Федеральное государственное образовательное бюджетное учреждение

высшего образования

«**Финансовый университет при Правительстве Российской Федерации»**

**(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)**

Департамент анализа данных и машинного обучения

**Дисциплина: «Теория сложных сетей в экономике»**

Направление подготовки: «Прикладная математика и информатика»

Профиль: «Анализ данных и принятие решений в экономике и финансах»

Факультет информационных технологий и анализа больших данных

Форма обучения очная

Учебный 2021/2022 год, 6 семестр

Курсовая работа на тему:

**«Построение рекомендательной системы с помощью графовых нейронных сетей использующих атрибуты товаров и пользователей»**

Выполнил:

студент гр. ПМ19-2

Фан А. Ч.

Научный руководитель:

к.т.н., доцент Догадина Е. П.

Москва – 2022

[Введение 2](#_Toc102500923)

[1. Предметная область 3](#_Toc102500924)

[1.1. Рекомендательные системы 3](#_Toc102500925)

[1.1.1 Методы, используемые при создании рекомендательных систем. 3](#_Toc102500926)

[1.1.1.1. Коллаборативная фильтрация 3](#_Toc102500927)

[1.1.1.2 Фильтрация на основе содержания 4](#_Toc102500928)

[1.2. Нейронные сети 4](#_Toc102500929)

[1.2.1. Задачи, решаемые нейронными сетями 4](#_Toc102500930)

[1.2.2. Модель искусственного нейрона 4](#_Toc102500931)

[1.2.3. Архитектура искусственных нейронных сетей 5](#_Toc102500932)

[1.2.3.1. Сети прямого распространения 6](#_Toc102500933)

[1.2.3.2. Рекуррентные сети 7](#_Toc102500934)

[1.2.3.3. Сверточные сети 7](#_Toc102500935)

[1.2.4. Обучение искусственной нейронной сети 8](#_Toc102500936)

[1.3. Графовые нейронные сети 9](#_Toc102500937)

[1.3.1. Архитектура GNN 10](#_Toc102500938)

[1.3.1.1. Сверточный оператор 11](#_Toc102500939)

[1.3.1.2. Оператор отбора соседей 12](#_Toc102500940)

[2. Формализация задачи и используемые алгоритмы 12](#_Toc102500941)

[2.1. Задача обучения представлений узлов графа 12](#_Toc102500942)

[2.2. Функция потерь 13](#_Toc102500943)

[2.3. модель Pinsage 13](#_Toc102500944)

[3. Технологическая часть 14](#_Toc102500945)

[3.1. Обзор данных 14](#_Toc102500946)

[3.2. Подготовка данных 15](#_Toc102500947)

[3.3. Адаптированная модель Pinsage 15](#_Toc102500948)

[3.4. Обучение модели 16](#_Toc102500949)

[3.5. Рекомендательная система 16](#_Toc102500950)

[Заключение 19](#_Toc102500951)

[Список использованных источников 20](#_Toc102500952)

[Пояснительная записка 21](#_Toc102500953)

# Введение

За последние несколько десятилетий, с появлением Youtube, Amazon, Netflix и многих других подобных веб-сервисов, рекомендательные системы стали занимать все большее место в нашей жизни. От интернет-магазинов (предлагать покупателям товары, которые могут их заинтересовать) до онлайн-рекламы (предлагать пользователям правильный контент, соответствующий их предпочтениям), рекомендательные системы сегодня можно встретить на каждом углу интернета.

Качественные рекомендации могут повысить доход компании, так как они заинтересуют клиента в покупке в случае интернет-магазинов, либо же привлекут новых клиентов за счет уровня сервиса в случае стриминговых сервисов.

Данная область активно развивается и глубокое обучение, как один из главных трендов IT-сферы, стремительно внедряется в эти системы. Все перечисленное выше подтверждает актуальной темы проекта.

Цель курсовой работы заключается в построении рекомендательной системы с помощью графовых нейронных сетей.

Данные взяты из соревнования «H&M Personalized Fashion Recommendations».

В работе будет использовать алгоритм «Pinsage» от компании «Pinterest», который сочетает эффективное случайное блуждание и свертку графа.

# 1. Предметная область

## 1.1. Рекомендательные системы

Рекомендательная система — это программное обеспечение, направленное на предложение релевантных товаров (items) пользователям.

### 1.1.1 Методы, используемые при создании рекомендательных систем.

Для решения задачи рекомендательной системы существует две традиционные группы методов: методы коллаборативной фильтрации и методы фильтрации на основе содержания.

#### 1.1.1.1. Коллаборативная фильтрация

Методы коллаборативной фильтрации — это методы, генерирующие рекомендации на основе данных о взаимодействиях пользователей и товаров.

Основная идея методов коллаборативной фильтрации, заключается в том, что этих прошлых взаимодействий пользователя и товаров достаточно для обнаружения похожих пользователей и / или похожих товаров и составления прогнозов на основе этих предполагаемых близостей.

Класс методов коллаборативной фильтрации делится на две подкатегории, которые обычно называются подходами, основанными на памяти, и подходами, основанными на моделях. Подходы, основанные на памяти, напрямую работают со значениями записанных взаимодействий, не предполагая никакой модели, и основаны на поиске ближайших соседей. Подходы, основанные на моделях, предполагают модель, с помощью которой можно получить новые представления пользователей и элементов. На основе новых представлений делаются рекомендации.

Главное преимущество коллаборативных подходов заключается в том, что они не требуют информации о пользователях или товаров, и поэтому их можно использовать во многих ситуациях. Более того, чем больше пользователей взаимодействуют с товарами, тем более точными становятся новые рекомендации: для фиксированного набора пользователей и товаров новые взаимодействия, регистрируемые с течением времени, приносят новую информацию и делают систему все более эффективной.

У данного класса методов есть очевидный недостаток. Рекомендательная система, построенная на основе коллаборативной фильтрации, не сможет посоветовать что-то новым пользователям или порекомендовать товары, которые еще никто не покупал. Существуют различные стратегии для решения этой проблемы: случайная стратегия, стратегия максимального ожидания, исследовательская стратегия и другие.

#### 1.1.1.2 Фильтрация на основе содержания

В отличие от коллаборативных методов, которые полагаются только на взаимодействие пользователя и элемента, подходы, основанные на содержании, используют признаковое описание пользователей или элементов, что позволяет избежать проблем с новыми пользователями или товарами. Если мы рассмотрим пример системы рекомендаций фильмов, признаками пользователя могут быть, например, возраст, пол, место работы или любая другая личная информация, а признаками элементов могут быть продолжительность, рейтинг, кол-во просмотром и другие характеристики фильмов.

