Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

_____Інститут електроенергетики_____
(інститут)
_____факультет інформаційних технологій_____
(факультет)
Кафедра_____інформаційних технологій та комп'ютерної інженерії_____
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня ___магістра__
(бакалавра, спеціаліста, магістра)

Студента_____Фурсикової Тетяни Володимирівни_____
(ПІБ)
академічної групи _____126м-22з-2_____
(шифр)
спеціальності _____126 «Інформаційні системи та технології»_____
(код і назва спеціальності)
за освітньо-професійною програмою _____

_____«Інформаційні системи та технології»_____
(офіційна назва)
на тему __Комплексна кваліфікаційна робота: Дослідження особливостей процесів_____
_____використання графів знань у системах на базі графових нейронних мереж__
_____
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Коротенко Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри

___інформаційних технологій___
_____та комп'ютерної інженерії___
(повна назва)

_____ _Гнатушенко В.В._
(підпис) (прізвище, ініціали)

«_____»_____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____магістр _____
(бакалавра, спеціаліста, магістра)

студенту _____Фурсиковій Т.В.____ **академічної групи** ___126м-22з-2_____
(прізвище та ініціали) (шифр)

спеціальності _____126 «Інформаційні системи та технології»_____ _____

за освітньою-професійною програмою _____

«Інформаційні системи та технології»_____

на тему ____Комплексна кваліфікаційна робота: Дослідження особливостей процесів___
_____використання графів знань у системах на базі графових нейронних мереж_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 09.10.2021 р. № 1228-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішення задачі	15.10.2023 – 20.10.2023
Розділ 2	Вивчення особливостей процесів використання графів знань	21.10.2023 – 30.11.2023
Розділ 3	Виконання досліджень та оформлення кваліфікаційної роботи	01.12.2023 – 14.01.2024

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____15.10.2023 р._____

Дата подання до екзаменаційної комісії ____17.01.2024 р.____

Прийнято до виконання _____ _____Фурсикова Т.В._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: стор. 80, рис. 6, додатки 4, джерела 36

Об’єкт дослідження: процес використання графів знань різних форматів під час навчання штучних нейронних мереж.

Предмет дослідження: представлення знань у вигляді графів, онтологій та штучних нейронних мереж, що здатні використовувати графи знань у якості джерела для набору даних.

Мета кваліфікаційної роботи: дослідити представлення знань в системах штучного інтелекту та процес їхнього перетворення у набори даних для машинного навчання.

Новизна отриманих результатів полягає у дослідженні способу перетворення графів знань у формат простих трійок та подальший процес навчання графової нейронної мережі з використанням графу знань у форматі трійок в якості джерела вхідних даних.

Пояснювальна записка містить опис форматів представлення знань в системах штучного інтелекту, опис процесу перетворення знань у набори даних для навчання графових нейронних мереж, опис класичної графової нейронної мережі, яка може використовувати ці дані та опис й реалізацію методів перетворення знань між форматами, знань у набір даних та відповідну графову нейронну мережу для їх обробки.

Список ключових слів: GNN, граф знань, RDF, OWL, Turtle, N-Triples, JSON-LD, RDF XML, KIF, embedding, агрегація, оновлення.

ABSTRACT

Explanatory note: pages 80, figures 6, applications 4, sources 36.

Object of research: is the process of using knowledge graphs of various formats when training artificial neural networks.

Subject of research: representation of knowledge in the form of graphs and ontologies and artificial neural networks that can use knowledge graphs as a source for a data set.

Purpose of research: to study the representation of knowledge in artificial intelligence systems and the process of transforming it into data sets for machine learning.

The novelty of the obtained results lies in the study of the method of converting knowledge graphs into the format of simple triples and the subsequent process of training a graph neural network using a knowledge graph in the format of triples as a source of input data.

The explanatory note contains a description of the formats for representing knowledge in artificial intelligence systems, a description of the process of converting knowledge into data sets for training graph neural networks, a description of a classical graph neural network that can use this data, and a description and implementation of methods for converting knowledge between formats, knowledge into a data set and, accordingly, a graph neural network for processing it.

Key words: GNN, knowledge graph, RDF, OWL, Turtle, N-Triples, JSON-LD, RDF XML, KIF, embedding, update, aggregation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – Artificial Intelligence (Штучний Інтелект)

GNN – Graph Neural Networks (графова нейронна мережа)

LSTM – Long short-term memory (Довга короткострокова пам'ять)

RDF – Resource Description Framework (середовище опису ресурса)

OWL – Web Ontology Language (мова веб-онтологій)

Turtle – Terse RDF Triple Language (мова стислих RDF трійок)

N-Triples – мова RDF триплетів

JSON-LD – JavaScript Object Notation for Linked Data (нотація JavaScript об’єктів для зв’язаних даних)

RDF – Resource Description Framework (Середовище Опису Ресурсу)

RDF XML – RDF eXtensible Markup Language (розширювана мова розмітки RDF)

KIF – Knowledge Interchange Format (формат обміну знаннями)

embedding – вбудовування

update – оновлення

aggregation – включення

node2vec – “node to vector” (алгоритм)

ШІ – Штучний Інтелект

ВСТУП

Актуальність роботи. Протягом доволі довгого часу системи штучного інтелекту розвивалися по двох напрямках, які перетиналися поміж собою тільки в окремих випадках. Системи так званого «semantic AI» спиралися на накопичення інформації у вигляді знань та створення систем обробки, що спираються на логіки різних порядків. Такі системи могли бути доволі ефективними, з одного боку, як частини систем прийняття рішень, а з іншого – як інтелектуальні помічники, проте накопиченні знання не завжди могли пояснити відхилення практичних результатів від теоретичних і тому точність прогнозу в таких системах могла б бути більш менш прийнятною у добре досліджених та описаних галузях знань.

Натомість нейронні мережі «neural AI» можуть досліджувати певний стан набору даних, проте виділити сенс того, що містить цей набір без додаткових інструментів штучного інтелекту, нейронна мережа не в змозі. Таким чином задача поєднання цих двох напрямів штучного інтелекту є актуальною, тим паче, що таке поєднання для різних предметних галузей та, навіть, різних задач в одній галузі, може відрізнятися.

Об’єктом досліджень є процес використання графів знань різних форматів під час навчання штучних нейронних мереж.

Предметом досліджень є представлення знань у вигляді графів, онтологій та штучних нейронних мереж, що здатні використовувати графи знань у якості джерела використовуваних наборів даних.

Мета роботи – дослідити представлення знань в системах штучного інтелекту та процес їхнього перетворення у набори даних для машинного навчання.

Пояснювальна записка містить опис форматів представлення знань в системах штучного інтелекту, опис процесу перетворення знань у набори даних для навчання графових нейронних мереж, опис класичної графової нейронної мережі, яка може використовувати ці дані та опис й реалізацію

методів перетворення знань між форматами, знань у набір даних та відповідну графову нейронну мережу для їх обробки.

Постановка задачі

Дослідити способи та формати представлення знань у системах штучного інтелекту. Дослідити процес перетворення графів знань на набори даних для навчання штучних нейронних мереж, дослідити процес навчання графової нейронної мережі з використанням графів знань та за результатами аналізу граматик графів знань дослідити спосіб перетворення графів знань різних форматів на формат простих трійок N-Triples з подальшим його перетворенням у векторний вигляд для навчання графової нейронної мережі GNN.

Висновок. У даній роботі будуть досліджені способи та формати у системах штучного інтелекту, спосіб перетворення графів знань формату N-Triples у векторний вигляд та подальший процес навчання та використання графової нейронної мережі, яка у якості джерела вхідних даних буде використовувати знання, уявлені у векторному вигляді. Будуть розроблені алгоритми конвертації різних форматів графів знань у формат N-Triples, спосіб векторизації графу знань N-Triples та за результатами цього буде навчена та випробувана нейронна мережа GNN.

РОЗДІЛ 1. АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1.1. Знання у реальному світі

Побудова сучасних систем штучного інтелекту базується на основній ідеї представлення знань, що передбачає виокремлення, опис та організацію знань предметної області системи в змістовний та структурований спосіб. Отримані знання є основою процесів навчання, міркування та прийняття рішень у системах штучного інтелекту.

Розуміння терміну «знання» вимагає певного розуміння терміну «інформація» та його зв'язком зі знанням. Деякі автори вважають, що інформація відрізняється від знання, тоді як інші вважають, що між ними мало відмінностей і вони є подібними. Крім того, ці два поняття використовуються в літературі як взаємозамінні. Концепція знань та інформації формується основними визначеннями цих двох термінів.

На практиці існує багато однаково правдоподібних визначень термінів знання та інформації. Термін «знання» відповідно до словника Вебстера, наприклад, визначає «факт чи стан володіння відомістю про щось, яку отримано через досвід або асоціації» [1]. Девенпорт і Прусак натомість визначають знання як «суміш плинного досвіду, цінностей, контекстної інформації та інтуїції, яка забезпечує структуру для оцінки та включення необробленого досвіду та інформації» [2]. Поширеним також є використання терміну знання у контексті ідеї або розуміння, якими володіє суб'єкт та які використовуються у виборі ефективної дії для досягнення мети суб'єкта.

Терра та Анджелоні відповідно визначають інформацію як інтерпретовані, впорядковані дані, що наділені релевантністю та метою, додаючи, що визначення знань набагато складніше, ніж визначення інформації. Вони вважали, що при визначенні знання буде доречно посилатися на корінь слова «епітема», що означає абсолютну істину [3]. З іншого боку Віг (1999) визначає інформацію як факти та дані, організовані для характеристики конкретної ситуації, а знання – як набір істин і вірувань, перспектив і концепцій, суджень і очікувань [4]. Це схоже на визначення знання, надане

Нонаки та Такеучі (1995), яке говорить: «Знання – це істинне й виправдане переконання» [5].

Узагальнюючи концепції знань та інформації, Ківумбі (2011) виділив наступне [6]:

- Інформація – це оброблені дані, тоді як знання – це інформація, яка моделюється, щоб бути корисною;
- Для можливості отримання знань у першу чергу потрібна інформація;
- Інформація має справу зі способом зв'язку даних, тоді як знання досліджує закономірності в межах даного набору інформації.

Усі наведені вище визначення дають змогу зробити висновок, що коли особа володіє обробленими даними, які також називають інформацією про якусь сутність, вважається, що ця особа володіє знаннями про цю сутність. Отже, знання можна визначити як володіння або передачу різними засобами (запам'ятовування, мистецтво, навички тощо) інформації або оброблених даних таким чином, щоб вони були корисними.

Таке визначення є основою концепції ієрархії, що має назву «піраміда знань», яка лежить в основі інформаційної науки та управління. За словами Фріке (2009), багато теоретиків з інформатики, інформаційних систем управління та бібліотечної справи, які зробили внесок у дискусію про управління знаннями, бачать інформацію в термінах «ієрархії» або «піраміди»: дані-інформація-знання-мудрість (DIKW) рисунок 1.1 [7].

Рисунок 1.1. Ієрархія DIKW або піраміда знань

Уся ця піраміда побудована на основі даних (здебільшого доступних для спостереження та вимірювання), які після обробки, тобто надання певного значення, стають інформацією. Це означає, що інформація виводиться з даних. Таким чином, інформаційні системи генерують, зберігають, витягують і обробляють дані. Обробка може бути статистичною або арифметичною (Ackoff, 1989) [8]. Згідно цієї піраміди інформація розглядається як продукт даних або як дані, які були доповнені чи уточнені, а знання відповідно розглядаються як продукт обробки інформації.

Розуміння визначення та концепції знань є важливим при побудові систем штучного інтелекту. Всі сучасні системи штучного інтелекту мають бази знань, що відображають знання системи про її домен, і задача полягає в тому як правильно здобути, інтерпретувати та зберегти знання для достовірного відображення за допомогою комп’ютера.

1.2. Представлення знань у системах штучного інтелекту

Представлення знань є ключовою концепцією в галузі штучного інтелекту (ШІ), яка передбачає створення моделей і структур для представлення інформації та знань для використання інтелектуальними системами [9]. Важливою особливістю цієї концепції є здатність створення таких моделей та структур, які дозволяють інтелектуальним системам не лише засвоювати інформацію, але й розуміти її контекст та використовувати для прийняття рішень. Це означає, що представлення знань в ШІ повинно бути не лише ефективним у зберіганні фактів, а й в подальшому використанні цих фактів для виявлення взаємозв'язків та вираження високорівневих концепцій.

Ключовою метою представлення знань є створення таких моделей, які дозволяють системам штучного інтелекту адаптуватися до нового контексту, вирішувати складні завдання та робити розумні висновки на основі зібраних даних. Такий підхід дозволяє машинам «мислити» та «вчитися», моделюючи процеси розумової діяльності подібні до людських [10].

Для досягнення цієї мети у галузі представлення знань та міркування визначені основні типи знань у штучному інтелекті:

- декларативні знання - описовий тип знань, які виражаються у вигляді фактів, правил чи припущень;
- процедурні знання - використовуються для виконання конкретних завдань або дій, виражаються у вигляді правил, стратегії, процедур, тощо і часто надаються за допомогою алгоритмів або мов програмування;
- метазнання - це знання про знання, які часто використовуються у процесі міркування і для підвищення продуктивності систем штучного інтелекту;
- евристичні знання - це тип знань, який відноситься до емпіричних правил або стратегій, які використовуються для швидкого та ефективного вирішення проблем та може бути виражений у вигляді експертних знань з предметної галузі;
- структурні знання - це знання про структуру проблеми або системи, що використовуються для декомпозиції складних проблем на простіші завдання та виражаються у описі відносини між такими поняттями, як вид, частина та угруповання чогось.

Використання різних типів знань у системах штучного інтелекту зазвичай відбувається за допомогою моделей представлення знань. Ці моделі включають в себе різні методи і підходи до організації інформації та знань, а також надають системам штучного інтелекту структурований та систематизований спосіб управління інформацією. Основні типи моделей представлення знань включають: логічні представлення, семантичні мережі, фрейми, сценарії, продукційні правила та онтології.

