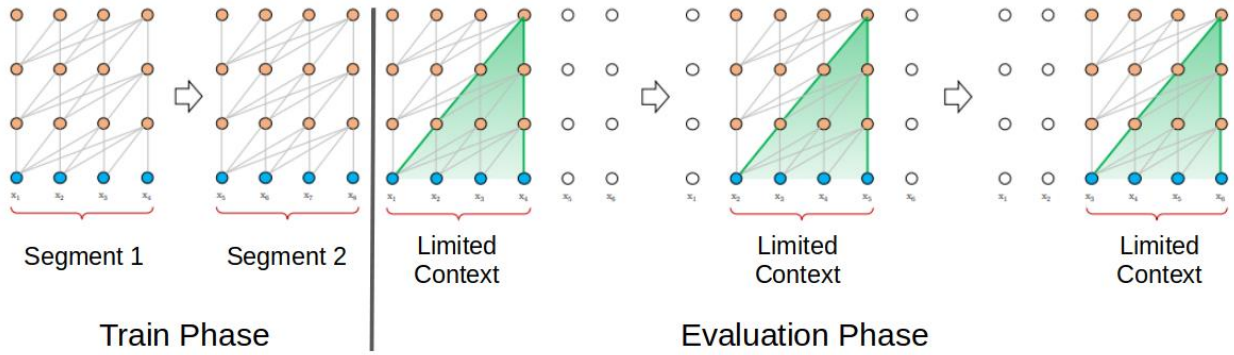

Рисунок 2.4.1 – Схематичне зображення матриці моделей для перекладу

Для архітектури моделі Transformer-XL проблеми зникнення градієнту не є суттєвою. Але фрагментація контексту обмежує можливості нейронної мережі до навчання довгостроковим залежностями [16]. Під час фази оцінки сегмент зміщується вправо лише на одну позицію. Новий сегмент потрібно обробляти повністю з нуля (див Рис 8). Цей метод оцінки, на жаль, досить трудомісткий та затратний з точку зору ресурсів.

Під час фази навчання в Transformer-XL прихований стан, обчислений для попереднього стану, використовується як додатковий контекст для

поточного сегмента. Цей механізм повторення Transformer-XL піклується про обмеження використання контексту фіксованої довжини.

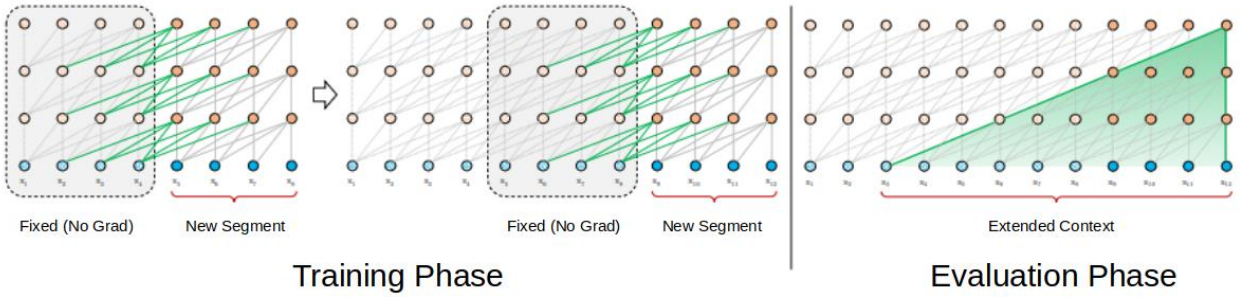

Рисунок 2.4.2 – Модель Transformer XL з довжиною сегмента 4[17]

Під час фази оцінки, результати обчислення з попередніх сегментів можна використовувати повторно замість обчислення цих даних з нуля (як у випадку моделі трансформатора). Це, багаторазово збільшує швидкість обчислень.

2.5 Типи моделей для перекладу

Важливим кроком у вирішення поставленої задачі, а саме побудови - моделі нейронної мережі для мультилігвістичного перекладу, є момент вибору підходу до побудови моделі перекладу. Існують два ефективних методи для мультилінгвістичного перекладу:

- Прямий переклад з однієї мові на іншу
- Переклад з використанням мови-посередника

2.5.1 Прямий переклад

При прямому перекладі з оригінальної мови на цільову, використовують модель при якій розрахунки вагових коефіцієнтів між словами представленими у словнику оригінальної мови та словнику цільової мови обраховуються зважаючи на статистичну характеристику частоти використання слова з цільового словника при використанні відповідного слова-оригіналу у групах речень обох мов.