## 1.2. Нейронные сети

Нейронные сети - подмножество алгоритмов машинного обучения (глубокое обучение). Их название и структура вдохновлены человеческим мозгом, имитируя способ, которым биологические нейроны передают сигналы друг другу.

### 1.2.1. Задачи, решаемые нейронными сетями

Нейронные сети можно адаптировать для решения многих задач машинного обучения, но наиболее эффективно они показали себя в следующих задачах:

* Классификация изображений
* Распознавание речи
* Распознавание рукописного текста
* Улучшение качества машинного перевода с одного языка на другой

### 1.2.2. Модель искусственного нейрона

Искусственный нейрон – математическая модель биологического нейрона, использующаяся как базовый элемент в нейронных сетях.

Искусственный нейрон принимает на вход вектор значений, производит скалярное произведение векторов весов и входных значений, далее полученное значение подается как аргумент в функцию активации:

, где n - размерность входного вектора, — вес i-го входа нейрона, — значение, поступающее на i-й вход нейрона;

, где Y — выходное значение нейрона, f(S) —функция активации.

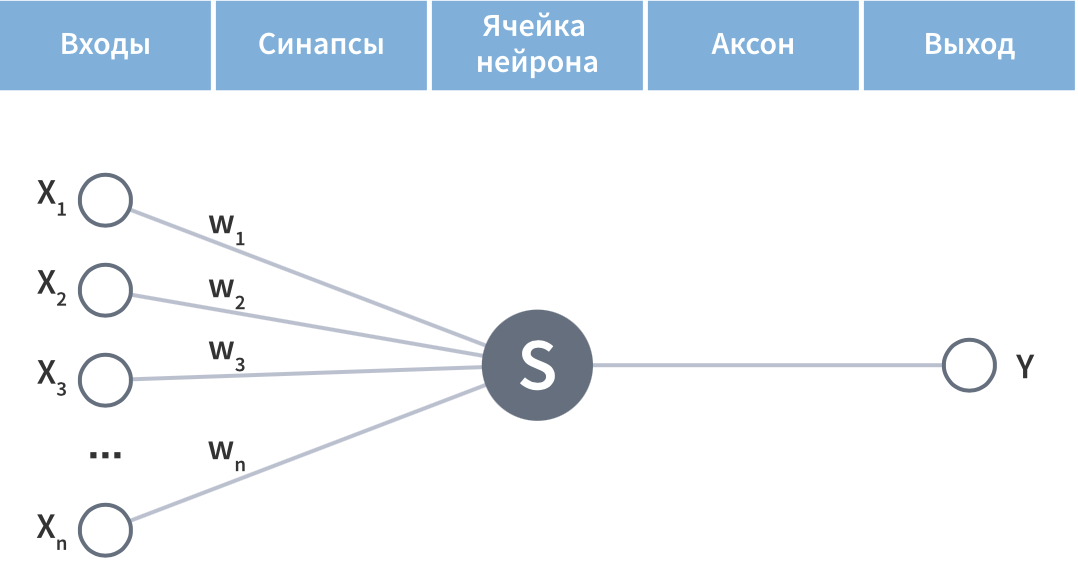


Рисунок 1. Модель искусственного нейрона.

### 1.2.3. Архитектура искусственных нейронных сетей

Еще один ключевой аспект нейронных сетей – задание архитектуры. Под архитектурой̆ понимается общая структура сети: сколько в ней̆ должно быть блоков и как эти блоки соединены между собой.

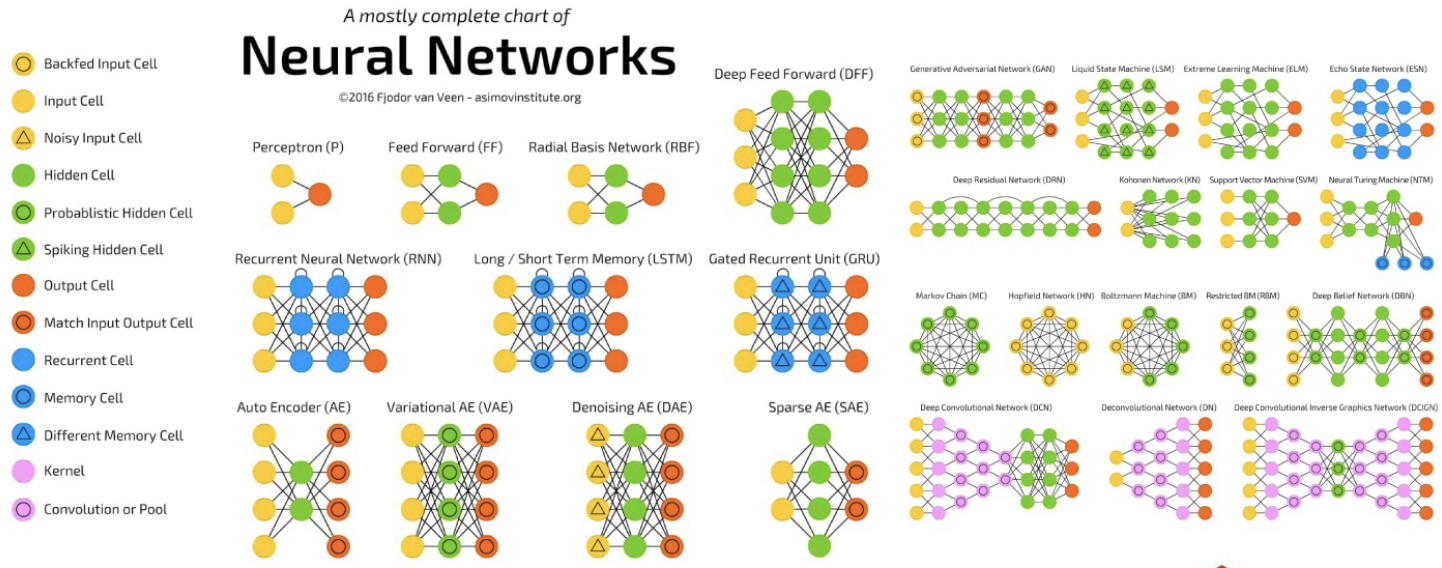
Большинство нейронных сетей организовано в виде групп блоков, именуемых слоями. В большинстве архитектур нейронных сетей слои собраны в цепочку, так что каждый слой является функцией от предыдущего. 

Рисунок 2. Многообразие архитектур нейронных сетей.

#### 1.2.3.1. Сети прямого распространения

Сети прямого распространения – архитектура, при которой информация движется в одном направлении (от входа к выходу). Простейшим примером является искусственный нейрон или перцептрон, описанный ранее. На практике же используют многослойные перцептроны (несколько нейронов в слое).

