КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теоретичної кібернетики

Кваліфікаційна робота
На здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ТРАНСФОРМЕРИ-АРХІТЕКТУРИ У ЗАДАЧІ КЛАСИФІКАЦІЇ ТЕКСТУ

(TRANSFORMERS-ARCHITECTURES IN THE PROBLEM OF CLASSIFICATION OF TEXT)

Виконав студент 4-го курсу групи ТК-41
Жолобіцький Владислав Олександрович

_____
(підпис)

Науковий керівник:
Професор, доктор фізико-математичних наук
Крак Юрій Васильович

_____
(підпис)

Засвідчую, що в цій
роботі немає запозичень
з праць інших авторів з
відповідних посилань
Студент _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи складає 44 сторінок, в ній містяться 26 ілюстрацій, 7 формул, використано 15 джерел посилань.

МАШИННЕ НАВЧАННЯ, МОДЕЛІ ТРАНСФОРМЕРИ, КЛАСИФІКАЦІЯ ТЕКСТУ.

Метою даного дослідження є спрощення процесу класифікації текстів за рахунок створення моделей машинного навчання.

Процес класифікації повідомлення. Для того щоб класифікувати повідомлення необхідно пройти такі етапи:

- ознайомитися із доступними категоріями, на які можна розділити повідомлення;

- ознайомитися із переліком текстів та прочитати кожен текст;

- проаналізувати текст та віднести його до конкретної категорії. Для автоматизації цього процесу необхідно вирішити задачу машинного навчання.

Об’єктом роботи є датасет з даними діалогів та їх характеристикою. Ціллю роботи є навчити модель визначати емоцію вкладену в повідомлення

Результати роботи: виконано розпізнавання емоцій,які були вкладені людиною в повідомлення, або речення, через навчання моделі.

ЗМІСТ

ДОДАТОК

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

NLP- Обробка природних мов

RNN- Рекурентна нейронна мережа

LSTM- довга короткочасна пам'ять

Supervised learning – навчання з вчителем

Unsupervised learning – навчання без вчителя

GPT - Generative Pre-Training

BERT - Bidirectional Encoder Representations from Transformer

XLM - Cross-Lingual Language Model

RoBERTa - Robustly Optimized BERT pre-training Approach

PLM - Permutation Language Model

ELMO - Embeddings from Language Models

ВСТУП

Штучний інтелект займається обробкою природної мови (NLP), який є одним із розділів інформатики та штучного інтелекту, що фокусується на створенні машин, зрозумілих та генеруючих людські мови. Найулюбленіші програми NLP, такі як система перекладу, пошукові системи, помічники природної мови та аналіз думок вирішують суспільні проблеми з рекордною швидкістю. Проте джерела даних доступні у різних формах – зображення, голос/мова, мови тіла, тексти, використання текстів стало популярним завдяки появі Web 2.0 та соціальних медіа. По порядку слів та їх відношенню у реченні, текстові дані є послідовними; звідси – порядок слів та його ставлення у реченнях, тобто контекст, грає життєву роль виведенні повного значення з тексту. Традиційні моделі машинного навчання без спостереження не враховують порядок чи взаємозв'язок слова у письмовому тексті, але також обмежені відносно невеликими фіксованими розмірами вхідними даними. У цьому випадку мотивація застосування обчислювально глибоких підходів у текстових даних. Рекурентна нейронна система (RNN) - це послідовна модель, здатна обробити послідовні дані, але обмежені повільним часом навчання та залежностями від наступної послідовності. Також існує можливість використання варіанта RNN, тривала коротка довготривала пам'ять (LSTM), яка є найважливішим фактором пом'якшення деяких з цих обмежень. У LSTM знайшли справедливе вирішення проблеми залежності послідовності з великою різницею в швидкості, але не враховується паралельне обчислення і був ще повільнішим, ніж RNM. Мережа уваги була запропонована для полегшення деяких проблем, пов'язаних із LSTM та RNN. У цьому полягає мета даної мережі – зосередити увагу найважливіших розділах даних, які викликають найбільший інтерес. І ще у 2017 році Васвані та інші. Запропонували модель трансформатора, що поєднує елементи як одноструктурну архітектуру. Використовуючи парадигму навчання переносу в якій навчання будується на отриманні знань з одного завдання і використовуючи його для інших суміжних завдань, модель усуває суміжні сегментарні парадигми навчання, що часто будуються з боку.

Досягнутого результату було досягнуто завдяки тому, що вона тренувалася на новому завданні з наявними вагами з відповідним завданням у ході попередньої підготовки; таким чином, роблячи проміжки навчання коротшим, ми отримуємо кращу точність. У 2018 році трансформер був задіяний для мовного моделювання, що уможливлює використання його для значної кількості NLP додатків. Виявлення емоцій – це розділ дослідження настроїв, який шукає для отримання тонких емоцій з промови/голосу, зображення та текстових даних. Спроба знайти емоційні реакції з текстів зазнала невдачі незалежно від джерел інформації. У цій проблемі частково винна нестача модуляції голосу, міміки та жестів, що може викликати сигнали для отримання сенсу та стосунків. Існує й інша частина, пов'язана з відсутністю практичного ставлення до отримання тексту в контексті. До того ж, необхідність для однозначності слів, що передають емоції, для перевірки класифікованих емоцій як справжні емоції становлять суттєву проблему на місцях, оскільки деякі тексти передають різні емоційні висловлювання.

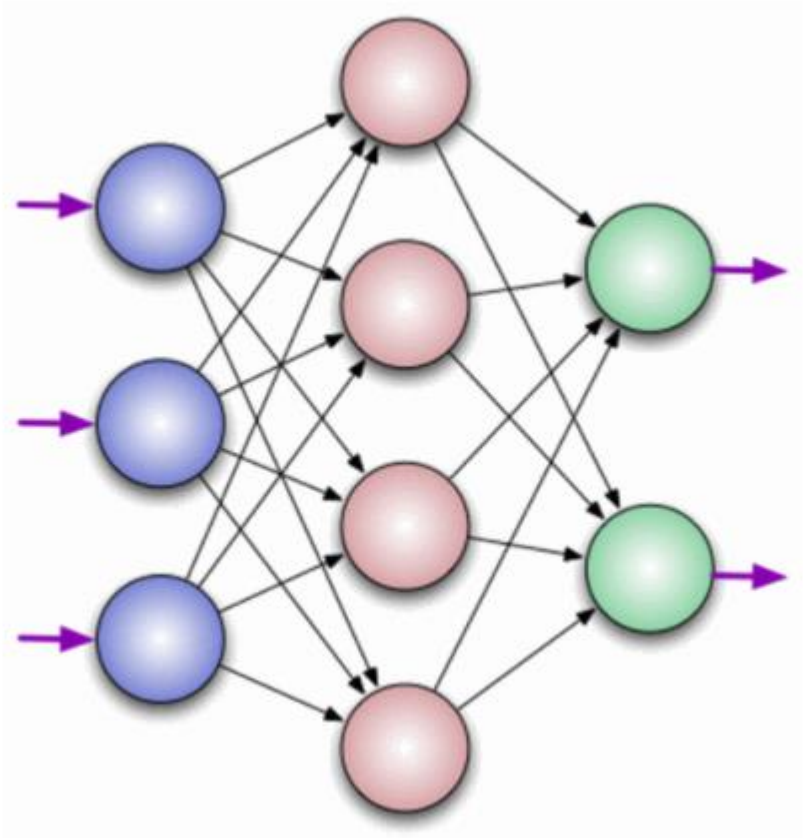
РОЗДІЛ 1: Нейронні мережі

Штучні нейронні мережі

Це обчислювальні системи, натхнені біологічними нейронними мережами, що є складовими мізків тварин. Системи навчаються завдання (поступово покращують свою продуктивність на них), розглядаючи приклади, загалом не використовуючи спеціального програмного забезбечення під задачу. Вони можуть навчитися ідентифікувати зображення з кішками, аналізуючи приклади зображень «кіт» та «не кіт», та використовувати результати для ідентифікації кішок в інших зображеннях. І це відбувається без жодного апріорного знання про котів. Наприклад, вони не знають, що у них є хутряні хвости та вуса чи котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони оброблюють. Нейронні мережі використовуються для вирішення складних завдань, які вимагають аналітичних обчислень подібних тим, що робить людський мозок. Серед найбільш популярних програм нейромереж є: Класифікація - розподіл даних за параметрами. Наприклад, на вхід даються люди і треба вирішити, кому з них надати кредит, а кому ні. Це може зробити нейронна мережа, аналізуючи такі дані як: вік, платоспроможність, кредитна історія і т. д. Прогноз - можливість передбачити наступний крок. Наприклад, зростання або падіння акцій, що ґрунтуються на ситуації на фондовому ринку. Розпізнавання сьогодні найбільш поширене застосування нейронних мереж. У Google, коли ви шукаєте фото або камеру телефону, воно визначає положення вашої особи та виділяє її.

Що таке нейрони

Ця одиниця, яка отримує інформацію, здійснює над нею прості обчислення і передає її далі. За способом їх виготовлення вони поділяються на три основні типи – вхідний (синій), що виходить (червоний) та вихідний (зелений).