1.2.1. Логічні представлення

Логічне представлення в штучному інтелекті визначається як мова з чіткими правилами, яка працює з чітко вираженими твердженнями без двозначності в поданні інформації. Цей підхід включає точно визначений синтаксис та семантику, які гарантують правильність логічних висновків та виводів системи. Існує декілька видів логічного представлення: пропозиційна логіка та логіка першого порядку.

Пропозиційна логіка, відома також як логіка висловлювань, представляє собою формальну систему логіки, що досліджує відносини між пропозиціями. Цей вид логіки базується на булевій системі, де пропозиції оцінюються як істинні або хибні. У логіці висловлювань пропозиції об'єднуються за допомогою логічних зв'язників, таких як «і», «або» і «ні». Складені висловлювання, отримані таким чином, також можна оцінити як істинні чи хибні, враховуючи істинність їх складників.

Логікою першого порядку у свою чергу називають розширення пропозиційної логіки, що дозволяє представляти складніші зв’язки між об’єктами. У даному виді логіці пропозиції будуються за допомогою предикатів, які є висловлюваннями, що описують властивості або відношення між об’єктами, і кванторів, які визначають область дії змінних у пропозиції.

Для представлення знань за допомогою мови логіки використовуються символи та оператори для представлення таких понять, як істина, заперечення, кон’юнкція, диз’юнкція, імплікація, квантифікація та тотожність. Семантика логічного представлення передбачає надання значення цим символам і формулам. Це робиться шляхом визначення набору аксіом і правил для маніпулювання цими символами.

Перевагою використання логічного представлення знань у системах штучного інтелекту є насамперед його простота та зручність у моделюванні логічних міркувань чи представлені простих понять у системі. Але дане

представлення має обмежену виразність при описі фактів та взаємозв’язків між ними, які складно інтерпретувати засобами математичної логіки.

1.2.2. Семантичні мережі

Семантичною мережею називають графічне представлення знань, де вузли представляють поняття або об’єкти, а дуги – зв’язки між ними. Синтаксис семантичної мережі складається з вузлів і посилань, а семантика передбачає визначення значення кожного вузла та посилання (рисунок 1.2).

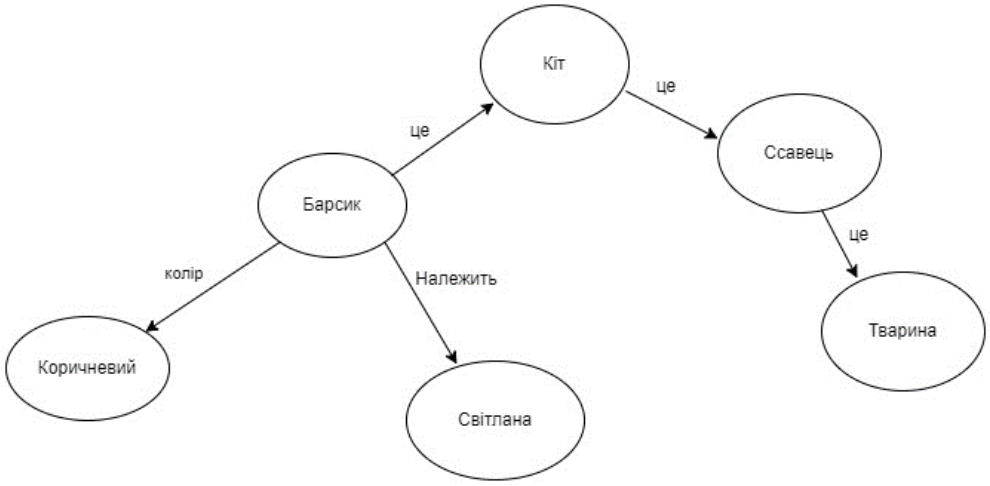

Рисунок 1.2. Приклад семантичної мережі

Однією з головних переваг семантичних мереж є візуальне представлення, що робить їх більш інтуїтивно зрозумілими, ніж логічні представлення. Крім того семантичні мережі призначені для моделювання семантики, що дозволяє більш природно виражати значення та взаємозв'язки між об'єктами.

Втім, існують певні недоліки, пов'язані із семантичним методом представлення. Наприклад, створення складних семантичних мереж може бути трудомістким завданням, особливо якщо система повинна моделювати великі області знань. З цієї проблеми також виходить проблема коштовності

великих семантичних мереж з точки зору обчислень при обробці та пошуку знань.

1.2.3. Фрейми

Фрейм – це структурований запис, який описує сутність у світі за допомогою набору атрибутів та їхніх відповідних значень. У штучному інтелекті фрейми служать структурою даних, яка розділяє знання на підструктури, представляючи типові ситуації. Він складається з набору слотів і значень слотів. Ці слоти можуть бути будь-якого типу та розміру. Слоти мають імена та значення, які називаються фасетами (рисунок 1.3.).

Слоти	Фільтри
Назва	Штучний інтелект
Жанр	Комп'ютерна наука
Автор	Пітер Норвіг
Редагування	Третє видання
рік	1996
Сторінка	1152

Рисунок 1.3. Представлення даних про книгу у вигляді фрейму

Фрейми з’явилися через розвиток семантичних мереж та слугували прототипом для класичних класів та об’єктів об'єктно орієнтованого підходу до створення структур даних. Система фреймів складається з набору з’єднаних кадрів. У кадрі знання про об’єкт або подію можуть зберігатися разом у базі знань. Технологія фреймів широко використовується для

представлення знань у різних програмах, включаючи обробку природної мови або машинне бачення.

Однією з переваг фреймового представлення є структурування інформації у вигляді концепційних блоків з атрибутами та відносинами між ними. Це сприяє легшій організації та розумінню знань. Також фрейми можуть мати загальні атрибути або властивості, які можуть успадковуватися, що спрощує моделювання схожих об'єктів чи концепцій.

З іншого боку при розробці великих систем фреймове представлення може стати складним через потребу управління багатьма фреймами та їх відносинами, що ускладнює розуміння системи. Крім того, узагальнений характер представлення кадру означає, що воно не завжди може найкраще підходити для представлення більш конкретних або складних ситуацій.

1.2.4. Сценарії

Сценарієм у ШІ називають структуроване представлення, що описує стереотипну послідовність подій у певному контексті. Вони використовують структуру, що нагадує фрейми, для упорядкування та систематизації інформації. Кожен сценарій може містити різні елементи, які включають в себе наступні компоненти:

- умова входу - це базова умова, яка повинна бути виконана до того, як події в сценарії можуть відбутися;
- результати - умова, яка буде істинною після того, як відбулися події в сценарії;
- реквізит - об’єкти, що беруть участь у подіях;
- ролі - це дії, які виконують окремі об’єкти;
- сцени - послідовність подій, що відбуваються;
- трек - варіації сцен за сценарієм. Різні треки можуть використовувати компоненти одних і тих самих сценаріїв;

За допомогою цих компонентів формується представлення конкретних ситуацій або процесів у інтелектуальних системах. Це дозволяє системам розуміти та використовувати стандартні шаблони дій у відповідних ситуаціях.

Сценарії можуть використовуватися, наприклад, у системах розуміння природної мови для організації бази знань, адже за допомогою них можна легко структурувати та узгоджувати незалежні окремі спостереження з контексту предметної області, такі як правила вимови, правопис, тощо. Також вони дозволяють системам штучного інтелекту моделювати та реагувати на типові чи стандартні ситуації, що спрощує розв'язання завдань у відповідних контекстах.

Незважаючи на переваги, сценарії в представленні знань мають свої недоліки, серед який можна виділити обмеженість їх використання у нестандартних або складних ситуаціях, через неможливість врахувати всі можливі варіанти або взаємозв'язки між подіями. Також для опису широкого спектру сценаріїв може знадобитися значна кількість даних, що може призвести до складнощів у керуванні обсягом та зберіганням інформації.

1.2.5. Продукційні правила

Продукційні правила - це основний компонент у представленні знань експертних систем, який визначає поведінку системи на основі умов та відповідних дій. Вони складаються з двох частин: умови (або «if») та дії (або «then»). Продукційні правила використовуються для створення систем, які можуть робити прийняття рішень на основі вхідних даних, порівнюючи їх з встановленими умовами та застосовуючи відповідні дії в залежності від виконання цих умов.

Представлення знань системи у вигляді продукційних правил надає переваги при редагуванні бази знань, адже виправлення або додавання нових продукційних правил не вимагає значних змін у системі, а також надає змогу швидко реагувати на вхідні дані при прийнятті рішень. Основним недоліком

систем, що використовують продукційні правила є відсутність можливостей до самостійного навчання, оскільки такі системи не передбачають зберігання результатів вирішення проблеми для подальшого використання та модифікації власних знань.

1.2.6. Онтології

Сучасним підходом, який поєднує у собі попередні моделі представлення знань, є представлення знань систем штучного інтелекту у вигляді онтологій. Онтології - це формальні моделі, які описують сутності концепцій та їх взаємозв'язки в конкретній області або домені.

Онтології подібно до семантичних мереж дозволяють представити знання у вигляді структурованих графів, де концепції або терміни представлені у вигляді вузлів, а їх відносини - у вигляді зв'язків між цими вузлами з можливістю додавання нових знань про предметну область. Основні елементи, що використовуються для побудови онтології включають у себе:

- класи - використовуються для представлення концептів у предметній області;
- екземпляри - об’єкти, що представляють собою певний клас.
- слоти - описують атрибути об’єктів;
- фасети - використовуються для опису обмежень слота;
- відношення - описують зв’язок між класом та екземпляром;
- правила - вирази типу «умова-дія», які вказують, які висновки можна зробити на основі наявної інформації або фактів;
- аксіоми - твердження, що містять загальну теорію предметної області;

Загалом побудова онтологій, включає в себе використання різних моделей представлення знань для конкретизації та організації знань в певній області. Для репрезентації об'єктів та їх характеристик у вигляді структурованих блоків з властивостями використовуються фрейми. Семантичні мережі використовуються для визначення відносин між об'єктами

або концептами у вигляді графів, а використання продукційних правил дозволяє описувати те, які дії чи висновки можна зробити на основі наявних фактів чи умов.

Таким чином гнучкість та чітке визначення термінів, концепцій та їх взаємозв'язків з можливістю створення нових знань, яку надають онтології зумовило зростання попиту на їх використання у сучасних системах штучного інтелекту як спосіб представлення знань системи про світ або його окремі аспекти. За допомогою онтологій можна створювати інтелектуальні системи, які здатні ефективно аналізувати, робити висновки та вирішувати складні проблеми, орієнтуючись на чітко визначені концепції та взаємозв'язки між ними, а для створення та управління безпосередньо самими онтологіями у зручний спосіб можна використовувати спеціальні мови онтологій або мови графів знань.

1.3. Мови графів знань

Онтології часто відображають мережу об'єктів, подій або концепцій у реальному світі та їх взаємозв'язки. Ця інформація може зберігатися в графових базах даних та відображатися у формі графових структур для зручності аналізу та використання. Для побудови онтологій були розроблені спеціальні формальні мови, які допомагають у кодуванні та представленні знань про конкретні предметні області. Ці мови забезпечують стандартизований спосіб опису об’єктів та їх взаємозв’язків у вигляді формальних структур.

Мови онтологій можна розділити на дві основні категорії: традиційні мови онтологій та веб-стандарти. До традиційних мов онтологій відносяться такі мови, як KIF, які спрямовані на формалізацію знань і відносин між ними у різних областях та забезпечують стандартизовані методи опису концепцій, відносин та атрибутів для побудови онтологій.

Мови веб-стандартів у свою чергу включають у себе мови розмітки, такі як XML (eXtensible Markup Language), і спеціалізовані мови, такі як JSON-LD (JSON for Linked Data) або Turtle, що використовуються для представлення та обміну онтологіями через мережу Інтернет. Ці стандарти дозволяють зручно та ефективно обмінюватись знаннями між різними системами та ресурсами у веб-середовищі.

Окрім вище зазначених категорій, також існують і гібридні мови онтологій. Вони поєднують у собі переваги традиційних мов для створення структурованих знань з ефективністю та доступністю веб-стандартів для обміну цими знаннями у мережі. Типовим прикладом гібридної мови онтологій є мова OWL (Web Ontology Language), яка поєднує у собі класичні методи опису онтології та використовується як веб-стандарт для обміну. Серед різних мов онтологій найбільш поширеними є KIF, RDF, OWL, DAML.

1.3.1. KIF

KIF (Knowledge Interchange Format) - це формальна мова, спроектована для обміну знаннями між різними системами. Вона базується на семантиці логіки предикатів та має синтаксичні елементи LISP. KIF дозволяє представляти різноманітні утвердження у логіці предикатів першого порядку. Ця мова була розроблена у рамках проекту Ontolingua, що мав на меті створення кооперативного конструктора онтологій.

У мові обміну знаннями (KIF), дані представляються у вигляді логічних виразів, що використовують логіку предикатів першого порядку. Вони описують твердження за допомогою різних логічних конструкцій та символів. Основні елементи представлення даних у KIF включають предикати, змінні, квантори, логічні операції та функції. Ці конструкції утворюють логічні вирази, що дозволяють виразно виражати різноманітні типи даних та відносини між ними.

Концепція KIF розроблена для обміну та представлення знань між комп'ютерними системами, а основний акцент робиться на формалізації логічних висловлювань та знань. Це обумовлює основні переваги мови KIF серед яких можна виділити наступні:

- формальність та логічна точність - KIF базується на математичних принципах логіки предикатів першого порядку, що дозволяє точно формалізувати знання та логічні висловлювання;
- семантика мови дозволяє розуміти значення виразів без необхідності використання інтерпретатора або конкретних операцій з цими виразами;
- Можливість представлення метаданих, що дозволяє включати нові структури знань без необхідності модифікувати саму мову.