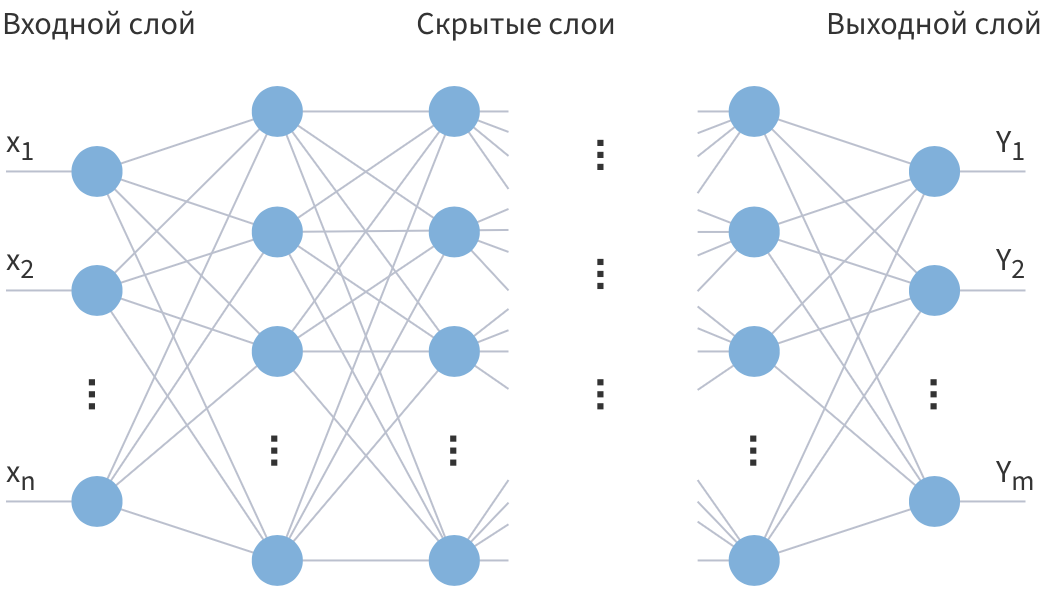


Рисунок 3. Многослойный перцептрон.

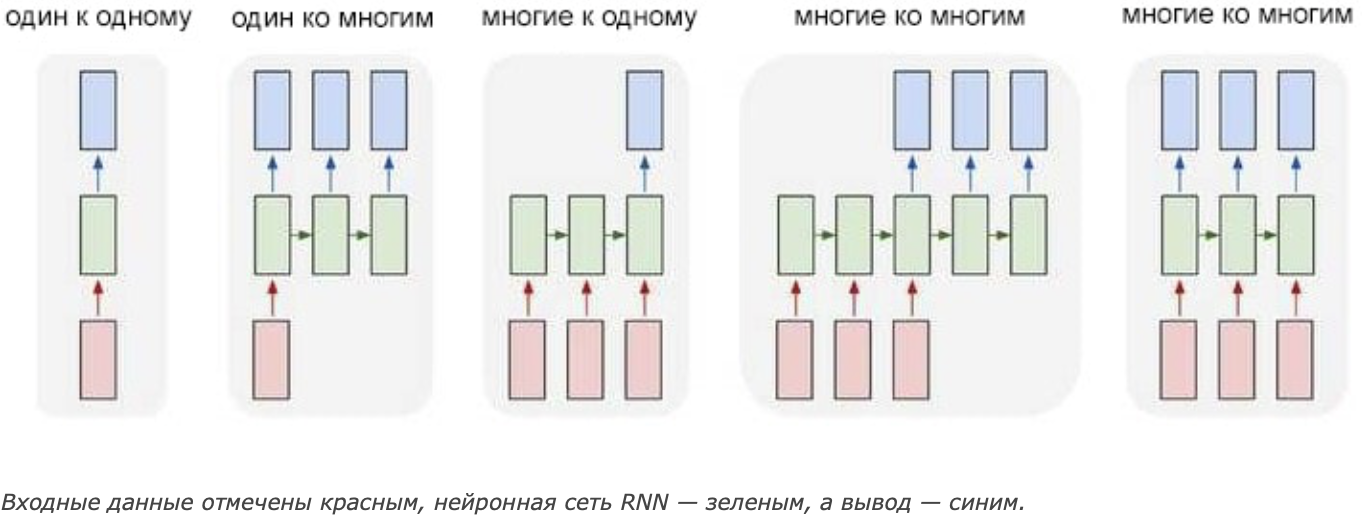
Эта архитектура, широко используемая в связи своей простоты, имеет характерную проблему при работе с большими данными:

* Большое кол-во параметров-весов для обучения, например для сети с одним скрытым слоем из 100 нейронов для обработки изображений 100\*100 ps потребуется 10 000 нейронов на входном слое. Получается уже 100 \* 10 000 = 1 000 000 параметров, не учитывая выходной слой. Для обучения такой сети потребуется очень много обучающих примеров. К тому же чем больше параметров, тем больше вероятность переобучения, т. е. модель запоминает обучающую выборку, но при этом плохо работает с данными, с которыми ни разу не встречалась.

#### 1.2.3.2. Рекуррентные сети

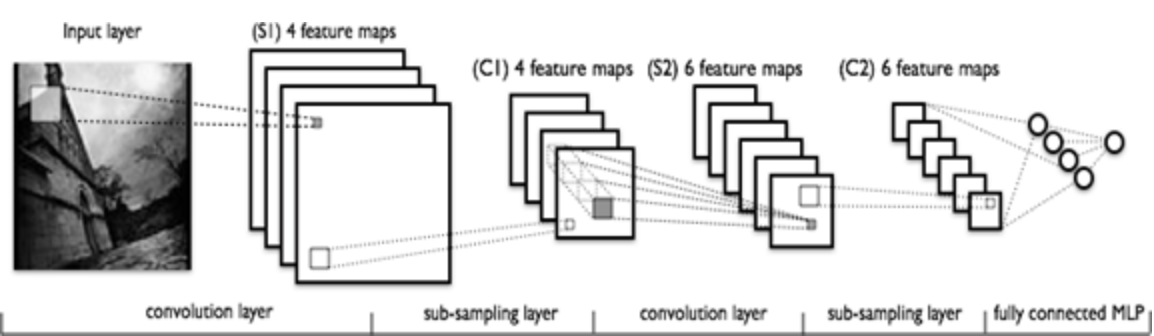
Рекуррентные нейронные сети, также известные как RNN, представляют собой класс нейронных сетей, которые позволяют использовать предыдущие выходные данные в качестве входных данных, имея при этом скрытые состояния.

Вот несколько примеров того, как может выглядеть рекуррентная нейронная сеть:



#### 1.2.3.3. Сверточные сети

Сверточные нейронные сети или CNN – архитектура нейронных сетей, использующая особые слои: сверточные (convolutional) слои, субдискретизирующие (subsampling) слои и многослойный перцептрон.



