Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет інформаційних систем та програмної інженерії
(назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту (роботи)

(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: **Розробка системи машинного перекладу на основі нейромережевих технологій з використанням вектора метрик якості**

Виконав: студент (ка) VI курсу, групи СПм-62

спеціальності (напряму підготовки) 121 - Інженерія програмного забезпечення
(шифр і назва спеціальності (напряму підготовки))

	(підпис)	Баранський М.О. (прізвище та ініціали)
Керівник	(підпис)	Пастух О.А. (прізвище та ініціали)
Нормоконтроль	(підпис)	Бойко І.В. (прізвище та ініціали)
Рецензент	(підпис)	Дмитроца Л.П. (прізвище та ініціали)

м. Тернопіль – 2019

ЗМІСТ

ВСТУП

Механізація перекладу була однією з найдавніших мрій людства. У XX столітті це стало реальністю у вигляді комп’ютерних програм, здатних перекладати найрізноманітніші тексти з однієї природної мови на іншу. Але, як ніколи, реальність не є досконалою. Немає «машин для перекладу», які за допомогою натискання декількох кнопок можуть приймати будь-який текст будь-якою мовою і створювати ідеальний переклад будь-якою іншою мовою без втручання чи допомоги людини. Це ідеал для далекого майбутнього, якщо воно навіть в принципі навіть досяжне, в чому багато хто сумнівається.

Досягнуто розроблення програм, які можуть створювати «сирі» переклади текстів у відносно чітко визначених предметних областях, які можуть бути переглянуті, щоб дати якісні перекладені тексти з економічно вигідною швидкістю, або які в їх невідредагованому стані можуть бути читають і розуміють фахівці з даної теми з інформаційною метою. У деяких випадках, за допомогою відповідного контролю мови вхідних текстів, переклади можуть бути виконані автоматично, які вищої якості потребують невеликого перегляду або не потребують перегляду [1].

Це суцільні здобутки у тому, що зараз традиційно називають машинним перекладом, але вони часто були затьмарені та неправильно зрозумілі. Громадське сприйняття МП спотворюється двома крайніми позиціями. З одного боку, є ті, хто не переконаний, що в аналізі мови є щось складне, оскільки навіть маленькі діти здатні так легко вивчати мови; і хто переконаний, що кожен, хто знає іноземну мову, повинен вміти легко перекладати. Отже, вони не в змозі оцінити труднощі завдання або скільки досягнуто. З іншого боку, є такі, хто вважає, що оскільки автоматичний переклад Шекспіра, Гете, Толстого та менших літературних авторів неможливий, немає жодної ролі для будь-якого комп'ютерного перекладу.

Вони не в змозі оцінити той внесок, який могли б зробити менш, ніж досконалий переклад, або у власній роботі, або в загальному поліпшенні міжнародної комунікації.

Більшість перекладів у світі не є текстами, які мають високий літературний та культурний статус. Переважна більшість професійних перекладачів зайнята для задоволення величезного і зростаючого попиту на переклади науково-технічних документів, комерційних та ділових операцій, адміністративних меморандумів, юридичної документації, інструкцій з експлуатації, підручників із сільського господарства та медичної медицини, промислових патентів, рекламних листівок, газет звіти тощо. Деякі з цих робіт є складними та складними. Але значна частина цього втомлива і повторювана, в той же час вимагає акуратності та послідовності. Попит на такі переклади зростає швидкістю, що значно перевищує можливості перекладацької професії. Допомога комп'ютера має чіткі та негайні привабливості. Практична корисність

Система МП визначається в кінцевому рахунку якістю її випуску. Але те, що вважається «хорошим» перекладом, чи то виготовленим людиною, чи машиною, - надзвичайно важко точно визначити поняття. Багато що залежить від конкретних обставин, за яких він зроблений, і конкретного одержувача, для якого він призначений. Вірність, точність, зрозумілість, відповідний стиль та реєстр - це всі критерії, які можна застосувати, але вони залишаються суб'єктивними судженнями. На практиці важливо, що стосується МП, - це те, що потрібно змінити, щоб досягти результатів до рівня, прийнятного для людського перекладача чи читача. З такою слизькою концепцією, як переклад, дослідники та розробники систем МП можуть зрештою прагнути лише створити переклади, які є "корисними" в конкретних ситуаціях - що зобов'язує їх визначити чіткі дослідницькі цілі - або, як альтернатива, вони шукають відповідних застосувань " переклади ", які насправді вони здатні робити.

Тим не менш, залишається вищий ідеал рівняння найкращого людського перекладу. МТ є частиною ширшої сфери «чистого дослідження» в обробці природними мовами на комп'ютерній основі в обчислювальній лінгвістиці та штучному інтелекті, які досліджують основні механізми мови та розуму шляхом моделювання та моделювання в комп'ютерних програмах. Дослідження МП тісно пов'язані з цими зусиллями, приймаючи та застосовуючи як теоретичні перспективи, так і оперативні методи перекладацьких процесів, і, в свою чергу, пропонують розуміння та вирішення конкретних проблем. Крім того, МТ може забезпечити більш широку шкалу тестування для теорій і методик, розроблених невеликими експериментами з обчислювальної лінгвістики та штучного інтелекту.

Основними перешкодами для перекладу за допомогою комп’ютера є, як це було завжди, не обчислювально, а мовно. Це - проблеми лексичної неоднозначності, синтаксичної складності, словникових відмінностей між мовами, еліптичних та «неграматичних» конструкцій, коротко кажучи, вилучення «значення» речень і тексти з аналізу письмових знаків та створення речень та текстів в іншій сукупності мовних символів з рівнозначним значенням. Отже, МП слід сподіватися сильно покластися на досягнення лінгвістичних досліджень, особливо тих галузей, які демонструють високий ступінь формалізації, і це дійсно є і буде продовжувати це робити. Але МП не може застосовувати лінгвістичні теорії безпосередньо: мовознавці переймаються поясненнями основних механізмів вироблення та осмислення мови, вони зосереджуються на важливих ознаках і не намагаються все описати чи пояснити. Навпаки, системи МП повинні мати справу з фактичними текстами. Вони повинні зіткнутися з усім спектром лінгвістичних явищ, складнощами термінології, неправильно написаними написаннями, неологізмами, аспектами «продуктивності», які не завжди стосуються абстрактної теоретичної лінгвістики. Коротше кажучи, МТ само по собі не є самостійним полем «чистого» дослідження. Це бере з лінгвістики,

інформатики, штучного інтелекту, теорії перекладу, будь-яких ідей, методів і прийомів, які можуть слугувати розробці вдосконалених систем. Це, по суті, "прикладні" дослідження, але це поле, яке все-таки створило значну кількість технік і концепцій, які, в свою чергу, можуть бути застосовані в інших сферах комп'ютерної обробки мови.

Термін "Машинний переклад" (МП) - це тепер традиційна і стандартна назва комп'ютеризованих систем, відповідальних за виробництво перекладів з однієї природної мови на іншу, з людською допомогою чи без неї. Раніші назви, такі як "механічний переклад" та "автоматичний переклад", зараз рідко використовуються англійською мовою; але їхні еквіваленти в інших мовах все ще поширені (наприклад, французька автоматизація перекладу, російська автоматизація переводу). Цей термін не включає комп’ютерні засоби перекладу, які підтримують перекладачів, забезпечуючи доступ до словників та віддалених баз даних термінології, полегшуючи передачу та отримання машиночитаних текстів або взаємодіючи з текстовим текстовим редактором, редагуванням тексту чи друком. Однак це включає системи, в яких перекладачі чи інші користувачі допомагають комп'ютерам у виробництві перекладів, включаючи різні комбінації підготовки тексту, он-лайн взаємодії та наступні зміни результатів. Межі між автоматизованим перекладом людини (МАНТ) та автоматизованим машинним перекладом (НАМТ) часто є невизначеними, а термін "комп'ютерний" (або комп'ютерний) переклад (обидва CAT) іноді може охоплювати і те, і інше. Але центральним ядром самого МП є автоматизація повного процесу перекладу.

Хоча ідеалом може бути створення високоякісних перекладів, на практиці вихід більшості систем МП переглядається (після редагування). У цьому відношенні вихід МТ трактуються не інакше, ніж вихід більшості людських перекладачів, який зазвичай переглядається іншим перекладачем перед розповсюдженням. Однак типи помилок що виробляються системами МП, дійсно відрізняються від таких у людських перекладачів. Хоча

розміщення пошти є нормою, є певні обставини, коли вихід МТ може бути залишений без змін (як необроблений переклад) або лише злегка виправлений, наприклад якщо він призначений лише для фахівців, знайомих з темою тексту. Вихідні дані також можуть слугувати грубим проектом для людського перекладача, як попередній переклад.

Технології, засновані на NLP, набувають все більшого поширення. Наприклад, телефони та портативні комп'ютери підтримують розпізнавання тексту та рукописного введення; веб-пошукові системи надають доступ до інформації, зачиненої в неструктурованому тексті; машинний переклад дозволяє отримувати тексти, написані китайською мовою, та читати їх іспанською мовою. Забезпечивши більш природні інтерфейси між людиною та машиною та більш досконалий доступ до збереженої інформації, обробка мови стала відігравати центральну роль у багатомовному інформаційному суспільстві.

1. РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до предметної області

1.1.1 Аналіз предметної області

В даному проєкті виконується розробка програмної системи машинного перекладу на основі нейромережевих технологій з використанням вектора метрики якості.

Машинний переклад (МП) - найдавніша проблема досліджень штучного інтелекту та інформатики. Протягом 1980-х років щодо винаходу персональних комп'ютерів змінився погляд дослідників МП на поводження з мовами. На веб-сайті про приїзд протягом 1990-х років з'явилося збільшення даних про Інтернет. Це швидке зростання даних в Інтернеті стало натхненням для дослідників МТ для розробки більш комерційних систем МП для забезпечення глобальної комунікації. Практична мета МТ - розробити системи, доступні для простого населення за номінальну вартість, щоб зменшити мовний бар'єр. Машинний переклад можна розглядати як область прикладних досліджень, яка черпає ідеї та методи з лінгвістики, інформатики, штучного інтелекту, теорії перекладу та статистики

Системи МП можуть бути розроблені або спеціально для двох мов, званих двомовною системою, або від однієї мови до багатьох мов, які називаються багатомовними системами. Зазвичай багатомовні системи є двонаправленими, але більшість двомовних систем є однонаправленими.

Незважаючи на те, що машинний переклад у 1950-х роках розглядався як комп’ютерна програма, а дослідження проводилися протягом 60 років, машинний переклад все ще вважається відкритою проблемою. Однак досі, незважаючи на значні зусилля, повністю автоматичний високоякісний

переклад не є реалістичною метою в осяжному майбутньому. Існує багато підходів до розробки систем МП, у кожної системи є свої переваги та недоліки. З цих підходів широко використовується статистичний машинний переклад [2]. Система статистичного машинного перекладу (SMT) виводить правила з двомовного паралельного корпусу, який містить велику кількість високоякісних вирівнюваних речень. Найбільшим завданням є отримання високоякісного корпусу через недостатнє джерело даних. Підхід на основі машинного перекладу (EBMT) застосовується, коли для мов доступна менша кількість лінгвістичних ресурсів. У прикладі перекладу на основі прикладу визначена система, яка містить набір пропозицій мови джерела та відповідних пропозицій цільової мови. Під час виконання перекладу на основі прикладу використовується двомовний корпус як його база даних. Ця база даних зберігається в пам'яті перекладу. Коли система стикається з тим самим реченням, система не потребує перекладу знову система безпосередньо отримує з пам'яті перекладу.

Перевага перекладу на основі прикладу полягає в тому, що пам'ять перекладу економить зусилля користувача при перекладі речення, і це економить час процесора, а також час користувача. Недоліком прикладного перекладу є витрата пам'яті для зберігання бази даних. Підхід до машинного перекладу на основі правил (RBMT) використовується для мов, які є граматично багатими. Система машинного перекладу на основі правил побудована з використанням великої кількості граматичних правил та високої якості двомовного словника з великою кількістю записів. Система RBMT розбирає текст і створює морфологічне подання тексту, а з морфологічної інформації мови-джерела генерується цільова мова. Цей процес вимагає широких двомовних словників із синтаксичною та семантичною інформацією та великими наборами граматичних правил. Система використовує граматичні правила передачі для перекладу мови-джерела на цільову мову. Системи машинного перекладу на основі правил дають високу якість перекладу, коли

він використовує великі словники та обширні граматичні правила. Для індійських мов, таких як каннада та телугу, немає заздалегідь визначених обчислювальних граматик, словників та великої кількості корпусу.

З аналізу поточного стану досліджень в машинному перекладі доведено, що двомовний словник з великою кількістю записів є основною вимогою для МП. Зараз система перекладу використовує багато здогадок замість точних знань протягом переклад з мови джерела на цільову мову через відсутність відповідної інформації для вирішення всіх ситуацій. Ступінь вірогідності наведеного твердження різниться, звичайно, між різними системами МП. Наприклад, це добре працює для статистичного машинного перекладу, оскільки модель перекладу дозволяє здогадуватися про наявний корпус, але якість перекладу обмежена. Якість можна покращити, збільшивши двомовний корпус, який охоплює всі аспекти мовної моделі.

Ідея, що лежить в основі підходу, що обговорюється тут, полягає в тому, що двомовні словники використовуються для відображення окремих слів мови-джерела на цільову мову. Подання зі словника самостійно не може дати відповідний переклад, оскільки лише словники не можуть дати точний вихід у всіх випадках. Для поліпшення якості машинного перекладу граматичні правила передачі виводу використовуються разом зі словником для отримання значущого результату. Правила граматики передачі розробляються на рівні тегів POS (частин мови), а для створення цільового слова використовується таблиця відображення суфіксів [3].

В даний час словники, корпусна та обчислювальна граматика є джерелом проблем з механізмами перекладу, але (навіть із орієнтованими на дані підходами), коли система повинна дати точний переклад, перераховані вище властивості відіграють велику роль. Ці властивості можна обробити, визначивши набір правил на основі властивостей окремого слова [6]. Цю систему можна вдосконалити на основі покращення якості двомовного

словника та набору правил граматики. Підхід, заснований на словнику, передбачає аналізатор морфи, двомовний словник, таблицю відображення суфіксів та генератор морфем.

Морфологічний аналізатор приймає введення як слово і виробляє вихід як аналіз слова. Вихід містить вміст морфологічної інформації слова. Система каннади Саари, розроблена Каві Нараяна Мурті в центральному університеті Хайдерабаду, використовується як морфологічний аналізатор.

Вихід морфологічного аналізатора дає морфологічну інформацію даного слова. Морфологічна структура дієслів каннади впливає на час, особу, стать та число. Іменники словосполучення для множини, косого, відмінка та відмінків. Структура дієслова є складною, і захоплення цієї складності в машино аналізованому та породжуваному форматі є складним завданням. До переліку дієслів каннада належать кінцеві, нескінченні, дієприкметникові, прислівникові та умовні позначення.

1.1.2 Постановка завдання

Кожна створена система є поставленою послідовністю, яку ми перевторюємо на послідовність загальних процесів для вирішення певної проблеми або задачі, яка перед нам стає, так само і при створенні певної програмної системи ми переслідуємо мету розв’язання певної проблеми чи задачі.

Програмна система - це система взаємодіючих комунікаційних компонентів, заснована на програмному забезпеченні, що є частиною комп'ютерної системи (поєднання апаратного та програмного забезпечення). Він "складається з ряду окремих програм, файлів конфігурації, які використовуються для налаштування цих програм, системної документації,

яка описує структуру системи, та документації користувача, яка пояснює, як користуватися системою".

Термін "програмна система" слід відрізняти від термінів "комп'ютерна програма" та "програмне забезпечення". Термін "комп'ютерна програма", як правило, відноситься до набору вказівок (вихідний або об'єктний код), які виконують конкретне завдання. Однак система програмного забезпечення, як правило, посилається на більш охоплюючу концепцію із значно більшою кількістю компонентів, таких як специфікація, результати випробувань, документація для кінцевих користувачів, записи технічного обслуговування тощо.

Використання терміну "програмна система" часом пов'язане із застосуванням підходів до теорії систем у контексті програмної інженерії. Програмна система складається з декількох окремих комп'ютерних програм та пов'язаних з ними файлів конфігурації, документації тощо, які працюють разом. Концепція використовується при вивченні великого і складного програмного забезпечення [4], оскільки фокусується на основних компонентах програмного забезпечення та їх взаємодії. Це також пов'язано зі сферою архітектури програмного забезпечення.

До основних категорій програмних систем належать ті, що базуються на розробці прикладного програмного забезпечення, програмного забезпечення, класичного для різних ОС та системного програмного забезпечення, хоча розрізнення іноді може бути складним. Прикладами програмних систем є операційні системи, комп'ютерні системи бронювання, системи управління повітряним рухом, системи військового командування та управління, телекомунікаційні мережі, системи управління контентом, системи управління базами даних, експертні системи, вбудовані системи тощо.

Після ретельного аналізу даної предметної області та знаходження основних проблем, потрібно сформувати завдання та мету для імплементування даного проєктного рішення.

У якості мети роботи було поставлено створити дві системи машинного перекладу з різними нейромережевими архітектурами такими як RNN та LSTM з перевіркою якості перекладу за допомогою алгоритму METEOR, що являє собою вектор метрики якості. Самі результати мають слугувати для аналізу та порівняння двох систем машинного перекладу та вияснення яка з цих архітектур видає кращі результати та є більш оптимальною для побудови точніших систем машинного навчання.

З огляду на те, що було описано, поставленні наступні задачі в рамках цієї роботи:

- Підготовка двомовного датасету на більше ніж 100 тисяч речень
- Розробити систему машинного перекладу на основі RNN
- Розробити систему машинного перекладу на основі LSTM
- Аналізувати результати виконання систем машинного навчання за допомогою метрики METEOR
- Сформувати висновок на основі результатів перевірки якості перекладу, їх порівняння з огляду на наданні результати перевірки метрикою METEOR

Тому відповідно до поставлених завдань потрібно зробити аналіз вибору відповідних інструментів для реалізації даної задачі, підбір найвідповідніших датасетів, які є у вільному доступі, які містять речення з різними граматичними формами, широким словниковим запасом та довжиною самих речень.