Якщо нейромережа складається з великої кількості нейронів, вводиться термін шару. Також існує вхідний шар, який отримує інформацію від прихованих шарів та вихідного шару для виведення результату. І в кожному випадку у нейронів є два основні параметри - вхідні дані та вихідні дані. Що стосується вхідного нейрона: input=output. У решті полів, у поле input потрапляє сумарна інформація кожного нейрона з попереднього шару і потім її можна відновити за допомогою функції активації. Це з'єднання двох нейронів, яке відбувається у мозку. Синапс має 1 параметр – вага. Завдяки йому вхідна інформація змінюється під час передачі від одного нейрона до іншого. Той нейрон, у якого вага буде більшою, інформації і головуватиме в наступному нейроні. Матриця, в якій знаходяться ваги, є сукупністю нейронної мережі або матриці ваги - це мозок всієї системи. Саме завдяки цим вагам, інформація, що входить, обробляється і перетворюється на результат.

Навчання нейронної мережі

Навчити нейронну мережу можна різними способами: з учителем, без учителя, з підкріпленням. При навчанні з учителем нейронна мережа навчається на розміченому наборі даних і пророкує відповіді, які використовуються для оцінки точності алгоритму на навчальних даних. При навчанні без учителя модель використовує нерозмічені дані, з яких алгоритм самостійно намагається витягти ознаки і залежності. Навчання з частковим залученням вчителя є чимось середнім. Воно використовує невелику кількість розмічених даних і великий набір нерозмічених. А навчання з підкріпленням тренує алгоритм за допомогою системи заохочень. Агент отримує зворотний зв'язок у вигляді винагород за правильні дії. Схожим чином дресирують тварин.

Навчання з вчителем

Навчання з учителем (supervised learning) передбачає наявність повного набору розмічених даних для тренування моделі на всіх етапах її побудови. Наявність повністю розміченого датасету означає, що кожному наприклад в навчальному наборі відповідає відповідь, який алгоритм і повинен отримати. Таким чином, розмічений датасет з фотографій квітів навчить нейронну мережу, де зображені троянди, ромашки або нарциси. Коли мережа отримає нове фото, вона порівняє його з прикладами з навчальної датасета, щоб передбачити відповідь.В основному навчання з учителем застосовується для вирішення двох типів задач: класифікації і регресії. У задачах класифікації алгоритм пророкує дискретні значення, що відповідають номерам класів, до яких належать об'єкти. У навчальному датасеті з фотографіями квітів кожне зображення буде мати відповідну мітку - «троянда», «ромашка» або «нарцис». Якість алгоритму оцінюється тим, наскільки точно він може правильно класифікувати нові фото з трояндами і ромашками. А ось завдання регресії пов'язані з безперервними даними. Один із прикладів, лінійна регресія, обчислює очікуване значення змінної y, враховуючи конкретні значення x. Більш утилітарні завдання машинного навчання задіють велику

кількість змінних. Як приклад, нейронна мережа, пророкує ціну квартири в Сан-Франциско на основі її площі, місця розташування і доступності громадського транспорту. Алгоритм виконує роботу експерта, який розраховує ціну квартири виходячи з тих же даних.

Навчання без вчителя

Ідеально розмічені і чисті дані дістати нелегко. Тому іноді перед алгоритмом стоїть завдання знайти заздалегідь не відомі відповіді. Ось де потрібно навчання без учителя. У навчанні без учителя (unsupervised learning) у моделі є набір даних, і немає явних вказівок, що з ним робити. Нейронна мережа намагається самостійно знайти кореллящії в даних, витягуючи корисні ознаки і аналізуючи їх.

РОЗДІЛ 2: РОЗПІЗНАВАННЯ ЕМОЦІЙ НА ОСНОВІ ТЕКСТУ

Модель Пола Екмана, яка розбиває емоції на шість основних категорій, тобто щастя, смуток, гнів, огиду, здивування та страх. Засновник цієї моделі стверджував, що ці емоції не залежать одне від інших, що вони є основними, і можуть викликати складні емоції за допомогою комбінацій. Модель Роберта Плутчика відповідає деяким умовам Екмана, про те що існують первинні емоції, та те, що їх поєднання може викликати більш складні емоції. Але він не погодився з кількістю первинних емоцій і визнавав лише 6 замість 8, які траплялися в протилежних парах. 8 основних емоцій, запропонованих Плутчиком були протилежними одне одному: радість проти смутку, довіра проти огиди, гнів проти страху і подив проти очікування. Ортоні, Клер та Коллінз не погодились з аналогією основних емоції , представлені Екманом та Плутчиком. Але вони погодились з тим, що емоції виникали завдяки тому, як люди сприймали події, і ці емоції варіювали залежно від ступеня їх виразності. Вони визначили 22 емоції, додаючи 16 емоцій до емоцій, які Екман визначив основними, з більшим спектром емоцій охоплюється таким чином набагато більш всебічне представлення емоцій, класи полегшення, заздрості, докору, самодокору, вдячності, сорому, жалю, розчарування, захоплення, надії, підтверджених страхів, горя, задоволення, злості, подобається і не подобається. Вимірні моделі, навпаки, поміщають емоції в одно- чи багатовимірний простір. Просторовий порядок відображає взаємозв'язок між емоціями та їх відносним ступенем виникнення. Запропонована модель Расселом, є глибоким прикладом моделі розмірних емоцій, яка розміщує емоції на двовимірному круговому колесі в області збудження. Також представив тривимірну емоційну модель, що складається з валентності, збудження та домінування з валентністю, що диференціює емоції за приємністю та неприємністю, збудження, що диференціює емоції за допомогою активацій та дезактивацій та домінування, що описує ступінь контролю дослідників свої емоції. Виявлення емоцій - це вилучення детальніших настроїв користувачів. Виявлення емоцій на основі тексту - це

підгалузь виявлення емоцій, яка фокусується на вилученні тонко виражених емоцій із написаних текстів.

Смуток:

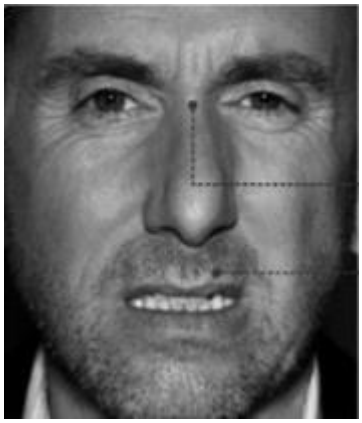
Гнів:

Щастя:

Страх:

Здивування:

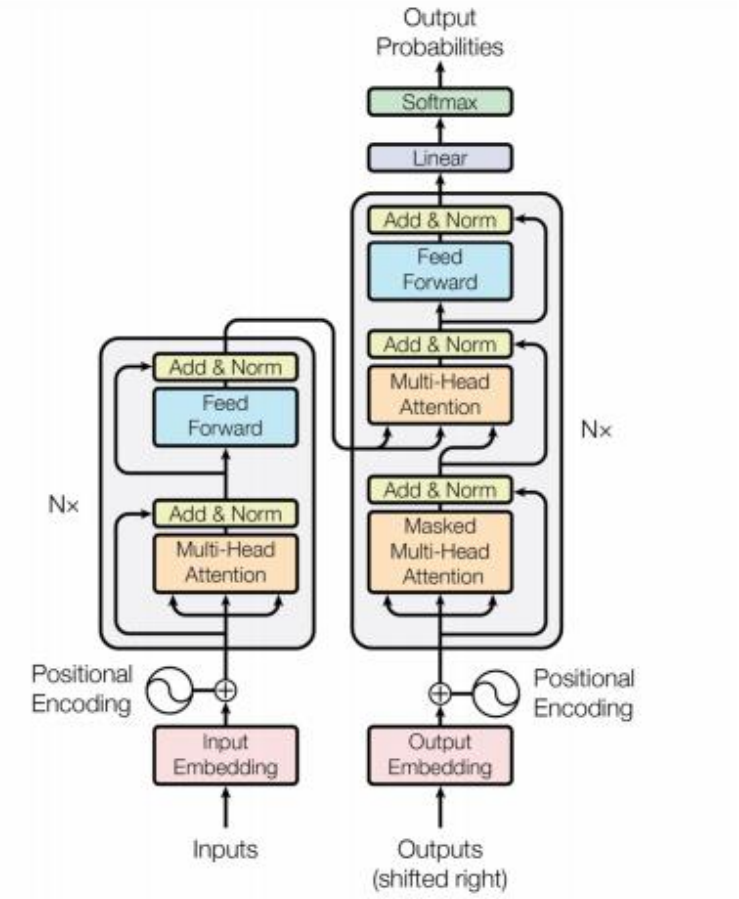
Огида:



РОЗДІЛ 3: ТРАНСФОРМЕРИ

Трансформери, забезпечують істотне вирішення давніх проблем, з якими стикаються послідовні маніпуляції, створює захоплюючий темп у дослідницькому просторі NLP. З моменту свого народження в 2017 році він дав багато результатів в деяких програмах NPL, особливо завдяки його складовим.Модель складається з блоків кодера та декодера з функцією активації softmax для нормалізації вихідних ймовірностей. Вхідними даними для моделі є послідовність даних. Вхідні слова вбудовуються і передаються через позиційні кодери, які присвоюють векторам слова на основі їх розташування в реченні; таким чином, витягуючи контекстне значення вхідних слів. Блоки кодера, що складаються з уваги з декількома головками та мережі прямої передачі, отримують вбудовування. Багатослойні шари уваги обчислюють вектори уваги для кожного входу, щоб представити, як кожне слово пов'язане з іншими словами в тому ж реченні, тобто, додатково фіксуючи контекстуальний зв'язок між словами в реченні. Вектори уваги передаються через мережу прямої передачі по одному вектору за раз до блоку декодера. Варто зазначити що паралелізація досягається в багатоголовому рівні уваги, оскільки мережі уваги незалежні. Модель трансформатора, спочатку розроблена для машинного перекладу, використовується для моделювання мови, завдяки чому вона застосовується для інших завдань NLP, таких як класифікація тексту, узагальнення документів, відповіді на запитання тощо.