Сверточный слой извлекает из исходных данных признаки, используя оператор свертки.

Субдискретизирующий слой уменьшает размерность данных полученных после сверточного слоя.

Операция свертки в общем виде:

, где – преобразование Фурье, - обратное преобразование Фурье, ⊗ - операция свертки.

### 1.2.4. Обучение искусственной нейронной сети

Обучение нейронной сети — это поиск наилучшего набора весов для минимизации функции потерь.

Обратное распространение ошибки — это метод вычисления градиента функции потерь с использованием цепного правила.

Пусть:

– вектор взвешенных входов слоя L, где W – матрица весов, b – вектор смещений,

– вектор активаций слоя L, где f – нелинейная функция,

C – минимизируемая функция потерь

Тогда:

где j – индекс нейрона в слое L, k – индекс нейрона в слое L - 1

Когда градиенты w и b посчитаны, используют градиентные методы для поиска оптимальных значений:

Где – скорость обучения, настраиваемый параметр.

## 1.3. Графовые нейронные сети

Весомую долю информации (изображения, аудио, видео и т. д.) можно описать в виде графов. Граф — это структура данных, состоящая из двух компонентов: вершин и ребер. Граф G описывается множеством вершин (узлов) V и ребер E, которые он содержит G = (V, E).

Ориентированные/Неориентированные графы. Ребра в ориентированных графах все направлены от одного узла к другому, что предоставляет больше информации, чем неориентированные графы.

Двудольный граф представляет собой граф, узлы которого могут быть разделены на два непересекающихся и независимых множества U и Q, то есть каждое ребро соединяет один из узлов U с одним из узлов Q.

Граф часто представляется A, матрицей смежности. Если граф имеет n узлов, A имеет размерность (n × n).

Иногда узлы имеют набор признаков (например, профиль пол/возраст пользователя). Если узел имеет f признаков, то матрица признаков узлов X имеет размерность (n × f).

Для использования традиционных методов машинного обучения необходимо преобразовывать графовые данные в другие структуры данных, которые принадлежат Евклидову пространству. Это может привести к потере части информации, заключающейся во взаиморасположении узлов сети.

Графовые нейронные сети (GNN) позволяют работать напрямую с графом. Слои GNN генерирует малоразмерные представления узлов/ребер/графов в Евклидовом пространстве.

Задачи, решаемые с помощью GNN, разделяются на три уровня:

* Уровень узлов: Классификация узлов, кластеризация узлов, обучение представлений узлов
* Уровень ребер: Классификация ребер, предсказания ребер
* Уровень графов: Классификация графов, обучение представлений графов

### 1.3.1. Архитектура GNN

Архитектура GNN может включать в себя следующие операторы в одном слое, либо как отдельные слои:

* Отбор соседей для сверточного оператора (Sampling Operator)
* Сверточный/рекуррентный оператор – генерация представлений (Conv/Recurrent Operator)
* Оператор, упрощающий граф (Pooling Operator)

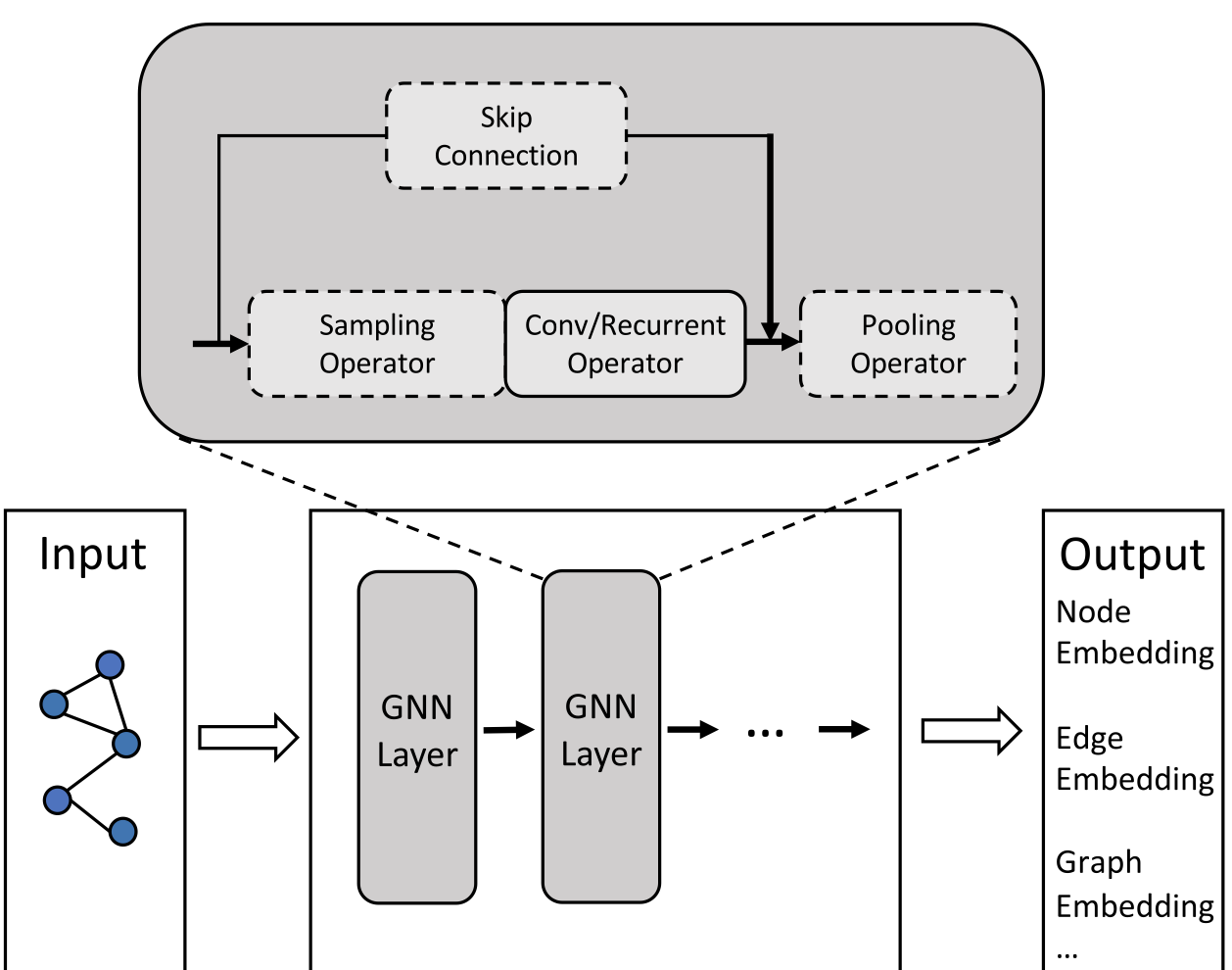


Рисунок 4. Архитектура GNN.