Хоча KIF є потужним інструментом для формалізації та обміну знаннями, його використання може бути важким через складність перекладу людського знання у формальні логічні висловлювання.

1.3.2. OWL

OWL є стандартною мовою онтологій для семантичних мереж, що розширює мовну схему RDF. З моменту появи семантичних мереж початкові спроби їх опису були зосереджені на RDF і схемі RDF; однак невдовзі було виявлено, що ці мови мають занадто обмежену виразну силу. Для вирішення цієї проблеми було створена мова веб-онтології OWL, яка стала рекомендацією W3C у лютому 2004 року.

Для опису онтологій у мові OWL використовуються класи і підкласи, екземпляри яких називаються індивідами. Індивіди, які є членами будь-якого класу, називаються його розширенням [36]. Клас в OWL – це класифікація індивідів на групи, які мають спільні характеристики. Для відображення класифікації об’єктів класи утворюють ієрархію, яку також називають таксономією. Індивіди класу мають властивості, які можуть бути двох типів:

- властивості об’єкта (ObjectProperty) пов’язують окремих індивідів (примірники) двох класів;
- властивості типу даних (DatatypeProperty) пов’язують окремі індивіди класів з літеральними значеннями;

Мова OWL сумісна з попередніми мовами онтологій, такими як SHOE, DAML+OIL, що дозволяє легко переносити або розширювати наявні онтології новими можливостями без втрати сумісності з іншими системами. Ці можливості дозволяють виконувати логічні висновки та отримувати більше знань про зв'язки між об'єктами в онтології, що сприяє більш точній, ефективній та зрозумілій формалізації знань та структуруванню онтологій.

Головним недоліком мови OWL можна вважати складність семантики, що вимагає у користувачів досвіду у інтерпретації знань у формальні мови.

1.3.3. DAML та OIL

Ініціатива DARPA Agent Markup Language (DAML) була створена в рамках підтримки уряду США з метою розробки спеціалізованої мови та інструментів для полегшення створення та управління семантичними мережами. DAML заснована на попередньому проекті SHOE та складається безпосередньо з мови онтології , яка визначає терміни та їх взаємозв'язки, і мови OIL (онтологічний рівень висновків), яка використовується для вираження обмежень та правил для здійснення логічних висновків. Основним нововведенням DAML було використання RDF і XML як основи, а також використання об'єктно-орієнтованого підходу, при якому структура даних визначається за допомогою класів і властивостей, що спрощує моделювання складних систем та відносин між об'єктами.

Головною перевагою цієї мови є наявність механізмів для представлення послуг, процесів та бізнес-моделей, що робить її корисною для опису складних бізнес-структур. Крім того DAML+OIL має можливості для розширення та

розвитку онтологій, дозволяючи вводити нові конструкції та правила без зміни основної мови.

1.3.4. RDF

Структура опису ресурсу (RDF) – це загальна структура для представлення взаємопов'язаних даних в Інтернеті, яку використовують найпоширеніші мови онтологій. Інструкції RDF використовуються для опису та обміну метаданими, що забезпечує стандартизований обмін даними на основі зв'язків. RDF використовується для опису ресурсів шляхом визначення їх властивостей та відносин між ними у вигляді триплетів, що складаються з суб'єкту, предикату та об'єкту (рисунок 1.4).

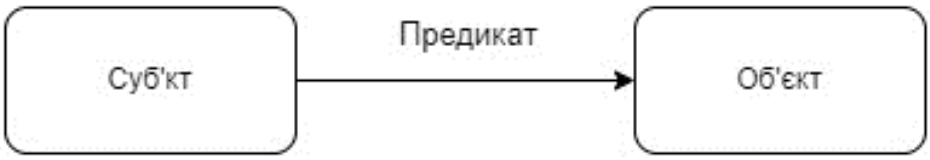

Рисунок 1.4. Представлення триплету у RDF

Суб'єкт і об'єкт є вузлами, які представляють сутності предметної області. Предикат представляє відношення між вузлами. Стандарт RDF передбачає три різні типи вузлів:

- Уніфікований ідентифікатор ресурсу (URI) – це стандартизований формат для ідентифікації ресурсу, абстрактного чи фізичного. Окремими випадками URI, що використовуються у RDF, також є уніфікований покажчик ресурсу (URL) та інтернаціоналізований ідентифікатор ресурсу (IRI).
- Літерал – це певне значення даних, яке може бути представлене у вигляді рядка, дати або числового значення. Літеральні значення виражаються за допомогою формату URI або IRI.

- Порожній ідентифікатор вузла також відомий як *анонімний ресурс* або *bnode* . Він представляє сутність, про яку нічого не відомо, крім її відносин з іншими сутностями у мережі.

Завдяки відкритим специфікаціям і широкому застосуванню RDF став найпоширенішим стандартом для публікації семантичних веб-даних і збирання семантичної інформації. RDF можна включати у веб-сторінки через мову гіпертекстової розмітки, або зберігати у окремих файлах, які потім можуть бути пов’язані з веб-контентом. У перших версіях оператори RDF були включені в документи XML, що було основним синтаксичним форматом. Проте з часом стали розвиватися інші синтаксичні формати. Наразі стандарти для кодування операторів RDF включають кілька різних синтаксичних варіантів:

- **Turtle** є найпопулярнішим текстовим синтаксисом для операторів RDF. W3C описує це як «компактну та природну текстову форму», яка включає скорочення для загальновживаних шаблонів.
- **JSON-LD** використовує синтаксис JSON для операторів RDF.
- **N-Triples** – це підмножина синтаксису Turtle, розроблена як простіший текстовий формат для операторів RDF для покращеної простоти використання.

До основних переваг стандарту RDF можна віднести:

- Стандартизований синтаксис RDF полегшує розробку програм, які використовують метадані для роботи з даними.
- На відміну від реляційних баз даних, дані RDF можуть надати більше інформації про сутності та їх зв'язки з іншими сутностями
- Пошукові системи можуть отримувати більш точні результати, використовуючи метадані, ніж при роботі зі збором повного тексту.

Але також використання RDF може мати деякі обмеження:

- Вибір найбільш відповідного синтаксичного формату для операцій RDF може потребувати досвіду. Різні формати можуть бути більш зручними для конкретних реалізацій.

Таким чином мови онтологій або мови графів знань надають широкий інструментарій для представлення знань, серед якого найпоширенішим стандартом є RDF, який забезпечує гнучкість у вираженні семантичної інформації та використовується для побудови семантичних веб-даних а також надає кілька форматів для представлення знань, що дає можливість вибирати оптимальний формат для конкретних потреб.

1.4. Формати представлення знань

Різні формати відрізняються в своїй зручності для читання, обробки та використання даних. Тут важливо розглянути не лише структуру та компактність, а й гнучкість у використанні, адже кожен формат має свої особливості. Наприклад, Turtle часто вибирають за легкістю читання та зручністю для людей, які аналізують дані вручну. З іншого боку, N-Triples може бути корисним у випадках, коли необхідно здійснювати обмін даними у найпростішому форматі. JSON-LD, зокрема через свій формат JSON, може бути популярним через широку підтримку у розробці веб-застосунків.

1.4.1. Turtle

Формат Turtle (Terse RDF Triple Language) є зручним та компактним способом представлення даних у вигляді тверджень (триплетів) у мові RDF. Основна його особливість - простий і зрозумілий для людини синтаксис, який легко читається та редагується.

Найпростіший триплет у мові Turtle– це послідовність термінів (суб’єкт, предикат, об’єкт), розділених пробілами та закінчених символом « .» після кожної трійки.

<http://example.org/#spiderman>
<http://www.perceive.net/schemas/relationship/enemyOf>
<http://example.org/#green-goblin>.

Для скорочення довгих ідентифікаторів, таких як URIs (Uniform Resource Identifiers) або IRI (Internationalized Resource Identifiers) у мові Turtle використовуються префікси. Це дозволяє замінити довгий ідентифікатор ресурсів коротшим іменем, з яким пов'язано певний простір імен.

@prefix ex: <http://example.org/#> .

@prefix rel: <http://www.perceive.net/schemas/relationship/> .

ex:spiderman rel:enemyOf ex:green-goblin .

Звичайні літерали у Turtle подаються в різних форматах, таких як рядки чи числа, і можуть мати мовний тег (якщо це рядок певної мови) або вказівку на тип даних. Символи, які можуть виникати у літералах, повинні бути коректно екрановані або оброблені у відповідних послідовностях для представлення даних [33]. Наприклад, у Turtle може бути представлення літерала у вигляді рядка з вказівкою на мову:

ex:example ex:label "Привіт, світ"@uk .

Цей триплет представляє рядковий літерал "Привіт, світ" українською мовою. Такі конструкції дозволяють зберігати та передавати дані з різними вимогами щодо мови та типів даних у форматі RDF.

1.4.2. N-Triples

N-Triples – формат для зберігання та передачі даних. Це рядковий формат серіалізації звичайного тексту для графів RDF, який є підмножиною формату Turtle. N-Triples не слід плутати з Notation3, який є надмножиною Turtle. N-Triples був розроблений як простіший формат, ніж Notation3 і Turtle, який легше аналізувати та генерувати. Однак, оскільки в ньому відсутні деякі ярлики, що надаються іншими RDF-серіалізаціями, введення вручну та читання великих обсягів даних може викликати труднощі. Простота N-Triples робить його тривіальними для аналізу/серіалізації, що обумовлює наявність

великої кількості бібліотек. Кожна трійка містить суб'єкт, присвоєну цьому суб'єкту властивість та значення цієї властивості.

<http://example.org/person123> <http://xmlns.com/foaf/0.1/name> "John Doe"

У цьому прикладі трійка показує, що суб'єкт має властивість name зі значенням "John Doe". З прикладу можна бачити головний недолік цього формату - відсутність префіксів і скорочень робить його довгим і дещо важким для читання.

1.4.3. JSON-LD

JSON-LD (JSON for Linked Data) - це розширення стандарту JSON, призначене для використання даних формату JSON з можливістю додавання контексту для перетворення їх в RDF. JSON-LD використовує контекст (@context) для визначення семантики даних. Це може включати посилання на класи та властивості, які забезпечують зв'язок зі схемами, онтологіями або іншими джерелами.

```
{
  "@context": {
    "dbpedia": "http://dbpedia.org/resource/",
    "schema": "http://schema.org/"
  },
  "@id": "https://www.w3.org/People/Berners-Lee/",
  "schema:birthDate": "1955-06-08",
  "schema:birthPlace": {
```

```
    "@id": "dbpedia:London"
  }
}
```

Синтаксис JSON-LD базується на стандарті JSON, але має деякі додаткові конструкції для роботи з семантичними даними. Окрім стандартних пар ключ-значення, а також операторів їх групування у списки, додатковим синтаксисом JSON-LD є:

- Контекст: Основна властивість JSON-LD - @context. Вона використовується для визначення контексту документа, у якому зазначаються терміни, простори імен, або відображення ключів на URL-адреси.
- Ланцюжки URI (IRI): JSON-LD дозволяє використовувати URL-адреси для ідентифікації ресурсів (IRI).
- Псевдоніми (префікси): Щоб скоротити IRI, JSON-LD, як і Turtle, дозволяє використовувати псевдоніми за допомогою @prefix.
- Директиви: Крім @context, JSON-LD також використовує деякі додаткові директиви, такі як @type, @id та інші, які використовуються для визначення типів ресурсів та їх ідентифікаторів.

Головною перевагою цього формату є сумісність зі стандартом JSON, що полегшує використання для розробників, звиклих до роботи з цим форматом. З іншого боку аналіз JSON-LD може бути складним і вимагати великої кількості додаткових операцій. Розбір JSON-LD для отримання даних в форматі RDF може бути трудомістким, особливо при обробці великих обсягів даних, що може вимагати розробки додаткового функціоналу кешування для збереження продуктивності[34].

Таким чином різні формати знань мають свої унікальні переваги для використання у різноманітних сценаріях. Для забезпечення сумісності та

узгодженості даних між різними системами чи платформами, які використовують різні формати, стандарт RDF включає підтримку можливості конвертування, дозволяючи перетворювати дані з одного формату в інший без втрати їх семантики чи структури.

1.5. Перетворення форматів графів знань

Як було зазначено вище перетворення форматів графів знань відіграє ключову роль у контексті інтеграції систем та обміну даними між ними. Воно дозволяє забезпечити сумісність між різними системами а також можливість об'єднувати дані з різних джерел. Це особливо корисно у випадках, коли важлива інформація знаходиться в різних системах з різними форматами даних.

Перетворення форматів графів знань (наприклад, перетворення між форматами RDF/XML, Turtle, JSON-LD і т. д.) може здійснюватися за допомогою різних інструментів та методів. Серед основних інструментів можна виділити наступні:

- ETL Інструменти (Extract, Transform, Load): Популярні ETL-інструменти, такі як Talend, Apache Nifi, Informatica, Pentaho, можуть обробляти великі обсяги даних, включаючи графи знань, і забезпечують можливість їх перетворення між форматами.
- Онтологічні редактори та платформи: Програми для роботи з онтологіями, такі як Protege, TopBraid Composer, GraphDB, Grakn, розроблені з урахуванням можливості конвертації між різними форматами онтологій.
- API та бібліотеки для програмування: Багато мов програмування мають бібліотеки для обробки графів знань та їх конвертації. RDFLib для Python, Jena для Java, RDF4J (раніше Sesame) - це лише деякі з бібліотек, які надають можливості роботи з різними форматами графів знань.

- SPARQL Запити: SPARQL запити можуть використовуватися для витягування даних з одного формату та перенесення їх у інший, використовуючи операції зчитування та запису.
- Конвертери онтологій: Інструменти спеціально розроблені для конвертації між різними мовами онтологій. Вони враховують специфіку кожного формату та забезпечують можливість перетворення без втрати інформації.