Будова моделі трансформера:



Процесс розпізнавання емоцій:

РОЗДІЛ 4: МОДЕЛІ НА ОСНОВІ ТРАНСФОРМЕРІВ

Моделі, які перетворилися на архітектуру трансформатора, сильні та слабкі сторони. Transformer-XL, Generative Pre-Training (GPT), Bidirectional Encoder Representations from Transformer (BERT), Cross-Lingual Language Model (XLM), XLNet, Robustly Optimized BERT pre-training Approach (RoBERTa), та DistilBERT.

4.1 Transformer-XL

Трансформатор-XL був запропонований для вирішення існуючих обмежень, пов'язаних з vanilla transformer'ом. Transformer-XL пропонує архітектуру, що виходить за рамки контексту фіксованої довжини. Це досягається за допомогою повторення рівня сегмента та відносного позиційного кодування. Техніка повторення на рівні сегмента дозволяє повторно використовувати раніше розраховані уявлення сегментів як розширення контексту при обробці нових сегментів. Таким чином, дозволяючи потік контекстної інформації через межі фіксованого сегмента. Схема відносного позиційного кодування підтверджує техніку повторення, і вона базується, головним чином, на відносній відстані між маркерами. Transformer-XL може краще захоплювати довгострокові залежності, ніж трансформатор RNN та Al-Rfou. Модель також здатна фіксувати короткочасні залежності. Крім того, виведення моделі для більш затяжних контекстів швидше за допомогою Transformer-XL. Модель досягає результатів для задач моделювання мови та аудіоаналізу. Незважаючи на сильні сторони, модель може використовуватися лише в обмежених областях застосування, тобто аудіоаналізі та машині переклади. Архітектура кодер-декодер як і попередні моделі seq2seq, оригінальна модель Transformer використовувала архітектуру кодер-декодер. Кодер складається з шарів кодування, які ітеративно обробляють вхідні дані один шар за іншим, тоді як декодер складається з шарів декодування, які роблять те ж саме з виходом кодера. Функція кожного шару кодера полягає у створенні кодування, що

містить інформацію про те, які частини вхідних даних є релевантними один одному. Він передає свої кодування наступному шару кодера як вхідні дані. Кожен рівень декодера робить протилежне, беручи всі кодування та використовуючи їх включену контекстну інформацію для створення вихідної послідовності. Для досягнення цього кожен рівень кодера та декодера використовує механізм уваги. Для кожного входу увага зважує релевантність будь-якого іншого входу і витягує з них для отримання результату. Кожен рівень декодера має додатковий механізм уваги, який отримує інформацію з виходів попередніх декодерів, перш ніж рівень декодера витягує інформацію з кодування. І шари кодера, і декодера мають нейронну мережу прямого зв'язку для додаткової обробки вихідних даних і містять залишкові зв'язки та кроки нормалізації шару. Масштабована увага до точки Будівельні блоки трансформатора — це масштабовані одиниці уваги точкового продукту. Коли речення передається в модель трансформатора, ваги уваги обчислюються між кожним маркером одночасно. Блок уваги створює вбудовування для кожного токена в контексті, що містить інформацію про сам токен разом із зваженою комбінацією інших відповідних маркерів, кожен зважений за своєю вагою уваги. Для кожної одиниці уваги модель трансформатора вивчає три вагові матриці; вагові коефіцієнти запиту W_Q , ключові вагові коефіцієнти W_K і вагові значення значення W_V. Для кожного маркера і вхідне слово вбудоване x_i множиться на кожну з трьох вагових матриць, щоб створити вектор запиту $q_i = x_i W_Q$, ключовий вектор $k_i = x_i W_K$ і вектор значення $v_i = x_i W_V$. Ваги уваги обчислюються за допомогою запиту та ключових векторів: вага уваги a_{ij} від токена і до маркера j є точковим добутком між q_i і k_j . Вагові коефіцієнти уваги поділені на квадратний корінь з розмірності ключових векторів, $\sqrt{d_k}$, що стабілізує градієнти під час тренування, і пройшов через софтмакс, який нормалізує ваги. Той факт, що W_Q, W_K є різними матрицями, дозволяє увазі бути несиметричним: якщо маркер 'і' звертається до маркера 'j'(тобто $q_i * k_j$ великий), це робить не обов'язково означає, що токен j буде

відвідувати маркер і (тобто $q_j * k_i$ може бути невеликим). Вихід одиниці уваги для токена і є зваженою сумою векторів значень усіх лексем, зваженої за a_{ij} ,увага від токена і до кожного токена.

Кодер

Кожен кодер складається з двох основних компонентів: механізму самоуваги та нейронної мережі з прямим зв'язком. Механізм самоуваги приймає вхідні кодування від попереднього кодера і зважує їх релевантність один одному, щоб створити вихідні кодування. Нейронна мережа з прямим зв'язком додатково обробляє кожне вихідне кодування окремо. Ці вихідні кодування потім передаються наступному кодеру як його вхід, а також декодерам. Перший кодер бере інформацію про позицію та вбудовування вхідної послідовності як вхід, а не кодування. Інформація про позицію необхідна трансформатору для використання порядку послідовності, тому що жодна інша частина трансформатора не використовує це.

Декодер

Кожен декодер складається з трьох основних компонентів: механізму самоуваги, механізму уваги над кодуваннями та нейронної мережі з прямим зв'язком. Декодер функціонує так само, як і кодер, але вставляється додатковий механізм уваги, який замість цього витягує відповідну інформацію з кодувань, згенерованих кодерами. Як і перший кодер, перший декодер бере інформацію про позицію та вбудовування вихідної послідовності як вхід, а не кодування. Трансформатор не повинен використовувати поточний або майбутній вихід для прогнозування виходу, тому вихідна послідовність повинна бути частково замаскована, щоб запобігти цьому зворотному потоку інформації. За останнім декодером слід остаточне лінійне перетворення та шар softmax, щоб отримати вихідні ймовірності по словнику.

4.2 Генеральна попереднє навчання(Generative Pre-Training (GPT))

Використання структурно маркованих даних у порівнянні з немаркованими дає кращі результати у більшості завдань машинного навчання. Цей ефект поставив контрольоване навчання на межі найбільш успішних робіт з машинного навчання . Однак із появою Web 2.0 існує величезна кількість немаркованих даних, доступних для різних завдань. Структуровані, позначені та використані в завданнях машинного навчання, ці неструктуровані дані здатні досягати хороших результатів. Тим не менше, ручні зусилля та тривалі періоди, необхідні для позначення цих неструктурованих даних, роблять завдання складним. Щоб подолати цю проблему, GPT використовує підхід під наглядом під наглядом для моделювання мови за допомогою трансформаторних декодерів. GPT, який в основному використовується для подання тексту, складається з 12 трансформаторних шарів, 12 декодер-трансформатора головки уваги, який використовує масивні немарковані набори даних, тобто набір даних BooksCorpus шляхом попередньої підготовки та точної настройки їх на обмежених контрольованих наборах даних. Єдине завдання GPT - передбачити наступний маркер у послідовності. Вхідні дані GPT - це вкладання ваги вхідних текстів плюс їх позиційні вбудовування для вилучення контексту. Вхідні дані передаються на рівень уваги з декількома головками в 12 шаруватих блоках декодера трансформатора, шар прямого подавання, а потім softmax виводить розподіл ймовірностей.

Математичний вигляд:

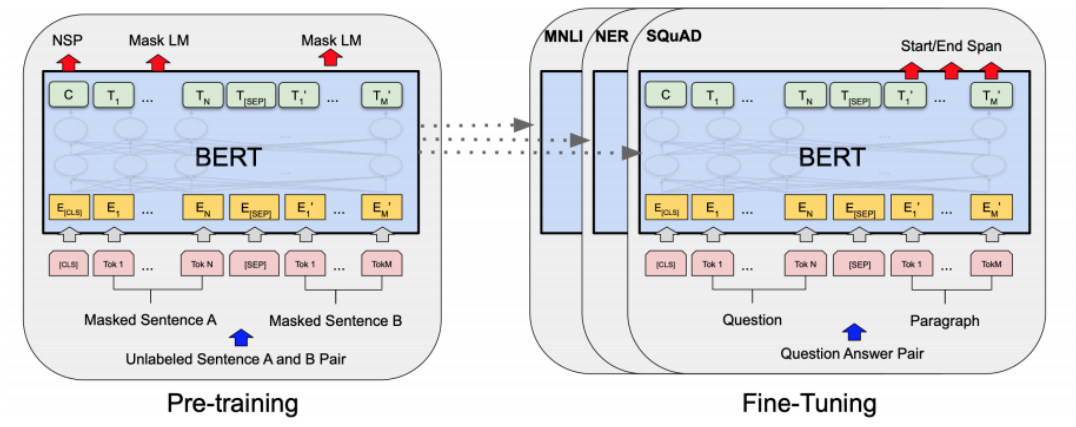
$$h_0 = U * W_e + W_p$$
$$h_l = transformer\_block(h_{l-1}) \forall i \in [1, n]$$
$$P(u) = softmax(h_n * W_e^T)$$