#### 1.3.1.1. Сверточный оператор

Основная идея сверточного оператора состоит в том, чтобы обобщить оператор свертки ( ) для графовых данных. Для достижения этой цели существует два подхода: спектральный (Spectral) и пространственный (Spatial).

Спектральные методы основаны на обработке графового сигнала x:

, где U – матрица собственных векторов нормализованной матрицы Лапласа где IN – единичная матрица размерности N\*N, D - диагональная матрица, которая содержит информацию о степени каждого узла, A – матрица смежности.

Пространственные методы итеративно генерируют представления узла на основе его информации его соседей:

где UPDATE – функция обновления представления узла, AGG – функция агрегации информации соседей.

#### 1.3.1.2. Оператор отбора соседей

Модели GNN агрегируют информацию соседей для каждого узла в каждом сверточном слое (в первом слое агрегируется информация соседей, во втором – информация соседей соседей). Интуитивно понятно, что, если мы построим несколько слоев GNN, кол-во соседей будет расти экспоненциально с увеличением глубины. Чтобы облегчить вычислительные затраты, эффективным и действенным способом является отбор соседей. Операторы для отбора соседей делятся на 3 вида:

* Отбор соседей фиксированного количества для каждого узла.
* Отбор множества допустимых соседей, используемого в слое
* Отбор соседей на основе создания подграфов

# 2. Формализация задачи и используемые алгоритмы

## 2.1. Задача обучения представлений узлов графа

Основной целью задачи обучения представлений узлов (Node representation learning) является получение новых представлений узлов в евклидовом пространстве с сохранением свойств топологических отношений между узлами.

Пусть:

vi – i-й узел графа G = (V, E), hi – представление узла в евклидовом пространстве, similarity(vi, vj) – функция определяющая схожесть двух вершин,

нужно найти такие представления, что similarity(vi, vj) .

## 2.2. Функция потерь

Нужно максимизировать схожесть представлений узлов и их соседей, для этого предлагаю ввести следующую функцию потерь:

где – представление i-го узла, – представление d-го схожего узла, – представление j-го не схожего узла. Основной смысл заключается в том, что мы хотим максимизировать скалярное произведение между схожими узлами (соседей) и минимизировать скалярное произведение между не схожими узлами.

Вычисление функции потерь для всех схожих и не схожих узлов потребует большое кол-во времени. Поэтому стоит определить положительные (выборка схожих узлов) и отрицательные выборки для каждого узла.

Для получения положительной выборки практикуется использование метода случайного блуждания.

Это метод, в котором происходит случайное блуждание от узла u на определенную глубину k, и все узлы, которые появляются на пути, считаются схожими узлами к u. Преимуществом метода является возможность определить самые схожие узлы к u по количеству «вхождений» в эти узлы.

Для получения отрицательной выборки наиболее релевантным способом является выборка из k случайных узлов. В больших графах вероятность того, что отобранные узлы являются отрицательными близка к 100%.

## 2.3. модель Pinsage

Pinsage – графовая сверточная сеть, использующая метод случайного блуждания. Модель представлена компанией «Pinterest» [3].

Для операции свертки модель использует следующие функции:

* , т. е. представление соседей прогоняется через простую нейронную сеть с функцией активации ReLU(x) = max(0, x), далее находится среднее значение или взвешенная сумма.
* , т. е. конкатенация нынешнего представления узла и значение, полученное после агрегации информации соседей, прогоняется через простую нейронную сеть с функцией активации ReLU.
* , т. е. для получения итогового представления модель отправляет в полносвязный и выходной слои.

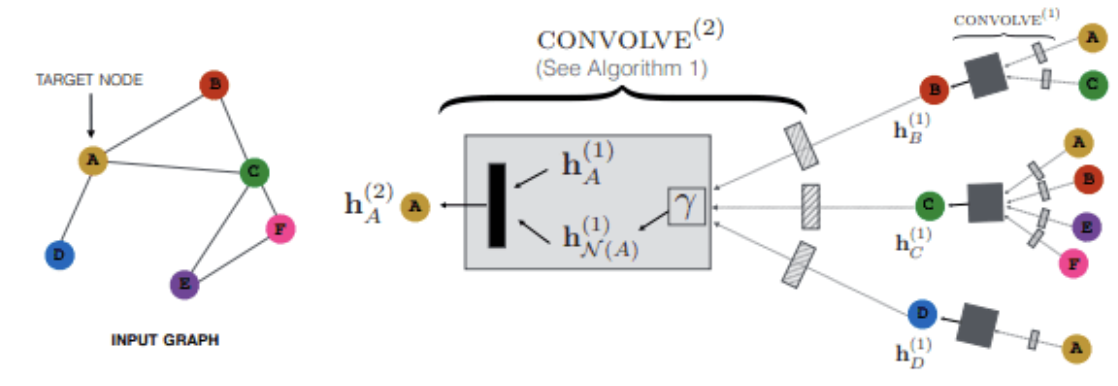


Рисунок 5. Принцип работы Pinsage

# 3. Технологическая часть

## 3.1. Обзор данных

В работе будут использоваться данные из соревнования «[H&M Personalized Fashion Recommendations](https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations)», которые включают в себя информацию о товарах, покупателей и историю транзакций.

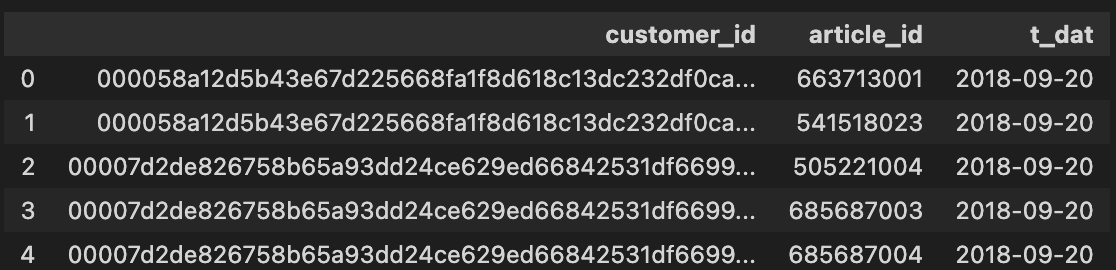


Рисунок 6. Пример истории транзакций.

Каждый товар имеет 11 категориальных признаков и уникальный идентификатор. Каждая транзакция включает в себя id товара, id пользователя и дату покупки.