До переваг даного методу можна віднести:

- Швидкість виконання перекладу
- Зменшені вимоги до кількості тренувальних даних
- Підвищена точності перекладу

Недоліки даного методу:

- Для кожної пари мов повинна існувати відповідна модель перекладу
- Використання одночасно кількох моделей потребує більшу кількість обчислювальних ресурсів.
- Необхідно навчати та зберігати модель для кожної пари мови-оригіналу та цільової мови.

Використання моделей для кожної пари мов породжує питання про кількість моделей, які необхідно навчити та зберегти для подальшого використання. Оскільки кількість моделей для кожної мови дорівнює N – 1, загальна кільскість моделей може бути обчислена за формулою N (N – 1), де N – кількість мов.

Таблиця 2.5.1 – Схематичне зображення матриці моделей для перекладу

	Англійська	Німецька	Французька	Італійська
Англійська		+	+	+
Німецька	+		+	+
Французька	+	+		+
Італійська	+	+	+	

2.6 Переклад з використанням мови-посередника

За умови використанні мови-посередника при перекладі з оригінальної мови на цільову, використовують модель при якій розрахунки вагових коефіцієнтів між словами представленими у словниках виконуються так само як і у випадку з прямим, проте обчислення виконуються між 3-ма словниками у 2 етапи.

Спочатку обчислення виконуються між словами оригінальної мови та словником мови-посередника, потім між словником мови-посередника та цільової мови.

До переваг даного методу можна віднести

- Менша кількість необхідних моделей для перекладу.
- Менше ресурсів необхідно для лінійного перекладу

Недоліки даного методу:

- Збільшений час виконання перекладу
- Збільшені вимоги до кількості тренувальних даних
- Менша точності перекладу при недостатній кількості тренувальних даних.

У даному підході використовується менша кількість моделей у порівняні з методом прямого перекладу для кожної пари проте точність перекладу залежить від кількості тренувальних даних значно більше. Оскільки кількість моделей для кожної мови дорівнює $N - 1$, загальна кількість моделей може бути обчислена за формулою $2*(N - 1)$, де N – кількість мов.

Таблиця 2.5.2 – Схематичне зображення матриці моделей для перекладу при мові - посереднику

	Англійська	Німецька	Французька	Італійська
Англійська		+	+	+
Німецька	+			
Французька	+			
Італійська	+			

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Обробка вхідних даних

Насамперед для початку роботи необхідно забезпечити механізм перетворення тексту природної мови у зрозумілу для машини форму, а саме - числове представлення, тобто провести токенізацію вхідного речення.

Першим кроком є відокремлення слів вхідного речення один від одного та зведення отриманого речення до масиву, який складається зі слів речення. Для цього імпортуємо бібліотеку обробки мови **spacy (version 3.0.4)**, встановлюємо мовний пакет для нашої мови та ініціалізуємо функцію токенізатора. (Додаток А.1)

Наступним кроком після створення функції токенізатора буде створення словника мови, який зберігає дані у вигляді {«слово»: «число»}.

Останнім кроком підготовки обробки вхідних даних є перетворення масиву слів вхідного речення на масив чисел використовуючи створений словник, змінюючи кожне слово вхідного речення на відповідне йому числове значення зі словника мови оригіналу.

Наступний крок підготовки даних для моделі – є розбиття даних на групи для передачі їх у майбутню модель нейронної мережі для тренування. Задля підвищення якості тренування моделі виконується групування речень за їх довжиною з метою отримання наборів речень приблизно однакової довжини у кожній групі. Для цього використовуємо функції **data_process** та **generate_batch** (Додаток А.2).

Створюємо модель нейронної мережі для мультилінгвістичного перекладу з визначенням кількості шарів енкодеру, декодеру там передачі словників мови оригіналу та цільовою мови перекладу. (Додаток А.3). Перевіряємо інформацію щодо будови створеної моделі (Додаток Б).