де U = (u − k, ..., u − 1) - вектор контексту, k - розмір вікна контексту, n - кількість шарів, W_e - матриця вбудовування токена, а W_p - вбудовування позиції матриця. Модель GPT може бути попередньо навчена та

відрегульована для інших завдань. Масштабування моделі GPT створило архітектуру GPT-2 та GPT-3. Як і модель GPT, друга версія, тобто GPT-2 , була розроблена для передбачення наступного речення у реченнях та встановлює, що мовні моделі можуть вивчати завдання без безпосереднього нагляду. Він має архітектуру, подібну до GPT, але з шаром нормалізації на вході кожного підблоку і після остаточного шару самоуваги. GPT-2 має чотири розміри моделей, великі та малі моделі розроблені з 40-гігабайтного тексту із приблизно 1,5 мільярда та 117 мільйонів параметрів, що піддаються навчанню, відповідно, пропонуючи масштаб до моделі GPT. Його більший склад даних робить його придатним для різноманітних завдань NLP, таких як відповіді на запитання, висновок природною мовою, класифікація тексту, оцінка семантичної подібності тощо. Модель GPT-3 масштабується на моделі GPT-2 додаткові 175 мільярдів параметрів, що піддаються навчанню. Його модельна архітектура така ж, як у GPT-2, за винятком того, що шари трансформатора мають черговані щільні та локально смугові розріджені схеми уваги. Загалом було підготовлено вісім різних розмірів моделі з діапазоном розмірів від 125 мільйонів до 175 мільярдів параметрів. Сильні сторони моделі GPT висвітлюються наступним чином. Під час застосування GPT покращується лексична стійкість. Попередньо навчена модель GPT може бути налаштована на виконання інших завдань без налаштування моделі. Модель GPT також перевершує різні моделі, навчені наборам даних для конкретних доменів, і дає результати щодо різноманітного завдання доменного моделювання мови. Що стосується GPT-2 та GPT-3, то вони взагалі не потребують точної настройки. Обмеження моделі GPT полягають у ресурсоємності моделі.

4.3 Bidirectional Encoder Representations from Transformers (BERT)

Представлення двонаправленого кодера від трансформаторів Девліна використовував кодери в трансформаторі, як підструктуру для попередньої підготовки моделей для задач NLP, таких як аналіз настрою, відповіді на запитання ,узагальнення тексту, тощо. Виконання BERT для цих завдань відбувається в два етапи, а саме. , попередня підготовка для розуміння мови та точне налаштування для конкретного завдання. BERT може розуміти мову, навчаючись на механізмах моделювання маскованих мов та механізму передбачення наступного речення. BERT взяв на себе сліпоту з моделюванню замаскованих мов, щоб вивчити двонаправлений контекст у реченнях. Таким чином, він бере як вхідні випадкові речення, маскує деякі слова в реченнях і реконструює замасковані слова з навколишніх текстів на виході. Його здатність вводити два речення одночасно та визначати, чи друге речення постає після першого, таким чином він передбачує наступне речення. Ця здатність допомагає моделі підтримувати віддалені стосунки між текстами. Після попередньої підготовки модель тренується щодо завдання NLP, виконуючи контрольоване навчання на наборі даних і замінюючи повністю підключений вихідний сигнал BERT новим набором вихідних шарів. Модель BERT тренується швидше, оскільки інші параметри моделі лише тонко налаштовані, крім вихідних параметрів, вивчених з нуля. Попереднє навчання та точне налаштування:



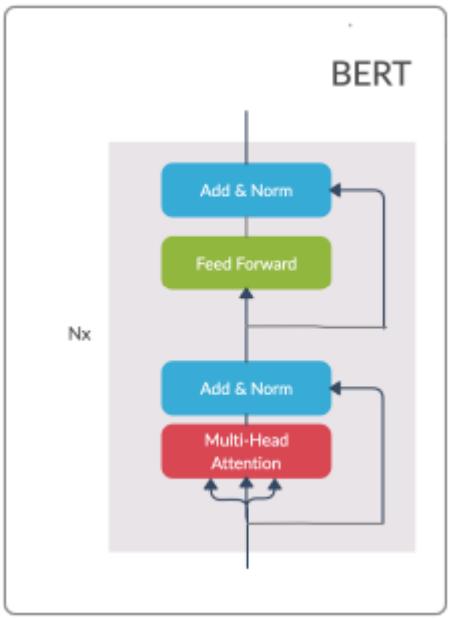
BERT має дві моделі, тобто BERT-база та BERT-великі моделі. Модель BERT-бази складається з 12-шарових блоків кодувальних трансформаторів, кожен з яких містить 12-ти головних шарів самообслуговування та 768 прихованих шарів, загалом виробляючи ≈ 110 мільйонів параметрів. З іншого боку, BERT-large складається з 24-шарових блоків кодувальних трансформаторів, кожен блок містить 24 головних шарів самоорієнтації і видає загалом ≈ 340 мільйонів параметрів. Продуктивність BERT залежить від типу моделі, тобто BERT-large може досягти вищої точності, ніж BERT-основа. Однак це покращення точності з використанням BERT-великого коштує за рахунок того, що потрібні більш широкі ресурси для завершення. Плюси BERT включають його здатність обробляти контекстну вилучення інформації завдяки її двонаправленій здатності; він швидше тренується і використовувався в широкому діапазоні програм мовного моделювання. Однак для BERT зберігаються наступні недоліки: він обмежений одномовною класифікацією, довжина вхідних речень також обмежує його, він страждає від прагматичного висновку, і використання BERT-large може мати тенденцію бути обчислювально дорогим. Мета будь-якої даної техніки НЛП полягає в тому, щоб зрозуміти людську мову так, як нею говорять природно. У випадку BERT це, як правило, означає передбачення слова в пустому місці. Для цього моделі зазвичай потрібно тренувати, використовуючи велике сховище спеціалізованих позначених навчальних даних. Це вимагає трудомісткого ручного маркування даних командами лінгвістів. BERT, однак, був попередньо навчений, використовуючи лише немаркований, простий текстовий корпус (всю англійську Вікіпедію). Він продовжує без нагляду вивчати текст без міток і вдосконалюватися, навіть якщо він використовується в практичних додатках (наприклад, пошук Google). Його попередня підготовка служить базовим шаром «знань», з яких можна побудувати. Звідси BERT може адаптуватися до постійно зростаючого вмісту та запитів для пошуку та бути точно налаштованим відповідно до специфікацій користувача. Цей процес відомий як трансферне навчання. Як зазначалося вище, BERT стало можливим

завдяки дослідженню Google Transformers. Трансформатор — це частина моделі, яка надає BERT підвищену здатність до розуміння контексту та неоднозначності мови. Трансформатор робить це, обробляючи будь-яке дане слово по відношенню до всіх інших слів у реченні, а не обробляє їх по одному. Переглядаючи всі навколишні слова, Transformer дозволяє моделі BERT зрозуміти повний контекст слова, а отже, краще зрозуміти наміри шукача. Це контрастує з традиційним методом обробки мови, відомим як вбудовування слів, у якому попередні моделі, такі як GloVe та word2vec, відображали кожне слово у вектор, який представляє лише один вимір, фрагмент значення цього слова. Ці моделі вбудовування слів вимагають великих наборів даних із мітками. Хоча вони вправні в багатьох загальних завданнях НЛП, вони не справляються з важким контекстом, прогнозним характером відповідей на запитання, оскільки всі слова в певному сенсі закріплені за вектором або значенням. BERT використовує метод замаскованого мовного моделювання, щоб утримати слово у фокусі, щоб воно не «бачило себе» – тобто, щоб воно мало фіксоване значення незалежно від його контексту. Потім BERT змушений ідентифікувати замасковане слово на основі лише контексту. У BERT слова визначаються оточенням, а не попередньо фіксованою ідентичністю.

Для чого використовується BERT:

- Завдання генерації мови від послідовності до послідовності, наприклад:
    - Відповідь на запитання
    - Реферат узагальнення
    - Прогноз речення
    - Формування розмовної відповіді
- Завдання на розуміння природної мови, такі як:
    - Багатозначність і кореференція
    - Розшифровка сенсу слова
    - Висновок природної мови
    - Класифікація настроїв

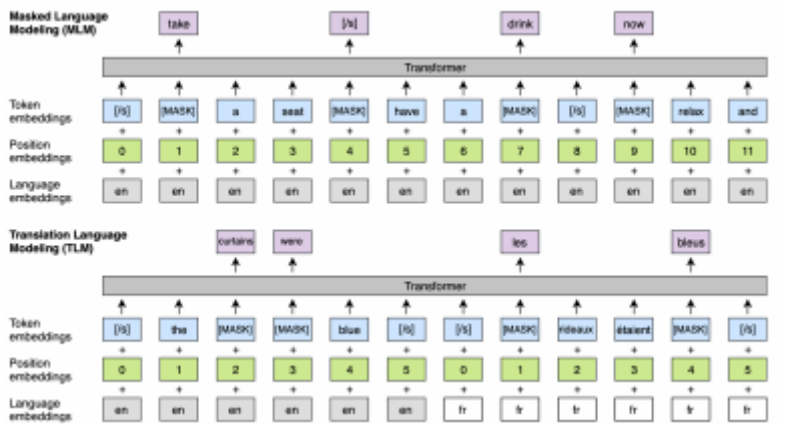
Структура моделі BERT:



4.4 Cross-Lingual Language Models

Модель BERT для досягнення багатообіцяючих результатів у задачах NLP вимагала одномовних даних. Cross-Lingual Language Models покращують BERT для завдань класифікації та перекладу шляхом попередньої підготовки міжмовних моделей для NLP. Модель забезпечує розширення, представлену в BERT, використовуючи концепцію перекладу моделі, запропоновану Ламплом. Корпуси складаються з одних і тих самих речень двома різними мовами. Речення, по одному з кожного мовного корпусу, подаються паралельно до модельних введення. Модель засвоює контекст, вивчаючи тексти на кожній мові та обох, щоб передбачити замасковані слова. Сильні сторони включають: модель враховує двомовні завдання класифікації, паралельне введення тексту стало можливим завдяки Cross-Lingual Language Models, і його можна попередньо навчити для досягнення результатів у двомовних завданнях. Однак Cross-Lingual Language Models страждає від обмеження фіксованої довжини і застосовувався в обмеженій кількості завдань NLP.