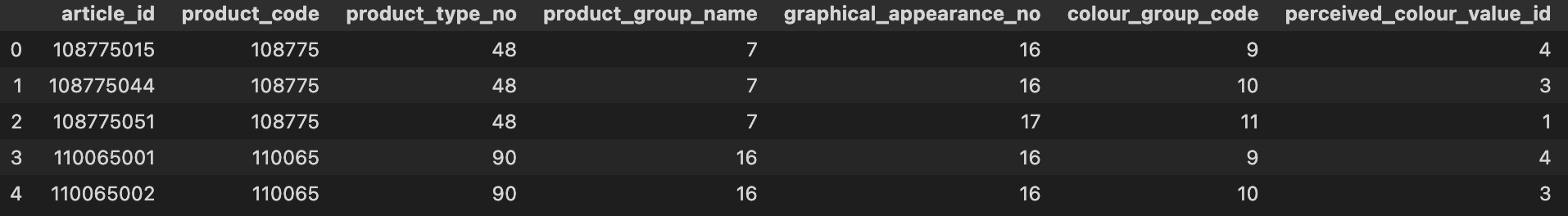


Рисунок 7. Первые 5 товаров в файле и их некоторые признаки.

## 3.2. Подготовка данных

Перед отправкой данных в модель необходимо закодировать признаки и представить подготовленные данные в виде двудольного графа пользователей и товаров. Для этого создадим вспомогающий модуль «builder.py» для создания графа и программу «data\_preprocessing.py», которая полностью подготавливает данные для модели.

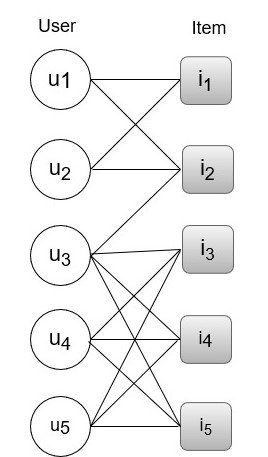


Рисунок 8. Пример двудольного графа.

## 3.3. Адаптированная модель Pinsage

Модель Pinsage предназначена для обучения с учителем, но большинство датасетов по рекомендательным системам не имеет вектор/матрицу ответом, включая выбранный для этой курсовой работы датасет. Поэтому будет использоваться адаптированная модель от разработчиков dgl, предназначенная для обучения без учителя. Эта реализация так же не использует полносвязный слой для получения итоговых представлений.

Для создания модели написаны следующие модули:

* «layers.py»: В этом модуле реализованы необходимые слои для модели
* «sampler.py»: В этом модуле реализованы различные функции генерации подвыборок

Модель определена в программе «model.py».

## 3.4. Обучение модели

Выбранные параметры для обучения:

* Кол-во случайных блужданий: 10
* Длина случайных блужданий: 2
* Кол-во сверточных слоев: 2
* Размерность итоговых представлений: 64
* Кол-во подвыборок в эпохе: 20000
* Кол-во узлов в подвыборке: 32
* Скорость обучения:

Модель обучалась 41 эпоху, итоговое значение функции потерь: 0.3630964728664607

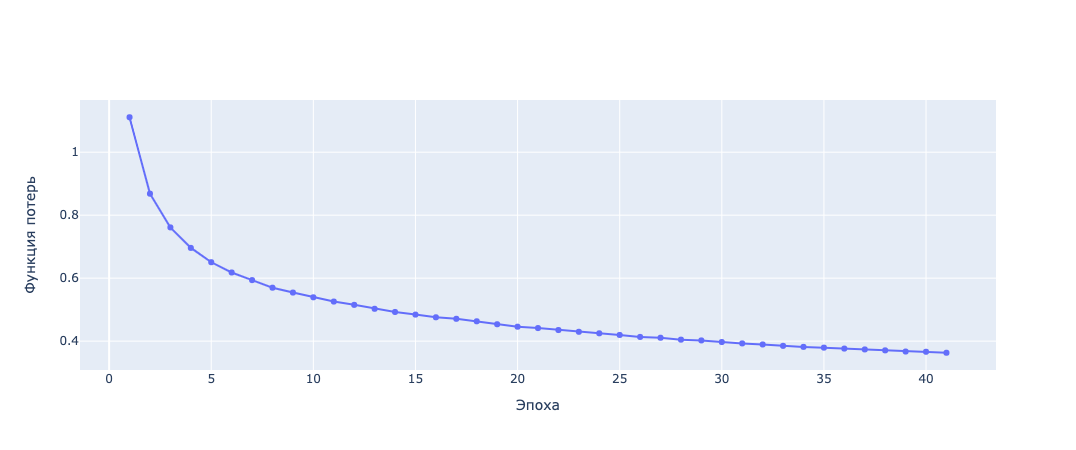


Рисунок 9. История обучения.

## 3.5. Рекомендательная система

Итогом обучения модели является получения новых представлений товаров. Чем больше скалярное произведение новых представлений двух вершин, тем более они схожи. На этом строится наша рекомендательная «оффлайн» система:

1. Обучения модели и генерация новых представлений вершин/товаров
2. Создание рекомендаций (нахождение соседей на основе новых представлений)

Пример рекомендаций для товара № 108775015:



Рисунок 10. Товар № 108775015.