Для оцінки якості роботи нейронної мережі за значенням валідаційної втрати (Val loss), яка представляє відсоток хибного перекладу набору для перевірки з використанням критерію крос-ентропії (CrossEntropyLoss).

У процесі навчання виконується обчислення матриці для кожного речення де колонки послідовно заповнюються значеннями слів з цільового словника.

Після обробки вхідних даних нейронною мережею, на виході повертається масив числових представлень слів з цільового словника і отримані дані порівнюються з масивом для перевірки.

Для перетворення вихідного масиву у природну мову використовуємо словник цільової мови у функції **greedy_decode_function**(Додаток А.4).

3.2 Результати роботи програми та тестування

У процесі навчання нейронної мережі на 29000 речень та перевірці на 1080 реченнях найкращий отриманий результат для значення Val_loss складає 1.756% (Рисунок Б.2).

Розглянемо роботу нейронної мережі на прикладі перекладу тексту з Англійської мови на Німецьку та Французьку

Спочатку токенізуємо речення та переводимо його у числовий вигляд (Таблиці 3.1).

Таблиця 3.2.1 – Приклад перетворення речення

Початкове речення	Розбиття речення на слова	Перетворення у числову форму
'A little girl climbing into a wooden playhouse.'	['A', 'little', 'girl', 'climbing', 'into', 'a', 'wooden', 'playhouse', '.']	['2','15', '646', '212', '126', '72', '283', '953', '5835', '908', '4']

Кількість елементів у числовій матриці збільшилась на два значення. Число 2 на початку масиву символізує про початок речення, а число 4 – про кінець речення.

Наступний крок – передача отриманого масиву на вхід до моделі нейронної мережі (Додаток Б.1).

У процесі обчислення ітераційно створюється матриця розмірністю N*(N+2), де N – довжина вхідного масиву. Значення, які знаходяться нижче головної діагоналі та на самій діагоналі прирівнюються до 0, значення вище головної діагоналі прирівнюються до найменшого можливого числа для подальших розрахунків. На початку матриця має вигляд як зображено на таблиці.

Таблиця 3.2.2 – Початковий вигляд матриці обчислень

0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
0	0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
0	0	0	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
0	0	0	0	-inf	-inf	-inf	-inf	-inf	-inf	-inf
0	0	0	0	0	-inf	-inf	-inf	-inf	-inf	-inf
0	0	0	0	0	0	-inf	-inf	-inf	-inf	-inf
0	0	0	0	0	0	0	-inf	-inf	-inf	-inf
0	0	0	0	0	0	0	0	-inf	-inf	-inf
0	0	0	0	0	0	0	0	0	-inf	-inf
0	0	0	0	0	0	0	0	0	0	-inf
0	0	0	0	0	0	0	0	0	0	0

У процесі обчислення на кожному кроці дані обчислюються с застосуванням механізму уваги, у результаті чого матриця поступово змінюється й у результаті отримується наступну матрицю для перекладу на Німецьку мову.

Таблиця 3.2.3 – Кінцевий вигляд матриці обчислень на німецьку мову

0	2	2	2	2	2	2	2	2	2	2
0	0	8708	642	5735	333	333	333	333	333	333
0	0	0	1610	7416	8782	623	623	623	623	623
0	0	0	0	8007	2910	251	2301	251	251	251
0	0	0	0	0	6157	3900	6368	1241	23	23
0	0	0	0	0	0	4136	8277	562	562	562
0	0	0	0	0	0	0	8700	7654	633	633
0	0	0	0	0	0	0	0	2428	626	626
0	0	0	0	0	0	0	0	0	3601	742
0	0	0	0	0	0	0	0	0	0	824
0	0	0	0	0	0	0	0	0	0	4

Аналогічна операція виконується для Французької мови й отримана матриця складається з числових представлень словника Французької мови.