В результаті цього аналізу було вирішено, що ці системи мають будуватися з використанням вільних бібліотек, які є найпопулярнішими на даний момент, та які містять алгоритми, які передбаченні вибраними

архітектурами нейронних мереж та алгоритмами самої метрики для перевірки та порівняння цих архітектур,

1.1.3 Пошук варіантів використання

Для того щоби проаналізувати які дії може робити користувач з програмною системою необхідно візуалізація цих дій, які відбуватимуться між користувачем та системою. Найбільш використовуваною такою візуалізацією таких дій є Use case діаграма.

Use case діаграма - це представлення взаємодії користувача з системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, а також часто супроводжуватись і іншими типами діаграм. Випадки використання представлені або колами, або еліпсами [5].

Хоча сама діаграма може детально описувати кожну можливість, діаграма може допомогти забезпечити високий рівень вигляду системи. Раніше було сказано, що "Use case діаграми є кресленнями вашої системи". Вони забезпечують спрощене та графічне зображення того, що система насправді повинна робити.

Завдяки своїй спрощеній природі use case діаграми можуть бути хорошим інструментом комунікації для зацікавлених сторін. Малюнки намагаються наслідувати реальний світ і дають уявлення зацікавленим особам зрозуміти, як планується проектувати систему. Інженери програмного забезпечення провели дослідження, щоб встановити, чи існувала справжня ситуація щодо діаграм випадків використання чи взагалі вони були непотрібними. Було встановлено, що Use case діаграми передавали наміри

системи більш спрощеним чином для зацікавлених сторін, і що вони "інтерпретувалися більш повно, ніж діаграми класів".

Мета Use case діаграм - просто забезпечити вигляд на високому рівні системи та донести вимоги до членів групи для зацікавлених осіб. Додаткові діаграми та документація можуть бути використані для забезпечення повного функціонального та технічного вигляду системи.

Так як для взаємодії з системою, що розробляється в рамках проєкту не передбачає постороньої взаємодії, для зображення цих дій буде використано лише два актора: користувач та система. Згідно того, що має робити система та для яких цілей вона використовуватиметься було спроєктовано діаграму, що зображено нижче:

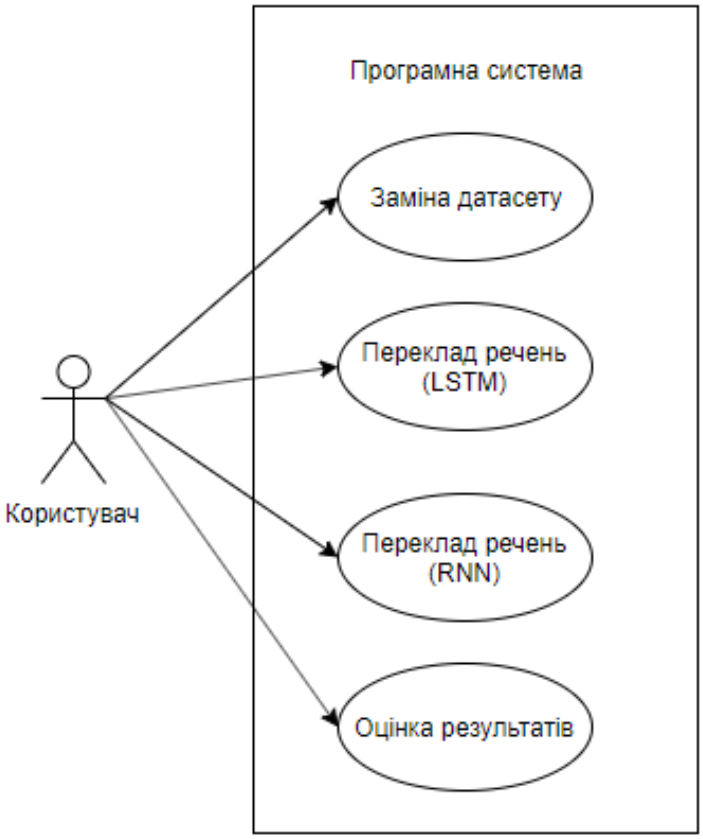

Рис. 1.1 – Use case діаграма системи машинного навчання

1.2 Проєктування системи

Rational Unified Process або RUP – це ітеративний процес розробки ПЗ, створений компанією IBM в 2003. RUP - це не єдиний конкретний процес розробки, а скоріше адаптований процес, призначений для розробки організацій розвитку та програмних команд проектів, які підберуть елементи процесу, які відповідають їх потребам.

Щоби описати вимог в RUP створюються прецеденти або варіанти використання (UseCases). Повний набір варіантів використання (ВВ) системи разом з логічними відношеннями між ними називається моделлю варіантів використання.

Кожний ВВ – це опис сценарію взаємодії користувача та системи, що повністю виконує конкретну задачу користувача. Зрозуміло, не має сенсу документувати у вигляді ВВ нефункціональні вимоги (до продуктивності, якості тощо). Проте, згідно RUP всі функціональні вимоги мають бути представлені у вигляді ВВ. Вважається, що модель ВВ дає більш цілісне уявлення про функціональність системи в порівнянні з традиційним описом вимог (перерахуванням функцій, якими повинна володіти система).

RUP заохочує використання візуальних засобів для аналізу і проектування. Як правило, використовується нотація і, відповідно, засоби моделювання UML (такі як RationalRose). Модель предметної області документується у вигляді діаграми класів, модель ВВ – за допомогою діаграми ВВ, взаємодія компонентів системи між собою описується діаграмою послідовності тощо

1.2.1 Обирання процесу реалізації системи та архітектури

1.2.2 Моделювання системи

Зачасту рішення з використанням машинного навчання використовують функціональний підхід, тому було вибрано саме такий шлях моделювання системи. У якості інструменту реалізації було обрано мову програмування Python у зв’язку з багатою бібліотекою, що включає у себе і різні математичні бібліотеки, і також через його велику гнучкість у реалізації рішень з будь-якою парадигмою [6]. Для побудови архітектури, яка відображатиме що у собі буде містити програмний код необхідно побудувати UML-діаграму.

Уніфікована мова моделювання (UML) - мова загального призначення для розробки, мова моделювання в галузі інженерії програмного забезпечення, яка покликана забезпечити стандартний спосіб візуалізації дизайну системи.

UML пропонує спосіб візуалізації креслення архітектури системи на діаграмі, включаючи такі елементи, як:

- будь-яка діяльність (робочі місця);
- окремі компоненти системи;
- вияснення як можуть взаємодіяти з іншими компонентами програмного забезпечення;
- як працюватиме система;
- як взаємодіють об'єкти з іншими (компоненти та інтерфейси);
- зовнішній інтерфейс користувача.

Хоча спочатку призначений для об'єктно-орієнтованої проектної документації, UML було розширено на більш великий набір проектної документації (як зазначено вище) і вважається корисним у багатьох контекстах.

Важливо розрізняти модель UML та набір діаграм системи. Діаграма - це часткове графічне зображення моделі системи. Набір діаграм не повинен повністю охоплювати модель, а видалення діаграми не змінює модель. Модель також може містити документацію, яка керує елементами моделі та діаграмами (наприклад, випадки письмового використання).

Діаграми UML представляють два різних представлення системної моделі:

- Статичний (або структурний) погляд: підкреслює статичну структуру системи з використанням об'єктів, атрибутів, операцій та відношень. Вона включає в себе діаграми класів і складові діаграми структури.
- Динамічний (або поведінковий) погляд: підкреслює динамічну поведінку системи, показуючи співпрацю між об'єктами та зміни внутрішніх станів об’єктів. Цей вид включає діаграми послідовності, діаграми активності та діаграми стану машини.

Моделі UML можна обміняти серед інструментів UML за допомогою формату обміну метаданими XML (XMI).

В UML одним з ключових інструментів моделювання поведінки є модель використання випадків, викликана OOSE. Випадки використання - це спосіб вказати необхідні звичаї системи. Зазвичай вони використовуються для охоплення вимог системи, тобто того, що система повинна робити [7].

Тому так як буде імплементовано дві системи машинного навчання, відповідно необхідно сконструювати дві діаграми, які відображають дві системи з різними архітектурами.

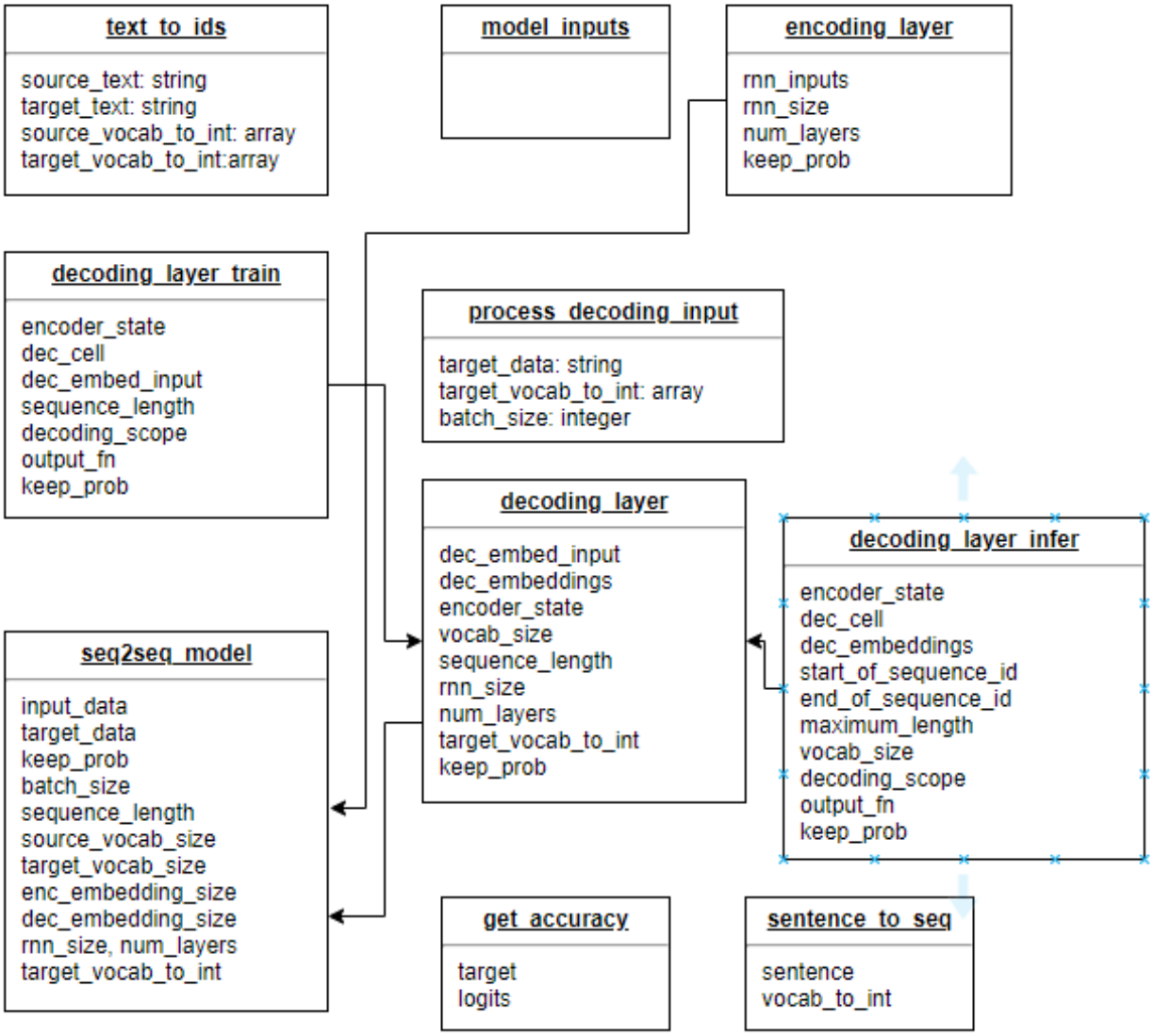

Рис. 1.2 – UML діаграма системи машинного навчання на основі RNN

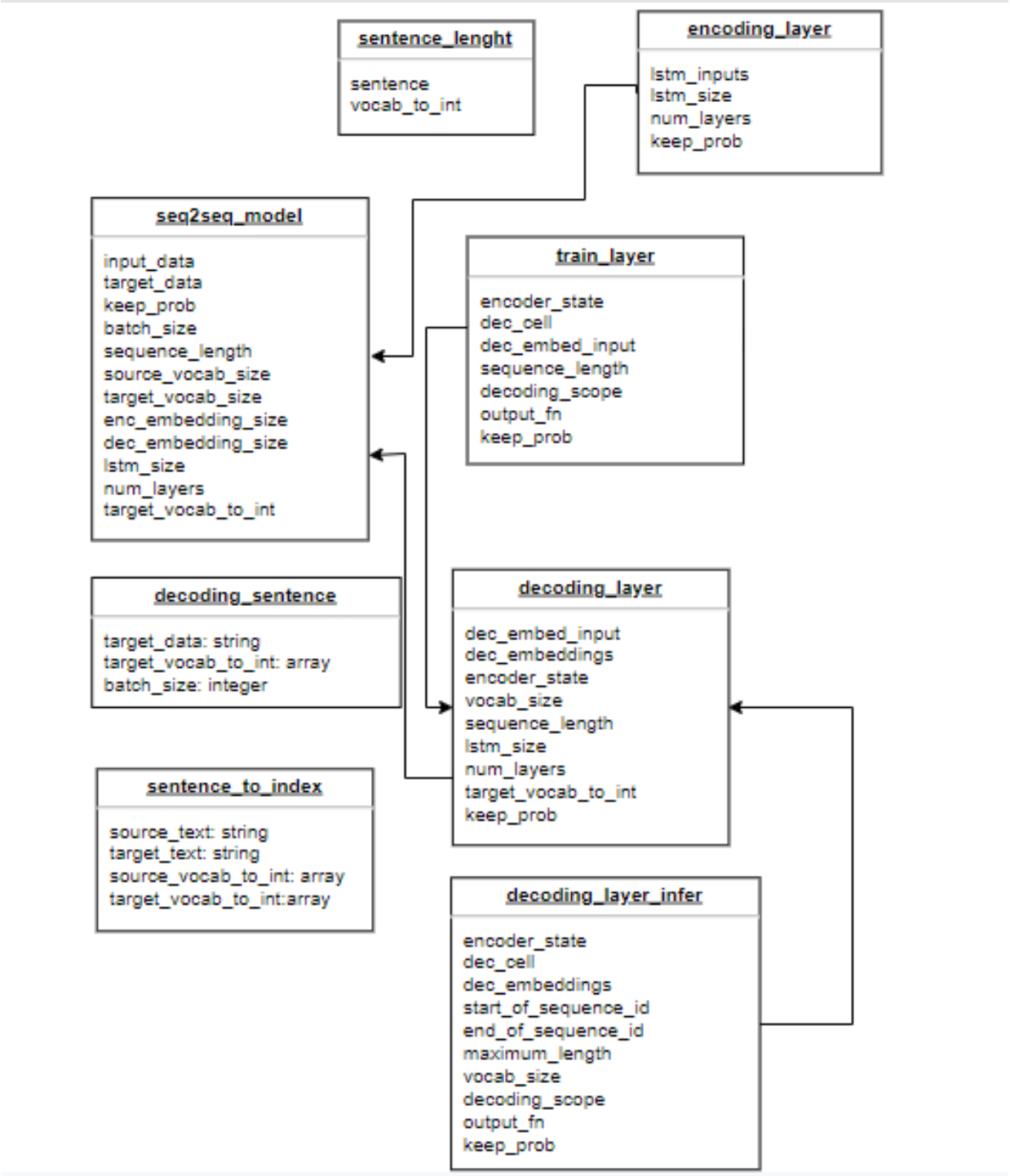

Рис. 1.3 – UML діаграма системи машинного навчання на основі LSTM

1.3 Конструювання програмної системи

1.3.1 Вибрані технології для конструювання

Для того, щоби приступити до реалізації програмної системи слід закцентувати увагу на виборі мови програмування та середовищі розробки. У залежності від вибору цих інструментів розробки залежатиме швидкість розробки та результати виконання.

Python - інтерпретована мова програмування високого рівня, загального призначення. Створена Гідо ван Россумом та вперше випущена в 1991 році, філософія дизайну Python підкреслює читабельність коду завдяки помітному використанню значного пробілу. Його мовні конструкції та об'єктно-орієнтований підхід мають на меті допомогти програмістам написати чіткий логічний код для малих та масштабних проектів.

Python динамічно типізована мова. Вона підтримує декілька парадигм програмування, включаючи процедурне, об'єктно-орієнтоване та функціональне програмування. Python часто описується мовою як мова у якої "все включено" завдяки своїй всебічній бібліотеці стандартів.

Python був задуманий в кінці 1980-х років як наступник мови ABC. Python 2.0, випущений у 2000 році, представив такі функції, саме поняття списків та систему збору сміття, здатну збирати еталонні цикли. Python 3.0, випущений у 2008 році, була основною редакцією мови, яка не є повністю зворотно-сумісною, і багато код Python 2 не працює без модифікацій у самому коді на Python 3.

Python - мова програмування багато парадигми. Об'єктно-орієнтоване програмування та структуроване програмування повністю підтримуються, і багато його функцій підтримують функціональне програмування та

орієнтоване на аспекти програмування (в тому числі метапрограмування та метаоб'єкти (магічні методи)). Багато інших парадигм підтримуються через розширення, включаючи дизайн за контрактом та логічне програмування.

Python використовує динамічне введення тексту та комбінацію підрахунку довідок та циклічне виявлення сміттєзбірника для управління пам'яттю. Він також має динамічне дозвіл імен (пізнє прив'язування), яке пов'язує імена методів та змінних під час виконання програми.

Дизайн Python пропонує деяку підтримку функціонального програмування в традиції Lisp. Він має функції фільтра, відображення та зменшення; перелічити розуміння, словники, набори та вирази генератора. Стандартна бібліотека має два модулі (itertools та functools), які реалізують функціональні інструменти, запозичені у Haskell та Standard ML.