Під час процесу конвертації форматів графів знань виникає ризик втрати контекстуальної інформації. Це може відбуватися через неповне або неправильне врахування особливостей форматів, що може вплинути на точність та коректність відтворення даних. Розбіжності у структурі форматів, а також у їх синтаксису також може вплинути на коректність та повноту конвертації даних [35]. Різні формати графів знань також можуть мати відмінну семантику, що в свою чергу може вплинути на правильність інтерпретації значення даних при перетворенні.

Незважаючи на ризики використання інструментів перетворення форматів, вони надають зручний інструментарій перетворення знань у формати, що придатні для аналізу та моделювання та можуть використовуватися для навчання моделей штучного інтелекту.

1.6. Постановка задачі

Метою кваліфікаційної роботи є дослідження використання наявних графів знань під час навчання систем штучного інтелекту. Для досягнення поставленої мети необхідно:

- проаналізувати наявні формати представлення знань у системах ШІ;
- вибрати формат придатний для навчання системи штучного інтелекту;
- розробити конвертер онтологій, який перетворює онтологію з популярних вхідних форматів у обраний базовий формат;

- використати обраний формат під час навчання графової нейронної мережі прямого розповсюдження задля перевірки придатності використання графів знань у машинному навчанні.

1.7. Висновок до розділу

У цьому розділі були проаналізовані основні мови та формати представлення знань у системах ШІ. Аналіз показав широке використання онтологій, які визначають велике різноманіття форматів даних. Наявність великої кількості цих форматів може викликати ускладнення під час навчання систем штучного інтелекту. Це ставить питання розробки уніфікованого формату представлення знань для використання у системах Ші, заснованих на графових нейронних мережах.

РОЗДІЛ 2. ВИВЧЕННЯ ОСОБЛИВОСТЕЙ ПРОЦЕСІВ ВИКОРИСТАННЯ ГРАФІВ ЗНАНЬ

2.1. Графи знань як сировина для навчання нейронних мереж

В Розділі 1 було детально розглянуто різні формати графів знань, проте розуміння, як стандартизувати вигляд графів знань задля навчання систем штучного інтелекту недостатньо. По-перше, потенційна трансформація графів знань з одного формату в інший може викликати часткове спотворення сенсу через можливу наявність у вихідних графах неповністю сформованих вузлів. Але ця проблема не має формального розв'язку, так само, як і перевірка достовірності наявної у графах інформації. Але залишається питання: що треба зробити, аби граф знань міг бути використаний під час навчання ШІ? Це питання породжує купу інших питань, пов'язаних з тим, чи всі системи ШІ можуть використовувати графи знань у якості вхідних даних, які саме задачі здатні розв'язувати такі системи ШІ, у який вигляд треба перетворити вхідний граф знань, аби нейронна мережа могла його використовувати в якості вхідних даних, чи потрібні додаткові шари нейронної мережі в такому випадку тощо.

За час існування нейронних мереж як різновиду систем ШІ було вигадано доволі багато абстракцій та цілком реальних архітектур на їх базі, які описують нейронні мережі для обробки такого штибу інформації. Зокрема, це графові нейронні мережі прямого розповсюдження, графові автоенкодери, графові мережі машинної уваги тощо. Звісно, ці графові нейронні мережі здатні розв'язувати різні задачі, але варто пам'ятати, що знання подані у вигляді графів, мають певні особливості під час їхнього використання у якості вхідних даних.

По-перше, на відміну від інших різновидів вхідних даних таких як зображення чи звук, графи знань містять інформацію про зв'язки між вузлами графа. Ступінь щільності цих зв'язків у графах знань зазвичай не зазначається. Однак, якщо враховувати цей фактор, тобто фактично моделювати зважений граф, виникає купа проблем пов'язаних з порядком подання інформації у

такому графі знань зі способом перетворення вмісту графу знань у векторний формат тощо. Крім того, на відміну від, наприклад, зображень, ступінь вузлів графа не обмежується значенням 8. У повнозв'язному графі наприклад, така ступінь дорівнює n-1, де n - кількість вузлів. І це створює велику проблему під час керованого навчання методом зворотнього розповсюдження помилки. Бо зміна одного вузла в такому випадку має якимось чином відбитися на всіх зв'язаних з ним інших вузлах.

Надалі у цій роботі буде представлено спосіб перетворення графа знань на вхідні дані для навчання нейронної мережі, а також базова архітектура класичної графової нейронної мережі GNN.

2.2. Підготовка вхідних даних

Першим кроком для забезпечення можливості використання графів знань у навчанні нейронних мереж є перетворення формату графа знань у компактний вигляд. Це питання буде розглянуто у наступному розділі, однак з наведеного опису зрозуміло, що одним з найзручніших варіантів є трійки RDF, зокрема формат N-Triples.

Далі список ребер, яким фактично є формат N-Triples, треба перетворити у векторний формат. Для цього треба утворити індексовані списки суб'єктів/об'єктів RDF та ребер (предикатів). З двох утворених списків треба утворити матрицю суміжності. Після цього кожному вузлу чи ребру треба поставити у відповідність певний вектор. Фактично, ця операція є відображенням сутностей та зв'язків з простору графа знань у прихований простір векторів. Надалі з отриманих векторів згідно з матрицею суміжності можна отримати вектор ознак або тензор залежно від того, який тип графової нейронної мережі буде використовувати ці вхідні дані.

Коли дані вже підготовлені, можна обрати архітектуру графової нейронної мережі для розв'язання певної задачі.

2.3. Графова нейронна мережа

Як зазначалося у роботі [11], вхідними даними для нейронних мереж можуть виступати дані із зображень, звуку, текстів, відео, вимірів певних величин тощо. При цьому слід розуміти, що ці дані можуть описувати як статичний стан даних, так і дані, що змінюються у часі. Деякі з цих даних, наприклад, як зображення фактично утворюють сітку рівноправних значень, що доволі суттєво спрощує роботу згорткових нейронних мереж. Змінні у часі дані можуть бути використані для навчання різних архітектур рекурентних мереж, таких, наприклад, як LSTM та GRU. Однак, навчання цих мереж ґрунтується на незалежності частин вхідного пакунку даних одна від одної та незалежності ознак, що входять до складу кожної з цих частин. Це дозволяє довільно переставляти дані у вхідному пакунку, як по рядках, так і по стовпцях і це жодним чином не впливає на результат роботи нейронної мережі. Однак, з опису наведеному у розділі 2.2 стає зрозуміло, що перестановка у векторизованому та згладженому представлені, що сформоване на базі графу знань неможливе. Це випливає з того, що якісь дані позначають об’єкт, інші суб’єкт, а решта предикат у випадку використання вхідного графа у форматі RDF. Таким чином, навчання традиційних нейронних мереж з використанням класичного підходу неможливе. Тож виникла потреба у створенні модифікованих архітектур в яких би існувала можливість контрольованого навчання чи то шляхом зворотного розповсюдження помилки, чи в інший спосіб, наприклад, через конкурентне навчання. Розмай таких архітектур отримав назву графові нейронні мережі й до цих архітектур можна віднести такі: GNN (Graph Neural Network), GGNN (Gated Graph Neural Network), GIN (Graph Isomorphism Network), GCN (Graph Convolutional Network), GAT(Graph Attention Network). Найпоширенішим методом оновлення стану матриць ваг у таких архітектурах став метод передачі повідомлень (message passing).

2.4. Передавання повідомлень

Основна ідея методу передавання повідомлень полягає в тому, що вузли графу агрегують інформацію про власних сусідів на кожній ітерації, збільшуючи відстань пошуку сусідніх вузлів. Основна мета, що переслідує цей метод, полягає в тому, щоб зробити дані виразнішими та набути можливість їх переставляти, як це роблять у класичних нейронних мережах.

Під час кожної ітерації передачі повідомлень у GNN, вкладення $\mathrm{h}_u^{(k)}$, що відповідає кожному вузлу $u \in \mathcal{V}$, оновлюється відповідно до інформації, зібраної з околу графа u $N(u)$ (Рисунок 2.1). Це оновлення з передачею повідомлень можна описати наступним чином:

$$\mathrm{h}_u^{(k+1)} = \mathrm{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \mathrm{AGGREGATE}^{(k)}\left(\left\{\boldsymbol{h}_v^{(k)}, \forall v \in N(u)\right\}\right)\right)$$

$$= \mathrm{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, m_{N(u)}^{(k)}\right) \qquad (2.1)$$

де UPDATE та AGGREGATE - довільні диференційовані функції (тобто нейронні мережі), а $\mathrm{m}_{N_{(u)}}$ - "повідомлення", яке агрегується з околиці графа $N(u)$. У формулі були використані надрядкові коефіцієнти, щоб розрізняти вкладки та функції на різних ітераціях проходження повідомлення. На кожній ітерації GNN функція AGGREGATE приймає на вхід множину вкладок вершин в околиці графа $N(u)$ і генерує повідомлення $\mathrm{m}_{N_{(u)}}$ на основі цієї агрегованої інформації про околиці. Функція оновлення UPDATE об'єднує повідомлення $\mathrm{m}_{N_{(u)}}$ з попереднім розміщенням h_u вершини u, щоб згенерувати оновлене розміщення. Початкові вбудовування при $k = 0$ дорівнюють вхідним характеристикам для всіх вершин, тобто такий вираз є справедливий:

$h_u^{(0)} = x_u \forall u \in V$. Після виконання K ітерацій проходження GNN-повідомлення, можливо використати вихід кінцевого шару для визначення вбудовувань для кожного вузла, тобто,

$$z_u = \mathrm{h}_u^{(K)}, \forall u \in V \tag{2.2}$$

Треба зауважити, що оскільки функція AGGREGATE приймає у вигляді вхідних даних множину, то графові нейронні мережі визначені у такий спосіб є еквівалентними за перестановкою.

Розгляньмо детальніше процес передавання повідомлення. На кожній ітерації кожен вузол збирає інформацію зі своєї локальної області, і в міру проходження цих ітерацій кожне вбудовування вузла містить все більше і більше інформації з більш віддалених ділянок графа. Точніше, після першої ітерації (k=1) кожне вбудовування вузла містить інформацію з околиці довжиною 1 крок, тобто кожне вбудовування вузла містить інформацію про особливості його найближчих сусідів по графу, до яких можна дістатися шляхом довжиною 1 у графі; після другої ітерації (k=2) кожне вбудовування вузла містить інформацію з околиці довжиною 2 кроки; і після k ітерацій кожне вбудовування вузла містить інформацію про околицю довжиною k кроків.

Інформація, що кодується у такий спосіб, буває двох видів. По-перше, це структурна інформація про граф. Наприклад, після k ітерацій проходження GNN-повідомлення, вкладення $\mathrm{h}_u^{(k)}$ вершини u може містити інформацію про степені всіх вершин у k-кроковому околі вершини u. Ця структурна інформація може бути корисною для багатьох задач. Наприклад, при аналізі молекулярних графів використовується інформація про ступені для визначення типів атомів.

На додаток до структурної інформації, іншим ключовим типом інформації, яку збирає вбудовування вузлів GNN, є інформація на основі ознак. Зазвичай висувається припущення, що всі вершини графа мають властивості або атрибути, пов'язані з ними, які використовуються для ініціалізації передачі GNN-повідомлень. Після t ітерацій передачі GNN-повідомлень, вбудовування для кожної вершини, таким чином, також кодують інформацію про всі ознаки в їхньому t-кроковому сусідстві. Така локальна поведінка агрегації ознак у GNN аналогічна поведінці ядер згортки у згорткових нейронних мережах (CNN). Однак, якщо CNN агрегують інформацію про особливості просторово визначених ділянок зображення, то GNN агрегують інформацію на основі локальних околиць графа.

2.5. GNN

Як зазначалося раніше, графова нейронна мережа GNN передбачає ітеративний процес описаною формулою 2.2. Найперші графові нейронні мережі з’явилися у 2009 році і за час свого існування цей клас нейронних мереж набув значного розвитку. Однак в рамках цієї атестаційної роботи одною з цілей є обробка графів знань за допомогою методів штучного інтелекту, тому для базового випробування цілком достатньо найпростішої графової нейронної мережі, описаної авторами першої публікації на цю тему Скарселлі та ін. [2009] [12]. Відповідно необхідно визначити та описати функції UPDATE та AGGREGATE, на які спирається формула 2.3.

Базова передача повідомлень GNN визначається наступним чином:

$$k_u^{(K)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neight}}^{(k)} \sum_{\upsilon \in N(u)} h_\upsilon^{(k-1)} + b^{(k)} \right), \tag{2.3}$$

де $\mathrm{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neight}}^{(k)} \in \mathrm{M}^{d^{(k)} \times d^{(k-1)}}$ - матриці параметрів, що навчаються, і σ позначає поелементну нелінійність (наприклад, tanh або ReLU). Член зсуву $b \in \mathrm{M}^{d^{(k)}}$

часто опускають для простоти запису, але він може бути важливим для досягнення високої продуктивності. Передача повідомлень у базовому фреймворку GNN аналогічна стандартному багатошаровому персептрону (MLP) або рекурентній нейронній мережі Елмана (Elman RNN), оскільки вона покладається на лінійні операції, за якими слідує одноелементна нелінійність. Спочатку підсумовуються повідомлення, що надходять від сусідів; потім комбінується інформація про сусідів з попереднім вбудовуванням вузла за допомогою лінійної комбінації; і, нарешті, застосовується поелементна нелінійність. Відповідно можна визначити базові функції UPDATE та AGGREGATE:

$$\mathbf{m}_{N_{(u)}} = \sum_{\upsilon \in N(u)} \mathbf{h}_{\upsilon} \tag{2.4}$$

$$\text{UPDATE}(\mathbf{h}_u, \mathbf{m}_{N_{(u)}}) = \sigma(\mathbf{W}_{\text{self}}\mathbf{h}_u + \mathbf{W}_{\text{neight}}\mathbf{m}_{N_{(u)}}), \tag{2.5}$$

де формула 2.6

$$\mathbf{m}_{N_{(u)}} = \mathbf{m}_{N_{(u)}} \tag{2.6}$$

є скорочення для позначення повідомлення, яке було агреговано з околиці графа u. Для скорочення запису надрядкову позначку ітерації було опущено. Параметри W_{self}, $\mathbf{W}_{\text{neight}}$ і b можна використовувати у різних ітераціях GNN ітерацій передавання повідомлень або навчатися окремо.