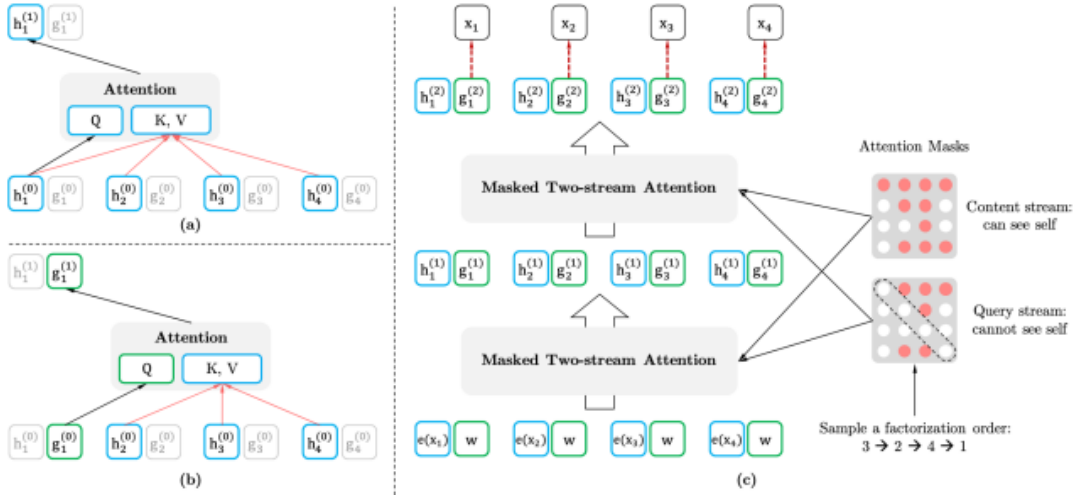
Структурна модель Cross-Lingual Language Models:



4.5 XLNet

XLNet є авторегресивною мовною моделлю, яка використовує концепції Permutation Language Model (PLM) та моделі Transformer-XL. Як варіант BERT, суттєва різниця між BERT та XLNet пов’язана з ціллю їх навчання. XLNet використовує мету перестановки, тоді як BERT маскує дані та намагається передбачити масковані дані за допомогою двонаправленого контексту. За допомогою PLM модель може вивчати двонаправлений контекст, навчаючи всі можливі перестановки слів у реченні. Потім використання позиційного механізму кодування та повторення в Transformer-XL викорінює проблему фіксованої довжини в BERT. До сильних сторін моделі XLNet належать: можливість витягувати контекстну інформацію завдяки впровадженню PLM у модель, як відомо, модель працює ефективніше, ніж BERT, у більш широкому діапазоні програм мовного моделювання. Найголовніше, що він викорінює обмеження BERT з фіксованою довжиною, а також може бути попередньо навчений для досягнення найсучасніших результатів . Однак усі ці переваги призводять до деяких обчислювальних складностей, що робить XLNet обчислювально інтенсивним.

Структурна модель XLNet:



4.6 Robustly Optimized BERT pre-training Approach (RoBERTa)

Надійно оптимізований підхід до підготовки BERT (RoBERTa) - це варіант BERT, який намагається остаточно оптимізувати BERT, змінивши різні методологічні параметри у початковій версії BERT. Ці налаштування допомогли дослідити сутність гіперпараметрів, таких як вплив більших наборів даних перед підготовкою, вплив статики на динамічний MLM, внесок розмірів пакетів, доречність кодування тексту та важливість техніки передбачання наступного речення у BERT. Динамічне маскування використовувалось замість статичного маскування, що використовується для BERT; це допомогло уникнути використання однієї маски для кожного етапу навчання. Використання динамічної маски дозволило трохи покращити багатозадачність, але не окремі завдання. Були визначені різні випадки для парних сегментів + передбачення наступного речення, пари речень + передбачення наступного речення, повних речень та речень документа, щоб дослідити необхідність передбаченнь наступного речення. Зменшенням порядку виконання були речення документа, повні речення, пари сегментів та пари речень відповідно. Однак випадок із повними реченнями був реалізований як найбільш ефективний. Переваги RoBERTa такі: більш масові дані попередньої підготовки дають кращі результати у виконанні різних завдань. RoBERTa також перевершує XLNet та BERT у подальших завданнях передбачення наступного речення. Однак мінуси RoBERTa мають ресурсомісткий характер, оскільки вимоги до даних є великими та збільшують обчислювальну складність. Чим вищий етап попередньої підготовки, тим краща продуктивність, але це стає обчислювально дорогим, вимагаючи більше часу для виконання.

RoBERTa ґрунтується на стратегії маскування мови BERT, у якій система вчиться передбачати навмисно приховані частини тексту в інших некоментованих прикладах мови. RoBERTa, який був реалізований в PyTorch, змінює ключові гіперпараметри в BERT, включаючи видалення мети попередньої підготовки до наступного речення BERT і навчання з набагато більшими міні-пакетами та швидкістю навчання. Це дозволяє RoBERTa покращити ціль моделювання замаскованої мови в порівнянні з BERT і веде до кращої продуктивності завдань нижче по ходу.