Основна філософія мови узагальнена в документі «Дзен Пітона» (PEP 20), який включає афоризми, такі як:

- Красиве краще, ніж потворне.
- Явне краще, ніж неявне.
- Простий - краще, ніж складний.
- Комплекс краще, ніж складний.
- Читання рахується.

Замість того, щоб вся його функціональність була вбудована в її основну основу, Python був розроблений як дуже розширюваний [8]. Ця компактна модульність зробила її особливо популярною як засіб додавання програмованих інтерфейсів до існуючих додатків. Бачення Ван Россума щодо невеликої основної мови з великою стандартною бібліотекою та легко розширюваним перекладачем випливало з його фрустрації з ABC, яка підтримувала протилежний підхід.

Велика стандартна бібліотека Python, яку зазвичай називають однією з найбільших її сильних сторін, пропонує інструменти, що підходять для багатьох завдань. Для програм, орієнтованих на Інтернет, підтримуються багато стандартних форматів та протоколів, таких як MIME та HTTP. Він включає модулі для створення графічних інтерфейсів користувачів, підключення до реляційних баз даних, генерації псевдовипадкових чисел, арифметики з десятковими знаками довільної точності, маніпулювання регулярними виразами та тестування одиниць.

Деякі частини стандартної бібліотеки охоплені специфікаціями (наприклад, реалізація інтерфейсу шлюзу веб-сервера (WSGI) wsgiref відповідає PEP 333), але більшість модулів не є. Вони визначаються кодом, внутрішньою документацією та тестовими наборами (якщо вони постачаються). Однак, оскільки більша частина стандартної бібліотеки є кросплатформним кодом Python, лише декілька модулів потребують зміни або переписування для варіантів реалізації.

Станом на листопад 2019 року, індекс пакетів Python (PyPI), офіційний сховище сторонніх програмних продуктів Python, містить понад 200 000 пакетів із широким спектром функціональності, включаючи:

- Графічний інтерфейс користувача
- Веб-рамки
- Мультимедіа
- Бази даних
- Мережі
- Тестові рамки
- Автоматизація
- Скребкування в Інтернеті
- Документація
- Адміністрація системи

- Наукові обчислення
- Обробка тексту
- Обробка зображень
- Машинне навчання
- Аналітика даних

У якості середовища розробки було вибрано Anaconda. Дистрибутив Anaconda з відкритим вихідним кодом є одним з найзручніших середовищ для Data Science та машинного навчання з використанням мов програмування як Python та R у таких популярних операційних системах як Windows, Mac OS та всіх дистрибутивів Linux.

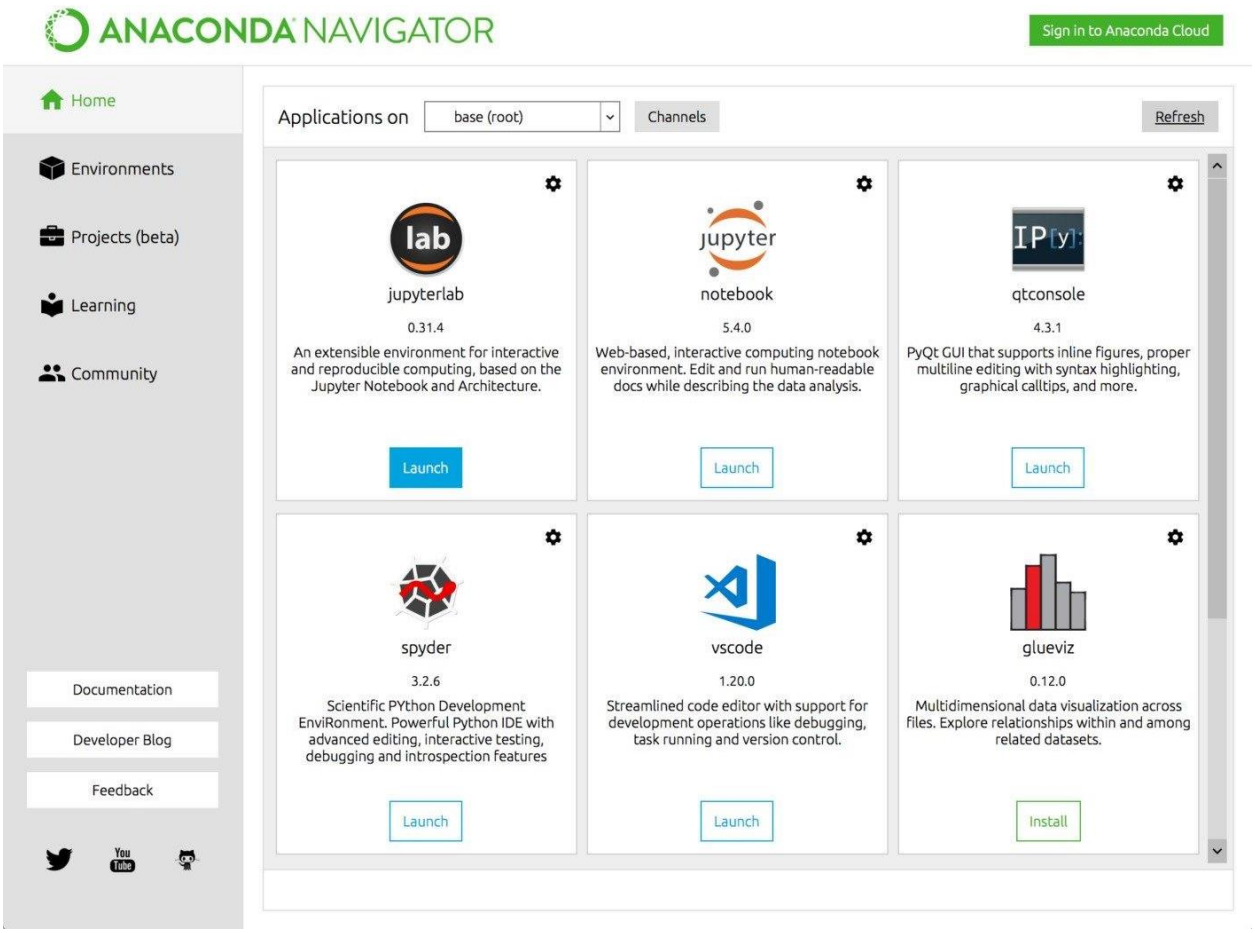

Рис. 1.4 – Вітальне вікно дистрибутиву Anaconda

Цим дистрибутивом користуються близько 15 мільйонів користувачів по всьому світу [9], і став вже фактично світовим стандартом для розробки, тестування та тренування на одному пристрої та включає в себе такі зручності як:

- Швидке завантаження понад 1500 бібліотек для мов Pythonта R
- Зручне керування бібліотеками, залежностями та середовищем з системним менеджером пакунків та середовища Conda
- Розробка, тренування машинного навчання та використання різних моделей глибинного навчання (Deep Learning) з scitik-learn, TensorFlow та Theano
- Аналіз даних з масштабуванням та високою продуктивністю з такими розширеннями як Dask, NumPy, pandas та Numba
- Візуалізація результатів з бібліотеками Matplotlib, Bokeh, Datashader та Holovlews

1.3.2 Реалізація машинного перекладу на основі RNN

Для реалізації рекурентної нейронної мережі потрібно побудувати алгоритм, які включають у себе наступні пункти та які дії в ньому включені:

- Попередня обробка: завантажити та аналізувати дані, чистка, токенізація, набивання
- Моделювання: побудувати, навчити та протестувати модель
- Прогнозування: генерувати конкретні переклади з двомовних даних та порівняти вихідні переклади з основними реальними перекладами
- Ітерація: ітерувати модель, експериментувати з різними архітектурами

Коли ми запускаємо кількість слів, ми можемо побачити, що словниковий запас набору даних досить малий [9]. Це було задумано для цього проекту. Це дозволяє нам тренувати моделі в розумний час (Ліст. 1.1) (Ліст 1.2) (Ліст 1.11).

Очищення. Ніякої додаткової чистки в цей момент не потрібно робити. Дані вже перетворені в малі та розділені, щоб між усіма словами та пунктуацією були пробіли.

Примітка. Для інших проектів NLP вам можуть знадобитися виконувати додаткові кроки, такі як: видалити теги HTML, видалити стоп-слова, видалити пунктуацію або перетворити на представлення тегів, позначити частини мови або здійснити вилучення сутності.

Токенізація. Далі нам потрібно токенізувати дані - тобто перетворити текст у числові значення. Це дозволяє нейронній мережі виконувати операції над вхідними даними. Для цього проекту кожному слову та розділовому знаку буде надано унікальний ідентифікатор. (Для інших проектів NLP може бути доцільним призначити кожному символу унікальний ідентифікатор.)

Коли ми запускаємо токенізатор, він створює індекс слів, який потім використовується для перетворення кожного речення у вектор.

Прокладки. Коли ми подаємо наші послідовності ідентифікаторів слів у модель, кожна послідовність повинна бути однакової довжини. Щоб досягти цього, підкладка додається до будь-якої послідовності, яка коротша за максимальну довжину (тобто коротше, ніж найдовше речення).

Моделювання. Спочатку розроблюється архітектура RNN на високому рівні (Ліст 1.3). Посилаючись на діаграму вище, є кілька частин моделі, про яку ми маємо знати:

Вхідні дані. Послідовності введення подаються в модель з одним словом на кожен крок часу. Кожне слово кодується як унікальне ціле число або

однокольоровий кодований вектор, який відображається до словника даних англійської мови.

Вкладання шарів. Вбудовування використовується для перетворення кожного слова у вектор. Розмір вектора залежить від складності словникового запасу.

Повторні шари (енкодер). Тут контекст із векторів слів у попередніх кроках часу застосовується до поточного вектору слів.

Щільні шари (декодер). Це типові повністю пов'язані шари, які використовуються для декодування закодованого вводу у правильну послідовність перекладу (Ліст 1.4).

Виходи. Виводи повертаються у вигляді послідовності цілих чисел або кодуються одним гарячим вектором, які потім можуть бути відображені у французький словник даних.

Вкладення. Вкладення дозволяють зафіксувати більш точні синтаксичні та семантичні зв’язки слів. Це досягається проектуванням кожного слова в n-мірний простір. Слова з подібним значенням займають подібні регіони цього простору; чим ближче два слова, тим вони схожіші. І часто вектори між словами представляють корисні стосунки, такі як гендер, час дієслова чи навіть геополітичні зв’язки.

Навчання вбудовування великого набору даних з нуля вимагає величезної кількості даних та обчислень. Отже, замість того, щоб робити це самостійно, ми зазвичай використовуємо попередньо підготовлений пакет для вбудовування, такий як GloVe або word2vec. При використанні цього способу вбудовування є формою трансферного навчання [10]. Однак, оскільки наш набір даних для цього проекту має невеликий словниковий запас і мало синтаксичних варіацій, ми будемо використовувати Keras для тренування вбудовування самостійно.

Кодер і декодер. Наша модель послідовності-послідовності пов'язує дві періодичні мережі: кодер і декодер. Кодер підсумовує вхід у зміну контексту, яка також називається станом. Потім цей контекст декодується і генерується вихідна послідовність.

Оскільки і кодер, і декодер є рецидивними, вони мають петлі, які обробляють кожну частину послідовності на різних часових етапах. Щоб уявити це, найкраще розкрутити мережу, щоб ми могли бачити, що відбувається на кожному кроці.

У наведеному нижче прикладі потрібно чотири кроки часу, щоб кодувати всю послідовність введення. На кожному кроці часу кодер "зчитує" введене слово і виконує перетворення на його прихованому стані. Потім він переходить цей прихований стан на наступний крок часу. Майте на увазі, що прихований стан являє собою відповідний контекст, що протікає по мережі. Чим більший прихований стан, тим більша навчальна здатність моделі, але також і більші вимоги до обчислень. Ми будемо говорити більше про перетворення в прихованому стані, коли ми охоплюватимемо повторювані одиниці з закритим типом (GRU).

Поки що зауважте, що для кожного часового кроку після першого слова у послідовності є два входи: прихований стан та слово із послідовності. Для кодера це наступне слово у послідовності введення. Для декодера це попереднє слово з вихідної послідовності.

Крім того, пам’ятайте, що, коли ми маємо на увазі «слово», ми дійсно маємо на увазі векторне представлення слова, яке походить із вкладеного шару.

1.3.3 Реалізація машинного перекладу на основі LSTM

У цій реалізації програмної системи машинного навчання буде побудованна на архітектурі LSTM. Потім потрібно для оптимльного результату тренувати послідовність до моделі послідовності на наборі даних з англійської та французької речень, які можуть перекласти нові речення з англійської на французьку.

Спочатку має бути зреалізована функція model_inputs() для створення TF-заповнювачів для нейронної мережі (Ліст. 1.5). Він повинен створити такі заповнювачі:

- Введіть текст-заповнювач тексту з назвою "введення", використовуючи параметр імені заповнювача TF з ранг 2.
- Цільовий заповнювач з ранг 2.
- Заповнювач рівня курсу навчання з рангом 0.
- Зберігайте ймовірність заповнювача імені імені "Keep_prob", використовуючи параметр імені заповнювача TF з рангом 0.

Повернення заповнювачів у наступному кортежі [11] (введення, цілі, швидкість навчання, збереження ймовірності).

Наступним кроком буде реалізація decoding_layer () для створення шару RNN декодера (Ліст. 1.7):

- Створюється комірка RNN для декодування за допомогою rnn_size та num_layers.
- Створюється вивідау функція за допомогою лямбда, щоб перетворити її вхід, logits, логіти класу.
- Використовуйте функцію decoding_layer_train (encoder_state, dec_cell, dec_embed_input, length_length, decoding_scope, output_fn, Keep_prob) для отримання навчальних логітів.
- Використовуйте свою функцію decoding_layer_infer (encoder_state, dec_cell, dec_embeddings, start_of_sequence_id, end_of_sequence_id,

максимальна_length, vocab_size, decoding_scope, output_fn, Keep_prob), щоб отримати дані висновку.

Найголовніша частина побудови перекладача з цією архітектурою – побудова самої нейронної мережі. Для цього потрібно застосувати описані вище функції (Ліст. 1.8)(Ліст. 1.9):

- Застосування вбудованих до вхідних даних кодера.
- Кодування входу, використовуючи encoding_layer (rnn_inputs, rnn_size, num_layers, Keep_prob).
- Оброблення цільових даних за допомогою функції process_decoding_input (target_data, target_vocab_to_int, batch_size).
- Застосування вбудованих до цільових даних для декодера.
- Розшифрування закодованого входу, використовуючи decoding_layer (dec_embed_input, dec_embeddings, encoder_state, vocab_size, length_length, rnn_size, num_layers, target_vocab_to_int, Keep_prob).

Але найголовніше для отримання хорошого результату потрібно застосування до даної нейронної мережі для тренування двомовний датасет, який містить велике число перекладених речень на цих мовах. Використовуючи датасет з малим числом перекладених речень хороший результат можна буде отримати лише при перекладі простих речень. Ви можете тренуватися на французько-англійському корпусі WMT10. Цей набір даних має більше словникового запасу та багатший на теми, що обговорюються. Однак на це вам знадобиться кілька днів, тому переконайтеся, що у вас є GPU, і нейронна мережа працює на наборі даних, які ми надали. Просто переконайтеся, що ви граєте з корпусом WMT10 після подання цього проекту.

1.4 Тестування і впровадження програмної системи

1.4.1 Тестування програмної системи машинного перекладу

Для того, щоби впевнитися у відсутності помилок у системі необхідно зробити тестування коду програми. Але так як програмні рішення побудовані на основі нейромережевих архітектур не піддаються звичайному тестуванню як інформаційні системи, було вирішено протестувати ці програмні системи за допомогою Blackbox-тестування.

Тестування Blackbox - це тестування функціональності програми, не знаючи деталей її реалізації, включаючи внутрішню структуру програми, структури даних тощо. Тестові приклади тестування blackbox створюються на основі специфікацій вимог. Тому його також називають тестуванням на основі специфікацій. Наступна схема представляє тестування Blackbox:

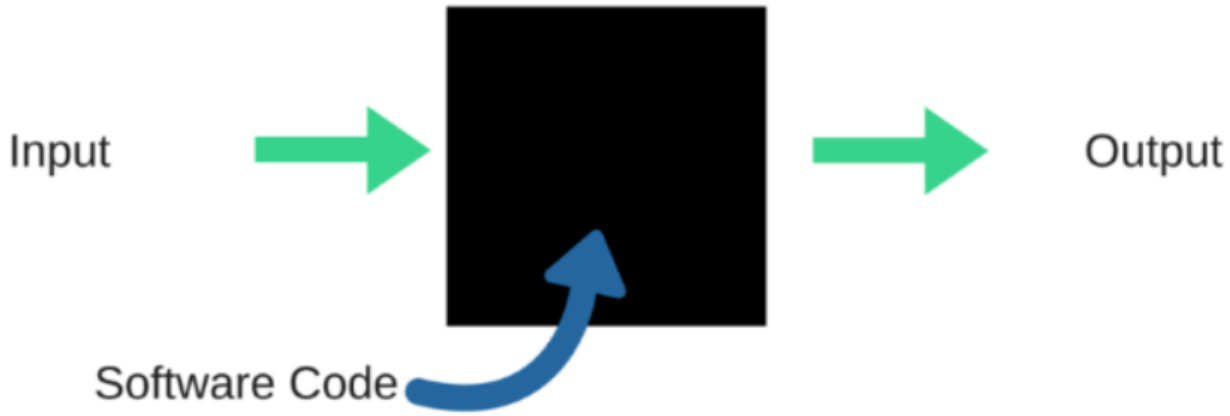

Рис. 1.5 – Схема Blackbox-тестування

Якщо застосовуватись до моделей машинного навчання, тестування blackbox означатиме тестування моделей Machine Learning, не знаючи внутрішніх деталей, таких як особливості моделі машинного навчання, алгоритм, який використовується для створення моделі тощо. Однак завдання

полягає в тому, щоб визначити тестовий оракул (джерело очікуваного результату), який може перевірити результат випробування на основі очікуваних раніше значень.

