МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»
Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.04.03 ПРИКЛАДНАЯ ИНФОРМАТИКА
(код и наименование направления подготовки)

Информационные системы и технологии корпоративного управления
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему Методика автоматизации анализа непоследовательных данных

Студент Д.А. Альшин
(И.О. Фамилия) (личная подпись)

Научный руководитель кандидат педагогических наук, доцент Е.В. Панюкова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4
Глава 1 ПРОБЛЕМЫ ОБУЧЕНИЯ НА НЕПОСЛЕДОВАТЕЛЬНЫХ ДАННЫХ 9
1.1 Анализ данных в виде множеств 10
1.2 Непоследовательные данные в задачах прогнозирования 12
1.3 Кодировка данных 14
1.4 Выделение образцовых подмножеств 15
1.5 Сети указателей и глубокие множества 17
1.6 Обучение на графах 18
1.7 Передача сообщений по нейросети 20
1.8 Алгоритмы SchNet 21
1.9 Использование графонов 23
1.10 Определение повторяющихся подграфов 24
Глава 2 Выборка подмножеств 26
2.1 Определение последовательной модели 26
2.2 Алгоритм последовательного выбора 29
2.3 Определение нелинейной модели 30
2.4 Генерация данных 32
2.5 Проведенные эксперименты 35
2.6 Результаты экспериментов 38
Глава 3 Вариативный автокодировщик с использованием графона 46
3.1 Определение модели 46
3.2 Проведенные эксперименты 52
3.3 Результаты экспериментов 55

Глава 4 Иерархический кодировщик графов 64
4.1 Интеллектуальный анализ подграфов 64
4.2 Алгоритм сегментации 65
4.3 Определение модели 69
4.4 Проведенные эксперименты 73
4.5 Результаты экспериментов 76
Глава 5 Методика внедрения работы с непоследовательными данными в системы erp 82
5.1 Особенности применения машинного обучения в ERP 82
5.2 Общая методика внедрения на примере SAP ERP 85
5.3 Работа с множествами для моделей SAP Leonardo 87
5.4 Добавление иерархичности для моделей SAP Leonardo 88
ЗАКЛЮЧЕНИЕ 92
Список используемой литературы и используемых источников 94

ВВЕДЕНИЕ

В существующих информационных системах накоплено огромное количество различных данных, хранящихся в виде баз данных, файлов, архивов, озер данных. При решении бизнес-задач для анализа большей части этих данных существуют и, во многих случаях, успешно применяются методы машинного обучения и интеллектуального анализа.

Однако, на низком уровне существенная часть данных хранится в неудобном для использования обычных способов машинного обучения виде – как множества и графы. Примерами могут быть любые данные хранимые в графовых базах данных, озерах данных.

Для из таких данных получения новой, полезной информации пока не было разработано эффективных, широко применимых методов и алгоритмов. Для достижения результатов при работе с непоследовательными данными, сравнимых с результатами работы с обычными последовательными данными требуется совершенствование уже существующих методов и разработка новых.

Решение этой задачи позволит получить до этого не достижимые результаты путем эффективной работы со сложными для анализа данными, что поможет эффективнее решать существующие бизнес-задачи.

Научной проблемой данного исследования является отсутствие или низкая эффективность методов и алгоритмов машинного обучения на непоследовательных данных. Эту проблему можно разделить на несколько частей:

– сложность решения задач прогнозирования, классификации, кластеризации на данных, хранимых в виде не упорядоченных множеств;

– отсутствие эффективных алгоритмов автокодировки данных в виде графов;

– сложность автоматического анализа данных в виде графов.

Для решения описанной выше проблемы требуется усовершенствовать уже имеющиеся методы работы с непоследовательными данными путем комбинирования существующих методов с новыми более эффективными подходами.

Цель исследования может быть сформулирована как: «разработать и внедрить в существующую информационную систему методику, позволяющую производить кодировку данных в виде множеств со скоростью близкой к $O(n)$ и уверенно выявлять высокоуровневые подграфы».

Объект исследования: анализ данных.

Предмет исследования: методика автоматизации анализа данных.

Гипотеза исследования: применение разработанной в рамках диссертационного исследования методики автоматизации позволит решать, до этого не решаемые задачи информационных систем, путем применения машинного обучения на непоследовательных данных.

Цель исследования будет достигнута, если:

– адаптировать методы глубоких множеств для работы с непоследовательными данными;

– использовать вариативный подход при автокодировке графов;

– внедрить иерархичность в модель, обучаемую на графах.