Таблиця 3.2.4 – Кінцевий вигляд матриці обчислень на Французьку мову

0	2	2	2	2	2	2	2	2	2	2
0	0	4719	4023	8639	2847	419	562	562	562	562
0	0	0	1963	2144	8378	251	137	137	137	137
0	0	0	0	1770	2001	251	8996	7414	34	34
0	0	0	0	0	8903	3564	5084	1241	1234	1234
0	0	0	0	0	0	6583	1489	8325	3173	3173
0	0	0	0	0	0	0	7006	8488	2346	2346
0	0	0	0	0	0	0	0	501	7	7
0	0	0	0	0	0	0	0	0	5744	134
0	0	0	0	0	0	0	0	0	0	71
0	0	0	0	0	0	0	0	0	0	76
0	0	0	0	0	0	0	0	0	0	4

Останній стовпець – вихідний масив числових представлень речення зі значеннями слів з словника цільової мови.

Таблиця 3.2.5 – Числові представлення речення на різних мовах

Мова	Числове представлення
Англійська	['2','15', '646', '212', '126', '72', '283', '953', '5835', '908', '4']
Німецька	['2', '333', '623', '251', '23', '562', '633', '626', '742', '824', '4']
Французька	['2', '562', '137', '34', '1234', '3173', '2346', '7', '134', '71', '4']

Отримані числові представлення передаємо разом зі словником цільової мови у функцію **greedy_decode_function** й отримуємо переклад у природній мові.

Таблиця 3.2.6 – Перекладані речення на різних мовах

Мова	Числове представлення
Англійська	A little girl climbing into a wooden playhouse.
Німецька	Ein kleines Mädchen, das in ein hölzernes Spielhaus klettert
Французька	Une petite fille s'élevant dans une maison de jeu en bois

Для покращення роботи нейронної мережі можна виконати наступні дії:

- Збільшити кількість шарів Енкодеру та Декодеру

- Збільшити кількість наборів тренувальних даних до кількохсот тисяч .
- Додавання зворотного перекладу під час обробки даних для корекції вагових коефіцієнтів у моделі.

ВИСНОВКИ

У даній роботі проведено аналіз існуючих підходів, методів та рішень у одному з напрямів обробки природної мови – мультилінгвістичному перекладі. На основі зібраної інформації було створено нейронну мережу типу Transformer для перекладу тексту.

Програмна реалізація включає в себе створення та налаштування функцій для попередньої обробки тексту з природної мови у числову, побудова моделі нейронної мережі та тренування з оцінюванням точності роботи нейронної мережі у задачах перекладу на кожній з навчальних епох. Історія тренування зображена на Рисунку Б.2.

Реалізація нейронної мережі виконана на мові програмування Python 3.12. Для обробки даних та побудови нейронної мережі використані інструменти бібліотек spacy, tensorflow, torch, torchtext.

Для навчання нейронної мережі було використано 29000 коротких текстів у якості тренувальних даних та додатково 1080 текстів для тестування.

Найкращий отриманий результат для моделі (Рисунок Б.1) складає 1.756% (Рисунок Б.2). Детальні покрокові пояснення до тестового запуску програми з використанням побудованої нейронної мережі описані в розділі 3.2. Варіанти для покращення результатів побудованої моделі для перекладу надані в кінці розділу 3.2.

СПИСОК ЛІТЕРАТУРИ

1. Luong M.-T. Addressing the Rare Word Problem in Neural Machine Translation / M.-T. Luong, I. Sutskever, G. V Quoc Le, W. Zaremba // Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2015.– p. 11–19.
2. Zennaki O. Inducing Multilingual Text Analysis Tools Using Bidirectional Recurrent Neural Networks / O. Zennaki, N. Semmar, L. Besacier // Conference: 26th International Conference on Computational Linguistics, COLING 2016. – p. 450-460.
3. Kalchbrenner N. Recurrent Continuous Translation Models / N. Kalchbrenner, P. Blunsom // Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013. – p. 1700-1709.
4. Vaswani A. Decoding with Large-Scale Neural Language Models Improves Translation / A. Vaswani, Y. Zhao, V. Fossum, D. Chiang // In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013. – p. 1387–1392.
5. Cho K. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation / K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014. – p. 1724–1734.
6. Datta D. Neural Machine Translation using Recurrent Neural Network Cloud Computing and IoT View project Neural Machine Translation using Recurrent Neural Network View project / D. Datta, P. Evangeline, A. Jain // Artic. Int. J. Eng. Adv. Technol., 2020. – №. 9. – p. 2249–8958.
7. Kudugunta S. Investigating Multilingual NMT Representations at Scale / S. Kudugunta, A. Bapna, I. Caswell, N. Arivazhagan, O. Firat, G. Ai // Proceedings of the 2019 Conference on Empirical Methods in Natural

Language Processing and the 9th International Joint Conference on Natural Language Processing, 2019. – p. 1565–1575.