Однак, замість запису 2.2, що визначає GNN на базі записів про вузли та ребра, GNN також можна визначити на рівні графу, наприклад формулою 2.7:

$$\mathrm{H}^{(t)} = \sigma(\mathrm{A}\mathrm{H}^{(t-1)}\mathbf{W}^{(t)}_{neight} + \mathrm{H}^{(t-1)}\mathrm{W}^{(t)}_{self}) \tag{2.7}$$

де $\mathrm{H}^{(t)} \in \mathrm{M}^{|V|\times d}$ позначає матрицю представлень вершин на рівні t у GNN (кожній вершині відповідає рядок у матриці), A - матриця суміжності графа. І член зсуву був опущений для спрощення запису. Хоча таке представлення на

рівні графа не завжди можна застосувати до всіх моделей GNN - наприклад, до моделей на основі уваги - воно часто є більш лаконічним, а також підкреслює, як багато GNN можна ефективно реалізувати, використовуючи невелику кількість розріджених матричних операцій.

2.6. Передача повідомлень за допомогою циклів

Для спрощення нейронного підходу до передачі повідомлень часто додають цикли до вхідного графа і не використовують явний крок оновлення. У цьому підході визначається передача повідомлення просто як

$$\mathbf{h}_u^{(k)} = \text{AGGREGATE}(\{\mathbf{h}_v^{(t-1)}, \forall v \in N(u) \cup \{u\}\}), \tag{2.8}$$

де тепер агрегація відбувається над множиною $N(u) \cup \{u\}$, тобто над сусідами вершини, а також над самою вершиною. Перевагою такого підходу є те, що більше не потрібно визначати явну функцію оновлення, оскільки оновлення неявно визначено за допомогою методу агрегації. Спрощення передачі повідомлень у такий спосіб часто може полегшити перенавчання, але це також сильно обмежує виразність GNN, оскільки інформація, що надходить від сусідів вузла, не може бути розрізнена від інформації про сам вузол.

У випадку базової GNN додавання циклів еквівалентне розподілу параметрів між матрицями $\mathbf{W}_{\text{self}}$ і $\mathbf{W}_{\text{neight}}$, що дає наступне оновлення на рівні графа:

$$\mathbf{H}^{(t)} = \sigma((\mathbf{A}+\mathbf{I})\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}) \tag{2.9}$$

2.7. Агрегація околиць

Сама по собі операція агрегації може бути виконана у різний спосіб. З одного боку це може бути здійснено за допомогою інваріантних функцій, які не враховують порядок сусідів. З іншого боку, є сучасніший підхід, який

усереднює агрегацію сусідів, виконану без використання інваріантної функції. Обидва ці підходи будуть стисло розглянуті надалі.

2.7.1. Нормалізація сусідства

Найпростіша операція агрегації сусідів (Рівняння 2.6) просто бере суму сусідніх вкладень. Однією з проблем такого підходу є те, що він може бути нестабільним і дуже чутливим до степенів вершин. Наприклад, якщо вершина u має у $100\times$ більше сусідів, ніж вершина u' (тобто набагато вищий ступінь), тоді можно очікувати, що $\left\|\sum_{\upsilon\in N(u)} \mathrm{h}_{\upsilon}\right\| >> \left\|\sum_{u'\in N(u')} \mathrm{h}_{\upsilon'}\right\|$ (для будь-якої розумної векторної норми $\|\cdot\|$), і ця різка відмінність у величині може призвести до чисельної нестабільності, а також до труднощів при оптимізації. Одне з простих розв'язань цієї проблеми полягає в тому, щоб просто нормалізувати операцію агрегації на основі ступенів долучених вузлів. Найпростіший підхід це просто взяти середнє значення, а не суму:

$$\mathrm{m}_{N_{(u)}} = \frac{\sum_{\upsilon\in N(u)} \mathrm{h}_{\upsilon}}{\left|N(u)\right|} \tag{2.10}$$

але дослідники також досягли успіху з іншими факторами нормалізації, такими як наступний підхід, застосований Кіпфом та Веллінгом [2016][13]:

$$\mathrm{m}_{N_{(u)}} = \sum_{\upsilon\in N(u)} \frac{\mathrm{h}_{\upsilon}}{\sqrt{\left|N(u)\right|\left|N(\upsilon)\right|}} \tag{2.11}$$

Припущення, що лежить в основі нормалізації Кіпфа, полягає в тому, що модель повинна зменшувати вагу повідомлень, які надходять від сусідів з дуже високими степенями, оскільки "сигнал", що надходить від цих сусідів з високими степенями, може бути менш точним та інформативним. Наприклад, у графі цитувань - тип даних, який проаналізували Кіпф і Веллінг [2016] - інформація з вузлів з дуже високими степенями (тобто статей) може бути не

дуже корисною для висновку про приналежність до спільноти на графі, оскільки ці статті можуть цитуватися тисячі разів у різних підгалузях. Нормалізація Кіпфа також може бути обґрунтована на основі спектральної теорії графів. Зокрема, поєднання нормалізованої агрегації Кіпфа (Рівняння 2.13) з базовою функцією оновлення GNN (Рівняння 2.7) призводить до наближення першого порядку згортки спектра графа.

Одна з найпопулярніших базових моделей графових нейронних мереж - графова згорткова мережа (GCN) - використовує нормалізоване агрегування Кіпфа, а також підхід самооновлення.Таким чином, модель GCN визначає функцію передачі повідомлення формулою 2.12.

$$\mathbf{h}_{\upsilon}^{(k)} = \sigma(\mathbf{W}^{(k)} \sum_{\upsilon \in N(u) \cup \{u\}} \frac{\mathbf{h}_{\upsilon}}{\sqrt{|N(u)||N(\upsilon)|}}) \tag{2.12}$$

Цей підхід був вперше описаний Кіпфом та Веллінгом [2016][13] і виявився однією з найпопулярніших та найефективніших базових GNN-архітектур.

Правильна нормалізація може бути важливою для досягнення стабільної та високої продуктивності під час використання GNN. Однак важливо зазначити, що нормалізація може також призвести до втрати інформації. Наприклад, після нормалізації може бути важко (або навіть неможливо) використовувати вивчені вбудовування для розрізнення вузлів різного ступеня, а різні інші структурні особливості графа можуть бути приховані нормалізацією. Насправді, базовий GNN, що використовує нормалізований оператор агрегації у рівнянні (2.12), є менш потужним, ніж базовий агрегатор суми у рівнянні (2.6).

Таким чином, використання нормалізації залежить від конкретного застосування. Зазвичай нормалізація є найбільш корисною в задачах, де інформація про властивості вузлів є набагато кориснішою за структурну

інформацію, або де існує дуже широкий діапазон ступенів вузлів, що може призвести до нестабільності під час обчислень.

2.7.2. Агрегація наборів

Нормалізація околиць є не єдиним інструментом покращення продуктивності нейронної мережі. Операція агрегації околиць по суті є функцією від множини. Це означає, що задана множина сусідніх вкладень $\{\mathbf{h}_\upsilon, \forall \upsilon \in N(u)\}$ й треба відобразити цю множину в єдиний вектор $\mathbf{m}_{N_{(u)}}$.

Той факт, що $\{\mathbf{h}_\upsilon, \forall \upsilon \in N(u)\}$ є множиною, насправді дуже важливий: не існує ніякого природного впорядкування сусідів вершин, і будь-яка функція агрегації, має бути інваріантною до перестановок.

Один з принципових підходів до визначення функції агрегації базується на теорії нейронних мереж, інваріантних до перестановок. Наприклад, Zaheer та ін. [2017] показують, що функція агрегації, яка може бути виражена багатошаровим персептроном (MLP) та має наступний вигляд, є універсальним апроксиматором функції множини (формула 2.13).

$$\mathbf{m}_{N_{(u)}} = \mathrm{MLP}_\theta\left(\sum_{\upsilon \in N(u)} \mathrm{MLP}_\phi(\mathbf{h}_\upsilon)\right), \tag{2.13}$$

Це означає, що будь-яка функція, яка не залежить від порядку елементів у множині та перетворює цю множину в одне вкладання, може бути наближена за допомогою моделі, заснованої на цьому принципі.

Дослідження Zaheer та співавторів [2017][14] використовує суму вкладень після застосування першого MLP. Однак, цю суму можна замінити на інші функції редукції, наприклад, вибір максимального або мінімального значення, як це було показано у дослідженні Qi та інших у 2017 році[15].

Також ці підходи можна комбінувати з методами нормалізації, розглянутими в інших дослідженнях, таких як GraphSAGE-pool[16].

Застосування об'єднання наборів за принципом, викладеним у рівнянні (2.15), часто призводить до покращення продуктивності моделей. Проте, існує ризик перенавчання, особливо якщо використовується MLP з великою кількістю шарів. Тому зазвичай рекомендується використовувати MLP з одним прихованим шаром, оскільки вони задовольняють теоретичним вимогам, не вдаючись до надмірної перепараметризації.

Підходи до агрегації сусідніх вузлів у графових нейронних мережах (GNN), які базуються на об'єднанні множин, по суті, включають додавання додаткових шарів MLP до стандартних архітектур агрегації. Ця методика, хоча і проста, підвищує теоретичну здатність GNN. Але існує альтернативний підхід, відомий як об'єднання Janossy[26], який також є ефективнішим порівняно з простим сумуванням або визначенням середнього значення сусідніх вкладень.

Ключовою проблемою в агрегації сусідів[27] є необхідність використання функції, незалежної від порядку елементів, оскільки природне впорядкування сусідів не існує. У підході об'єднання множин досягають інваріантності до перестановок шляхом застосування таких операцій, як сума, середнє або максимум. Цю модель було покращено шляхом інтеграції з MLP. На відміну від цього, об'єднання Janossy використовує інший підхід, застосовуючи функцію, чутливу до порядку елементів, та усереднюючи результати за декількома можливими перестановками.

У цьому підході, множина вкладень перетворюється у послідовність, яка потім обробляється функцією, чутливою до порядку, такою як LSTM. Якщо в рівнянні об'єднання Janossy враховувати всі можливі перестановки, то такий агрегатор також стає універсальним апроксиматором для функцій множин, подібно до попереднього підходу. Однак, через велику кількість можливих перестановок, на практиці це нереалістично, тому зазвичай використовують один із двох методів: обирають випадкову підмножину перестановок для

кожного застосування агрегатора або застосовують канонічне впорядкування сусідів.

Дослідження Murphy та співавторів у 2018 [17] році детально обговорює та порівнює ці методи, а також інші способи апроксимації. Їхні результати вказують, що об'єднання Janossy може бути ефективнішим у порівнянні з об'єднанням множин у деяких випадках.

2.8. Узагальнені методи оновлення

Найчастіше для породження нових архітектур графових нейронних мереж, зміні піддається саме функція агрегації. Це обумовлено тим, що архітектури задежать від характеру даних й відповідно дані залежно від процесу, що їх породжує, по різному пов’язані між собою. Однак функція оновлення для обробки графів за допомогою GNN є не менш важливою.

У базовій моделі GNN оновлене приховане представлення $\mathrm{h}_u^{(t)}$ вузла u обчислюється на кожному кроці проходження повідомлення за допомогою лінійної комбінації агрегованого повідомлення $\mathrm{m}_{N_{(u)}}$, що надходить від сусідів вузла u, а також попереднього прихованого представлення вузла $\mathrm{h}_u^{(t-1)}$ [28]. Хоча цей простий і ефективний у багатьох ситуаціях підхід до оновлення має серйозні недоліки.

Найбільш помітно, що при лінійному поєднанні $\mathrm{m}_{N_{(u)}}$ з $\mathrm{h}_u^{(t-1)}$ інформація з різних ітерацій передачі повідомлень стає заплутаною після кожного раунду передачі повідомлень. При використанні цієї простої лінійної комбінації GNN не має простого способу відокремити локальну інформацію, яка є специфічною для певного раунду передачі повідомлень, від більш нелокальної інформації, отриманої в наступних ітераціях передачі повідомлень. Ця лінійна плутанина може призвести (серед іншого) до проблеми, відомої як надмірне згладжування. Після великої кількості ітерацій

передачі повідомлень, представлення для всіх вершин графа стають дуже схожими одне на одне.

Нарешті, простий стиль оновлення, який використовується в базовому підході GNN, може призвести до числової нестабільності (наприклад, вибухових градієнтів [29]) через багаторазове застосування множення матриць на кожному шарі. Ця проблема аналогічна чисельній нестабільності, що спостерігається для базових рекурентних нейронних мереж (RNN) моделей, таких як RNN Елмана [19].

Питання надмірного згладжування в GNN можна формалізувати, визначивши вплив вхідної характеристики кожної вершини $\mathrm{h}_u^{(0)} = x_u$ на кінцевий шар вбудовування всіх інших вершин графа, тобто $\mathrm{h}_\upsilon^{(K)}, \forall \upsilon \in \boldsymbol{V}$.

Зокрема, для будь-якої пари вершин u та υ можливо кількісно оцінити вплив вершини u на вершину υ в GNN, дослідивши величину відповідної матриці Якобіана [Xu et al, 2018b] [19]:

$$I_K(u,\upsilon) = 1^T \left(\frac{\partial \mathrm{h}_\upsilon^{(K)}}{\partial \mathrm{h}_u^{(0)}} \right) 1, \tag{2.14}$$

де 1 - вектор всіх одиниць. Значення $I_K(u,\upsilon)$ використовує суму елементів матриці Якобіана $\frac{\partial \mathrm{h}_\upsilon^{(K)}}{\partial \mathrm{h}_u^{(0)}}$ як міру того, наскільки сильно початкове вбудовування вершини u впливає на кінцеве вбудовування вершини v в GNN.