Задачами исследования являются:

– провести анализ существующих методов машинного обучения на непоследовательных данных для выявления путей их дальнейшего совершенствования;

– экспериментально проверить эффективность применения методов приведения множеств к массивам;

– найти наиболее оптимальные архитектуры нейронных сетей для решения поставленных задач;

– проверить применимость методов глубоких множеств для выборки несходных подмножеств;

– разработать автокодировщик данных в виде графов;

– проверить эффективность применения иерархичного подхода для модели прогнозирования свойств данных в виде графов;

– разработать методику внедрения результатов проведенной работы в существующие информационные системы.

Наличие в существующих информационных системах большого количества накопленных данных в виде графов и неупорядоченных множеств на данный момент не ставится под сомнение. Также не ставится под сомнение сложность получения потенциально достижимой ценности для бизнеса из таких данных [7, 35].

При анализе современных международных источников по применению нейронных сетей и моделей машинного обучения видно, что уже совершались попытки расширения методов обучения на применение к непоследовательным данным [3, 19, 22].

Однако, результаты применения предложенных методов и алгоритмов оказались не удовлетворительными или применимыми только к узкому набору данных.

Самым развитым, на данный момент, направлением применения нейронных сетей к данным в виде графов и множеств является прогнозирование молекулярных свойств в химических информационных системах [17, 22, 40].

Используя описанные в данных работах наборы данных и методы обучения, можно проверить выдвигаемые в данной работе гипотезы.

Современными направлениями в развитии машинного обучения, потенциально применимыми для решения задач этой работы, являются глубокое обучение, графоны, рекуррентные нейронные сети [54], вариативные автокодировщики [2] и другие, использованные в данных направлениях методы.

Описание методологического аппарата исследования

В ходе работы использовались:

– анализ и синтез существующих методов и алгоритмов;

– сравнение архитектур обучаемых моделей;

– проведение экспериментов по обучению различных моделей;

– модели теории графов и динамического программирования.

Описание основных результатов исследования и результатов его апробации.

Исследование можно разделить следующие этапы:

1) анализ существующих подходов, потенциально применимых для решения задач исследования;

2) проведение экспериментов по использованию теории глубокого обучения для выборки несходных подмножеств;

3) создание модели вариативной автокодировки графов и анализ показателей эффективности этой модели;

4) проведение экспериментов для оценки применимости иерархического подхода обучении на графах.

Обоснование научной новизны, теоретической и практической значимости результатов исследования

В ходе работы были улучшены и дополнены методы автокодировки данных в виде графов.

Была экспериментально доказана низкая, вопреки данным нескольких существующих источников, эффективность применения глубоких множеств для работы с непоследовательными данными при различных архитектурах нейронных сетей.

В первые был применен иерархических подход для кодировки графов, добавивший новые свойства существующим моделям и позволяющий существенно ускорить работу некоторых существующих архитектур моделей. Были показаны направления дальнейшего исследования применения моделей с иерархической кодировкой.

В результате работы был расширен горизонт применения машинного обучения применительно к анализу накопленных в информационных системах данных.

Теоретическая значимость исследования выражается во вкладе исследования в информатику путем расширения знаний о машинном обучении применительно к информационным системам.

Практическая значимость исследования выражается в разработанной методике внедрения полученных знаний в существующие информационные системы. Данная методика может быть применена для получения, до этого недостижимых, результатов в виде добавления возможности эффективной работы с непоследовательными данными.

Положения, выносимые на защиту:

1) Методика внедрения работы с непоследовательными данными в информационные системы
2) Иерархичная модель кодировки данных в виде графов
3) Модель вариативной автокодировки с помощью графона

Диссертационное исследование состоит из введения, 5 глав, заключения и библиографии.

Работа изложена на 97 страницах, содержит 11 рисунков, 8 таблиц.

Глава 1 ПРОБЛЕМЫ ОБУЧЕНИЯ НА НЕПОСЛЕДОВАТЕЛЬНЫХ ДАННЫХ

В течение последних лет стремительно растет применение алгоритмов машинного обучения для решения разнообразных задач. Самоуправляемые автомобили появились благодаря технологиям по распознаванию изображений, использующим сверточные нейронные сети [23]. Благодаря научным прорывам в распознавании речи, сейчас никого не удивить использование голосовых помощников [42]. Многие финансовые компании на постоянной основе используют машинное обучение для моделирования сложных взаимодействий на финансовых рынках.