8. Cheng J. Long Short-Term Memory-Networks for Machine Reading / J. Cheng, L. Dong, M. Lapata // Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016. – p. 551–561.
9. Schuster M. Bidirectional Recurrent Neural Networks / M. Schuster, K. K. Paliwal // IEEE Trans. SIGNAL Process., 1997. – № 11(45). – p. 2673-2681.
10. Vaswani A. Attention Is All You Need / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin // 31st Conference on Neural Information Processing Systems (NIPS 2017). – p. 1-11.
11. Suleiman D. Deep Learning Based Technique for Plagiarism Detection in Arabic Texts / D. Suleiman, A. Awajan, N. Al-Madi // The International Conference on new Trends in Computing Sciences (ICTCS2017), 2017. – p. 1 – 6.
12. Chernyavskiy A. Transformers: ‘The End of History’ for Natural Language Processing? / A. Chernyavskiy, D. Ilvovsky, P. Nakov, // Machine Learning and Knowledge Discovery in Databases. Research Track, 2021. – p. 1-16.
13. Al-Rfou R. Character-Level Language Modeling with Deeper Self-Attention / R. Al-Rfou, D. Choe, M. Guo, L. Jones // The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. – p. 3159-3166, AAAI Press, 2019.
14. Webster J. J. Tokenization As The Initial Phase In NLP / J. J. Webster C. Kit // The 14th International Conference on Computational Linguistics (COLING 1992), 1992. – № 4. – p. 13-21.

15. Pennington J. GloVe: Global Vectors for Word Representation / J. Pennington, R. Socher, C. D. Manning // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014. – p. 1532-1543.
16. Dai Z. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context / Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V Le, R. Salakhutdinov // Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. – p. 2978-2988.
17. Dai Z. Transformer-XL: Language Modeling With Longer-Term Dependency / Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, R. Salakhutdinov // ICLR, 2019. – p. 1-16.

ДОДАТКИ

Додаток А

Додаток А.1

#Завантаження та встановлення модулю spacy та мовних пакетів для Англійської, Німецькой та Французької мов

In [ ]:
```
!python -m pip install spacy
!python -m spacy download en
!python -m spacy download de
!python -m spacy download fr


url_base_link ='/content/drive/MyDrive/Colab Notebooks/Magister/'
train_urls_link =['train_batch_sent.en','train_batch_sent.de', 'train_batch_sent.fr']
val_urls_link = ['val_batch_sent.en','val_batch_sent.de', 'val_batch_sent.fr']
test_urls_link = ['test_2018_batch_sent.en','test_2018_batch_sent.de', 'test_2018_batch_sent.fr']

def read_file(path):
  lines = []
  with open(path,encoding='utf-8') as f:
    lines = f.readlines()
  return lines

train_lang_pack = [read_file(url_base_link+url) for url in train_urls_link]
val_lang_pack = [read_file(url_base_link+url) for url in val_urls_link]
test_lang_pack = [read_file(url_base_link+url) for url in test_urls_link]

for arr in range(len(train_lang_pack)):
  train_lang_pack[arr].extend(val_lang_pack[arr])
  train_lang_pack[arr].extend(test_lang_pack[arr])
```

#Ініціалізація токенізаторів

In [ ]:
```
train_filepaths = [[],[]]
val_filepaths =  [[],[]]
test_filepaths =  [[],[]]
```