Враховуючи це визначення впливу, Сюй та ін. [2018b] [19] довели наступну теорему: для будь-якої GNN-моделі, що використовує підхід з самооновленням циклу та функцію агрегації вигляду

$$\text{AGGREGATE}\left(\left\{\mathrm{h}_{\upsilon}, \forall_{\upsilon} \in N(u) \cup \{u\}\right\}\right) = \frac{1}{f_n(|\, N(u) \cup \{u\}\,|)} \sum_{\upsilon \in N(u) \cup \{u\}} \mathrm{h}_{\upsilon}, \quad (2.15)$$

де $f : \mathrm{M}^{+} \to \mathrm{M}^{+}$ - довільна диференційовна функція нормалізації, маємо, що

$$I_K(u,\upsilon) \propto p_{\varsigma,K}(u \,|\, \upsilon), \quad (2.16)$$

де $p_{\varsigma,K}(u \,|\, \upsilon)$ позначає ймовірність відвідування вершини υ під час випадкового блукання довжиною K, що починається з вершини u.

Ця теорема є прямим наслідком іншої теореми в роботі Сюй та ін. [2018b][19]. А ця теорема своєю чергою стверджує, що при використанні K-шарової моделі в стилі GCN вплив вузла u і вузла υ пропорційний ймовірності досягнення вузла v на K-кроковому випадковому блуканні, що починається з вузла u. Важливим наслідком цього, однак, є те, що при $K \to \infty$ вплив кожної вершини наближається до стаціонарного розподілу випадкових блукань по графу, що означає, що інформація про локальну околицю втрачається. Більше того, у багатьох реальних графах, які містять вершини високих степенів і нагадують так звані "експандерні" графи, потрібно лише $k = O(log\left(|\boldsymbol{\mathcal{V}}\,|\right)$ кроків, щоб випадкові блукання, починаючи з будь-якої вершини, збіглися до майже рівномірного розподілу [Hoory et al., 2006][20].

Ця теорема застосовується безпосередньо до моделей, що використовують підхід самооновлення[30], але результат також може бути поширений в асимптотичному сенсі для базового оновлення GNN (тобто рівняння 2.7), якщо $\| \mathrm{W}_{\text{self}}^{(k)} \| < \| \mathrm{W}_{\text{neight}}^{(k)} \|$ на кожному шарі k. Таким чином, при використанні простих моделей GNN - особливо тих, що використовують підхід самооновлення - побудова глибших моделей може насправді погіршити продуктивність. З додаванням нових шарів втрачається інформація про локальні сусідні структури, і раніше зроблені вбудовування стають надмірно згладженими, наближаючись до майже рівномірного розподілу.

2.8.1. Конкатенація та пропуск з'єднань

Основною проблемою базової функції оновлення GNN є те, що вона використовує лінійну комбінацію для оновлення представлення прихованих вузлів на кожній ітерації передачі повідомлення. Найбільш очевидним способом розв'язання цієї проблеми є введення різноманітних конкатенаційних або пропускних з'єднань, які намагаються безпосередньо зберегти інформацію з попередніх раундів передачі повідомлень під час кроку оновлення.

Ці методи конкатенації та пропуску з'єднань можна використовувати у поєднанні з більшістю інших підходів до оновлення GNN. Таким чином, для узагальнення, використовується UPDATE_{base} для позначення базової функції оновлення. Наприклад, можна припустити, що UPDATE_{base} задано рівнянням 2.17, і було визначено різні оновлення з пропускними з'єднаннями над цією базовою функцією.

Одне з найпростіших оновлень пропуску з'єднань використовує конкатенацію для збереження більшої кількості інформації на рівні вузлів під час передачі повідомлення:

$$\mathrm{UPDATE}_{concat}(\mathrm{h}_u, \mathrm{m}_{N(u)}) = \left[\mathrm{UPDATE}_{base}(\mathrm{h}_u, \mathrm{m}_{N(u)}) \oplus \mathrm{h}_u\right], \tag{2.17}$$

де об’єднуються вихід функції оновлення бази з вузлами попереднього шару представлення. Знову ж таки, припущення тут полягає в тому, що модель змушена буде відокремлювати інформацію поточного вузла від інформації про сусідів.

З'єднання пропусків на основі конкатенації було запропоновано у фреймворку GraphSAGE[16]. І це була перша робота, яка висвітлила можливі переваги таких модифікацій функції оновлення [Hamilton et al. 2017a] [16]. Однак, окрім конкатенації, можливо використовувати інші форми пропускних

з'єднань, такі як метод лінійної інтерполяції, запропонований Pham та ін. [2017] [21]:

$$\mathrm{UPDATE}_{\mathrm{int}erpolate}(\mathrm{h}_u, \mathrm{m}_{N_{(u)}}) = \alpha_1 \boxtimes \mathrm{UPDATE}_{base}(\mathrm{h}_u, \mathrm{m}_{N_{(u)}}) + \alpha_2 \boxtimes \mathrm{h}_u, \quad (2.18)$$

де $\alpha_1, \alpha_2 \in [0,1]^d$ - вектори стробування, де $\alpha_2 = 1 - \alpha_1$ і $\boxtimes$ позначає поелементне множення. У цьому підході остаточне оновлене представлення є лінійною інтерполяцією між попереднім представленням і представленням, яке було оновлене на основі інформації про околиці. Параметри стробування α_1 можна дізнатися разом з моделлю різними способами. Наприклад, Фам та ін. [2017] [21] генерують α_1 як вихід окремого одношарового GNN, який використовує поточні представлення прихованого шару як ознаки. Але також можуть бути застосовані й інші простіші підходи, такі як безпосереднє навчання параметрів α_1 для кожного шару, що передає повідомлення, або використання MLP на поточних представленнях вузлів для генерації цих параметрів стробування.

Загалом, ці конкатенаційні та залишкові зв'язки є простими стратегіями, які можуть допомогти полегшити проблему надмірного згладжування в GNN, а також покращити чисельну стабільність оптимізації. Дійсно, подібно до корисності залишкових зв'язків у згорткових нейронних мережах (CNN) [He et al., 2016][22], застосування цих підходів до GNN може полегшити навчання набагато глибших моделей. На практиці ці методи, як правило, найбільш корисні для задач класифікації вузлів з помірно глибокими GNN (наприклад, 2-5 шарів), і вони досить добре справляються із завданнями, які демонструють гомофільність, тобто, де прогноз для кожного вузла сильно пов'язаний з особливостями його локального оточення.

2.8.2. Оновлення з обмеженнями

Методи оновлення описані в пункті 2.8.1 схожі на ці методи поліпшення продуктивності, що застосовуються у глибоких нейромережах машинного зору. Однак існує інший погляд на цю проблему, який схожий тим, що використовується для поліпшення стабільності класичних рекурентних нейронних мереж (RNN)[31]. Зокрема, один із способів розглянути алгоритм передачі повідомлень GNN полягає в тому, що функція агрегації отримує "спостереження" від сусідів, які потім використовуються для оновлення "прихованого стану" кожного вузла. Виходячи з цього, можна безпосередньо застосовувати методи, що використовуються для оновлення прихованого стану архітектур RNN на основі спостережень. Наприклад, одна з найперших архітектур GNN [Li et al., 2015] [23] визначає функцію оновлення як

$$\mathrm{h}_u^{(k)} = \mathrm{GRU}\left(\mathrm{h}_u^{(k-1)}, \mathbf{m}_{N_{(u)}}^{(k)}\right) \tag{2.19}$$

де GRU позначає рівняння оновлення комірки зі штриховим рекурентним блоком (GRU) [Cho et al., 2014] [24]. Також використовуються оновлення на основі архітектури LSTM [Selsam et al., 2018][25].

Загалом, будь-яка функція оновлення, визначена для RNN, може бути використана в контексті GNN. Просто відбувається заміна аргументу прихованого стану функції оновлення RNN (зазвичай позначається $\mathrm{h}^{(t)}$ на прихований стан вузла, а вектор спостереження (зазвичай позначається $\mathrm{x}^{(t)}$) на повідомлення, агреговане з локальної околиці[32]. Важливо, що параметри цього оновлення в стилі RNN завжди є спільними для всіх вузлів (тобто, використовується та сама комірка LSTM або GRU для оновлення кожного вузла). На практиці найчастіше розподіляють параметри функції оновлення між рівнями передачі повідомлень у GNN.

Ці закриті оновлення дуже ефективні для підтримки глибоких архітектур GNN (наприклад, понад 10 шарів) і запобігання надмірному згладжуванню. Загалом, вони найбільш корисні для застосунків GNN, де

завдання прогнозування вимагає складних міркувань над глобальною структурою графа, таких як застосунки для аналізу програм [Li et al., 2015] [23] або комбінаторної оптимізації [Selsam et al., 2018][25].

2.8.3. Перехід між зв'язками знань

Досі в цій роботі вважалося, що використовуються результати останнього шару GNN:

$$z_u = h_u^{(K)}, \forall u \in V \tag{2.20}$$

Таке припущення робиться у багатьох підходах GNN, і обмеження цієї стратегії зумовили потребу у залишкових та обмежених оновленнях, щоб унеможливити надмірне згладжування.

Однак, додатковою стратегією для покращення якості кінцевих представлень вузлів є просто використання представлень на кожному рівні проходження повідомлень, а не лише на виході кінцевого рівня. У цьому підході ми визначаємо кінцеві представлення вузлів z_u як

$$z_u = f_{JK}(h_u^{(0)} \oplus h_u^{(1)} \oplus ... \oplus h_u^{(K)}), \tag{2.21}$$

де f_{JK} - довільна диференційовна функція. Ця стратегія відома як додавання зв'язків зі стрибкоподібними знаннями (JK) і була вперше запропонована та проаналізована в [Xu et al., 2018b][19]. У багатьох застосунках функцію f_{JK} можна просто визначити як функцію тотожності, що означає, що просто об'єднуються вкладення вузлів з кожного шару, але Сюй та ін. [2018b][19] також досліджують інші варіанти, такі як підходи з максимальним об'єднанням та шари уваги LSTM. Попри простоту, цей підхід часто призводить до послідовних покращень у широкому спектрі завдань і є загалом корисною стратегією для застосування.

2.9. Висновок до розділу

Граф знань може бути використаний у якості джерела для навчання графових нейронних мереж GNN. Однак, спочатку інформація з графу знань має бути перетворена у векторний формат, який міститиме інформацію про самі вузли та зв’язки між ними. Графова нейронна мережа класичної архітектури, яка здатна навчатися на таких даних, містить дві ключові операції: агрегації та оновлення. Їх конкретну реалізацію треба обирати залежно від характеру та якості вхідних даних.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ПРОЦЕСІВ ВИКОРИСТАННЯ ГРАФІВ ЗНАНЬ У СИСТЕМАХ НА БАЗІ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ

3.1. Підготовка вхідних даних

Враховуючи кількість графів даних розповсюджених у різноманітних форматах в мережі інтернет, для використання у машинному та глибокому навчанні та у системах штучного інтелекту неминуче постають кілька основних проблем:

- формати геть різні й здебільшого не придатні для використання у якості наборів даних через наявність зайвого синтаксису;
- графи знань мають різний розмір вузлів і через це можуть виникати складнощі під час навчання систем штучного інтелекту;
- природа графів знань і характер даних, що вони зберігають, можуть стати на заваді побудові ефективних систем ШІ таких, наприклад, як згорткові графові нейронні мережі.

В рамках цієї роботи було запропоновано розв’язок перших двох з описаних проблем та показано, як графи знань можна використати під час навчання нейронної мережі прямого розповсюдження.

Аналіз синтаксису графів знань наведених у різних форматах показує, що найменшу кількість зайвих синтаксичних конструкцій містить формат N-Triples. Тому було прийнято рішення створити бібліотеку конвертації графів знань різних форматів до формату N-Triples та використати зконвертовані таким чином графи в процесі машинного навчання. Таким чином бібліотека конвертера мала містити функції конвертації з форматів Turtle, OWL.