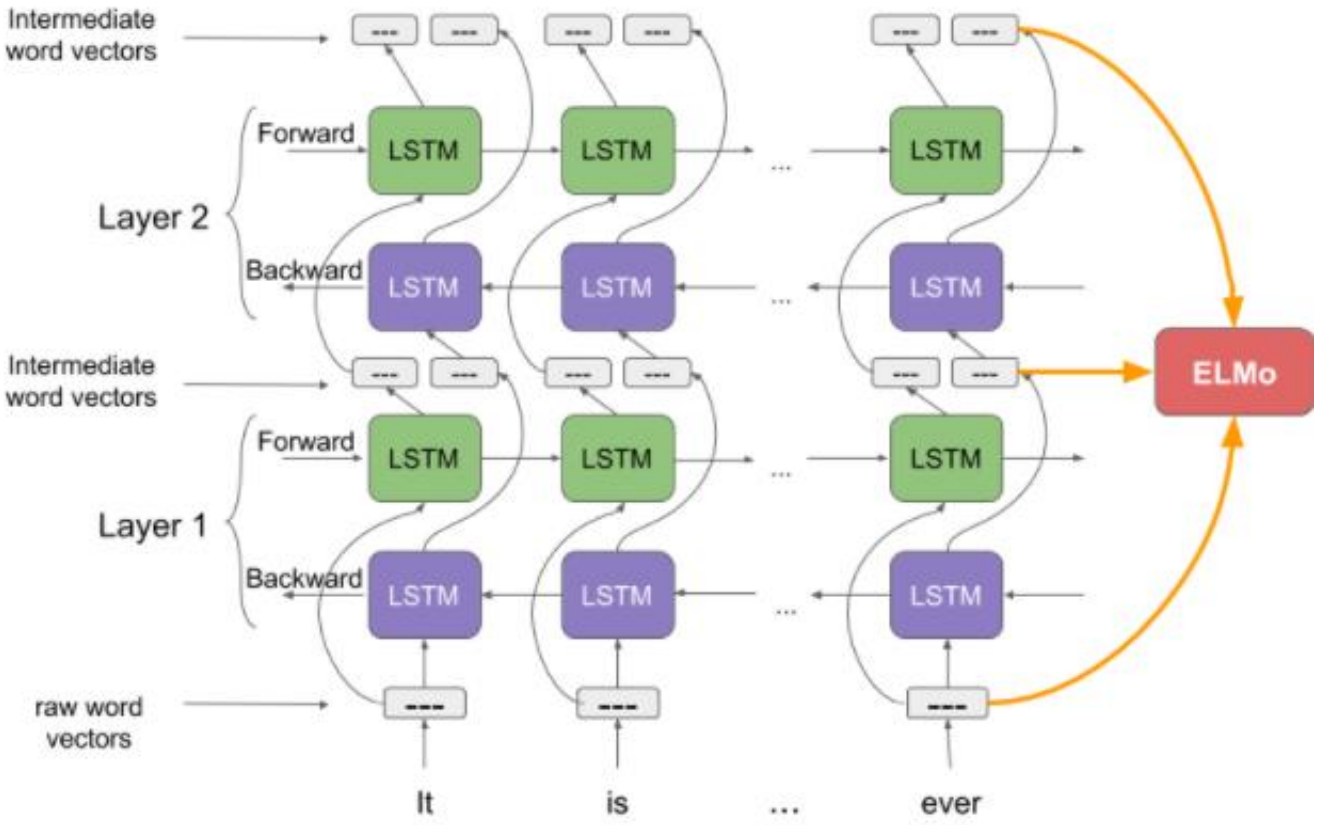
4.7 DistilBERT

Концепція дистиляції в нейронних мережах, запропонована, спрямована на пришвидшення моделей. Він досягає цього, замінюючи більш масивні архітектури широкими параметрами на полегшену версію тієї ж архітектури, яка має меншу кількість параметрів. DistilBERT працює таким чином. Він бере архітектуру початкової версії BERT, зменшує кількість шарів у базовій моделі BERT у 2 рази, видаляє вбудовані маркери та пулери, щоб отримати набагато меншу та швидшу версію BERT для загального використання. Він застосовує динамічне маскування та ігнорує передбачення наступного речення, запропоновані для кращого висновку. Базовий показник загальної мовної оцінки розуміння(General Language Understanding Evaluation (GLUE)) був використаний для оцінки ефективності моделі щодо подальших завдань та порівняно добре працював. DistilBERT спеціально зберіг 97% ефективності роботи BERT, навіть коли було використано на 40% менше параметрів. DistilBERT також був на 60% швидшим за BERT. DistilBERT здатний моделювати мови та може пройти попередню підготовку з інших завдань моделювання мови. Модель швидша та легша за оригінальну BERT. Якості демонструють сильні сторони DistilBERT. Однак проблема обмеження BERT фіксованої довжини зберігається у DistilBERT.
Можна навчити класифікатор, використовуючи вихідні дані іншого класифікатора з реальними значеннями як цільову значення, ніж використання фактичних міток реальної істини.

- Навчена модель класифікатора призначає ймовірності всім міткам
- Відносна величина цих неправильних ймовірностей впливає на здатність моделі до узагальнення.
- Наприклад: зображення автобуса можна помилково прийняти за автомобіль за моделлю класифікації зображень, але навряд чи буде помилково прийнятий за крісло.

4.8 ELMO (Embeddings from Language Models)

ELMo - це новий спосіб представити слова у векторах або вбудованих елементах. Один з найбільших проривів у цьому плані досяг завдяки ELMo, сучасному фреймворку NLP, розроблений AllenNLP. Ці вбудовані слова корисні для досягнення найсучасніших результатів у ряді завдань NLP. Вектори слів ELMo обчислюються поверх двошарової двонаправленої моделі мови. Ця модельмає два шари, складені разом. Кожен шар має 2 проходи - прохід вперед і назад:



Розділ 5 :Порівняння BERT, RoBERTa та DistilBERT:

Порівняння	BERT	RoBERTa	DistilBERT
Параметри	Base:110M Large:340M	Base:125 Large:355	Base:66
Layers/Hidden Dimensions/Self-Attension Heads	Base:12/768/12 Large:24/1024/16	Base:12/768/12 Large:24/1024/16	Base:6/768/12
Час тренування	Base:8xV100x12d Large:280xV100x1d	1024xV100x1d	Base:8xV100x3.5d
Продуктивність	Перевиповнення	88.5 on GLUE	97% від Базової BERT продуктивності

BERT

Моделі BERT використовують неконтрольоване вивчення великої кількості немічених даних (маскованих даних) для попередньої підготовки, і ці попередньо навчені моделі можуть бути використані для точного налаштування невеликих контрольованих даних для досягнення великої точності. Bert використовує трансформатори і складає безліч трансформаторних кодерів поверх кожного. Він використовував двонаправлене навчання на противагу спрямованим моделям. BERT намагається зрозуміти контекст кожного слова зліва та справа.

Це подбає про завантаження, кешування та завантаження моделі та необхідних налаштувань. Бібліотека побудована навколо трьох класів в pytorch.

1) класи моделей, які є моделями піторхів для кожної 6 архітектур, наприклад: BertModel

2) класи конфігурації: Усі файли конфігурації для побудови моделі

3) класи токенізера: в яких зберігається словниковий запас для кожної моделі.

RoBERTa

RoBERTa, розроблена Facebook, перевершила XLNET та BERT у тесті GLUE. Вони вдосконалили MLM BERT за допомогою стратегії, коли вони видаляють передбачення наступного речення в BERT та вводять динамічне маскування таким чином. Вони також покращили налаштування гіперпараметрів для BERT, а також навчили модель BERT, використовуючи набагато більші міні-пакети та швидкість навчання.

DistilBERT

Коли всі згадані моделі давали нам збільшення продуктивності. Так, DistilBERT призначений для порушнння точності прогнозування з меншими обчислювальними витратами. Це може бути в основному використано на виробництві, оскільки можуть використовуватися пристрої графічного процесора з відносно низьким рівнем живлення. Тут більша модель BERT використовується для перегонки знань, що є технікою стиснення, при якій малу модель навчають відтворювати поведінку більшої моделі.

Розділ 6 : Модель на основі BERT для текстового виявлення емоцій:

```
import numpy as np
import pandas as pd


import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
import torch.nn.functional as F
from transformers import BertTokenizer, BertConfig,AdamW, BertForSequenceClassification,get_linear_schedule_with_warmup


import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score,matthews_corrcoef

from tqdm import tqdm, trange,tnrange,tqdm_notebook
import random
import os
import io


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
n_gpu = torch.cuda.device_count()
torch.cuda.get_device_name(0)

SEED = 19

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if device == torch.device("cuda"):
    torch.cuda.manual_seed_all(SEED)

device = torch.device("cuda")
```

```
df_train = pd.read_csv('C://Users//Scandiy//Desktop//train.txt', names=['text', 'emotion'], sep=';')
df_val = pd.read_csv('C://Users//Scandiy//Desktop//val.txt', names=['text', 'emotion'], sep=';')
df_test = pd.read_csv('C://Users//Scandiy//Desktop//test.txt', names=['text', 'emotion'], sep=';')
train_data.head()
```

```
df = pd.concat([df_train,df_test,df_val])

df['label'].unique()
```

```
array(['sadness', 'anger', 'love', 'surprise', 'fear', 'joy'],
      dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['label_enc'] = labelencoder.fit_transform(df['label'])
```

```
df[['label','label_enc']].drop_duplicates(keep='first')
```

	label	label_enc
0	sadness	4
2	anger	0
3	love	3
6	surprise	5
7	fear	1
8	joy	2

```
df.rename(columns={'label':'label_desc'},inplace=True)
df.rename(columns={'label_enc':'label'},inplace=True)
```

```
sentences = df.sentence.values

print("Distribution of data based on labels: ",df.label.value_counts())

MAX_LEN = 256

tokenizer = BertTokenizer.from_pretrained (' bert-base-uncased ', do_lower_case = True)
```

```
input_ids=[tokenizer.encode(sent,add_special_tokens=True,max_length=MAX_LEN,pad_to_max_length=True) for sent in sentences]
labels = df.label.values

print("Actual sentence before tokenization: ",sentences[2])
print("Encoded Input from dataset: ",input_ids[2])

attention_masks = []
attention_masks = [[float(i>0) for i in seq] for seq in input_ids]
print(attention_masks[2])
```

```
Distribution of data based on labels:  2    6761
4    5797
0    2709
1    2373
3    1641
5     719
```

```
train_inputs,validation_inputs,train_labels,validation_labels=
train_test_split(input_ids,labels,random_state=41,test_size=0.1)

train_masks,validation_masks,_,_=
train_test_split(attention_masks,input_ids,random_state=41,test_size=0.1)


train_inputs = torch.tensor(train_inputs)
validation_inputs = torch.tensor(validation_inputs)
train_labels = torch.tensor(train_labels)
validation_labels = torch.tensor(validation_labels)
train_masks = torch.tensor(train_masks)
validation_masks = torch.tensor(validation_masks)

batch_size = 32

train_data = TensorDataset(train_inputs,train_masks,train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader=DataLoader(train_data,sampler=train_sampler,batch_size=batch_size)

validation_data=TensorDataset(validation_inputs,validation_masks,validation_labels)
validation_sampler = RandomSampler(validation_data)
```

```
validation_dataloader = DataLoader(validation_data,sampler=validation_sampler,batch_size=batch_size)


model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=6).to(device)

lr = 2e-5
adam_epsilon = 1e-8

epochs = 2

num_warmup_steps = 0
num_training_steps = len(train_dataloader)*epochs

optimizer=AdamW(model.parameters(),lr=lr,eps=adam_epsilon,correct_bias=False
scheduler=get_linear_schedule_with_warmup(optimizer,num_warmup_steps=num_warmup_steps, num_training_steps=num_training_steps)


train_loss_set = []
learning_rate = []

model.zero_grad()

for _ in tnrange(1,epochs+1,desc='Epoch'):
  print("<" + "="*22 + F" Epoch {_} "+ "="*22 + ">")
  batch_loss = 0

  for step, batch in enumerate(train_dataloader):
    model.train()


    batch = tuple(t.to(device) for t in batch)

    b_input_ids, b_input_mask, b_labels = batch

    outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
    loss = outputs[0]

    loss.backward()

    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
```

```
    optimizer.step()

    scheduler.step()

    optimizer.zero_grad()

    batch_loss += loss.item()

  avg_train_loss = batch_loss / len(train_dataloader)

  for param_group in optimizer.param_groups:
    print("\n\tCurrent Learning rate: ",param_group['lr'])
    learning_rate.append(param_group['lr'])

  train_loss_set.append(avg_train_loss)
  print(F'\n\tAverage Training loss: {avg_train_loss}')

  model.eval()


  eval_accuracy,eval_mcc_accuracy,nb_eval_steps = 0, 0, 0

    for batch in validation_dataloader:
    batch = tuple(t.to(device) for t in batch
    b_input_ids, b_input_mask, b_labels = batch

    with torch.no_grad():

      logits = model(b_input_ids, token_type_ids=None,
attention_mask=b_input_mask)

    logits = logits[0].to('cpu').numpy()
    label_ids = b_labels.to('cpu').numpy()

    pred_flat = np.argmax(logits, axis=1).flatten()
    labels_flat = label_ids.flatten()


df_metrics=pd.DataFrame({'Epoch':epochs,'Actual_class':labels_flat,'Predicted_cla
ss':pred_flat})

    tmp_eval_accuracy = accuracy_score(labels_flat,pred_flat)
    tmp_eval_mcc_accuracy = matthews_corrcoef(labels_flat, pred_flat)
```

```
    eval_accuracy += tmp_eval_accuracy
    eval_mcc_accuracy += tmp_eval_mcc_accuracy
    nb_eval_steps += 1

  print(F'\n\tValidation Accuracy: {eval_accuracy/nb_eval_steps}')
  print(F'\n\tValidation MCC Accuracy: {eval_mcc_accuracy/nb_eval_steps}')
```

```
 <====================== Epoch 1 ======================>

         Current Learning rate:  1.3333333333333333e-05

         Average Training loss: 0.39324043376195916

         Validation Accuracy: 0.933531746031746

         Validation MCC Accuracy: 0.9128122385092313
 <====================== Epoch 2 ======================>

         Current Learning rate:  6.666666666666667e-06

         Average Training loss: 0.11971633532622145

         Validation Accuracy: 0.933531746031746

         Validation MCC Accuracy: 0.9137053098662156
```

```
from sklearn.metrics import confusion_matrix,classification_report
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    import itertools
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
```

```
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