Зважаючи на те, що моделі машинного навчання були віднесені до категорії неперевірених, це дає завданню виконати тестування моделей ML для Blackbox.

У традиційній розробці програмного забезпечення припущенням часто є наявність тестового оракула, який є не що інше, як тестери / тестові інженери (людині) або якась форма механізмів тестування, включаючи програму тестування, яка могла б перевірити вихід комп'ютерної програми на очікуване значення що відомо заздалегідь. Що стосується моделей машинного навчання, заздалегідь не очікувані значення, оскільки вихід ML-моделей є певним прогнозом. Зважаючи на те, що результат моделей машинного навчання є передбаченням, непросто порівняти чи перевірити прогнозування з якоюсь очікуваною вартістю, яка не відома заздалегідь. Зважаючи на це, під час фази розробки (побудови моделі) науковці перевіряють продуктивність моделі, порівнюючи результати моделі (передбачувані значення) з фактичними значеннями. Це не те саме, що тестування моделі на будь-який вхід, де очікуване значення не відомо.

В лістину викладений скрипт, який перевіряє системи на наявність помилок.

1.4.2 Використання програмної системи машинного перекладу

Після порівняння архітектур можна виявнити яка найкраще підходить для побудови повноцінної системи машинного навчання. Тому розроблені системи можна потім реалізувати як застосунок, або як веб-сервіс, тому цю

систему можна використати в багатьох сферах, де використовуються іноземні мови.

Також його сфери використання можуть бути такі:

- попередній переклад великих проектів для подальшого редагування фахівцями з метою прискорення процесу перекладу;
- переклад в реальному часі для служби технічної підтримки, чату і т.п .;
- переклад питань і відповідей, а також інтернет-форумів;
- переклад матеріалів по спеціалізованій тематиці при наявності великого обсягу паралельних текстів - зазвичай для наукових або юридичних областей;
- внутрішня кореспонденція;
- чітко структуровані каталоги.

1.4.3 Порівняння програмних систем машинного перекладу

За допомогою метрики METEOR та результатів обох систем машинного перекладу. З огляду на представленні дані зображені на рис та рис. можна зробити висновок, що серед вибраних архітектур найкращою для побудови системи машинного перекладу є LSTM система.

```
Score: 0.9977 = Fmean: 1.0000 * (1 - Penalty: 0.0023)
Fmean: 1.0000 = 10 * Precision: 1.0000 * Recall: 1.0000 /  (Recall: 1.0000 + 9 * Precision: 1.0000)
Penalty: 0.0023 = 0.5 * (Fragmentation: 0.1667 ^3)
Fragmentation: 0.1667 = Chunks: 1.0000 / Matches: 6.0000
Sentence (Machine): Ich meine, daß der Grundsatz der relativen Stabilität einen elementaren Rechtsgrundsat
z der gemeinsamen Fischereipolitik darstellt und ein Vorschlag, diesen zu unterlaufen, rechtlich unzulässi
g wäre.
Sentence (Etalon): Ich meine, daß der Grundsatz der relativen Stabilität einen elementaren Rechtsgrundsatz
der gemeinsamen Fischereipolitik darstellt und ein Vorschlag, diesen zu unterlaufen, rechtlich unzulässig
wäre.
```

Рис. 1.6 – Результати оцінки метрикою METEOR перекладу за допомогою LSTM

```
Score: 0.9654 = Fmean: 0.9836 * (1 - Penalty: 0.0185)
Fmean: 0.9836 = 10 * Precision: 0.8571 * Recall: 1.0000 / (Recall: 1.0000 + 9 * Precision: 0.8571)
Penalty: 0.0185 = 0.5 * (Fragmentation: 0.3333 ^3)
Fragmentation: 0.3333 = Chunks: 2.0000 / Matches: 6.0000
Sentence (Machine): Ich glaube, dass der Grundsatz der relativen Stabilität ein grundlegendes Rechtsprinzi
p der Gemeinsamen Fischereipolitik ist und ein Vorschlag, ihn zu untergraben, rechtlich unzulässig wäre.
Sentence (Etalon):Ich meine, daß der Grundsatz der relativen Stabilität einen elementaren Rechtsgrundsatz
der gemeinsamen Fischereipolitik darstellt und ein Vorschlag, diesen zu unterlaufen, rechtlich unzulässig
wäre.
```

Рис. 1.7 – Результати оцінки метрикою METEOR перекладу за допомогою RNN

2. СПЕЦІАЛЬНА ЧАСТИНА

2.1 Нейромережеві архітектури

Люди та інші тварини обробляють інформацію за допомогою нейронних мереж. Вони утворюються з трильйонів нейронів (нервових клітин), що обмінюються короткими електричними імпульсами, які називаються потенціалами дії. Комп'ютерні алгоритми, що імітують ці біологічні структури, офіційно називаються штучними нейронними мережами, щоб відрізнити їх від зухвалих речей всередині тварин. Однак більшість вчених та інженерів не є офіційними та використовують термін нейронна мережа для включення як біологічних, так і небіологічних систем.

Дослідження нейронної мережі мотивовано двома бажаннями: краще зрозуміти людський мозок та розробити комп'ютери, які можуть вирішувати абстрактні та погано визначені проблеми [12]. Наприклад, звичайні комп’ютери мають проблеми з розумінням мови та розпізнаванням облич людей. Для порівняння, людина надзвичайно добре справляється з цими завданнями.

Було випробувано багато різних структур нейронної мережі, деякі засновані на імітації того, що бачить біолог під мікроскопом, а деякі на основі більш математичного аналізу проблеми. Найбільш часто використовується структура показана на рис. 26-5. Ця нейронна мережа складається з трьох шарів, званих вхідним шаром, прихованим шаром та вихідним шаром. Кожен шар складається з одного або декількох вузлів, представлених на цій діаграмі

невеликими колами. Рядки між вузлами вказують на потік інформації з одного вузла в інший. У цьому конкретному типі нейронної мережі інформація протікає лише з входу на вихід (тобто зліва направо). Інші типи нейронних мереж мають більш складні зв'язки, такі як шляхи зворотного зв'язку.

Вузли вхідного шару пасивні, тобто вони не змінюють дані. Вони отримують єдине значення на своєму введенні та дублюють це значення на своїх кількох результатах. Для порівняння, вузли прихованого та вихідного шару активні. Це означає, що вони змінюють дані, як показано на рис. 26-6. Змінні: X11, X12… X115 містять дані, що підлягають оцінці (див. Рис. 2.1). Наприклад, це можуть бути піксельні значення із зображення, зразки авдіосигналу, ціни на фондових біржах у наступні дні тощо. Вони також можуть бути результатом якогось іншого алгоритму, наприклад, класифікаторів у нашому прикладі виявлення раку: діаметр, яскравість, різкість краю і т.д.

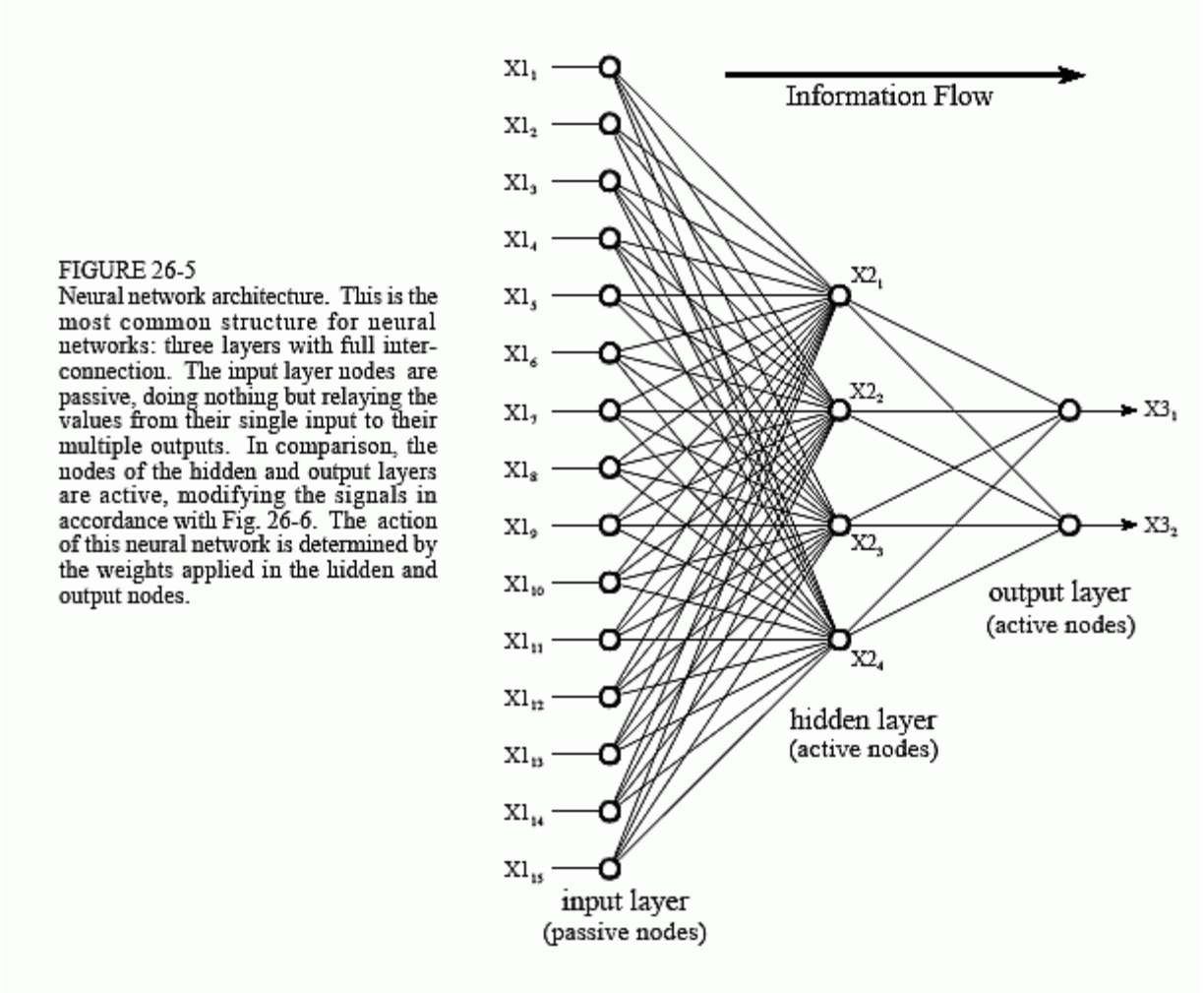

Рис. 2.1 – Приклад схеми нейронної мережі

Кожне значення з вхідного шару дублюється і надсилається всім прихованим вузлам. Це називається повністю взаємопов'язаною структурою. Як показано на рис. 26-6, значення, що входять до прихованого вузла, множать на ваги, набір заздалегідь визначених чисел, що зберігаються в програмі. Потім зважені входи додаються для отримання єдиного числа. Це показано на діаграмі символом ∑. Перед виходом із вузла це число передається через нелінійну математичну функцію, яку називають сигмоїдою. Це крива "s" у формі, що обмежує вихід вузла [13]. Тобто вхід до сигмоїди - це значення між -∞ і + ∞, тоді як його вихід може бути лише між 0 і 1.

Виходи з прихованого шару представлені на схемі потоку (рис. 26-5) змінними: X21, X22, X23 і X24. Як і раніше, кожне з цих значень дублюється і застосовується до наступного шару. Активні вузли вихідного шару об'єднують і модифікують дані для отримання двох вихідних значень цієї мережі, X31 та X32.

Нейронні мережі можуть мати будь-яку кількість шарів і будь-яку кількість вузлів на шарі. Більшість програм використовують тришарову структуру з максимум кількома сотнями вхідних вузлів. Прихований шар зазвичай становить приблизно 10% від розміру вхідного шару. У разі виявлення цілі вихідному шару потрібен лише один вузол. Вихід цього вузла пороговий, щоб забезпечити позитивну або негативну індикацію наявності або відсутності цілі у вхідних даних.

2.1.1 RNN

Рекурентна нейронна мережа (RNN) - це клас штучних нейронних мереж, де з'єднання між вузлами утворюють спрямований графік по часовій послідовності. Це дозволяє йому проявляти часову динамічну поведінку. На відміну від нейронних мереж подачі, RNN можуть використовувати свій внутрішній стан (пам'ять) для обробки послідовностей входів. Це робить їх корисним для таких завдань, як несегментоване, пов'язане розпізнавання рукописного тексту або розпізнавання мовлення.

Термін "рекурентна нейронна мережа" використовується без розбору для позначення двох широких класів мереж з подібною загальною структурою, де одна структура - кінцевий імпульс, а інша - нескінченний імпульс. Обидва класи мереж демонструють часову динамічну поведінку. Кінцева імпульсна рекурентна мережа - це спрямований ациклічний графік, який можна розкрутити і замінити суворо подається нейронною мережею, тоді

як нескінченна імпульсна рекурентна мережа - це спрямований циклічний графік, який неможливо розкрутити.

Як кінцеві імпульсні, так і нескінченні повторювані імпульсні мережі можуть мати додатковий збережений стан, і сховище може знаходитися під безпосереднім контролем нейронної мережі. Сховище також може бути замінено іншою мережею або графіком, якщо це включає затримки в часі або має петлі зворотного зв'язку. Такі керовані стани називаються затворними або закритими пам'яттю і є частиною довготривалих мереж пам'яті (LSTM) та рецидивованих одиниць, що закриваються. Це також називається нейронною мережею зворотного зв’язку.

Базові RNN - це мережа нейроноподібних вузлів, організованих у послідовні "шари". Кожен вузол у заданому шарі з'єднаний спрямованим (одностороннім) з'єднанням з кожним іншим вузлом у наступному послідовному шарі. [Потрібне цитування] Кожен вузол (нейрон) має активацію реальної цінності, що залежить від часу. Кожне з'єднання (синапс) має змінити реальну вагу. Вузли - це або вузли введення (отримання даних за межами мережі), вузли виведення (дають результати), або приховані вузли (які змінюють дані на шляху від введення до виводу).

Для контрольованого навчання в дискретних настройках часу послідовності дійсних векторів входу надходять на вхідні вузли, по одному вектору. На будь-якому етапі часу кожен невхідний блок обчислює свою поточну активацію (результат) як нелінійну функцію зваженої суми активацій усіх одиниць, що підключаються до неї. Цільові активації, задані керівництвом, можуть подаватися для деяких вихідних блоків протягом певних етапів часу. Наприклад, якщо вхідна послідовність є мовним сигналом, відповідним розмовній цифрі, кінцевим вихідним цільовим результатом в кінці послідовності може бути мітка, що класифікує цифру.

У налаштуваннях навчання підкріплення жоден викладач не надає цільових сигналів. Натомість функцію фітнесу або функцію винагороди періодично використовують для оцінки працездатності RNN, яка впливає на її вхідний потік через вихідні блоки, підключені до виконавчих механізмів, що впливають на навколишнє середовище [14]. Це може бути використано для гри в гру, в якій прогрес вимірюється кількістю виграних очок.

Кожна послідовність створює помилку як суму відхилень усіх цільових сигналів від відповідних активацій, обчислених мережею. Для навчального набору з численних послідовностей загальна помилка - це сума помилок усіх окремих послідовностей.

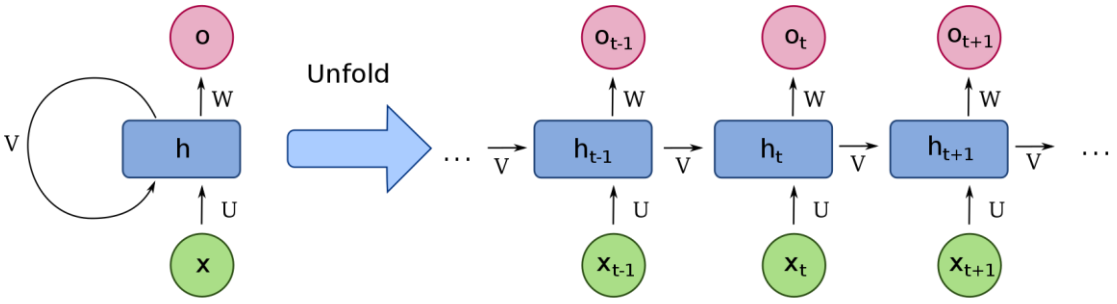

Рис. 2.2 – Архітектура RNN

Тренування ваг в нейромережі можна моделювати як нелінійну проблему глобальної оптимізації. Цільова функція може бути сформована для оцінки придатності або похибки певного вагового вектора так: Спочатку ваги в мережі встановлюються відповідно до вагового вектора. Далі мережа оцінюється відповідно до тренувальної послідовності. Як правило, різниця суми в квадраті між прогнозами і цільовими значеннями, зазначеними в тренувальній послідовності, використовується для представлення похибки

поточного весового вектора. Тоді довільні методи глобальної оптимізації можуть бути використані для мінімізації цієї цільової функції.

Найпоширенішим методом глобальної оптимізації для навчання RNN є генетичні алгоритми, особливо в неструктурованих мережах.

Спочатку генетичний алгоритм кодується вагами нейронної мережі заздалегідь визначеним чином, коли один ген у хромосомі являє собою одну вагову ланку. Вся мережа представлена у вигляді єдиної хромосоми. Фітнес-функція оцінюється наступним чином:

- Кожна вага, закодована в хромосомі, присвоюється відповідній ваговій ланці мережі.
- Набір тренінгу представлений мережі, яка поширює вхідні сигнали вперед.
- Середня квадратична помилка повертається до функції фітнесу.
- Ця функція керує процесом генетичного відбору.

Багато хромосом складають популяцію; тому розвивається багато різних нейронних мереж [15], поки не буде виконано критерій зупинки. Загальна схема зупинки:

- Коли нейромережа засвоїла певний відсоток даних тренінгу або
- Коли мінімальне значення середньоквадратичної помилки задоволено або
- Коли буде досягнуто максимальної кількості навчальних поколінь.