3.2. Конвертація з формату Turtle

Як зазначалося в розділі 1.3, формат Turtle це також трійки виду “об'єкт - предикат - суб'єкт”, які на відміну від N-Triples відокремлюють простори імен аби зменшити обсяг текстової інформації. Також, трійки Turtle можуть містити порожні вузли, що може стати на заваді формуванню трійок N-Triples. І нарешті, Turtle може містити коментарі (рядки, що починаються з #), чого не передбачає N-Triples. Таким чином, конвертація Turtle в N-Triples мовою Python виглядає так (Лістинг 3.1):

Лістинг 3.1

```
def turtle_ntriples_bridge(turtle_data):
    lines = turtle_data.split('\n')
    triples = []

    for line in lines:
        line = line.strip()
        if not line or line.startswith("#"):
            continue

        subject, predicate, obj = line.split(maxsplit=2)

        # Обробка порожніх вузлів
        if obj == "[]":
            obj = "_:blank_node"
        elif obj.startswith("[") and obj.endswith("]"):
            obj = "_:blank_node"

        triple = f"{subject} {predicate} {obj} ."
        triples.append(triple)

    ntriples_data = '\n'.join(triples)
    return ntriples_data
```

3.3. Конвертор OWL XML у N-Triples

У OWL XML містить набагато більше семантичної інформації, ніж формат N-Triples. Крім того, за структурою їх формати геть не схожі. Тому під час перетворення OWL в N-Triples, вузли OWL мають бути подріблені. Відповідно конвертацію наведено у лістингу 3.2:

Лістинг 3.2.

```
import xml.etree.ElementTree as ET


def parse_owl(owl_file):
    triples = []

    # Зчитуємо XML-структуру з OWL-файлу
    tree = ET.parse(owl_file)
    root = tree.getroot()

    # Отримуємо простори імен
    namespaces = {'owl': 'http://www.w3.org/2002/07/owl#', 'rdf': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'}

    # Проходимося по всіх елементах у дереві
    for elem in root.iter():
        if '}' in elem.tag:
            # Розбираємо тег у вигляді {namespace}tag
            tag_parts = elem.tag.split('}')
            namespace = tag_parts[0][1:]
            tag = tag_parts[1]

            # Перетворюємо простір імен у префікс, наприклад, owl:Class
            prefix = [key for key, value in namespaces.items() if value == namespace][0]
            full_tag = f"{prefix}:{tag}"
```

```
        # Отримуємо значення елементу
        value = elem.text if elem.text else ''

        # Формуємо триплет (суб'єкт, предикат, об'єкт)
        triple = f"{elem.attrib['{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about']} {full_tag} {value} ."

        # Додаємо триплет до списку
        triples.append(triple)

    return triples
```

3.4. Конвертер OWL json в N-Triples:

OWL json теж мало схожий на N-Triples, як і OWL XML. Фактично він містить той самий обсяг семантично зв’язаної інформації, що й OWL XML. Конвертація OWL json в N-Triples, виглядає як на лістингу 3.3:

Лістинг 3.3.

```
import json

def parse_owl_json(owl_json)
    triples = []

    for entry in owl_json
        subject = entry.get('subject', '')
        predicate = entry.get('predicate', '')
        obj = entry.get('object', '')

        if subject and predicate and obj
            triples.append((subject, predicate, obj))
```

```
    return triples

def convert_to_nt(triples)
    nt_lines = [f{triple[0]} {triple[1]} {triple[2]} . for triple in triples]
    return 'n'.join(nt_lines)
```

3.5. Конвертор KIF у N-Triples

Формат KIF фактично містить опис знань та правил висновку. Цей формат зовсім не схожий на трійки RDF, тому повна та безпомилкова конвертація є доволі складною. Тож спрощений конвертер KIF у N-Triples наведено у лістингу 3.4:

Лістинг 3.4.

```
def kif_to_nt(kif_text):
    triples = []

    lines = kif_text.split('\n')
    for line in lines:
        line = line.strip()
        if line.startswith('(') and line.endswith(')'):
            parts = line[1:-1].split()
            if len(parts) == 3 and parts[1].lower() == '=>' and '(' not in parts[2]:
                subject = f"<{parts[0]}>"
                predicate = "<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>"
                obj = f"<{parts[2]}>"
                triples.append((subject, predicate, obj))

    return triples

def convert_to_nt(triples):
    nt_lines = [f"{triple[0]} {triple[1]} {triple[2]} ." for triple in triples]
    return '\n'.join(nt_lines)
```

3.6. Конвертор JSON-LD у N-Triples

JSON-LD є спорідненим RDF форматом, але трійки на відміну від Turtle подані у вигляді словників. Також наявні простори імен. Конвертація JSON-LD виглядає як на лістингу 3.5:

Лістинг 3.5

```
import json


def json_ld_to_nt(json_ld_data):
    triples = []


    def parse_jsonld_object(obj, subject=None):
        for key, value in obj.items():
            if isinstance(value, list):
                for item in value:
                    parse_jsonld_object({key: item}, subject)
            elif isinstance(value, dict):
                parse_jsonld_object(value, subject)
            else:
                predicate = key
                obj_value = value


                if isinstance(obj_value, str) and obj_value.startswith("http"):
                    obj_value = f"<{obj_value}>"
                elif isinstance(obj_value, (int, float)):
                    obj_value = f'"{obj_value}"'
                elif isinstance(obj_value, bool):
                    obj_value = "true" if obj_value else "false"
                else:
                    obj_value = f'"{obj_value}"'
```

```
            triples.append((subject, predicate, obj_value))


    parse_jsonld_object(json_ld_data)


    nt_lines = [f"{triple[0]} {triple[1]} {triple[2]} ." for triple in triples]
    return '\n'.join(nt_lines)
```

3.7. Векторизація графів знань

Наступним етапом підготовки графу знань до навчання графової нейронної мережі є етап векторизації або створення так званих вбудовувань. Процес векторизації може бути реалізований у різний спосіб, наприклад Python має у репозиторії pip доволі потужну бібліотеку node2vec. Однак, під час розробки цієї атестаційної роботи була використана бібліотека PyTorch, тому побудова коду для векторизації відбувалася з використанням її засобів (лістинг 3.6):

Лістинг 3.6:

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Node2Vec(nn.Module):
    def __init__(self, num_nodes, embedding_dim, p, q):
        super(Node2Vec, self).__init__()
        self.num_nodes = num_nodes
        self.embedding_dim = embedding_dim
        self.p = p
        self.q = q


        self.W = nn.Embedding(num_nodes, embedding_dim)
```

```
def forward(self, nodes):
    walks = self.generate_walks(nodes)
    nodes_embedding = self.get_node_embeddings(walks)
    return nodes_embedding

def generate_walks(self, nodes):
    walks = []
    for node in nodes:
        walk = []
        for step in range(self.p + self.q):
            if step < self.p:
                walk.append(node)
            elif step >= self.p and step < self.p + self.q:
                walk.append(self.choose_neighbor(walk[-1]))
            else:
                walk.append(None)
        walks.append(walk)
    return walks

def choose_neighbor(self, node):
    neighbors = []
    for neighbor in self.W[node].t().nonzero(as_tuple=False):
        neighbors.append(neighbor[1])
    if neighbors:
        return neighbors[torch.randint(0, len(neighbors))]
    else:
        return None

def get_node_embeddings(self, walks):
    nodes_embedding = []
    for walk in walks:
```

```
            nodes_embedding.append(self.get_walk_embedding(walk))
        return torch.stack(nodes_embedding)


    def get_walk_embedding(self, walk):
        walk_embedding = torch.zeros(self.embedding_dim)
        for node in walk:
            if node is not None:
                walk_embedding += self.W[node]
        return walk_embedding / len(walk)
```

3.8. Базова GNN

В рамках цієї атестаційної роботи була створена базова графова нейронна мережа з використанням бібліотеки PyTorch у лістингу 3.7:

Лістинг 3.7

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class GNNLayer(nn.Module):
    def __init__(self, in_features, out_features):
        super(GNNLayer, self).__init__()
        self.linear = nn.Linear(in_features, out_features)


    def forward(self, x, edge_index):
        # Агрегування повідомлень від сусідів
        x = self.linear(x)
        x = F.relu(x)
        out = torch.matmul(edge_index, x)


        # Нормування за кількістю сусідів
```

```
        row, col = edge_index
        deg = torch.bincount(col, minlength=x.size(0))
        deg_inv_sqrt = deg.pow(-0.5)
        deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
        out = deg_inv_sqrt.view(-1, 1) * out

        return out

class GNN(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super(GNN, self).__init__()
        self.layer1 = GNNLayer(in_features, hidden_features)
        self.layer2 = GNNLayer(hidden_features, out_features)

    def forward(self, x, edge_index):
        x = self.layer1(x, edge_index)
        x = self.layer2(x, edge_index)
        return x
```

3.9. Навчання та тестування

Для навчання та тестування був завантажений граф знань з ресурсу freebase (fb15k). Цей датасет містить інформацію про людей, а саме їх професії, вподобання тощо, у вигляді трійок. Загальна довжина навчального датасету склала 17500 записів. Загальна довжина тестової вибірки склала 1000 записів. Загальна довжина вибірки склала 1000 записів. На графіках нижче наведені функції втрат процесу навчання та тестування (рис. 3.1, 3.2):

Рисунок 3.1. Функція втрат під час навчання GNN

Рисунок 3.2. Функція втрат під час тестування GNN

Як видно з графіку, хоча процес навчання характеризувався високими значеннями функції втрат, проте за відносно короткий час (100 епох), мережа навчилася достатньо добре аби на тестовій вибірці показати гарний результат.

3.10. Висновок до розділу

За результатами аналізу різних форматів графів знань в якості джерела вхідних даних для навчання нейронної мережі було обрано формат N-Triples. Цей формат є найпростішим з точки зору формалізації. Для навчання

нейронної мережі вхідні графи знань необхідно перетворювати у векторний формат, що було запропоновано зробити за допомогою алгоритму node2vec. Отримані вбудування вузлів графа були використані як вхідні дані для графової нейронної мережі GNN та були отримані доволі якісні результати навчання, що доведено відносно малими значеннями функції втрат на тестовій виборці даних. Таким чином, встановлено, що такий спосіб використання графів знань у якості джерела вхідних даних для систем машинного навчання є припустимим і може бути використаним на практиці.

ВИСНОВОК

В ході виконання атестаційної роботи були проаналізовані різні формати представлення знань у системах штучного інтелекту. Був обґрунтований вибір RDF N-Triples, як кращого формату для використання у якості джерела вхідних даних для навчання нейронної мережі. Був розглянутий спосіб підготовки вхідних даних та архітектура базової нейронної мережі GNN. Для побудови RDF N-Triples з інших форматів графів знань, був створений набір методів конвертації. Була створена реалізація алгоритму node2vec для перетворення знань у вхідні дані для нейронної мережі. Базова графова нейронна мережа GNN була навчана та перевірена на даних графу знань RDF N-Triples. Розроблений спосіб використання графів знань для машинного та глибокого навчання, може бути розповсюджений на графові нейронні мережі, архітектура яких є похідною від базової GNN.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Csikszentmihalyi M. Flow: The psychology of optimal experience / Csikszentmihalyi M. – New York: Harper Perennial, 1990. – URL: https://core.ac.uk/download/42629223.pdf
2. Toward A Unified Modeling of Learner's Growth Process and Flow Theory / [Challco, G. & Andrade, F. & Borges et al.] // Educational Technology & Society. – 2016. – 19(2). – P. 215-227. – URL: https://www.researchgate.net/publication/301233159_Toward_A_Unified_Modeling_of_Learner's_Growth_Process_and_Flow_Theory
3. Akers, S. B. A graphical approach to production scheduling problems / Akers, S. B. // Operations Research. – 1956. – Vol. 4, issue 2. – P. 244–245
4. Sowa, J. F. Conceptual graphs for a database interface / Sowa, J. F. // IBM Journal of Research and Development. – 1976. – 20:4. – P. 336-357.
5. Colmerauer, A. The birth of Prolog / Colmerauer, A., Roussel, P. // November, – 1992. – URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.7438&rep=rep1&type=pdf
6. Pearl, J. Bayesian Networks / Pearl, J., Russell, S. // Lecture notes, University of California. – URL: https://www.cs.ubc.ca/~murphyk/Teaching/CS532c_Fall04/Papers/hbtnn-bn.pdf
7. Sowa, J. F. Semantic Networks. In Stuart C Shapiro. Encyclopedia of Artificial Intelligence / Sowa, J. F. – Wiley, New York. – 1987. – URL: http://www.jfsowa.com/pubs/semnet.htm
8. Sowa, J. F. Principles of Semantic Networks. / Sowa, J. F. SanMateo. – 1992.
9. Berners-Lee, T. The Semantic Web / Berners-Lee, T., Hendler, J., Lassila, O. // Scientific American Magazine. – 2001.
10. Web Ontology Language (OWL). W3C Semantic Web : website. – URL: https://www.w3.org/OWL/

11. https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_5-GNNs.pdf
12. "The Graph Neural Network Model" by Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, published in 2009: https://ro.uow.edu.au/infopapers/3165/
13. Thomas N. Kipf, Max Welling "Semi-Supervised Classification with Graph Convolutional Networks" 2016: https://arxiv.org/pdf/1609.02907.pdf
14. Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, Alexander J. Smola "Deep Sets" 2017:https://papers.nips.cc/paper_files/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html
15. Guoray Cai, Yimu Pan "Understanding the Imperfection of 3D point Cloud and Semantic Segmentation algorithms for 3D Models of Indoor Environment" 2017: https://www.researchgate.net/figure/PointNet-framework-Qi-et-al-2017_fig2_361240571
16. William L. Hamilton, Rex Ying, Jure Leskovec "GraphSAGE: Inductive Representation Learning on Large Graphs" 2017: https://ar5iv.labs.arxiv.org/html/1706.02216
17. Ryan L. Murphy, Vinayak Rao, Balasubramaniam Srinivasan, Bruno Ribeiro "Janossy Pooling: Learning Deep Permutation-invariant Functions for Variable-size Inputs" 2018: https://ar5iv.labs.arxiv.org/html/1811.01900
18. https://pabloinsente.github.io/the-recurrent-net
19. Lei Xu, Xiaoqing Ru, Rong Song "Application of Machine Learning for Drug–Target Interaction Prediction"
20. https://www.ams.org/journals/bull/2006-43-04/S0273-0979-06-01126-8/home.html
21. Pham, DH, Zhang, C., Yin, C «Using zebrafish to model liver diseases-Where do we stand?» 2017
22. Каймін Хе , Сян'ю Чжан , Шаоцін Рен , Цзянь Сунь «Deep Residual Learning for Image Recognition» 2016

23. https://pubs.giss.nasa.gov/abs/li03700v.html
24. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio «Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling» 2014
25. Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, David L. Dill, Leonardo de Moura “Learning a SAT Solver from Single-Bit Supervision” 2018
26. Facebook Graph Search. Wikipedia : website. – URL: https://en.wikipedia.org/wiki/Facebook_Graph_Search
27. Malhotra, M. Evolution of Knowledge Representation and Retrieval Techniques. International Journal of Intelligent Systems and Applications. / Malhotra, M., Gopalakrishnan Nair, T.R // 2015. Vol 07. P.8-28. – URL: https://www.researchgate.net/publication/280578562_Evolution_of_Knowledge_Representation_and_Retrieval_Techniques
28. Hobbes, T. Elements of Law, Natural and Political / Hobbes, T. // Routledge. – 1969
29. Constructivist Learning Theory. Educational Technology : website. – URL: https://educationaltechnology.net/constructivist-learning-theory/
30. Davis, R. What is a Knowledge Representation? / Davis, R., Shrobe, H., and Szolovits P. // AI Magazine. – 14(1):17-33, 1993. – URL: https://groups.csail.mit.edu/medg/ftp/psz/k-rep.html#kr
31. Wang, Y. On Cognitive Informatics / Wang, Y. – 2003. – Brain and Mind, 4. – P. 151-167
32. Kiely, K. Cognitive function/ Kiely, K. – In Michalos. – Encyclopedia of Quality of Life and Well-Being Research. – 2014. – Springer. – P. 974–978
33. What are cognitive abilities and skills, and can we boost them? Sharpbrains : website. – URL: https://sharpbrains.com/what-are-cognitive-abilities/
34. Bloom, B.S. Taxonomy of Educational Objectives, Handbook: The Cognitive Domain / Bloom, B.S. – 1956. – David McKay, New York