```
df[['label','label_desc']].drop_duplicates(keep='first')
```

	label	label_desc
0	4	sadness
2	0	anger
3	3	love
6	5	surprise
7	1	fear
8	2	joy

```
              precision    recall  f1-score   support

     sadness    1.00000   1.00000   1.00000         3
         joy    1.00000   0.50000   0.66667         2
       anger    1.00000   1.00000   1.00000         5
        fear    0.83333   1.00000   0.90909         5
    surprise    1.00000   1.00000   1.00000         1

    accuracy                        0.93750        16
   macro avg    0.96667   0.90000   0.91515        16
weighted avg    0.94792   0.93750   0.92992        16
```

Розділ 7 : Модель на основі RoBERTa для текстового виявлення емоцій

Імпортуємо бібліотеки:

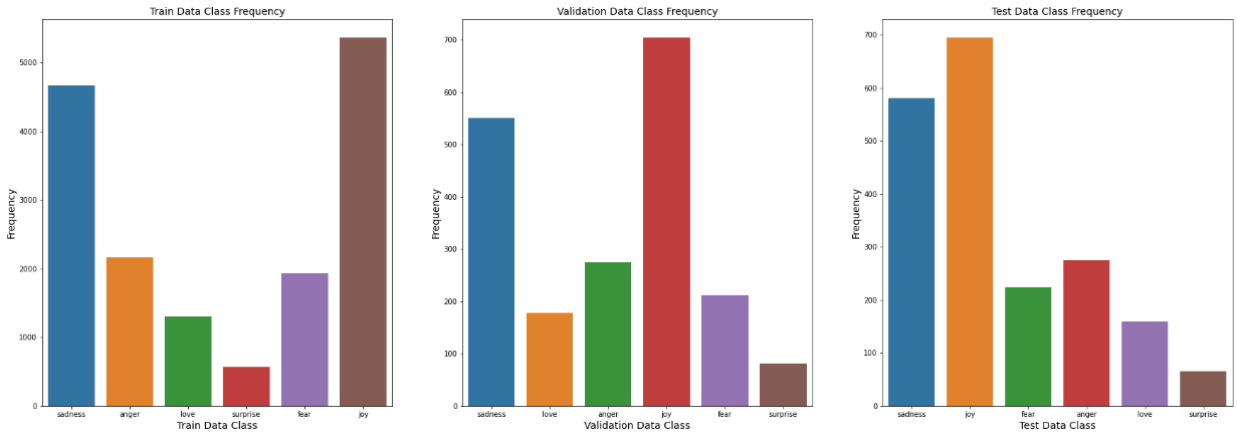
```
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
!pip install contractions
import contractions
import re
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import seaborn as sns
```

Завантажуємо дані:

```
train_data = pd.read_csv('C://Users//Scandiy//Desktop//train.txt', names=['text',
'emotion'], sep=';')
val_data = pd.read_csv('C://Users//Scandiy//Desktop//val.txt', names=['text',
'emotion'], sep=';')
test_data = pd.read_csv('C://Users//Scandiy//Desktop//test.txt', names=['text',
'emotion'], sep=';')
train_data.head()
data = {'Train Data': train_data, 'Validation Data': val_data, 'Test Data': test_data}
for temp in data:
    print(temp)
    print(data[temp].isnull().sum())
    print('*'*20)
```

Будуємо глафіки, які показують скільки випадків кожної емоції:

```
bar, ax = plt.subplots(1,3, figsize=(30, 10))
for index, temp in enumerate(data):
    sns.countplot(ax = ax[index],x = 'emotion', data = data[temp])
    ax[index].set_title(temp+' Class Frequency', size=14)
    ax[index].set_ylabel('Frequency', size=14)
    ax[index].set_xlabel(temp+' Class', size=14)
```



Визначаємо негативні слова:

```
def preprocess(sentence):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    sentence = re.sub('[^A-z]', ' ', sentence)
    negative = ['not', 'neither', 'nor', 'but', 'however', 'although', 'nonetheless',
'despite', 'except', 'even though', 'yet']
    stop_words = [z for z in stop_words if z not in negative]
    preprocessed_tokens = [lemmatizer.lemmatize(contractions.fix(temp.lower()))
for temp in sentence.split() if temp not in stop_words]
    return ' '.join([x for x in preprocessed_tokens]).strip()


train_data['text'] = train_data['text'].apply(lambda x: preprocess(x))
val_data['text'] = val_data['text'].apply(lambda x: preprocess(x))
test_data['text'] = test_data['text'].apply(lambda x: preprocess(x))
```

Тренуємо:

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
train_x, train_y = ros.fit_resample(np.array(train_data['text']).reshape(-1, 1),
np.array(train_data['emotion']).reshape(-1, 1))
train = pd.DataFrame(list(zip([x[0] for x in train_x], train_y)), columns = ['text',
'emotion'])

from sklearn import preprocessing
le = preprocessing.OneHotEncoder()
y_train= le.fit_transform(np.array(train['emotion']).reshape(-1, 1)).toarray()
y_test= le.fit_transform(np.array(test_data['emotion']).reshape(-1, 1)).toarray()
y_val= le.fit_transform(np.array(val_data['emotion']).reshape(-1, 1)).toarray()
```

```
from transformers import RobertaTokenizerFast
tokenizer = RobertaTokenizerFast.from_pretrained("roberta-base")


def roberta_encode(data,maximum_length) :
  input_ids = []
  attention_masks = []


  for i in range(len(data.text)):
      encoded = tokenizer.encode_plus(

        data.text[i],
        add_special_tokens=True,
        max_length=maximum_length,
        pad_to_max_length=True,

        return_attention_mask=True,

      )

      input_ids.append(encoded['input_ids'])
      attention_masks.append(encoded['attention_mask'])
  return np.array(input_ids),np.array(attention_masks)



max_len = max([len(x.split()) for x in train_data['text']])
train_input_ids,train_attention_masks = roberta_encode(train, max_len)
test_input_ids,test_attention_masks = roberta_encode(test_data, max_len)
val_input_ids,val_attention_masks = roberta_encode(val_data, max_len)
```

Створюємо модель :

```
def create_model(bert_model, max_len):
    input_ids = tf.keras.Input(shape=(max_len,),dtype='int32')
    attention_masks = tf.keras.Input(shape=(max_len,),dtype='int32')

    output = bert_model([input_ids,attention_masks])
    output = output[1]

    output = tf.keras.layers.Dense(6, activation='softmax')(output)
    model = tf.keras.models.Model(inputs = [input_ids,attention_masks],outputs =
output)
```

```
    model.compile(Adam(lr=1e-5), loss='categorical_crossentropy',
metrics=['accuracy'])
    return model


from transformers import TFRobertaModel
roberta_model = TFRobertaModel.from_pretrained('roberta-base')



model = create_model(roberta_model, max_len)
model.summary()
```