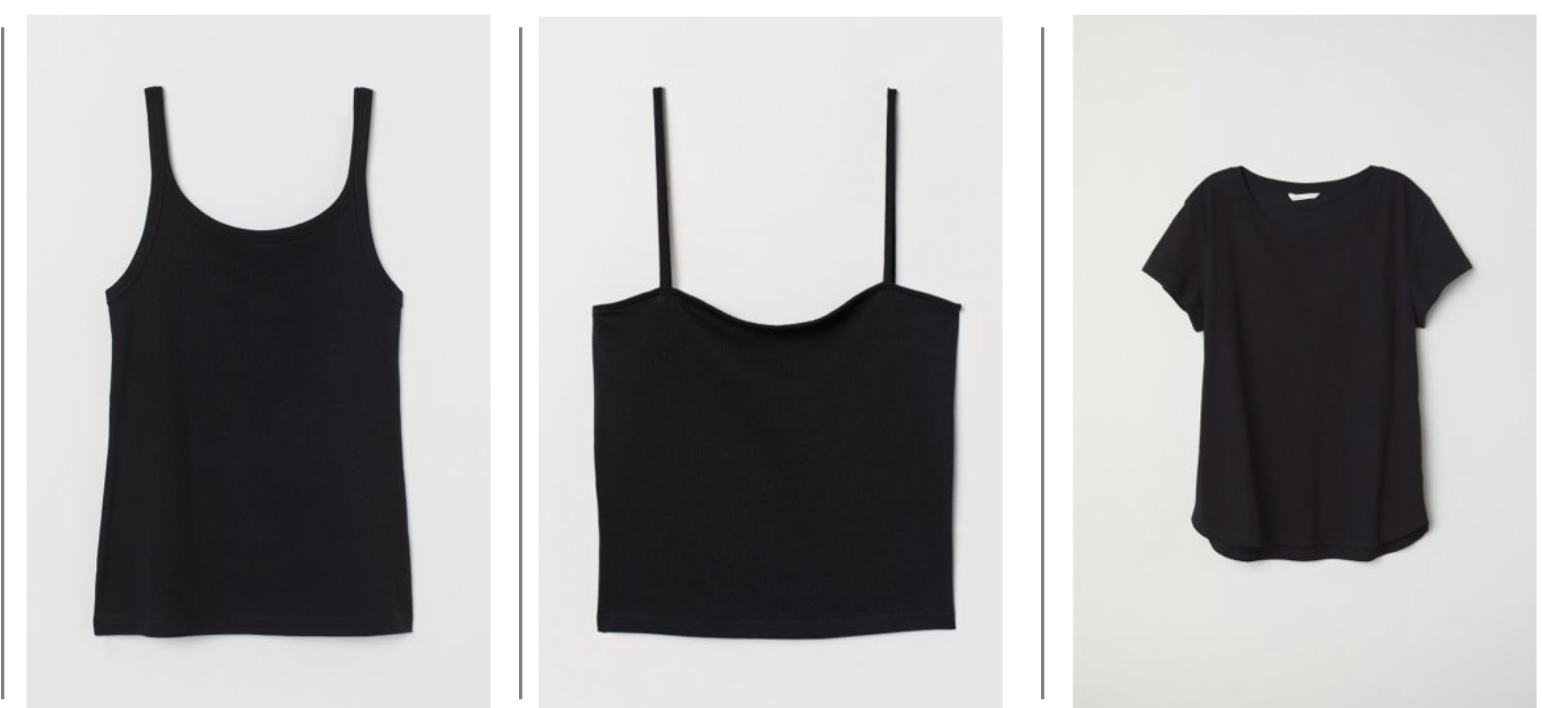


Рисунок 11. Топ 3 схожих товаров.

# Заключение

Мы показали, как можно использовать графовую нейронную сеть для задачи рекомендательной системы. Используя наборы данных H&M, мы построили двудольный граф (пользователи и товары). С помощью модели Pinsage мы сгенерировали новые представления товаров, на основе этих представлений мы можем найти схожие товары для каждого продукта.

Графовые нейронные сети хорошо себя показали в области рекомендательных система: они учитывают стороннюю информацию (например признаки вершин) в отличии от методов коллаборативной фильтрации, так же они более вычислительно эффективны чем методы, основанные на содержании, которые обычно используют матричную факторизацию. Данное утверждение подтверждает использование GNN крупными компаниями: Pinterest – Pinsage [3], Alibaba – Aligraph [4], Uber Eats – GraphSage [5].

# 

# Список использованных источников

1. Jie Zhou. Graph neural networks: A review of methods and applications / Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun / / AI Open. - 2020. - № 1. -C. 57-81
2. Дьяконов А. Г. Графовые нейронные сети [Электронный ресурс]. URL: <https://dyakonov.org/2021/12/30/gnn/>. (Дата обращения: 20.04.2022)
3. Rex Ying. Graph Convolutional Neural Networks for Web-Scale Recommender Systems [Электронный ресурс]. / Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec. URL: https://arxiv.org/abs/1806.01973/. (Дата обращения: 20.04.2022)
4. Rong Zhu. AliGraph: A Comprehensive Graph Neural Network Platform [Электронный ресурс]. / Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, Jingren Zhou. URL: https://arxiv.org/abs/1902.08730/. (Дата обращения: 20.04.2022)
5. William L. Hamilton. Inductive Representation Learning on Large Graphs [Электронный ресурс]. / William L. Hamilton, Rex Ying, Jure Leskovec. URL: https://arxiv.org/pdf/1706.02216.pdf. (Дата обращения: 20.04.2022)

# Пояснительная записка

Модель обучалась на разных видеокартах:

* Tesla T4: Среднее время, требуемое на 1 эпоху ~ 24 минуты
* K80: Среднее время, требуемое на 1 эпоху ~ 35 минут
* Geforce GTX 1060 3gb: Среднее время, требуемое на 1 эпоху ~ 45 минут

Листинг программ:

Обработка данных и создание двудольного графа: hm\_data.py

import pandas as pd

from sklearn.preprocessing import LabelEncoder

import numpy as np

import os

import argparse

import scipy.sparse as ssp

import pickle

from utils.data\_utils import \*

from utils.builder import PandasGraphBuilder

if \_\_name\_\_ == '\_\_main\_\_':

articles = pd.read\_csv('data/articles.csv')

transactions = pd.read\_csv('data/transactions\_train.csv')[['customer\_id', 'article\_id', 't\_dat']]

customers = pd.DataFrame({'customer\_id': transactions['customer\_id'].drop\_duplicates()})

articles = articles.drop(columns = ['product\_type\_name', 'graphical\_appearance\_name', 'colour\_group\_name', 'perceived\_colour\_value\_name',

'perceived\_colour\_master\_name', 'index\_name', 'index\_group\_name', 'section\_name',

'garment\_group\_name', 'prod\_name', 'department\_name', 'detail\_desc'])

for col in ['index\_code', 'product\_group\_name']:

number = LabelEncoder()

articles[col] = number.fit\_transform(articles[col].astype('str'))

for col in ['product\_type\_no', 'graphical\_appearance\_no', 'colour\_group\_code', 'perceived\_colour\_value\_id', 'perceived\_colour\_master\_id']:

number = LabelEncoder()

articles[col] = number.fit\_transform(articles[col].astype('int64'))

transactions['t\_dat'] = transactions['t\_dat'].values.astype('datetime64[s]').astype('int64')

graph\_builder = PandasGraphBuilder()

graph\_builder.add\_entities(customers, 'customer\_id', 'customer')

graph\_builder.add\_entities(articles, 'article\_id', 'article')

graph\_builder.add\_binary\_relations(transactions, 'customer\_id', 'article\_id', 'bought')

graph\_builder.add\_binary\_relations(transactions, 'article\_id', 'customer\_id', 'bought-by')

g = graph\_builder.build()

for col in articles.columns:

if col == 'article\_id':

continue

else:

g.nodes['article'].data[col] = torch.LongTensor(articles[col].values)

g.edges['bought'].data['t\_dat'] = torch.LongTensor(transactions['t\_dat'].values)

g.edges['bought-by'].data['t\_dat'] = torch.LongTensor(transactions['t\_dat'].values)

train\_indices, val\_indices, test\_indices = np.load('data/train\_indices.npy'), np.load('data/val\_indices.npy'), np.load('data/test\_indices.npy')

train\_g = build\_train\_graph(g, train\_indices, 'customer', 'article', 'bought', 'bought-by')

val\_matrix, test\_matrix = build\_val\_test\_matrix(

g, val\_indices, test\_indices, 'customer', 'article', 'bought')

dataset = {

'train-graph': g,

'val-matrix': val\_matrix,

'test-matrix': test\_matrix,

'item-texts': {},

'item-images': None,

'user-type': 'customer',

'item-type': 'article',

'user-to-item-type': 'bought',

'item-to-user-type': 'bought-by',

'timestamp-edge-column': 't\_dat'}

with open('data/full\_graph', 'wb') as f:

pickle.dump(dataset, f)