Додаток А.2

#Створення функції зчитування файлів з парами речень кількох мов

In [ ]:
```
def data_process(filepaths,voc_src,voc_trg,src_tokenizer,trg_tokenizer):
  raw_de_iter = iter(io.open(filepaths[0], encoding="utf8"))
  raw_en_iter = iter(io.open(filepaths[1], encoding="utf8"))
  data = []
  for (raw_de, raw_en) in zip(raw_de_iter, raw_en_iter):
    #print(raw_de)
    de_tensor_ = torch.tensor([voc_src[token] for token in src_tokenizer(raw_de.rstrip("n"))],
    dtype=torch.long)
    en_tensor_ = torch.tensor([voc_trg[token] for token in trg_tokenizer(raw_en.rstrip("n"))],
    dtype=torch.long)
    data.append((de_tensor_, en_tensor_))
  return data

def generate_batch(data_batch):
  de_batch, en_batch = [], []
  for (en_item,de_item) in data_batch:
    de_batch.append(torch.cat([torch.tensor([BOS_IDX]), de_item, torch.tensor([EOS_IDX])], dim=0))
    en_batch.append(torch.cat([torch.tensor([BOS_IDX]), en_item, torch.tensor([EOS_IDX])], dim=0))
  de_batch = pad_sequence(de_batch, padding_value=PAD_IDX)
  en_batch = pad_sequence(en_batch, padding_value=PAD_IDX)
  return en_batch,de_batch

train_data_prep = [url_base_link+lang for lang in train_urls_link]
val_data_prep = [url_base_link+lang for lang in val_urls_link]
test_data_prep = [url_base_link+lang for lang in test_urls_link]

train_data = data_process((train_data_prep[0],train_data_prep[1]),en_vocab,de_vocab,en_tokenizer,de_tokenizer)
val_data = data_process((url_base_link+val_urls_link[0],url_base_link+val_urls_link[1]),en_vocab,de_vocab,en_tokenizer,de_tokenizer)
test_data = data_process((url_base_link+test_urls_link[0],url_base_link+test_urls_link[1]),en_vocab,de_vocab,en_tokenizer,de_tokenizer)

train_iter = DataLoader(train_data, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=generate_batch)
valid_iter = DataLoader(val_data, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=generate_batch)
test_iter = DataLoader(test_data, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=generate_batch)
```

Вибір обробника для подальших обчислень

In [ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Збереження розміру поточного батчу та індексів спеціальних токенів

In [ ]:
```
BATCH_SIZE = 128
PAD_IDX = en_vocab['<pad>']
BOS_IDX = en_vocab['<bos>']
EOS_IDX = en_vocab['<eos>']
```

Додаток А.3

In [ ]:
```
class Sequence_To_Sequnce_Transformer(nn.Module):
    def __init__(self, num_encoder_layers: int, num_decoder_layers: int,
                 emb_size: int, src_vocab_size: int, tgt_vocab_size: int,
                 dim_feedforward:int = 512, dropout:float = 0.1):
        super(Sequence_To_Sequnce_Transformer, self).__init__()
        encoder_layer = TransformerEncoderLayer(d_model=emb_size, nhead=NHEAD,
                                                dim_feedforward=dim_feedforward)
        self.transformer_encoder = TransformerEncoder(encoder_layer, num_layers=num_encoder_layers)
        decoder_layer = TransformerDecoderLayer(d_model=emb_size, nhead=NHEAD,
                                                dim_feedforward=dim_feedforward)
        self.transformer_decoder = TransformerDecoder(decoder_layer, num_layers=num_decoder_layers)
        self.generator = nn.Linear(emb_size, tgt_vocab_size)
        self.src_tok_emb = Token_Embedding_Function(src_vocab_size, emb_size)
        self.tgt_tok_emb = Token_Embedding_Function(tgt_vocab_size, emb_size)
        self.positional_encoding = Positional_Encoding_Layer(emb_size, dropout=dropout)

    def forward(self, src: Tensor, trg: Tensor, src_mask: Tensor,
                tgt_mask: Tensor, source_padding_mask: Tensor,
                target_padding_mask: Tensor, memory_key_padding_mask: Tensor):
        src_emb = self.positional_encoding(self.src_tok_emb(src))
        tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))
        memory = self.transformer_encoder(src_emb, src_mask, source_padding_mask)
        outs = self.transformer_decoder(tgt_emb, memory, tgt_mask, None,
                                        target_padding_mask, memory_key_padding_mask)
        return self.generator(outs)

    def encode(self, src: Tensor, src_mask: Tensor):
        return self.transformer_encoder(self.positional_encoding(
                            self.src_tok_emb(src)), src_mask)

    def decode(self, tgt: Tensor, memory: Tensor, tgt_mask: Tensor):
        return self.transformer_decoder(self.positional_encoding(
                          self.tgt_tok_emb(tgt)), memory,
                          tgt_mask)
```

```
class Positional_Encoding_Layer(nn.Module):
    def __init__(self, emb_size: int, dropout, maxlen: int = 5000):
        super(Positional_Encoding_Layer, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2) * math.log(10000) / emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)
        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding: Tensor):
        return self.dropout(token_embedding +
                            self.pos_embedding[:token_embedding.size(0),:])

class Token_Embedding_Function(nn.Module):
    def __init__(self, vocab_size: int, emb_size):
        super(Token_Embedding_Function, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.emb_size = emb_size
    def forward(self, tokens: Tensor):
        return self.embedding(tokens.long()) * math.sqrt(self.emb_size)

def generate_square_subsequent_mask_sent(sz):
    mask = (torch.triu(torch.ones((sz, sz), device=DEVICE)) == 1).transpose(0, 1)
    mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask == 1, float(0.0))
    return mask
def create_mask(src, tgt):
    src_sequecne_len = src.shape[0]
    tgt_sequence_len = tgt.shape[0]

    tgt_mask = generate_square_subsequent_mask_sent(tgt_sequence_len)
    src_mask = torch.zeros((src_sequecne_len, src_sequecne_len), device=DEVICE).type(torch.bool)

    source_padding_mask = (src == PAD_IDX).transpose(0, 1)
    target_padding_mask = (tgt == PAD_IDX).transpose(0, 1)
    return src_mask, tgt_mask, source_padding_mask, target_padding_mask
```