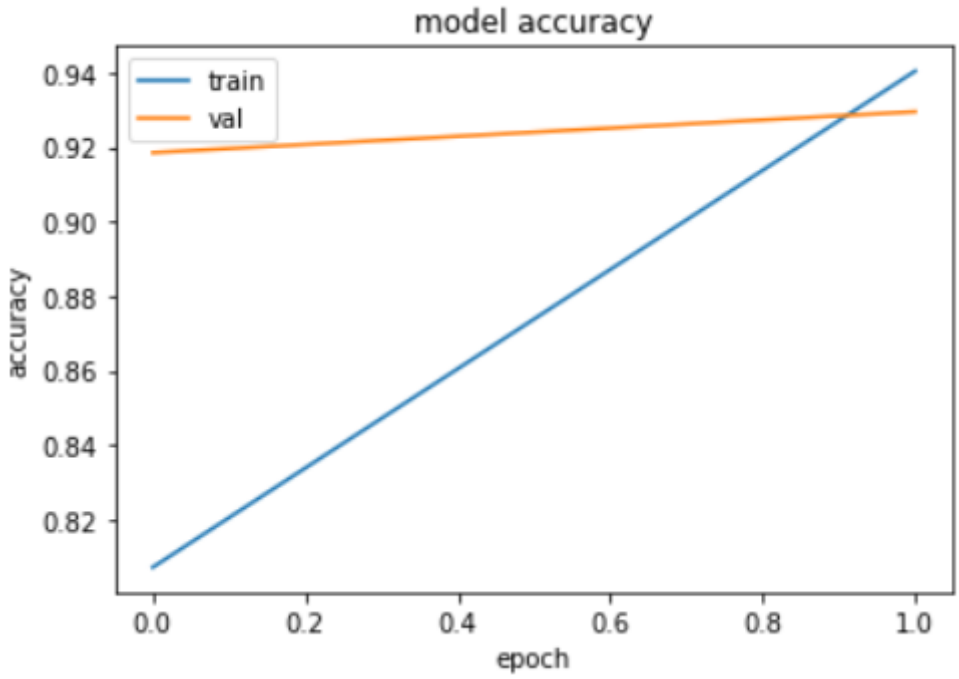
Тренуємо дані :

```
history = model.fit([train_input_ids,train_attention_masks], y_train,
validation_data=([val_input_ids,val_attention_masks], y_val),
epochs=2,batch_size=100)
```

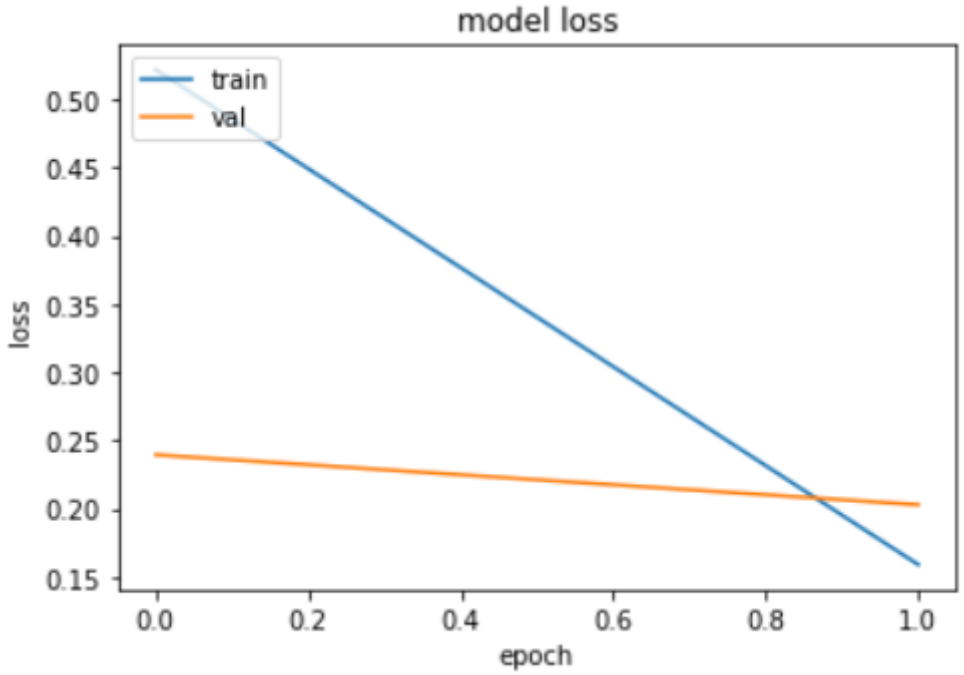
```
Epoch 1/2
322/322 [==============================] - 6933s 22s/step - loss: 0.5216 - accuracy: 0.8071 - val_loss: 0.2395 - val_accuracy:
0.9185
Epoch 2/2
322/322 [==============================] - 6483s 20s/step - loss: 0.1591 - accuracy: 0.9404 - val_loss: 0.2029 - val_accuracy:
0.9295
```

Будуємо графіки :

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
result = model.predict([test_input_ids,test_attention_masks])
y_pred = np.zeros_like(result)
y_pred[np.arange(len(result)), result.argmax(1)] = 1
```

```
from sklearn.metrics import accuracy_score, f1_score
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy ', accuracy)
f1 = f1_score(y_test, y_pred, average = 'macro')
print('F1 Score :', f1)
```

```
Accuracy  0.921
F1 Score : 0.8886050899811894
```

```
model.save_weights('my_checkpoint')
```

```
def plot_result(result):
   sns.barplot(x = 'Category', y = 'Confidence', data = result)
   plt.xlabel('Categories', size=14)
```

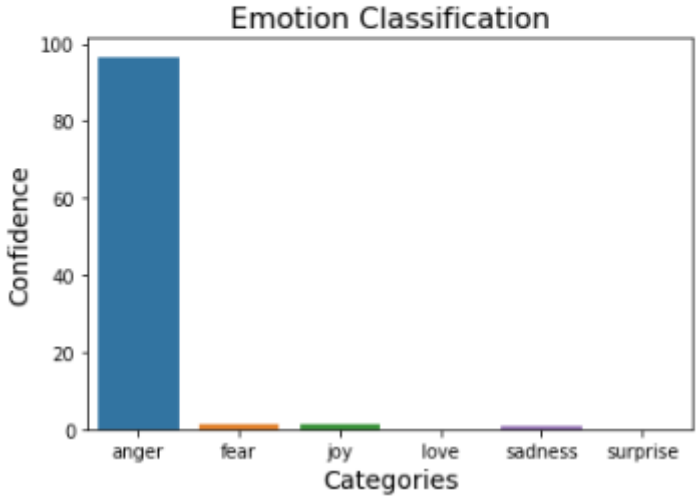
```
    plt.ylabel('Confidence', size=14)
    plt.title('Emotion Classification', size=16)


def roberta_inference_encode(data,maximum_length) :
    input_ids = []
    attention_masks = []



    encoded = tokenizer.encode_plus(
    data,
    add_special_tokens=True,
    max_length=maximum_length,
    pad_to_max_length=True,

    return_attention_mask=True

    )

    input_ids.append(encoded['input_ids'])
    attention_masks.append(encoded['attention_mask'])
    return np.array(input_ids),np.array(attention_masks)
```

Висновок :

```
def inference(text_sentence, max_len):
    preprocessed_text = preprocess(text_sentence)
    input_ids, attention_masks = roberta_inference_encode(preprocessed_text,
maximum_length = max_len)
    model = create_model(roberta_model, 43)
    model.load_weights('my_checkpoint')
    result = model.predict([input_ids, attention_masks])
    #le.categories_[0] = ['anger' 'fear' 'joy' 'love' 'sadness' 'surprise']
    result = pd.DataFrame(dict(zip(list(le.categories_[0]), [round(x*100, 2)for x in
result[0]])).items(), columns = ['Category', 'Confidence'])
    plot_result(result)
    return result
```

```
result = inference("I am mad", max_len)
print(result)
```

	Category	Confidence
0	anger	96.64
1	fear	1.44
2	joy	1.15
3	love	0.01
4	sadness	0.65
5	surprise	0.11



Висновки

Було розглянуто моделі на основі трансформаторів, які в даний час виробляють найкращі результати у програмах NLP. Розглянуті моделі на основі трансформаторів включали GPT, модель Transformer-XL, міжмовні моделі (XLM) та двонаправлені подання кодерів від трансформаторів (BERT). Було розглянуто сильні сторони, і слабкі сторони різних моделей трансформаторів. Було розроблено програми на основі BERT та RoBERTa для виявлення, що є кращим. Основним критерієм можна назвати точність і з нею вони показали майже однаковий результат.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. “A survey of sentiment analysis in social media,” Knowledge and Information Systems
2. ” Handbook of cognition and emotion, vol. 98, 1999
3. “The hourglass of emotions,” in Cognitive behavioural systems, Springer, 2012.
4. “Xlnet: Generalized autoregressive pretraining for language understanding,” 2019.
5. “Roberta: A robustly optimized bert pretraining approach,” 2019.
6. “Emotionlines: An emotion corpus of multi-party conversations”, 2019.
7. “Semeval-2019 task 3: Emocontext contextual emotion detection in text,” , 2019.
8. “Glove: Global vectors for word representation,” in Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014
9. “Multi-task learning network for emotion recognition in conversation,” arXiv preprint 2020.
10. “Evaluation datasets for twitter sentiment analysis: a survey and a new dataset, the sts-gold,” ,2013.
11. “Finding principal paths in data space,”, 2018.
12. https://medium.com/analytics-vidhya/nlp-language-models-bert-gpt2-t-nlg-changing-the-rules-of-the-game-3334b23020a9
13. https://neurohive.io/ru/tutorial/bert-klassifikacya-teksta/
14. http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_%28курс_лекций%2C_К.В.Воронцов%29
15. https://robotdreams.cc/blog/69-novaya-era-v-nlp-kak-model-bert-vse-izmenila