35. Anderson, L. W. A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives / Anderson, L. W., Krathwohl, D. R. – 2001. – Allyn & Bacon. – Boston, MA (Pearson Education Group)
36. Longo, F. Learning and Memory: How it Works and When it Fails. Stanford. Youtube : website. URL: https://www.youtube.com/watch?v=a_HfSnQqeyY

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

		Позначення	Найменування	Кільк. аркушів	Примітка
	1				
	2		Документація		
	3				
	4	**ІТКІ.ДП 18.01.ДА.ПЗ**	Пояснювальна записка	80	
	5				
	6		Диск CD-R з презентацією	1	

Зм	Лист	№ докум.	Підпис	Дата	**ІТКІ.ДП 18.01.ДА.ПЗ**			
Розроб.		Фурсикова			**Матеріали кваліфі-каційної роботи**	Літ.	Аркуш	Аркушів
Керівник		Коротенко					1	1
Рецензент		Ширін				НТУ «ДП» 8; 126м-22з-2		
Н.контр.		Коротенко						
Зав. каф.		Гнатушенко						

ДОДАТОК Б

КОД ПРОГРАМИ

convert.py

```
def turtle_ntriples_bridge(turtle_data):
    lines = turtle_data.split('\n')
    triples = []

    for line in lines:
        line = line.strip()
        if not line or line.startswith("#"):
            continue

        subject, predicate, obj = line.split(maxsplit=2)

        # Обробка порожніх вузлів
        if obj == "[]":
            obj = "_:blank_node"
        elif obj.startswith("[") and obj.endswith("]"):
            obj = "_:blank_node"

        triple = f"{subject} {predicate} {obj} ."
        triples.append(triple)

    ntriples_data = '\n'.join(triples)
    return ntriples_data

import xml.etree.ElementTree as ET

def parse_owl(owl_file):
    triples = []

    # Зчитуємо XML-структуру з OWL-файлу
    tree = ET.parse(owl_file)
```

```
    root = tree.getroot()

    # Отримуємо простори імен
    namespaces = {'owl': 'http://www.w3.org/2002/07/owl#', 'rdf': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'}

    # Проходимося по всіх елементах у дереві
    for elem in root.iter():
        if '}' in elem.tag:
            # Розбираємо тег у вигляді {namespace}tag
            tag_parts = elem.tag.split('}')
            namespace = tag_parts[0][1:]
            tag = tag_parts[1]

            # Перетворюємо простір імен у префікс, наприклад, owl:Class
            prefix = [key for key, value in namespaces.items() if value == namespace][0]
            full_tag = f"{prefix}:{tag}"

            # Отримуємо значення елементу
            value = elem.text if elem.text else ''

            # Формуємо триплет (суб'єкт, предикат, об'єкт)
            triple = f"{elem.attrib['{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about']} {full_tag} {value} ."

            # Додаємо триплет до списку
            triples.append(triple)

    return triples

import json

def parse_owl_json(owl_json)
    triples = []
```

```
    for entry in owl_json
        subject = entry.get('subject', '')
        predicate = entry.get('predicate', '')
        obj = entry.get('object', '')

        if subject and predicate and obj
            triples.append((subject, predicate, obj))

    return triples



def kif_to_nt(kif_text):
    triples = []

    lines = kif_text.split('\n')
    for line in lines:
        line = line.strip()
        if line.startswith('(') and line.endswith(')'):
            parts = line[1:-1].split()
            if len(parts) == 3 and parts[1].lower() == '=>' and '(' not in parts[2]:
                subject = f"<{parts[0]}>"
                predicate = "<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>"
                obj = f"<{parts[2]}>"
                triples.append((subject, predicate, obj))

    return triples

def convert_to_nt(triples):
    nt_lines = [f"{triple[0]} {triple[1]} {triple[2]} ." for triple in triples]
    return '\n'.join(nt_lines)

def json_ld_to_nt(json_ld_data):
    triples = []
```

```
    def parse_jsonld_object(obj, subject=None):
        for key, value in obj.items():
            if isinstance(value, list):
                for item in value:
                    parse_jsonld_object({key: item}, subject)
            elif isinstance(value, dict):
                parse_jsonld_object(value, subject)
            else:
                predicate = key
                obj_value = value

                if isinstance(obj_value, str) and obj_value.startswith("http"):
                    obj_value = f"<{obj_value}>"
                elif isinstance(obj_value, (int, float)):
                    obj_value = f'"{obj_value}"'
                elif isinstance(obj_value, bool):
                    obj_value = "true" if obj_value else "false"
                else:
                    obj_value = f'"{obj_value}"'

                triples.append((subject, predicate, obj_value))

    parse_jsonld_object(json_ld_data)

    nt_lines = [f"{triple[0]} {triple[1]} {triple[2]} ." for triple in triples]
    return '\n'.join(nt_lines)
```

GNN.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
class GNNLayer(nn.Module):
    def __init__(self, in_features, out_features):
        super(GNNLayer, self).__init__()
        self.linear = nn.Linear(in_features, out_features)

    def forward(self, x, edge_index):
        # Агрегування повідомлень від сусідів
        x = self.linear(x)
        x = F.relu(x)
        out = torch.matmul(edge_index, x)

        # Нормування за кількістю сусідів
        row, col = edge_index
        deg = torch.bincount(col, minlength=x.size(0))
        deg_inv_sqrt = deg.pow(-0.5)
        deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
        out = deg_inv_sqrt.view(-1, 1) * out

        return out

class GNN(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super(GNN, self).__init__()
        self.layer1 = GNNLayer(in_features, hidden_features)
        self.layer2 = GNNLayer(hidden_features, out_features)

    def forward(self, x, edge_index):
        x = self.layer1(x, edge_index)
        x = self.layer2(x, edge_index)
        return x
```

node2vec.py

```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F


class Node2Vec(nn.Module):
    def __init__(self, num_nodes, embedding_dim, p, q):
        super(Node2Vec, self).__init__()
        self.num_nodes = num_nodes
        self.embedding_dim = embedding_dim
        self.p = p
        self.q = q

        self.W = nn.Embedding(num_nodes, embedding_dim)

    def forward(self, nodes):
        walks = self.generate_walks(nodes)
        nodes_embedding = self.get_node_embeddings(walks)
        return nodes_embedding

    def generate_walks(self, nodes):
        walks = []
        for node in nodes:
            walk = []
            for step in range(self.p + self.q):
                if step < self.p:
                    walk.append(node)
                elif step >= self.p and step < self.p + self.q:
                    walk.append(self.choose_neighbor(walk[-1]))
                else:
                    walk.append(None)
            walks.append(walk)
        return walks

    def choose_neighbor(self, node):
        neighbors = []
```

```
    for neighbor in self.W[node].t().nonzero(as_tuple=False):
        neighbors.append(neighbor[1])
    if neighbors:
        return neighbors[torch.randint(0, len(neighbors))]
    else:
        return None


def get_node_embeddings(self, walks):
    nodes_embedding = []
    for walk in walks:
        nodes_embedding.append(self.get_walk_embedding(walk))
    return torch.stack(nodes_embedding)


def get_walk_embedding(self, walk):
    walk_embedding = torch.zeros(self.embedding_dim)
    for node in walk:
        if node is not None:
            walk_embedding += self.W[node]
    return walk_embedding / len(walk)
```

ДОДАТОК В

ВІДГУК
на комплексну кваліфікаційну роботу рівня магістра «Дослідження особливостей процесів використання графів знань у системах на базі графових нейронних мереж» студента групи 126м-22з-2 Фурсикової Тетяни Володимирівни

1 Мета даної кваліфікаційної роботи – дослідження використання наявних графів знань під час навчання систем штучного інтелекту..

2 Обрана тема актуальна тому, що системи так званого «semantic AI» спиралися на накопичення інформації у вигляді знань та створення систем обробки, що спираються на логіки різних порядків. Такі системи могли бути доволі ефективними, з одного боку, як частини систем прийняття рішень, а з іншого – як інтелектуальні помічники, проте накопиченні знання не завжди могли пояснити відхилення практичних результатів від теоретичних і тому точність прогнозу в таких системах могла б бути більш менш прийнятною у добре досліджених та описаних галузях знань.

3 Тема кваліфікаційної роботи відповідного рівня безпосередньо пов'язана з об'єктом діяльності магістра спеціальності 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології» – створення інформаційних технологій різного призначення і застосування.

4 Явища і процеси, що досліджуються в даній кваліфікаційній роботі і обрані для моделювання, оцінювання та реалізації – віднесені в освітньо-кваліфікаційній характеристиці магістрів до класу дослідних та евристичних, рішення яких заснована на знаково-понятійних уміннях.

5 Робота складається з трьох розділів. Перший розділ присвячений аналізу теми дослідження та постановці задачі. У другому розділі наведено проектну складову вирішення завдання. Третій розділ присвячено дослідженню особливостей процесів використання графів знань у системах на базі графових нейронних мереж. Оригінальність отриманих в роботі наукових результатів та їх наукова новизна полягають у наступному:

- виконано аналіз різних форматів графів знань;
- в якості джерела вхідних даних для навчання нейронної мережі було обрано формат N-Triples;
- отримані вбудування вузлів графа були використані як вхідні дані для графової нейронної мережі GNN та були отримані доволі якісні результати навчання, що доведено відносно малими значеннями функції втрат на тестовій виборці даних;
- для побудови RDF N-Triples з інших форматів графів знань, був створений набір методів конвертації.

6 Практичне значення результатів роботи полягає в обранні потрібних для вирішення поставленої задачі алгоритмів на основі порівняння групи найбільш відомих.

7 Практичні результати кваліфікаційної роботи отримані із застосуванням відповідних технічних і програмних засобів, мови Python, а також програмних продуктів MS Word і MS PowerPoint на інформаційно-технологічній платформі Windows.

8 Оформлення графічних матеріалів до кваліфікаційної роботи рівня магістр виконано на сучасному рівні і відповідає вимогам, що пред'являються до рівня виконання робіт даної кваліфікації.

9 Ступінь самостійності виконання кваліфікаційної роботи достатньо висока.

10 Незважаючи на велику кількість розглянутих у роботі форматів графів знань, було встановлено, що для навчання нейронної мережі вхідні графи знань необхідно перетворювати у векторний формат, що було запропоновано зробити за допомогою алгоритму node2vec.

Розроблені алгоритми достатньо кваліфіковано реалізовані автором за допомогою мови програмування Python.

У якості зауваження слід відзначити недостатньо повний опис розроблених алгоритмів.

У підсумку, можна зробити висновок, що дана кваліфікаційна робота є закінченим науковим дослідженням і в цілому заслуговує оцінки 96 (відмінно) та присвоєння здобувачу відповідної кваліфікації.

Керівник кваліфікаційної роботи,
проф. кафедри ІТКІ, д.т.н. Г.М. Коротенко

ДОДАТОК Г

РЕЦЕНЗІЯ
на комплексну кваліфікаційну роботу рівня магістра
«Дослідження особливостей процесів використання графів знань у системах на базі графових нейронних мереж»
студента групи 126м-22з-2 Фурсикової Тетяни Володимирівни

Розглянута робота присвячена дослідженню використання наявних графів знань під час навчання систем штучного інтелекту.

Завдання і зміст кваліфікаційної роботи відповідає головній цілі - перевірці знань і ступеня підготовленості студента за фахом 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології».

Зміст пояснювальної записки кваліфікаційної роботи відповідає необхідним критеріям та затвердженій темі.

Актуальність обраної теми обумовлена тим, що дослідження ефективності застосування інформаційних технологій і відповідних програмних засобів у створенні компонентів систем штучного інтелекут продовжують розвиватися на базі розширення їхнього спектру застосувань.

Повнота і глибина вирішення задач, поставлених в завданні на кваліфікаційну роботу є достатньою.

Оформлення пояснювальної записки кваліфікаційної роботи виконано в повній відповідності з діючими стандартами і нормативними вимогами.

Наукова новизна результатів кваліфікаційної роботи роботи визначається тим, що отримані вбудування вузлів графа знань були використані як вхідні дані для графової нейронної мережі GNN та були отримані доволі якісні результати навчання, що доведено відносно малими значеннями функції втрат на тестовій виборці даних.

Практичне значення результатів роботи полягає у тому, що запропоновано спосіб використання графів знань у якості джерела вхідних даних для систем машинного навчання, що може бути використано на практиці.

До числа загальних зауважень і недоліків роботи слід віднести:

1) не до кінця розкриті функціональні особливості реалізації та результатів практичного застосування розробленої технології;

2) дещо спрощений опис кінцевих результатів роботи.

Однак, зазначені зауваження не здійснюють істотного впливу на підсумкові результати кваліфікаційної роботи і не знижують її безумовну практичну та наукову цінність.

Таким чином, слід зробити висновок, що кваліфікаційна робота в цілому заслуговує оцінки «_____», а її виконавець присвоєння відповідної кваліфікації.

Рецензент, доцент кафедри програмного
забезпечення комп'ютерних систем НТУ
«ДП», к.т.н. А.Л. Ширін