Додаток А.4

#Функція переведення з масиву чисел у масив слів природної мови

In [ ]:
```
def greedy_decode_to_sent(model, source, source_mask, max_len, start_symbol):
    source = source.to(device)
    source_mask = source_mask.to(device)
    memory_layer = model.encode(source, source_mask)
    result_sentence = torch.ones(1, 1).fill_(start_symbol).type(torch.long).to(device)
    for i in range(max_len-1):
        memory_layer = memory_layer.to(device)
        memory_mask_layer = torch.zeros(ys.shape[0], memory_layer.shape[0]).to(device).type(torch.bool)
        target_mask = (generate_square_subsequent_mask_sent(ys.size(0))
                                    .type(torch.bool)).to(device)
        out_layer = model.decode(ys, memory_layer, target_mask)
        out_layer = out.transpose(0, 1)
        probability = model.generator(out[:, -1])
        _, next_word_predict = torch.max(probability, dim = 1)
        next_word_predict = next_word_predict.item()

        ys = torch.cat([ys,
                        torch.ones(1, 1).type_as(source.data).fill_(next_word_predict)], dim=0)
        if next_word_predict == EOS_IDX:
          break
    return ys
```

#Функція для переведення тексту

In [ ]:
```
def translation(model, source, source_vocab, target_vocab, source_tokenizer):
    model.eval()
    tokens = [BOS_IDX] + source_vocab.forward(source_tokenizer(source)) + [EOS_IDX]
    num_tokens = len(tokens)
    source = (torch.LongTensor(tokens).reshape(num_tokens, 1))
    source_mask = (torch.zeros(num_tokens, num_tokens)).type(torch.bool)
    target_tokens = greedy_decode_to_sent(model, source, source_mask, max_len=num_tokens + 5, start_symbol=BOS_IDX).flatten()
    return " ".join(target_vocab.lookup_tokens(target_tokens.tolist())).replace("<bos>", "").replace("<eos>", "").replace("\n ", "")
```