Критерій зупинки оцінюється функцією фітнесу, оскільки він отримує зворотну помилку середнього квадрату від кожної мережі під час тренування. Тому мета генетичного алгоритму - максимально використовувати функціональність, зменшуючи середньоквадратичну помилку.

Інші глобальні (та / або еволюційні) методи оптимізації можуть бути використані для пошуку гарного набору ваг, таких як імітація відпалу або оптимізація рою частинок.

2.1.2 LSTM

Довга короткочасна пам'ять (LSTM) - це архітектура штучної періодичної нейронної мережі (RNN), що використовується в галузі глибокого навчання. На відміну від стандартних нейронних мереж подачі, LSTM має зв'язки зворотного зв'язку. Він може обробляти не лише окремі точки даних (наприклад, зображення), а й цілі послідовності даних (наприклад, мова або відео). Наприклад, LSTM застосовний до таких завдань, як несегментоване, підключене розпізнавання рукописного тексту, розпізнавання мови та виявлення аномалії в мережевому трафіку або IDS (системи виявлення вторгнень).

Теоретично класичні RNN можуть відслідковувати довільні довгострокові залежності у вхідних послідовностях. Проблема ванільних RNN носить обчислювальний (або практичний) характер: при тренуванні ванільної РНН із застосуванням зворотного розповсюдження градієнти, які розповсюджуються назад, можуть «зникати» (тобто вони мають тенденцію до нуля) або «вибухати» (тобто вони мають тенденцію до нескінченності) через обчислення, що беруть участь у процесі, в яких використовуються кінцеві точні числа. RNN, що використовують блоки LSTM, частково вирішують проблему градієнта, що зникає, оскільки одиниці LSTM також дозволяють градієнтам текти без змін.

Загальний блок LSTM складається з комірки, вхідних воріт, вихідних воріт та шлюзу забуття. Осередок запам'ятовує значення протягом довільних часових інтервалів, а три ворота регулюють потік інформації в клітинку і поза нею.

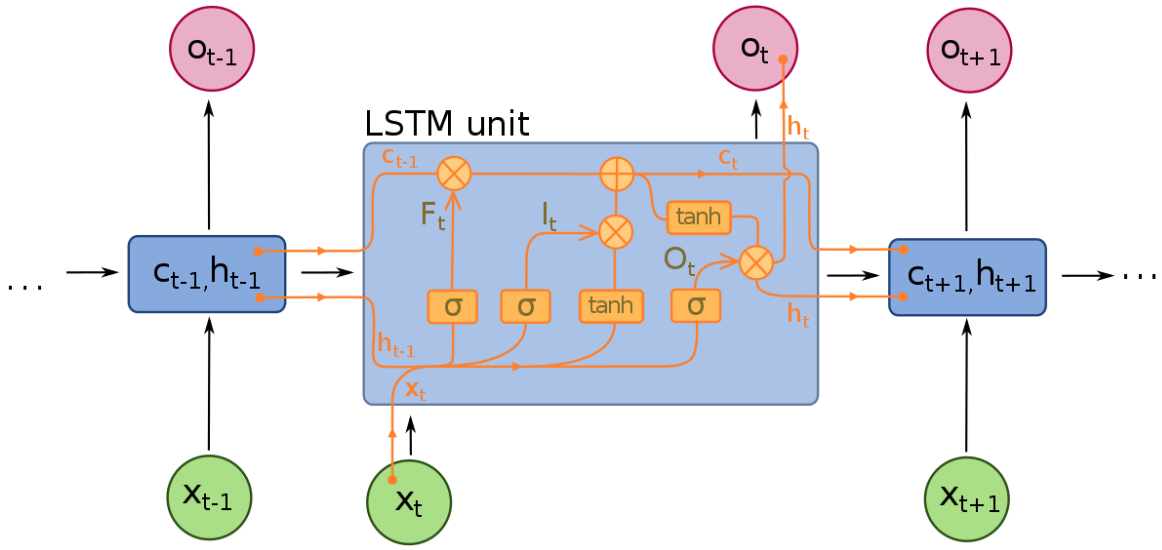

Рис. 2.3 – Архітектура LSTM

Мережі LSTM добре підходять для класифікації, обробки та прогнозування на основі даних часових рядів, оскільки між важливими подіями в часовому ряді можуть бути відставання невідомої тривалості. LSTM були розроблені для вирішення вибухових та зникаючих проблем з градієнтом, які можуть виникнути при навчанні традиційних RNN. Відносна нечутливість до довжини зазору є перевагою LSTM над RNN, прихованими моделями Маркова та іншими методами навчання послідовності у численних програмах.

Існує кілька архітектурних підрозділів LSTM. Загальна архітектура складається з комірки (частина пам'яті блоку LSTM) і трьох "регуляторів", зазвичай званих воротами, потоку інформації всередині блоку LSTM: вхідний затвор, вихідний затвор і затвор забуття. Деякі варіанти блоку LSTM не мають однієї або декількох цих воріт або, можливо, мають інші ворота. Наприклад, одиничні повторювані одиниці (GRU) не мають вихідного коду.

Інтуїтивно зрозуміла, що комірка відповідає за облік залежностей між елементами вхідної послідовності. Вхідний затвор керує тим, наскільки нове значення впадає в комірку [16], затвор забуття керує тим, наскільки значення залишається в комірці, а вихідний затвор керує тим, наскільки значення в комірці використовується для обчислення виводу активація блоку LSTM. Функція активації воріт LSTM часто є логістичною сигмоїдною функцією.

Існують підключення до та вихідних воріт LSTM, деякі з яких є повторними. Ваги цих з'єднань, які потрібно дізнатися під час тренування, визначають, як працюють ворота.

У рівняннях, наведених нижче, змінні у нижньому регістрі представляють вектори. Матриці У рівняннях, наведених нижче, змінні у нижньому регістрі представляють вектори. Матриці W_q та U_q містять відповідно ваги вхідних та періодичних з'єднань, де індекс q може бути або вхідним воротом i , вихідним воротом o, воротом забуття f або пам'яттю комірка c, залежно від обчисленої активації. У цьому розділі ми використовуємо "позначення вектора". Так, наприклад,$c_t \in R^h$ - це не одна комірка LSTM-одиниця, а містить h комірок LSTM одиниць.

Компактними формами рівнянь для прямого проходу блоку LSTM з воротом забуття є формули 2.1 – 2.5:

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right) \quad (2.1)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (2.2)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (2.3)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c) \quad (2.4)$$

$$h_t = \sigma_h(o_t \circ c_t) \quad (2.5)$$

де початкові значення $c_0 = 0$ і $h_0 = 0$, а оператор $\circ$ - позначає добуток Адамара (продукту, що належить до елементів). Коефіцієнт t індексує крок часу.

Таблиця 2.1 – Коефіцієнти, які використовуються для воріт забуття

$x_t \in R^d$	Вхід вектора до LSTM юніта
$f_t \in R^h$	Вектор активації забувального вентиля
$i_t \in R^h$	Вектор активації вхідного вентиля
$o_t \in R^h$	Вектор активації вихідного вентиля
$h_t \in R^h$	Вектор прихованого стану, який відомий як вектор LSTM юніта
$c_t \in R^h$	Вектор стану комірки
$W \in R^{h \times d}$, $U \in R^{h \times h}$ та $b \in R^h$	Матриця ваг і параметри вектора зміщення, які потрібні для навчання під час тренінгу

RNN, що використовує одиниці LSTM, можна навчати під контролем, набір навчальних послідовностей, використовуючи алгоритм оптимізації, такий як спуск градієнта, поєднаний із зворотним розповсюдженням у часі для обчислення градієнтів, необхідних під час процесу оптимізації, з метою зміни кожної ваги мережі LSTM пропорційно похідній помилки (на вихідному шарі мережі LSTM) щодо відповідної ваги.

Однак, для одиниць LSTM, коли значення помилок поширюються із вихідного шару, помилка залишається в комірці блоку LSTM. Ця "карусель помилок" постійно повертає помилку до кожного з воріт пристрою LSTM, поки вони не навчаться скорочувати значення.

2.2 Метрики якості перекладу

Оцінка систем машинного перекладу (МТ) є життєво важливою сферою досліджень, як для визначення ефективності існуючих систем МТ, так і для оптимізації продуктивності систем МТ. Ця частина описує низку різних підходів до оцінювання, які використовуються в GALE-спільноті, та вводить протоколи та методики оцінювання, що використовуються в програмі. Ми обговорюємо розробку та використання автоматичних, людських, на основі завдань та напівавтоматичних методів оцінки машинного перекладу, орієнтуючись на використання рівня помилок перекладу, опосередкованого людиною, HTER як стандарт оцінки використовується в GALE. Ми обговорюємо робочий процес, пов'язаний із використанням цього заходу, включаючи публікацію після редагування, контроль якості та оцінку. Ми документуємо завдання оцінювання, дані, протоколи та результати останніх оцінок GALE MT. Крім того, ми представляємо цілу низку різних підходів до оптимізації систем МТ на основі різних заходів. Ми окреслюємо вимоги та конкретні проблеми при використанні різних підходів до оптимізації та описуємо, як характеристики різних метрик МТ впливають на оптимізацію. Нарешті, ми описуємо нову нещодавню та поточну роботу над розробкою повністю автоматичних метрик оцінювання МТ, які можуть мати потенціал істотного підвищення ефективності оцінювання та оптимізації систем МТ.

Прогрес у галузі машинного перекладу покладається на оцінку якості нової системи шляхом систематичного оцінювання, таким чином, щоб нова система могла показати, що вона працює краще, ніж попередні системи. Складність виникає при визначенні кращої системи. Оцінюючи якість перекладу, немає єдиної правильної відповіді; швидше, може бути будь-яка кількість можливих правильних перекладів. Крім того, коли два переклади є лише частково правильними - але різними способами - важко відрізнити якість. Крім того, оцінка якості може залежати від призначеного використання для перекладу, наприклад, тон перекладу може бути вирішальним у деяких програмах, але не має значення для інших програм.

Традиційно існує дві парадигми оцінювання машинного перекладу: (1) Оцінка скляної скриньки, яка вимірює якість системи на основі внутрішніх властивостей системи, та (2) Оцінка чорної скриньки, яка вимірює якість системи, що ґрунтується виключно на її вихід, без поваги до внутрішніх механізмів системи перекладу. Оцінка Glass Box фокусується на вивченні мовної системи, затвердженої для публічного випуску; Поширення - це необмежене охоплення та теорії, що використовуються для боротьби з тими мовними явищами [17]. Окремі мовні компоненти системи можуть бути досліджені та піддані оцінці чорного поля. Цей метод оцінювання насамперед був орієнтований на експертні системи, засновані на правилах, а не на статистичні системи.

З іншого боку, оцінка чорної скриньки стосується лише об'єктивної поведінки системи за попередньо визначеним набором оцінок. Цей метод оцінювання є лише справедливим порівнянням систем, якщо системи, які тестуються, були розроблені для роботи над даними, які мають той самий характер, що і набір оцінювання, або, якщо ні, особа, яка тестує системи, має на меті перевірити надійність у всіх різні типи даних з різними структурами, жанром та стилем. Цей метод виявився безцінним для галузі машинного перекладу, що дозволяє порівняти системи на одних і тих же тестових наборах, щоб визначити, чи є дана зміна системи насправді вдосконаленням. Метод фактичного вимірювання працездатності системи на тестовому наборі все ще залишається дуже активною дослідницькою областю, і такі показники оцінювання є основою цієї частини.

2.2.1 METEOR

METEOR (Metric for Evaluation of Translation with Explicit ORdering) — метрика для оцінки якості машинного перекладу. Метеор оцінює переклад, обчислюючи бал, заснований на явній відповідності слово в слово між

перекладом та заданим перекладом. Якщо доступно більше одного еталонного перекладу, переклад оцінюється незалежно від кожної посилання, і використовується найкраща пара балів. Давши пару рядків для порівняння, Meteor створює вирівнювання слів між двома рядками. Вирівнювання - це зіставлення між словами, таким чином, що кожне слово в кожній рядку відображає щонайменше одне слово в іншому рядку. Це вирівнювання поступово виробляється послідовністю модулів відображення слів. "Точний" модуль відображає два слова, якщо вони абсолютно однакові. Модуль «носія стебла» відображає два слова, якщо вони однакові після того, як вони виявляються за допомогою стебмера Портера. Модуль “Синонімія WN” відображає два слова, якщо вони вважаються синонімами, виходячи з того, що вони обидва належать до одного і того ж “synset” у словниковій мережі.

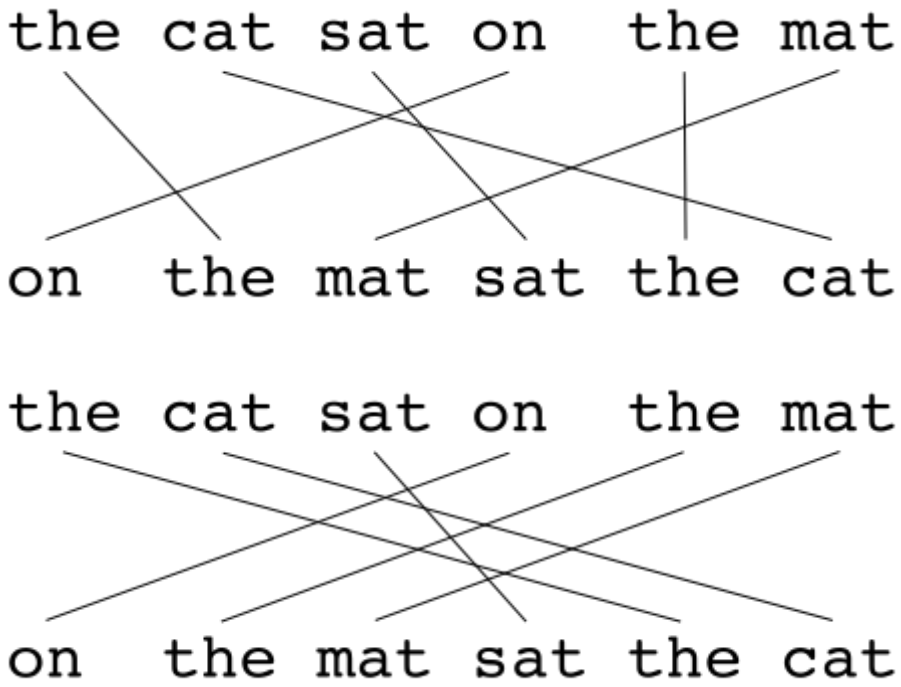

Рис. 2.4 – Приклади схем вимірювання за допомогою метрики METEOR

Модулі картографічного відображення спочатку ідентифікують усі можливі збіги слів між парами рядків. Потім ми визначаємо найбільший підмножина цих відображень слів, таким чином, що отриманий набір являє

собою вирівнювання, як визначено вище. Якщо знайдено більше одного максимального вирівнювання кардинальності, METEOR вибирає вирівнювання, для якого порядок слів у двох рядках є найбільш подібним (відображення, що має найменшу кількість відображень уніграмових схрещувань). Порядок запускання модулів відображає вподобання словосполучення. Замовлення за замовчуванням полягає в тому, щоб спочатку застосувати модуль «точного» відображення (Формула 2.6), після чого «портир, що випливає», а потім «синонімію WN».

$$P = \frac{m}{w_t} \tag{2.6}$$

Після того, як буде зроблено остаточне вирівнювання між системним перекладом та еталонним перекладом, оцінка METEOR для цього спарювання обчислюється наступним чином. На основі кількості відображених уніграмів, знайдених між двома рядками (m), загальної кількості уніграм у перекладі (t) та загальної кількості уніграм у посиланні (r), обчислюємо точність уніграму P = m / t і Уніграм виклику R = m / r. Потім обчислюємо параметризоване середнє гармонічне значення P і R (van Rijsbergen, 1979):

$$F_{mean} = \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R} \tag{2.7}$$

Точність, відкликання та Fmean (Формула 2.7) ґрунтуються на одностроях. Для врахування того, наскільки відповідні уніграми у двох рядках знаходяться в одному порядку слова, METEOR обчислює штраф за задане вирівнювання наступним чином. По-перше, послідовність узгоджених уніграм між двома рядками поділяється на найменшу кількість можливих «шматків»,

так що узгоджені уніграми в кожному фрагменті є суміжними (в обох рядках) і в однаковому порядку слів. Кількість фрагментів (ch) і кількість співпадінь (m) потім використовуються для обчислення фрагмента фрагментації: $frag = ch/m$. Потім штраф нараховується як на формулі 2.8:

$$Pen = \gamma \cdot frag^{\beta} \tag{2.8}$$

Значення γ визначає максимальний розмір штрафу ($0 \leq \gamma \leq 1$). Значення β визначає функціональне співвідношення між фрагментацією та штрафом. Нарешті, оцінка METEOR для вирівнювання між двома рядками обчислюється як на формулі 2.9:

$$score = (1 - Pen) \cdot F_{mean} \tag{2.9}$$

У всіх попередніх версіях METEOR значення трьох параметрів, згаданих вище, встановлювали так: α = 0,9, β = 3,0 і γ = 0,5 на основі експериментів, проведених на початку 2004 року. В останньому випуску ми налаштували ці параметри для оптимізації кореляції з судженнями людини на основі більш обширних експериментів.

У початковій версії METEOR (Banerjee and Lavie, 2005) встановлені значення для трьох параметрів у метриці: один для контролю відносної ваги точності та відкликання при обчисленні балу Fmean (α); одна, що регулює форму покарання як функції фрагментації (β), і одна для відносної ваги, присвоєної штрафу за фрагментацію (γ). У всіх версіях Meteor на сьогодні ці параметри були встановлені зі значеннями α = 0,9, β = 3,0 і γ = 0,5, засновані на ранніх експериментах з даними. Нещодавно ми провели більш ретельне

дослідження, спрямоване на налаштування цих параметрів на основі декількох наявних наборів даних, з метою встановлення параметрів параметрів, що дозволяють максимально співвіднестись із судженнями людини. Судження людини складаються у вигляді кількісних показників "адекватності" та "ефективності". У наших експериментах ми розглядали оптимізацію параметрів для кожного з цих типів людських суджень окремо, а також оптимізацію параметрів для суми адекватності та ефективності. Адаптація параметрів також є проблемою для новостворених екземплярів METEOR для інших мов. Були підозри, що параметри, оптимізовані для максимальної кореляції з людськими судженнями для англійської мови, не обов'язково будуть оптимальними для інших мов.