Модель: model.py

import pickle

import argparse

import numpy as np

import torch

import torch.nn as nn

from torch.utils.data import DataLoader

import torchtext

import dgl

import tqdm

import utils.sampler as sampler\_module

from utils import layers, evaluation

class PinSAGEModel(nn.Module):

def \_\_init\_\_(self, full\_graph, ntype, textsets, hidden\_dims, n\_layers):

super().\_\_init\_\_()

self.proj = layers.LinearProjector(full\_graph, ntype, textsets, hidden\_dims)

self.sage = layers.SAGENet(hidden\_dims, n\_layers)

self.scorer = layers.ItemToItemScorer(full\_graph, ntype)

def forward(self, pos\_graph, neg\_graph, blocks):

h\_item = self.get\_repr(blocks)

pos\_score = self.scorer(pos\_graph, h\_item)

neg\_score = self.scorer(neg\_graph, h\_item)

return (neg\_score - pos\_score + 1).clamp(min=0)

def get\_repr(self, blocks):

h\_item = self.proj(blocks[0].srcdata)

h\_item\_dst = self.proj(blocks[-1].dstdata)

return h\_item\_dst + self.sage(blocks, h\_item)

def train(dataset, args):

g = dataset['train-graph']

val\_matrix = dataset['val-matrix'].tocsr()

test\_matrix = dataset['test-matrix'].tocsr()

item\_texts = dataset['item-texts']

user\_ntype = dataset['user-type']

item\_ntype = dataset['item-type']

user\_to\_item\_etype = dataset['user-to-item-type']

timestamp = dataset['timestamp-edge-column']

device = torch.device(args.device)

# Assign user and movie IDs and use them as features (to learn an individual trainable

# embedding for each entity)

g.nodes[user\_ntype].data['id'] = torch.arange(g.number\_of\_nodes(user\_ntype))

g.nodes[item\_ntype].data['id'] = torch.arange(g.number\_of\_nodes(item\_ntype))

# Prepare torchtext dataset and vocabulary

fields = {}

examples = []

for key, texts in item\_texts.items():

fields[key] = torchtext.legacy.data.Field(include\_lengths=True, lower=True, batch\_first=True)

for i in range(g.number\_of\_nodes(item\_ntype)):

example = torchtext.legacy.data.Example.fromlist(

[item\_texts[key][i] for key in item\_texts.keys()],

[(key, fields[key]) for key in item\_texts.keys()])

examples.append(example)

textset = torchtext.legacy.data.Dataset(examples, fields)

for key, field in fields.items():

field.build\_vocab(getattr(textset, key))

#field.build\_vocab(getattr(textset, key), vectors='fasttext.simple.300d')

# Sampler

batch\_sampler = sampler\_module.ItemToItemBatchSampler(

g, user\_ntype, item\_ntype, args.batch\_size)

neighbor\_sampler = sampler\_module.NeighborSampler(

g, user\_ntype, item\_ntype, args.random\_walk\_length,

args.random\_walk\_restart\_prob, args.num\_random\_walks, args.num\_neighbors,

args.num\_layers)

collator = sampler\_module.PinSAGECollator(neighbor\_sampler, g, item\_ntype, textset)

dataloader = DataLoader(

batch\_sampler,

collate\_fn=collator.collate\_train,

num\_workers=args.num\_workers)

dataloader\_test = DataLoader(

torch.arange(g.number\_of\_nodes(item\_ntype)),

batch\_size=args.batch\_size,

collate\_fn=collator.collate\_test,

num\_workers=args.num\_workers)

dataloader\_it = iter(dataloader)

# Model

model = PinSAGEModel(g, item\_ntype, textset, args.hidden\_dims, args.num\_layers).to(device)

# Optimizer

opt = torch.optim.Adam(model.parameters(), lr=args.lr)

# For each batch of head-tail-negative triplets...

for epoch\_id in range(args.num\_epochs):

model.train()

for batch\_id in tqdm.trange(args.batches\_per\_epoch):

pos\_graph, neg\_graph, blocks = next(dataloader\_it)

# Copy to GPU

for i in range(len(blocks)):

blocks[i] = blocks[i].to(device)

pos\_graph = pos\_graph.to(device)

neg\_graph = neg\_graph.to(device)

loss = model(pos\_graph, neg\_graph, blocks).mean()

opt.zero\_grad()

loss.backward()

opt.step()

# Evaluate

model.eval()

with torch.no\_grad():

item\_batches = torch.arange(g.number\_of\_nodes(item\_ntype)).split(args.batch\_size)

h\_item\_batches = []

for blocks in dataloader\_test:

for i in range(len(blocks)):

blocks[i] = blocks[i].to(device)

h\_item\_batches.append(model.get\_repr(blocks))

h\_item = torch.cat(h\_item\_batches, 0)

#print(evaluation.evaluate\_nn(dataset, h\_item, args.k, args.batch\_size))

torch.save(h\_item, 'data/h\_item')

if \_\_name\_\_ == '\_\_main\_\_':

# Arguments

parser = argparse.ArgumentParser()

parser.add\_argument('dataset\_path', type=str, default='data/full\_graph')

parser.add\_argument('--random-walk-length', type=int, default=2)

parser.add\_argument('--random-walk-restart-prob', type=float, default=0.5)

parser.add\_argument('--num-random-walks', type=int, default=10)

parser.add\_argument('--num-neighbors', type=int, default=3)

parser.add\_argument('--num-layers', type=int, default=2)

parser.add\_argument('--hidden-dims', type=int, default=4)

parser.add\_argument('--batch-size', type=int, default=16)

parser.add\_argument('--device', type=str, default='cpu') # can also be "cuda:0"

parser.add\_argument('--num-epochs', type=int, default=1)

parser.add\_argument('--batches-per-epoch', type=int, default=200)

parser.add\_argument('--num-workers', type=int, default=0)

parser.add\_argument('--lr', type=float, default=3e-5)

parser.add\_argument('-k', type=int, default=3)

args = parser.parse\_args()

# Load dataset

with open(args.dataset\_path, 'rb') as f:

dataset = pickle.load(f)

train(dataset, args)