Додаток Б

```
Sequence_To_Sequnce_Transformer(
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (1): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (2): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (transformer_decoder): TransformerDecoder(
    (layers): ModuleList(
      (0): TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
```

```
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
      (1): TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
      (2): TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (generator): Linear(in_features=512, out_features=11238, bias=True)
  (src_tok_emb): Token_Embedding_Function(
    (embedding): Embedding(20133, 512)
  )
  (tgt_tok_emb): Token_Embedding_Function(
    (embedding): Embedding(11238, 512)
  )
  (positional_encoding): Positional_Encoding_Layer(
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Додаток В

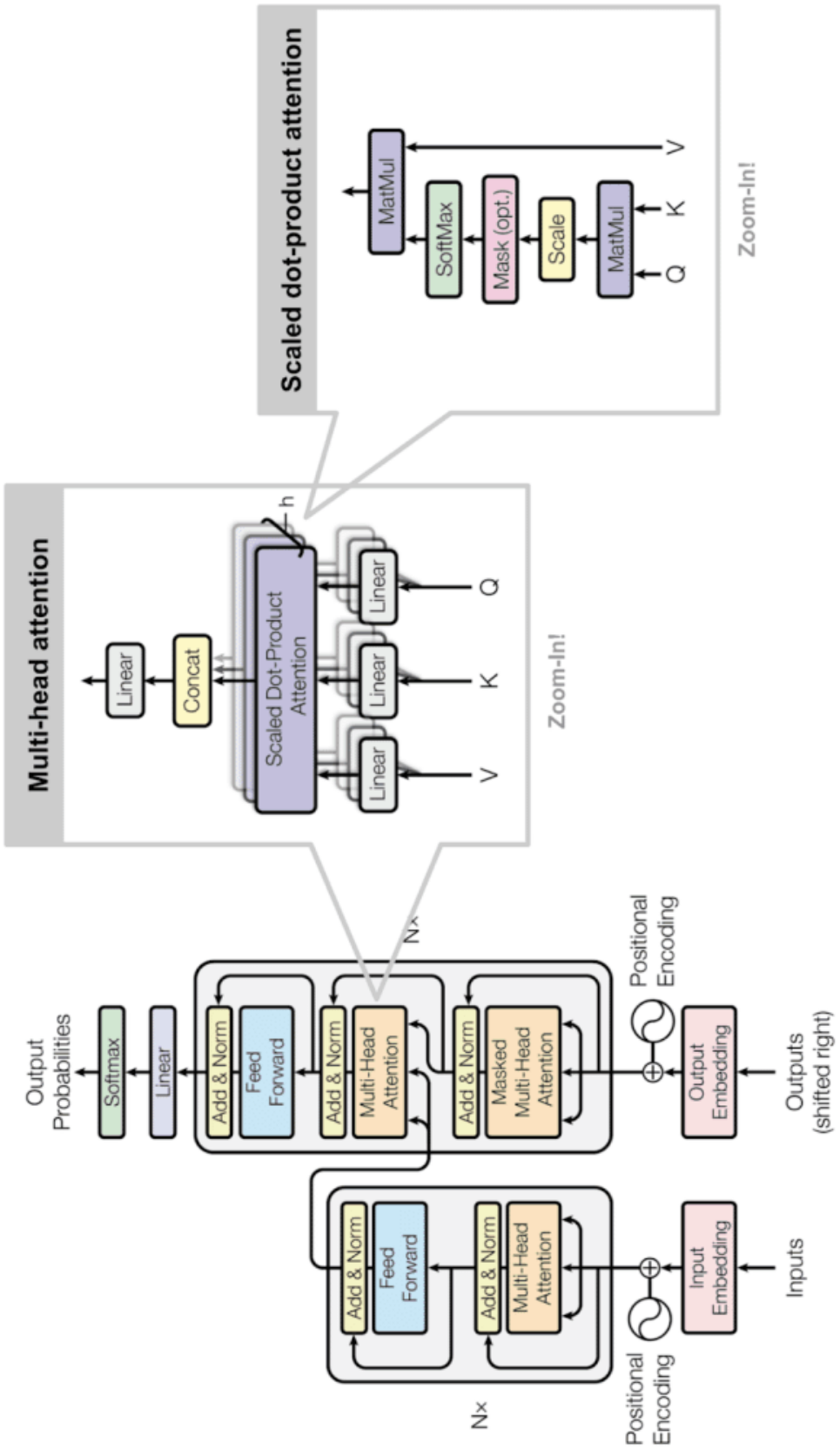

Рисунок В.1Детальне зображення моделі Трансформер