В результаті було проведено пошук на так би мовити «сходження на гору», щоб виявити параметри, які досягають максимальної кореляції з людськими судженнями на тренувальному наборі. Використовувалися коефіцієнт кореляції Пірсона як міру кореляції. Ми дотримувались тренувальної процедури «випустити поза», щоб уникнути перенасичення. Коли n систем було доступно для певної мови, ми тренуємо параметри n разів, залишаючи одну систему у кожному навчанні та об’єднуючи сегменти з усіх інших систем. Кінцеві значення параметрів обчислюються як середнє значення з n отриманих наборів підготовлених параметрів. Оцінюючи набір параметрів на тестових даних, ми обчислюємо кореляцію рівня сегмента з судженнями людини для кожної з систем тестового набору, а потім повідомляємо про середнє значення для всіх систем.

3. ОРАГНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

3.1 Загальний підхід до визначення економічної ефективності розробки

Обов’язковою складовою частиною будь-якого інженерного проєкту, в тому числі софтверного, є фінансові витрати на різних етапах виконання робіт. Відповідно, важливо вірно здійснити фінансову оцінку передбачуваних витрат, продуктивність, корисність та, в результаті, економічну ефективність проєкту.

Оцінка вартості дослідницьких розробок базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації. Так, як результати роботи у вигляді математичних моделей та реалізованого на їх основі ПЗ не буде використовуватися в комерційних цілях та не підлягатиме продажу, а становить наукову та інтелектуальну цінність, то доходу від продажі ПЗ та розробки як такого не передбачається. Іншими словами, всі вкладені кошти та витрати на розробку даного рішення є не взаємоокупними, що несуть лише витрати у кількості залучених ресурсів та матеріальних засобів.

Згідно Статті 8 Закону № 3792-12 передбачено, що твори наукового характеру та комп’ютерні програми є об’єктами авторського права[18]. У разі реєстрації виконаної роботи як інтелектуальної власності та авторського права у державній службі інтелектуальної власності України необхідно буде сплатити збори за державну реєстрацію (382,5 грн.).

Для отримання відмінних результатів експериментів та доцільності розробки такого спеціалізованого ПЗ потрібні відповідні затрати на дослідження та розробку. Це і становитиме основу витрат, які будуть здійснені протягом підготовки та виконання реалізації даного рішення. Уявно модель

витрат можна поділити на дві основні частини: витрати, пов’язанані на дослідження предметної області, побудову математичних моделей, отримання попередніх результатів експериментів, та частину реалізації програмної системи, архітектури та тестування.

До створення ПЗ можуть бути залучені позаштатні програмісти як зареєстровані, так і не зареєстровані підприємцями. В обох випадках співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Щодо оподаткування виплат за договором підряду, то все залежить від того, чи зареєстрований виконавець підприємцем. Важливим етапом розробки ПЗ є його тестування, яке виконується тестувальником за певну винагороду. Якщо тестувальники не перебувають з підприємством у трудових відносинах, оплата виконується на основі договору підряду на виконання робіт з тестування ПЗ. Сам же результат розробки не оподатковується, адже не є комерційним проєктом і не спрямований на продаж.

3.2 Розрахунок вартості процесу розробки та оцінка економічної ефективності проєкту

Виконання розробки програмного забезпечення з огляду економічної моделі можна виконувати двома способами: процедурним та об’єктно-орієнтованим. Обидва підходи потребують залучення ресурсів у вигляді програмісті-розробників, тестувальників, керівника проєкту, наукового ресурсу. Різниця виникає в самій схемі розробки, тривалості періоду розробки та відповідній вартості. Процедурний підхід для розробки ПЗ в основі якого лежать процедури і функції передбачає розробку ПЗ як монолітного композиту, що в подальшому, як правило, вимагає великих витрат на супровід та модернізацію. Об’єктно-орієнтований підхід, що ґрунтується на основі об’єктів певних класів, що описують певну область, описують певну поведінку

(методи) та володіють властивостями (атрибутами), орієнтовані на варіанти використання та покроковий процес розробки [19].

Для початку робіт необхідно скласти технічне завдання на розробку, яке є основним документом, що регламентує подальшу роботу, та містить докладний опис необхідних функцій програми, інтерфейс, технології, інше. Вартість складання технічного завдання переважно складає до 10% від планованої вартості розробки. Роботу зі складання технічного завдання веде керівник проєкту разом із програмістами та консультуючись із замовником.

Усі програмісти, що працюють у штаті підприємства-розробника мають встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість (днів) і основна заробітна плата кожного учасника техпроцесу представлено у таблиці 4.1. Всі суми наведені в національній валюті – в гривні.

Таблиця 4.1 – Розрахункова вартість технологічного процесу розробки

Посада	Місячний оклад, грн.	Денна зар. плата, грн.	Об'єктно-орієнтований підхід		Процедурний підхід	
			Днів	Сума, грн.	Днів	Сума, грн.
Керівник	13310	605	15	9075	16	9680
Програміст	11990	545	21	11445	25	13625
Тестувальник	9966	453	7	3171	7	3171
Додаткова зар. плата 20%			43	4738,2	48	5295,2
Фонд оплати праці 36,77%				8711,18		9735,23
Всього витрат на зар. плату				32402,18		36211,23
Військовий збір 1,5%				486,03		543,17
Єдиний соціальний внесок 3.6%				1166,48		1303,60
ПДВ, 15%				4860,33		5431,68
Всього				**38915,02**		**43489,68**

Згідно вимог та прорахованої кількості необхідних ресурсів на виконання, розробку, тестування та дослідницьку роботу було отримано основні часові рамки роботи над проєктом. Так для об'єктно-орієнтованого підходу загальна тривалість роботи над ПЗ становить 38 робочих днів (під робочим днем розуміється 8-ми годинний робочий день), що включає роботу програміста, який, в свою чергу, являється і керівником розробки, роботу тестувальника та наукового працівника. Сума витрат на заробітню плату становить 51222,21 гривень включаючи всі види додаткових оплат. Для процедурного підходу до розробки суми дещо більші, адже затрачається більше часу на розробку. Так, при використанні процедурного підходу сумарна тривалість часу розробки становить 42 робочі дні, та витрати у вигляді виплат заробітної плати становлять 54571,95 гривень.

Витрати на науково-дослідницьку роботу та здійснення розробки програмних продуктів і об'єктно-орієнтованим, і процедурним способом включають:

Основна заробітна плата:

$$ЗП_{осн\ 1} = 23691 \text{ грн}; \qquad ЗП_{осн\ 2} = 26476 \text{ грн}.$$

Додаткова заробітна плата обчислюється як $ЗП_{дод} = 0,2 \cdot ЗП_{осн}$.

$$ЗП_{дод\ 1} = 0,2 \cdot 23691 = 4738,20 \text{ грн}; \quad ЗП_{дод\ 2} = 0,2 \cdot 26476 = 5295,20 \text{ грн}.$$

Нарахування на фонд оплати праці (ФОП):

$$ФОП_{ЄСВ} = 0,3677 \cdot ФЗП$$

$$ФОП_{ЄСВ1} = 0,3677 \cdot 28429,20 = 8711,18 \text{ грн};$$

$$ФОП_{ЄСВ1} = 0,3677 \cdot 31771,20 = 9735,23 \text{ грн}.$$

Всього витрат:

$$В_{ЗП1} = ЗП_1 + ФОП_{ЄСВ1} + ЗП_{дод1} = 38915,02 \text{ грн};$$

$$В_{ЗП2} = ЗП_2 + ФОП_{ЄСВ2} + ЗП_{дод2} = 43489,68 \text{ грн}.$$

З цієї суми утримуються обов'язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, який складає 15% від суми

нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов'язкове соціальне страхування та податкової соціальної пільги, військовий збір у розмірі 1,5%, від суми нарахувань.

До окремих витрат також відносяться витрати на куповані вироби (матеріальне забезпечення) та спец обладнання для підтримки експерименту, накладні витрати. Витрати, що будуть супроводжувати проєкт розробки, порівнюватимемо в двох можливих підходах розробки.

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (формула 4.1).

$$M_{Bi} = q_i \cdot p_i \ , \tag{4.1}$$

де q_i – кількість витраченого матеріалу i-го виду; p_i – ціна матеріалу i-го виду.

Матеріальні витрати в рамках проєкту наведені в таблиці 4.2. Загальна сума матеріальних витрат становить 1189 гривень.

Таблиця 4.2 – Матеріальні витрати

Найменування ресурсу	Кількість, шт.	Ціна одиниці, грн	Загальна сума, грн
Флешки	3	195,00	585,00
Папір для друку А4, арк	500	0,166	83,00
Тонер для принтера	1	53,00	53,00
Дошка для записів	1	435,00	435,00
Перманентний маркер	3	11,00	33,00
Всього			1189,00

Розрахунок витрат на електроенергію одиниці обладнання визначаються за формулою:

$$З_в = W * T * S, \quad (4.2)$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії, S = 2,50 грн/кВт·год.

$З_{в1}$ =0.7 * 344 * 2.5 = 602 грн;

$З_{в2}$ =0.7 * 384 * 2.5 = 672 грн;

Розрахунок суми амортизаційних відрахувань. Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 %, вартість яких перевищує 1000 грн. і визначається:

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{ГОД}} \quad (4.3)$$

де $C_Б$ – балансова вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, %; $T_{ФАК}$ – фактичний час роботи обладнання по написанню програми, год; $T_{ГОД}$ – річний робочий фонд часу, год.

У даній формулі норма відрахувань на амортизацію рівна N_A = 0,6. Балансова вартість обладнання вказана в таблиці 4.3 і рівна $C_Б$ = 22780 гривень. Річний робочий фонд часу приймемо за $T_{ГОД}$ = 2120 годин. З них реальний фактичний робочий час становить $T_{ФАК}$ = 344 години згідно об'єктно-орієнтованого підходу та $T_{ФАК}$ = 384 годин згідно процедурного підходу.

Згідно вищезгаданої формули витрати на амортизацію становлять 1959,94 гривень та 2166,25 гривень для кожного підходу відповідно.

Таблиця 4.3 – Перелік необхідного обладнання

Найменування	Кількість, шт	Ціна, грн	Сума, грн	
Комп’ютер	2	9830,00	19660,00	
Принтер	1	3120,00	3120,00	
Середовища розробки	2	безкоштовно	безкоштовно	
Операційна система (Ubuntu Linux)	2	безкоштовно	безкоштовно	
Всього більше 1000 грн.			22780,00	
Всього витрат на амортизацію			1959,94	2166,25
Всього			24739,94	24946,25

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління та створення необхідних умов праці та закупівлю ресурсів та обладнання для розробки наведені в таблиці 4.3.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників. Нехай вона буде дорівнювати 40%, що становить 11371,68 грн для об’єктно-орієнтованого і 12708,48 грн для процедурного підходу розробки.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток. Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни.

Можна отримати загальні значення витрат на розробку та реалізацію проєкту, враховуючи всі вище описані затрати та нарахування. Цей вид витрат складається з сум витрат на оплату праці (всього витрати на оплату праці),

матеріальні затрати, затрати на електроенергію, накладні витрати, витрати на обладнання, враховуючи амортизації обладнання на час виконання проєкту.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 76743,38 грн використовуючи об'єктно-орієнтований підхід, 82916,71 грн при процедурному підході розробки.

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (Е) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E = \frac{П}{Св} \tag{4.4}$$

де $П$ – прибуток, $П = В - Св$; $Св$ – собівартість.

У випадку даної розробки, маючи некомерційний проєкт без економічно корисного результату, можна прогнозувати, що економічна ефективність прямує до 0 у обох випадках. Однак це не є причиною для негативного економічного висновку щодо даного проєкту, адже такого плану розробки приносять користь у вигляді інтелектуальних ресурсів, і, переважно, фінансуються або виконуються на замовлення організацій, зацікавлених в отриманні результатів досліджень та розробок. Фінансування не можна вважати отриманим доходом від реалізації. Однак за надходження коштів на реалізацію ззовні можна вважати ефективність проєкту рівною 1 (Е = 1), що означає перекриття витрат на розробку у повній мірі, тобто фінансування.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Поряд

із економічною ефективністю розраховують термін окупності капітальних вкладень ($T_{ок}$):

$$T_{ок} = \frac{1}{E} \tag{4.5}$$

У нашому випадку прямого прибутку не існує. Прибуток можна прогнозувати на підприємствах, організаціях чи відомствах, що зацікавлені в дослідженні. Окупність же для розробки даного ПЗ за ефективності рівній одиниці можна вважати теж рівній 1 згідно формули 4.4.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за об'єктно-орієнтованим методом 25% (19185,84 грн) від початкових витрат, а за процедурним – 30% (24875,01 грн).

Однак варто зауважити, що розробка спрямована на короткотривалу підтримку та не прогнозує модернізації. У разі ж необхідності розробки такого ж або суміжного ПЗ можна вважати за доцільно розпочинати розробку з початкових етапів, що потягне за собою нові витрати у повній мірі. Тобто у разі короткотермінової підтримки все ж за доцільніше обирати об'єктно-орієнтовану модель розробки, адже вартість короткотермінової підтримки згідно цієї моделі не вплине значно на сукупну вартість розробки.

Здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. Об'єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізувати програми, що значно скорочує подальші витрати на супровід і модернізацію [20].

Сумарні дані економічного розрахунку розробки даного проєкту наведені в таблиці 4.4.

Таблиця 4.4 – Загальні витрати

Вид витрат	Об’єктно-орієнтований підхід, грн	Процедурний підхід, грн
Зарплата основна	23691,00	26476,00
Зарплата додаткова	4738,20	5295,20
Фонд заробітної плати	28429,20	31771,20

Продовження таблиці 4.4

Відрахування на ФОП	8711,18	9735,23
Разом на оплату праці	38915,02	43489,68
Матеріальні витрати	1189,00	1189,00
Електроенергія	602,00	672,00
Амортизація	1959,94	2166,25
Накладні витрати	11371,68	12708,48
Обладнання	22780,00	22780,00
Разом на ін. витрати	36390,84	37598,90
Собівартість	76817,64	83005,41
Прибуток	відсутній	відсутній
Вартість розробленого ПЗ	**76817,64**	**83005,41**
Модернізація і супровід	19204,41	24901,62
Загальні витрати на розробку	96022,05	107907,03
Економія ($ЗВ_1$-$ЗВ_2$)		**11884,99**

Загальна вартість пропонованих робіт становить 107907,03 гривень для процедурного і 96022,05 гривень для об’єктно-орієнтованого підходів розробки. В даному випадку реалізації проєкту варто вибрати об’єктно орієнтований підхід для розробки даного ПЗ, адже фінансово це більш вигідно. Також у даній методиці розробки кращі часові рамки та перспективи підтримки і модернізації. У оцінці вартості продукту варто враховувати можливість фінансування та спонсорування проєкту, що дозволить гнучко та ефективно підійти до розробки та організації праці.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Розробка програмну систему машинного перекладу на основі нейромережевих технологій здійснювалась з використанням вектора метрики якості з використанням портативних пристроїв та персональних комп'ютерів. Відповідно, в ході виконання експлуатації доведеться працювати з комп'ютерною технікою. Тому надзвичайно важливим фактором безпеки праці є дотримання правил користування технікою, норм та правил охорони праці. Потрібно забезпечити користувачам максимально комфортні та безпечні умови для їх перебування в приміщенні та якісного, ефективного виконання поставлених завдань.

Для забезпечення норм охорони праці та безпеки використання програмної системи користувачі повинні проходити первинний інструктаж з охорони праці на робочому місці та бути проінформованими щодо правил користування відповідних приладів. В подальшому вони проходять повторні інструктажі з охорони праці на робочому місці раз в півріччя.

Відповідно до встановлених санітарно-гігієнічних норм (ГОСТ 12.1.005-88, ДСН 3.3.6.042-99) регламентуються вимоги до приміщення лабораторії, де користуватимуться розроблюваною системою:

Для внутрішнього оздоблення приміщень з персональними комп'ютерами слід обирати світлі нейтральні кольори стін. Покриття підлоги та поверхня має бути рівною, неслизькою, з антистатичними властивостями. Ці умови оздоблення необхідно враховувати, адже при розробці системи машинного перекладу вимагає забезпечення максимально комфортного середовища. Це сприятиме максимальній ефективності отримання результатів тестування пацієнта та роботи працівника лабораторії.

Основним нормативним документом, що регулює забезпечення охорони праці користувачів комп'ютерної техніки є «Державні санітарні норми електронно-обчислювальних машин» ДСанПіН 3.3.2.007-98, «Державні санітарні правила і норми. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно- обчислювальних машин». У виробничих приміщеннях та на робочих місцях з ВДТ та ПК мають бути забезпечені оптимальні значення параметрів мікроклімату - температури повітря, відносної вологості, швидкості руху повітря. Для цього приміщення, в яких розташовані комп'ютеризовані робочі місця повинні бути обладнані системами опалення, кондиціонування, які автоматично підтримують задані параметри мікроклімату.

Потрібно створити сприятливі умови для зорової роботи, які б мінімізували втому очей, виникнення професійних захворювань та сприяли підвищенню продуктивності праці. Тому освітлення повинне відповідати вимогам ДБН В.2.5-28:2018 «Природне і штучне освітлення». Основною вимогою є необхідність створення на робочій поверхні освітленості, що відповідає характеру зорової роботи і знаходиться в межах встановлених норм. Освітлення у приміщенні має бути суміщеним. Відтак, недостача денного природного освітлення компенсується необхідною для приміщення кількістю штучного освітлення. Як джерело штучного освітлення в приміщеннях, де встановлено комп'ютерну техніку, бажано використовувати люмінесцентні лампи. Освітленість робочого місця у горизонтальній площині на висоті 0,8 м від рівня підлоги повинна бути не менше 400 лк. Для захисту від прямих сонячних променів повинні бути передбачені сонцезахисні пристрої, жалюзі, штори.

При розробці програмного продукту та в ході його експлуатації користуються лініями електромереж. Для побудови системи потрібно використовувати лише якісні та сертифіковані пристрої та засоби. Персональні комп'ютери і периферійні пристрої повинні підключатися до

електромережі тільки за допомогою справних штепсельних з’єднань і електророзеток заводського виготовлення. В них, окрім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Усі електроприлади, згідно з ДНАОП 0.00-1.21-98, повинні бути заземленні за допомогою нульового захисного провідника.

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), мають бути надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння працівника під напругу. Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією.

Приміщення мають бути оснащені системою автоматичної пожежної сигналізації і вогнегасниками відповідно до вимог чинного законодавства України. Проходи до засобів пожежогасіння мають бути вільними. Згідно техніки пожежної безпеки пристрої повинні розташовуватися не ближче одного метра від джерел тепла. Також на них не повинні падати прямі сонячні промені, щоб виключити можливість перегріву компонентів та вбудованих акумуляторів.

Користувачі програмної системи машинного перекладу на основі нейромережевих технологій з використанням вектора метрики якості повинні дотримуватися правил та норм поведінки з комп’ютерною технікою та портативними пристроями. У разі виникнення ситуацій, які суперечать нормам охорони праці та можуть бути причиною негативних наслідків чи завдати шкоди, потрібно припинити виконання діагностичного тесту та повідомити керівника роботи чи особу, що відповідає за охорону праці в лабораторії, про порушення та проблеми.

Під час розробки, тестування та впровадження системи машинного перекладу були дотримані всі вимоги, норми та державні стандарти з охорони праці.

4.2 Електробезпека користувачів персонального комп’ютера

Сучасний комп’ютер є електротехнічним пристроєм загального користування, не є засобом підвищеної небезпеки, а ремонт його внутрішнього електронного обладнання можливий тільки у сервісних центрах за наявності спеціального обладнання.

Вимоги електробезпеки у приміщеннях, де встановлені електронно-обчислювальні машини і персональні комп'ютери (далі — ЕОМ) відображені у ДНАОП 0.00-1.31-99. Відповідно до цього нормативного документу під час проектування систем електропостачання, монтажу основного електрообладнання та електричного освітлення будівель та приміщень для ЕОМ необхідно дотримуватись вимог Правил влаштування електроустановок (ПВЕ), ГОСТ 12.1.006-84, ГОСТ 12.1.019-79, ГОСТ 12.1.030-81, ГОСТ 12.1.045-84, ПТЕ, ПБЕ, ВСН 59-88 "Электрооборудованиежилых и общественных зданий. Нормыпроектирования", СН 357-77 "Инструкция по проектированию силового осветительногооборудованияпромишленныхпредприятий", Правил пожежної безпеки в Україні та інших нормативних документів, що стосуються штучного освітлення і електротехнічних пристроїв, а також вимог нормативно-технічної експлуатаційної документації заводу-виробника.

Приміщення із робочими місцями користувачів комп’ютерів для забезпечення електробезпеки обладнання, а також для захисту від ураження електричним струмом самих користувачів ПК повинні мати достатні технічні засоби захисту.

Під час монтажу та експлуатації ліній електромережі необхідно повністю унеможливити виникнення електричного джерела загоряння внаслідок короткого замикання та перевантаження проводів, обмежувати застосування проводів з легкозаймистою ізоляцією і, за можливості, перейти на негорючу ізоляцію.

Лінія електромережі для живлення ЕОМ, периферійних пристроїв ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ виконується як окрема групова трипровідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів і прокладається від стійки групового розподільчого щита, розподільчого пункту до розеток живлення

Використання нульового робочого провідника як нульового захисного провідника забороняється. Нульовий захисний провід прокладається від стійки групового розподільчого щита, розподільчого пункту до розеток живлення. Не допускається підключення на щиті до одного контактного затискача нульового робочого та нульового захисного провідників. Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі повинна бути не менше площі перерізу фазового провідника.

Усі провідники повинні відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам Правил налаштування електроустанов.

У приміщенні, де одночасно експлуатується або обслуговується більше п'яти персональних ЕОМ, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

ЕОМ, периферійні пристрої ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ повинні підключатися до електромережі тільки з допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників повинні мати спеціальні контакти для підключення нульового захисного провідника. Конструкція їх має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше ніж приєднання фазового та нульового робочого провідників. Порядок роз'єднання при відключенні має бути зворотним. Необхідно унеможливити з'єднання контактів фазових провідників з контактами нульового захисного провідника.

Неприпустимим є підключення ЕОМ, периферійних пристроїв ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ до звичайної двопровідної електромережі, в тому числі — з використанням перехідних пристроїв.

Електромережі штепсельних з'єднань та електророзеток для живлення ЕОМ, периферійних пристроїв слід виконувати за магістральною схемою, по 3…6 з’єднань або електророзеток в одному колі. Штепсельні з'єднання та електророзетки для напруги 12 В та 36 В за своєю конструкцією повинні відрізнятися від штепсельних з'єднань для напруги 127 В та 220 В і мають бути пофарбовані в колір, який візуально значно відрізняється від кольору штепсельних з'єднань, розрахованих на напругу 127 В та 220 В.

Індивідуальні та групові штепсельні з'єднання та електророзетки необхідно монтувати на негорючих або важкогорючих пластинах з урахуванням вимог Правил налаштування електроустанов та Правил пожежної безпеки в Україні.

Електромережу штепсельних розеток для живлення ЕОМ, периферійних пристроїв ЕОМ при розташуванні їх уздовж стін приміщення прокладають по

підлозі поряд зі стінами приміщення, як правило, в металевих трубах і гнучких металевих рукавах з відводами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання.

При розташуванні в приміщенні за його периметром до 5 персональних ЕОМ, використанні трипровідникового захищеного проводу або кабелю в оболонці з негорючого або важкогорючого матеріалу дозволяється прокладання їх без металевих труб та гнучких металевих рукавів.

Електромережу штепсельних розеток для живлення ЕОМ при розташуванні їх у центрі приміщення, прокладають у каналах або під знімною підлогою в металевих трубах або гнучких металевих рукавах. При цьому не дозволяється застосовувати провід і кабель в ізоляції з вулканізованої гуми та інші матеріали, що містять сірку. Відкрита прокладка кабелів під підлогою забороняється. Металеві труби та гнучкі металеві рукави повинні бути заземлені. Заземлення повинно відповідати вимогам НПАОП 40.1-1.21-98.

Для підключення переносної електроапаратури застосовують гнучкі проводи в надійній ізоляції. Тимчасова електропроводка від переносних приладів до джерел живлення виконується найкоротшим шляхом без заплутування проводів у конструкціях машин, приладів та меблях. Доточувати проводи можна тільки шляхом паяння з наступним старанним ізолюванням місць з'єднання.

Металеві труби та гнучкі металеві рукави повинні бути заземлені. Заземлення повинно відповідати вимогам ДНАОП 0.00-1.21-98 "Правила безпечної експлуатації електроустановок споживачів". Заземлені конструкції, що знаходяться у приміщеннях (батереї опалення, водопровідні труби, кабелі із заземленим відкритим екраном тощо), мають бути надійно захищені діелектричними щитками або сітками від випадкового дотику.

Конструкція знімної підлоги повинна бути такою, щоб забезпечувались:

- вільний доступ, до кабельних комунікацій під час обслуговування;
- стійкість до горизонтальних зусиль при частково знятих плитах;
- вирівнювання поверхні підлоги за допомогою регулювальних опорних елементів;
- взаємозамінюваність плит.

Є неприпустимими:

- експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості за час експлуатації, ізоляцією; залишення під напругою кабелів та проводів з неізольованими провідниками;
- застосування саморобних продовжувачів, які не відповідають вимогам ПВЕ до переносних електропроводок;
- застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання;
- користування пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має сліди затемнення або випинання;
- підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною та іншими горючими матеріалами, експлуатація їх зі знятими ковпаками (розсіювачами);
- використання електроапаратури та приладів в умовах, що не відповідають вказівкам (рекомендаціям) підприємств-виготовлювачів.

Заходи щодо усунення небезпеки ураження електричним струмом зводяться до правильного розміщення устаткування та електричних кабелів. Інші заходи щодо забезпечення електробезпеки, збігаються з загальними заходами пожежо- та електробезпеки.

В якості профілактичних заходів для забезпечення пожежної безпеки слід використовувати скриту електромережу, надійні розетки з пожежобезпечних матеріалів, силові мережі живлення устаткування виконувати кабелями, розрахованими на підключення в 3-5 разів більшого навантаження, включати й виключати живлення обладнання за допомогою штатних вимикачів. Треба регулярно робити очистку внутрішніх частин комп'ютерів, іншого устаткування від пилу, розташовувати комп'ютери на окремих неспалюваних столах. Для запобігання іскріння необхідно рідше встромляти і виймати штепсельні вилки з розеток.

ВИСНОВКИ

В дипломній роботі магістра було розроблено систему машинного перекладу на основі нейромережевих технологій з використанням вектора метрик якості. Було розглянуто проблему різних видів машинного навчання.

Використання машинного перекладу важливіше, ніж ми можемо подумати. Можна стверджувати, що ресурси, доступні для перекладача через інформаційні технології, передбачають зміну взаємозв'язку між перекладачем і текстом, тобто новий спосіб перекладу. Однак є розвиток нових можливостей, що призводить до того, щоб ми вказали на ряд істотних аспектів ситуації, що склалася. Переклад за допомогою комп’ютера, безумовно, не те саме, що працювати виключно на папері та з паперовими виробами, такими як звичайні словники, тому що комп’ютерні інструменти забезпечують нам ставлення до тексту, яке набагато гнучкіше, ніж чисто лінійне читання. Крім того, Інтернет з його універсальним доступом до інформації та миттєвим спілкуванням між користувачами створив фізичну та географічну свободу для перекладачів, які були немислимі в минулому.

Перекладачам потрібно прийняти нові технології та навчитися використовувати їх з максимальним потенціалом як засіб для підвищення продуктивності та підвищення якості. Як ми вже згадували, існують проблеми неоднозначності при роботі з МТ, і ці проблеми також є спільними для нас. Яскравим прикладом можуть бути переклади з іспанської на баскську.

У даній програмній системі були розроблені дві технології машинного навчання. Користувач може наочно порівняти ці дві технології, або добавити інший датасет певного розширення та форматування для перекладу іншими мовами, а зад допомогою метрики якості було порівняно результати цих двох систем на точність та якість перекладу.

В ході розробки дипломної роботи магістра були отримані результати, які показують, що використання машинного перекладу на основі технології довгої короткотривалої пам’яті (LSTM) показує кращі результати ніж рекурентна нейронна мережа (RNN).

Розроблена інформаційна система містить в собі всі необхідні функціональні можливості для впровадження його в програмні системи, де необхідно використовувати переклади, або використовуються тексти з декількома мовами та їх обробка. Так як ця система дає хороші результати, тому таким чином, розроблена інформаційна система доводить свою життєздатність, ефективність і необхідність її впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Philosophical Transactions − the world's first science journal. – Режим доступу: http://rstl.royalsocietypublishing.org/. – Дата доступу: 05.05.2017
2. A. M. TURING I.—COMPUTING MACHINERY AND INTELLIGENCE/ A.M. Тюрінг // Mind. – 1950. – № 236. – С. 433-460.
3. Stochastic neural analog reinforcement calculator. – Режим доступу: http://cyberneticzoo.com/mazesolvers/1951-maze-solver-minsky-edmondsamerican/.
4. How Many Computers to Identify a Cat? – Режим доступу: http://www.nytimes.com/2012/06/26/technology/in-a-big-network-ofcomputers-evidence-of-machine-learning.html. – Дата доступу: 05.05.2017
5. In a huge breakthrough, google's ai beats a top player at the game of go. – Режим доступу: https://www.wired.com/2016/01/in-a-huge-breakthroughgoogles-ai-beats-a-top-player-at-the-game-of-go
6. Онлайн журнал engadget (Google DeepMind AI wins final Go match for 4-1 series win). – Режим доступу: https://www.engadget.com/2016/03/14/thefinal-lee-sedol-vs-alphago-match-is-about-to-start/. – Дата доступу: 05.05.2017
7. Comparison of the DQN agent with the best reinforcement learning methods. – Режим доступу: http://www.nature.com/nature/journal/v518/n7540/fig_tab/nature14236_F3.html.
8. Офіційний сайт Keras. – Режим доступу: https://keras.io.
9. Моделирование процессов обучения в нейронных сетях. – Режим доступу: http://old.exponenta.ru/soft/others/mvs/stud3/3.asp.
10. CS234: Reinforcement Learning – Режим доступу: http://web.stanford.edu/class/cs234/index.html.
11. Alex Galea, Luis Capelo «Applied Deep Learning with Python. Use scikit-learn, TensorFlow, and Keras to create intelligent system and machine learning solutions» 2018 p. c.214-215, c.216-217.

12. Steven Bird, Ewan Klein, Edward Loper “Natural Language Processing with Python. Analyzing Text with the Natural Language Toolkit”, c 292-304.
13. Mnih, V. Playing Atari with deep reinforcement learning. Technical Report / Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller M. – DeepMind Technologies, 2013 – C. 7
14. Schematic illustration of the convolutional neural network. – Режим доступу: http://www.nature.com/nature/journal/v518/n7540/fig_tab/nature14236_F1.html.
15. Comparison of games scores obtained by DQN. – Режим доступу: http://www.nature.com/nature/journal/v518/n7540/fig_tab/nature14236_ST2.html.
16. Pooyan J. Self-Learning Cloud Controllers: Fuzzy Q-Learning for Knowledge Evolution / Pooyan J., Amir S., Claus P., Andreas M., Giovani E. – International Conference on Cloud and Autonomic Computing, 2015 – C. 8
17. Gers, F. A.; Schmidhuber, E. LSTM recurrent networks learn simple context-free and context-sensitive languages // IEEE Transactions on Neural Networks: journal. — 2001. — November (vol. 12, no. 6). — P. 1333—1340. — ISSN 1045-9227. — DOI:10.1109/72.963769
18. Форма №2 «Звіт про фінансові результати»: методика підготовки[Електронний ресурс]. – Режим доступу: URL: http://osvita.ua/vnz/reports/accountant/17368/.
19. Лаврищева К.М. «Програмна інженерія» - С. 95 – Режим доступу: http://www.programsfactory.univ.kiev.ua/content/books/2/28
20. Методичні вказівки для виконання розділу дипломної роботи щодо техніко-економічного обґрунтування вибору проєктного рішення розробки та оцінки якості програмного забезпечення/ Упор. Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І. Пулюя. – 2013. -34 с.

21. Правила безпечної експлуатації електроустановок споживачів [Текст] : ДНАОП 0.00-1.21-98. - Київ : Держнаглядохоронпраці, 2003. - 383 с.
22. Жидецький В. Ц. Охорона праці користувачів комп'ютерів. – Львів: Афіша, 2000. - 176 с.
23. Наказ Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду «Про затвердження Правил охорони праці під час експлуатації електронно-обчислювальних машин» від 26.03.2010 № 65 – Режим доступу: URL: http://zakon2.rada.gov.ua/laws/show/z0293-10;

ДОДАТКИ

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою

програмної інженерії

"___ "_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської дипломної роботи

на тему: «Розробка програмного забезпечення адміністрування бібліотечних даних на основі використання системи Koha із вбудованим модулем каталогізації.»

Баранський Максим Олександрович _____ ТЗ

Керівник роботи:
д.т.н., професор Пастух О.А.
"___" _____ 2019р.
_____

Виконавець:
студент групи СПм-62
Баранський Максим Олександрович
"___" _____ 2019р.
_____

м. Тернопіль – 2019

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка системи машинного перекладу на основі нейромережевих техноголій з використанням вектора метрик якості.».

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Дипломна робота присвячена створенню програмної системи машинного перекладу на основі нейромережевих техноголій з використанням вектора метрик якості.

Предметом дослідження: є програмна система машинного перекладу на основі двох різних нейромережевих техноголій так як RNN та LSTM з використанням вектора метрик якості METEOR.

Мета роботи дослідити яка з нейромережевих техноголій є кращою для машинного перекладу, перевірка чого буде здійснена за допомогою метрики METEOR.

За результатами виконаної роботи необхідно отримати програмне забезпечення для машинного перекладу та можливістю перевірити результати точності перекладу

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмне забезпечення має виконувати наступні дії:

- Заміна користувачем датасету з потрібною мовою;
- Переклад за допомогою технології
- Переклад за допомогою технології
- Перевірка точності та коректності перекладу

3.2 Склад та параметри технічних засобів

ПК з 8 Гб оперативної пам‘яті, встановленою системою Windows, Linux. Не менше 1000 Мб вільного місця на жорсткому диску. Двоядерний процесор з тактовою частотою від 2.4 GHz і більше.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати в різних операційних системах. Розроблювана програмна система повинна бути пристосована до використання у різних комп’ютерах. Розробку виконувати з використанням мови Python, з використанням дистрибутиву Anacondata бібліотеки Keras.

4. СТАДІЇ РОЗРОБКИ

В ходів реалізації роботи проект повинен пройти крізь наступні стадії розробки:

- аналіз предметної області;
- проектування архітектури;
- реалізація класів і модулів;
- тестування результатів розробки;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Пояснювальна записка;
- Технічне завдання;
- Презентаційний матеріал;
- Інструкція користувача;
- Додатки.

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Приймання проводиться спеціально створеною екзаменаційною комісією в термін до:

“24” грудня 2019р.

УДК 004.415.5
М. Баранський ст. гр. СПм-62.
Тернопільський національний технічний університет імені Івана Пулюя

Розробка системи машинного перекладу на основі нейромережевих техноголій з використанням вектора метрик якості

M. Baranskyi
Ternopil Ivan Puluj National Technical University

Development of machine translation system based on technology of neural networks with using a quality metric vector

Доповідь присвячена програмній реалізації системи машинного перекладу на основі нейромережевих технологій з використанням вектора метрик якості. Проведено аналіз предметної області. Після проведення перед проектного дослідження предметної області, поставлено завдання на розробку системи, сформульовано вимоги до ресурсу.

Професійних перекладачів вчать передавати зміст своїми словами, не прив'язуючись до структури речень оригінального тексту. Адекватний переклад повинен прагнути від дослівній і пофразовій передачі до сенсових трансформацій. Правильна «конвертація» граматичних конструкцій однієї мови засобами іншої це межа для статистичного машинного перекладу. Це обмеження не скасовується навіть на базі споріднених мов і створює більше проблем в інтерпретації, але і менше граматичної схожості мов в парі.

Як і статистичний переклад, нейронний переклад вимагає для навчання паралельні корпуси, що дозволяють порівняти автоматичний переклад з еталонним «людським», тільки в процесі навчання оперує не окремими фразами і словосполученнями, а цілими реченнями [1]. Основна проблема в тому, що для тренування такої системи потрібно істотно більше обчислювальних потужностей.

Наступний етап полягав у виборі архітектур для розробки системи машинного навчання та тренування нейронних мереж кожної з архітектур. Наступним кроком є визначення точності перекладу за допомою метрики METEOR - метрика для оцінювання якості машинного перекладу. Метрика базується на використанні n-gram та орієнтована на використання статистичної та точної оцінки вихідного тексту[2].

Далі було створено систему машинного перекладу, використовуючи мову програмування Python. Для реалізації архітектури було обрано бібліотеку управління базами даних Keras та дистрибутив розробки Anaconda, яка включає у собі дану бібліотеку. У якості даних для навчання будуть використовуватися двісті тисяч перекладів речень з англійської на німецьку.

Розробка архутектури системи машинного перекладу є одним з ключових моментів його створення, Практичне значення отриманих результатів дослідження полягає у тому, що створена комп’ютерна система допомагає визначити котра з обраних архітектур є найбільш точною для використання при розробці різних систем машинного перекладу.

Література:

1. Alex Galea, Luis Capelo «Applied Deep Learning with Python. Use scikit-learn, TensorFlow, and Keras to create intelligent system and machine learning solutions» 2018 p. c.214-215, c.216-217.
2. Steven Bird, Ewan Klein, Edward Loper “Natural Language Processing with Python. Analyzing Text with the Natural Language Toolkit”, c 292-304.

Лістинг 1.1 – Завантаження датасету

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper
import problem_unittests as tests

source_path = 'data/small_vocab_en'
target_path = 'data/small_vocab_fr'
source_text = helper.load_data(source_path)
target_text = helper.load_data(target_path)
```

Лістинг 1.2 – Метод view_sentence_range()

```
view_sentence_range = (0, 10)

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np

print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word:
None for word in source_text.split()})))

sentences = source_text.split('\n')
word_counts = [len(sentence.split()) for sentence in sentences]
print('Number of sentences: {}'.format(len(sentences)))
print('Average number of words in a sentence:
{}'.format(np.average(word_counts)))

print()
print('English sentences {} to
{}:'.format(*view_sentence_range))
print('\n'.join(source_text.split('\n')[view_sentence_range[0]:v
iew_sentence_range[1]]))
print()
print('French sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(ta’rget_text.split('\n')[view_sentence_range[0]:
view_sentence_range[1]]))
```

Лістинг 1.3 – Метод text_to_ids()

```
def text_to_ids(source_text, target_text, source_vocab_to_int,
target_vocab_to_int):
```

```
    """
    Convert source and target text to proper word ids
    :param source_text: String that contains all the source
text.
    :param target_text: String that contains all the target
text.
    :param source_vocab_to_int: Dictionary to go from the source
words to an id
    :param target_vocab_to_int: Dictionary to go from the target
words to an id
    :return: A tuple of lists (source_id_text, target_id_text)
    """

    # source_id_text: nested list comprehension. Return value
from dict, for word in line, for line in text
    # target_id_text: nested list comprehension. Return value
from dict, for word in line, add <EOS> for line in text

    source_id_text = [[source_vocab_to_int[word] for word in
(line).split()]
            for line in source_text.split("\n")]

    target_id_text = [[target_vocab_to_int[word] for word in
line.split()] + [target_vocab_to_int[ "<EOS>"]]
            for line in target_text.split("\n")]

    # Input:  Raw text
    # print("source_text", source_text)

    # Get value from dict to end of line element, a stop " ..in
april ."
    # print("source_vocab_to_int", source_vocab_to_int["."])

    # Print the first sentence --> last element is value 21 (a
stop)
    # print("source_id_text", source_id_text[:1],
type(source_id_text))

    # Print the first target sentence, last element is the added
by us, <EOS>
    # print("target_id_text", target_id_text[:1],
type(target_id_text))

    # print(target_vocab_to_int["<GO>"])

    return source_id_text, target_id_text

"""
```

Лістинг 1.4 – Створення Checkpoint’у

```
"""
```

```
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np
import helper

(source_int_text, target_int_text), (source_vocab_to_int,
target_vocab_to_int), _ = helper.load_preprocess()
```

Лістинг 1.5 – Метод view_model()

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
from distutils.version import LooseVersion
import warnings
import tensorflow as tf

# Check TensorFlow Version
assert LooseVersion(tf.__version__) in [LooseVersion('1.0.0'),
LooseVersion('1.0.1')], 'This project requires TensorFlow
version 1.0  You are using {}'.format(tf.__version__)
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please use a GPU to train your
neural network.')
else:
print('Default GPU Device:
{}'.format(tf.test.gpu_device_name()))

def model_inputs():
    """
    Create TF Placeholders for input, targets, and learning
rate.
    :return: Tuple (input, targets, learning rate, keep
probability)
    """
    input = tf.placeholder(tf.int32, [None, None], name="input")
    targets = tf.placeholder(tf.int32, [None, None],
name="target")
    learning_rate = tf.placeholder(tf.float32,
name="learning_rate")
    keep_prob = tf.placeholder(tf.float32, name="keep_prob")

    return input, targets, learning_rate, keep_prob

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
```

```
"""
tests.test_model_inputs(model_inputs)
```

Лістинг 1.5 – Метод process_decoding_input ()

```
def process_decoding_input(target_data, target_vocab_to_int,
batch_size):
    """
    Preprocess target data for dencoding
    :param target_data: Target Placehoder
    :param target_vocab_to_int: Dictionary to go from the target
words to an id
    :param batch_size: Batch Size
    :return: Preprocessed target data
    """
    # tf.strided_slice(input_, begin, end, strides=None)
    # http://stackoverflow.com/questions/41380126/what-does-tf-
strided-slice-do

    # print("target_data.shape", target_data.shape) #
target_data.shape (2, 3)
    ending = tf.strided_slice(target_data, [0, 0], [batch_size,
-1], [1, 1])
    # print("ending.shape", ending.shape) # ending.shape (2, 2)

    dec_input = tf.concat([tf.fill([batch_size, 1],
target_vocab_to_int['<GO>']), ending], 1) # dec_input (2, 3)
    # print("dec_input", dec_input.shape)

    return dec_input

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_process_decoding_input(process_decoding_input)
```

Лістинг 1.6 – Метод encoding_layer()

```
def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob):
    """
    Create encoding layer
    :param rnn_inputs: Inputs for the RNN
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param keep_prob: Dropout keep probability
    :return: RNN state
```

```
    """
    LSTM = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    enc_cell = tf.contrib.rnn.MultiRNNCell([LSTM]*num_layers)
    drop = tf.nn.dropout(rnn_inputs, keep_prob)
    RNN_output, enc_state = tf.nn.dynamic_rnn(enc_cell, drop,
dtype=tf.float32)

    return enc_state

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_encoding_layer(encoding_layer)
```

Лістинг 1.7 – Метод decoding_layer_train()

```
def decoding_layer_train(encoder_state, dec_cell,
dec_embed_input, sequence_length, decoding_scope,
                         output_fn, keep_prob):
    """
    Create a decoding layer for training
    :param encoder_state: Encoder State
    :param dec_cell: Decoder RNN Cell
    :param dec_embed_input: Decoder embedded input
    :param sequence_length: Sequence Length
    :param decoding_scope: TensorFlow Variable Scope for
decoding
    :param output_fn: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: Train Logits
    """
    train_decoder_fn =
tf.contrib.seq2seq.simple_decoder_fn_train(encoder_state
    train_pred, _, _ = tf.contrib.seq2seq.dynamic_rnn_decoder(
                                dec_cell, train_decoder_fn,
dec_embed_input, sequence_length, scope=decoding_scope)
    train_pred =  output_fn(train_pred)
    train_logits = tf.contrib.layers.dropout(train_pred,
keep_prob)

    return train_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_train(decoding_layer_train)
```

Лістинг 1.8 – Метод decoding_layer_infer()

```
def decoding_layer_infer(encoder_state, dec_cell,
dec_embeddings, start_of_sequence_id, end_of_sequence_id,
                         maximum_length, vocab_size,
decoding_scope, output_fn, keep_prob):
    """
    Create a decoding layer for inference
    :param encoder_state: Encoder state
    :param dec_cell: Decoder RNN Cell
    :param dec_embeddings: Decoder embeddings
    :param start_of_sequence_id: GO ID
    :param end_of_sequence_id: EOS Id
    :param maximum_length: The maximum allowed time steps to
decode
    :param vocab_size: Size of vocabulary
    :param decoding_scope: TensorFlow Variable Scope for
decoding
    :param output_fn: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: Inference Logits
    """
    decoder_infer =
tf.contrib.seq2seq.simple_decoder_fn_inference(output_fn,

encoder_state,

dec_embeddings,

start_of_sequence_id,

end_of_sequence_id,

maximum_length,

vocab_size,

dtype=tf.int32)

    output_infer, state_infer, _ =
tf.contrib.seq2seq.dynamic_rnn_decoder(dec_cell,

decoder_infer,

scope=decoding_scope)

    inference_logits = tf.contrib.layers.dropout(output_infer,
keep_prob)
    return inference_logits


"""
```

```
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_infer(decoding_layer_infer)
```

Лістинг 1.9 – Метод decoding_layer()

```
def decoding_layer(dec_embed_input, dec_embeddings,
encoder_state, vocab_size, sequence_length, rnn_size,
                   num_layers, target_vocab_to_int, keep_prob):
    """
    Create decoding layer
    :param dec_embed_input: Decoder embedded input
    :param dec_embeddings: Decoder embeddings
    :param encoder_state: The encoded state
    :param vocab_size: Size of vocabulary
    :param sequence_length: Sequence Length
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target
words to an id
    :param keep_prob: Dropout keep probability
    :return: Tuple of (Training Logits, Inference Logits)
    """
    dec_cell =
tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.BasicLSTMCell(rnn_si
ze)] * num_layers)

    with tf.variable_scope('decoding') as decoding_scope:
        output_fn = lambda x:
tf.contrib.layers.fully_connected(x, vocab_size, None, scope =
decoding_scope)

        train_logits = decoding_layer_train(encoder_state,
dec_cell, dec_embed_input, sequence_length,
                                            decoding_scope,
output_fn, keep_prob)

    with tf.variable_scope('decoding', reuse=True) as
decoding_scope:
        infer_logits = decoding_layer_infer(encoder_state,
dec_cell, dec_embeddings, target_vocab_to_int['<GO>'],

target_vocab_to_int['<EOS>'], sequence_length, vocab_size,
                                            decoding_scope,
output_fn, keep_prob)

    return train_logits, infer_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
```

```
"""
tests.test_decoding_layer(decoding_layer)
```

Лістинг 1.10 – Метод seq2seq_model()

```
def seq2seq_model(input_data, target_data, keep_prob,
batch_size, sequence_length, source_vocab_size,
target_vocab_size,
                  enc_embedding_size, dec_embedding_size,
rnn_size, num_layers, target_vocab_to_int):
    """
    Build the Sequence-to-Sequence part of the neural network
    :param input_data: Input placeholder
    :param target_data: Target placeholder
    :param keep_prob: Dropout keep probability placeholder
    :param batch_size: Batch Size
    :param sequence_length: Sequence Length
    :param source_vocab_size: Source vocabulary size
    :param target_vocab_size: Target vocabulary size
    :param enc_embedding_size: Decoder embedding size
    :param dec_embedding_size: Encoder embedding size
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target
words to an id
    :return: Tuple of (Training Logits, Inference Logits)
    """
    rnn_inputs = tf.contrib.layers.embed_sequence(input_data,
source_vocab_size, enc_embedding_size)
    enc_cell = encoding_layer(rnn_inputs, rnn_size, num_layers,
keep_prob)

    dec_cell = process_decoding_input(target_data,
target_vocab_to_int, batch_size)
    dec_embeddings =
tf.Variable(tf.random_uniform([target_vocab_size,
dec_embedding_size]))
    dec_embed_input = tf.nn.embedding_lookup(dec_embeddings,
dec_cell)

    training_logits, inference_logits =
decoding_layer(dec_embed_input, dec_embeddings, enc_cell,
target_vocab_size, sequence_length, rnn_size,
                               num_layers, target_vocab_to_int,
keep_prob)

    return training_logits, inference_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
```

```
"""
tests.test_seq2seq_model(seq2seq_model)
```

Лістинг 1.11 – Створення графів

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
save_path = 'checkpoints/dev'
(source_int_text, target_int_text), (source_vocab_to_int,
target_vocab_to_int), _ = helper.load_preprocess()
max_source_sentence_length = max([len(sentence) for sentence in
source_int_text])

train_graph = tf.Graph()
with train_graph.as_default():
    input_data, targets, lr, keep_prob = model_inputs()
    sequence_length =
tf.placeholder_with_default(max_source_sentence_length, None,
name='sequence_length')
    input_shape = tf.shape(input_data)

    train_logits, inference_logits = seq2seq_model(
        tf.reverse(input_data, [-1]), targets, keep_prob,
batch_size, sequence_length, len(source_vocab_to_int),
len(target_vocab_to_int),
        encoding_embedding_size, decoding_embedding_size,
rnn_size, num_layers, target_vocab_to_int)

    tf.identity(inference_logits, 'logits')
    with tf.name_scope("optimization"):
        # Loss function
        cost = tf.contrib.seq2seq.sequence_loss(
            train_logits,
            targets,
            tf.ones([input_shape[0], sequence_length]))

        # Optimizer
        optimizer = tf.train.AdamOptimizer(lr)

        # Gradient Clipping
        gradients = optimizer.compute_gradients(cost)
        capped_gradients = [(tf.clip_by_value(grad, -1., 1.),
var) for grad, var in gradients if grad is not None]
        train_op = optimizer.apply_gradients(capped_gradients)
```

Лістинг 1.12 – Метод get_accuracy()

```
"""
```

```
DON'T MODIFY ANYTHING IN THIS CELL
"""
import time

def get_accuracy(target, logits):
    """
    Calculate accuracy
    """
    max_seq = max(target.shape[1], logits.shape[1])
    if max_seq - target.shape[1]:
        target = np.pad(
            target,
            [(0,0),(0,max_seq - target.shape[1])],
            'constant')
    if max_seq - logits.shape[1]:
        logits = np.pad(
            logits,
            [(0,0),(0,max_seq - logits.shape[1]), (0,0)],
            'constant')

    return np.mean(np.equal(target, np.argmax(logits, 2)))

train_source = source_int_text[batch_size:]
train_target = target_int_text[batch_size:]

valid_source =
helper.pad_sentence_batch(source_int_text[:batch_size])
valid_target =
helper.pad_sentence_batch(target_int_text[:batch_size])

with tf.Session(graph=train_graph) as sess:
    sess.run(tf.global_variables_initializer())

    for epoch_i in range(epochs):
        for batch_i, (source_batch, target_batch) in enumerate(
                helper.batch_data(train_source, train_target,
batch_size)):
            start_time = time.time()

            _, loss = sess.run(
                [train_op, cost],
                {input_data: source_batch,
                 targets: target_batch,
                 lr: learning_rate,
                 sequence_length: target_batch.shape[1],
                 keep_prob: keep_probability})

            batch_train_logits = sess.run(
                inference_logits,
                {input_data: source_batch, keep_prob: 1.0})
            batch_valid_logits = sess.run(
                inference_logits,
```

```
                {input_data: valid_source, keep_prob: 1.0})

            train_acc = get_accuracy(target_batch,
batch_train_logits)
            valid_acc = get_accuracy(np.array(valid_target),
batch_valid_logits)
            end_time = time.time()
            print('Epoch {:>3} Batch {:>4}/{} - Train Accuracy:
{:>6.3f}, Validation Accuracy: {:>6.3f}, Loss: {:>6.3f}'
                  .format(epoch_i, batch_i, len(source_int_text)
// batch_size, train_acc, valid_acc, loss))

    # Save Model
    saver = tf.train.Saver()
    saver.save(sess, save_path)
print('Model Trained and Saved')
```

Лістинг 1.13 – Метод sentence_to_seq()

```
def sentence_to_seq(sentence, vocab_to_int):
    """
    Convert a sentence to a sequence of ids
    :param sentence: String
    :param vocab_to_int: Dictionary to go from the words to an
id
    :return: List of word ids
    """

    word_id = lambda word: vocab_to_int[word] if word in
vocab_to_int else vocab_to_int['<UNK>']
    word_to_int = [word_id(word.lower()) for word in
sentence.split()]

    # print(type(x))
    # print(word_to_int)
    return word_to_int


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_sentence_to_seq(sentence_to_seq)




translate_sentence = 'he saw a old yellow truck .'
```

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
translate_sentence = sentence_to_seq(translate_sentence,
source_vocab_to_int)

loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_path + '.meta')
    loader.restore(sess, load_path)

    input_data = loaded_graph.get_tensor_by_name('input:0')
    logits = loaded_graph.get_tensor_by_name('logits:0')
    keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')

    translate_logits = sess.run(logits, {input_data:
[translate_sentence], keep_prob: 1.0})[0]

print('Input')
print('  Word Ids:      {}'.format([i for i in
translate_sentence]))
print('  English Words: {}'.format([source_int_to_vocab[i] for i
in translate_sentence]))

print('\nPrediction')
print('  Word Ids:      {}'.format([i for i in
np.argmax(translate_logits, 1)]))
print('  French Words: {}'.format([target_int_to_vocab[i] for i
in np.argmax(translate_logits, 1)]))
```