Причина такого успеха применения машинного обучения в данных областях деятельности заключается в том, что существующие методы могут быть легко применены на данных, используемых в этих областях. Данные, используемые в большинстве областей последовательные. Управление автомобилем, например, основано на видео данных, которые, в упрощенном виде, представляют собой последовательный набор изображений. Изображения, в свою очередь, — это массив точек, последовательность между строками и столбцами которых, точно определена. Для получения более точных прогнозов при работе с изображениями была разработана специальная методика, использующая сверточные нейронные сети. Голосовые помощники, например, обрабатывают аудио сигнал, представляющий собой цепочку данных, распределенных по оси времени. Финансовые данные подпадают под ту же категорию. Для распознавания текста были разработаны технологии обработки естественного языка (Natural Language Processing, NLP). Здесь модель данных немного сложнее: текст — это набор слов, каждое из которых может содержать произвольное число знаков. Однако, набор всех слов уже заранее известен и помещен в специальные словари, которые могут быть сконвертированы в последовательность векторов.

В приведенных примерах прослеживается общий мотив – машинное обучение обычно применяется к наборам последовательно структурированных данных, таких как векторы(массивы), зависимости между данными которых заранее известны. Использование таких форматов данных широко распространено.

Однако, как показано в данной работе, существуют намного больше интересующих нас типов данных, которые не могут быть легко приведены в привычный последовательный формат. Социальные сети обычно представляются как графы, с вершинами, представляющими людей, а ребра – взаимодействия между этими людьми. Химические молекулы тоже представляются как графы из атомов и связей между ними. Группы вещей, такие как коллекции фильмов или аудиозаписей, обычно представляют собой множества – не упорядоченный тип данных. Именно тот факт, что множества не упорядочены – ключевой фактор, отличающий множества от массивов при работе с ними.

Задача этой работы – расширение границ машинного обучения для работы с такими не последовательно структурированными данными. Для функционирования моделей, работающих с множествами необходимо чтобы данные на выходе из модели, не зависели от последовательности данных на входе. Такая независимость последовательности на входе не поддерживается существующими моделями машинного обучения, работающими на последовательных данных. Графы, в свою очередь, структурированы отлично от последовательностей. Одна вершина потенциально может иметь связи с несколькими другими вершинами графа. Исходя из этого, для работы с графами требуются иные алгоритмы.

1.1 Анализ данных в виде множеств

Множество – не упорядоченный набор данных. Множествами обычно представляются группы вещей. Фотоальбомы чаще всего представляются множеством фотографий, нежели упорядоченным списком. Группы новостей

также не упорядочены по своей природе. Сделки на фондовой бирже описываются как неупорядоченный набор транзакций, а не последовательность.

Несмотря на то, что методы машинного обучения на множествах изучались, методология для такого обучения еще не сформировалась так четко как, например, методология распознавания текста или изображений.

Данная работа сфокусирована на одной из проблем этого направления - выделение образцовых подмножеств. Предположим, есть множество и требуется выделить подмножество различных элементов, максимально сохранив репрезентативность исходного множества. Например, требуется отобрать небольшое подмножество из большого множества данных фондовой биржи для определения в какие типы акций лучше инвестировать. Для этого потребуется использовать достижения в глубоком обучении, применяя методики, изначально разработанные для множеств. Такой подход позволит решить, до этого сложно решаемые задачи, путем комбинации новых алгоритмов репрезентативной выборки данных и уже проверенных подходов к обучению на этих данных.

Одно из направлений, где машинное обучение имеет большой потенциал — это химическая отрасль. За многие годы химиками собраны обширные данные о молекулах, их составляющих и взаимодействию между собой. Машинное обучение может открыть для систем, связанных с химией, новые горизонты в понимании о том, как взаимодействуют между собой химически элементы, открыть новые пути прогнозирования свойств молекул и генерации молекул с заданными свойствами.

Однако, машинное обучение на данных о молекулах в корне отличается от обучения на последовательности изображений. Молекула представляет собой граф, с определенным количеством вершин, соединенных между собой ребрами. Машинное обучение на графах – развивающееся направление, и его методы пока не так развиты, как методы и алгоритмы для последовательных данных [22]. Вершины графов сложно

связанны, поэтому в большинстве случаев их нельзя представить в виде последовательного массива фиксированного размера. Существующие методы для множеств плохо применимы к графам [22, 17].

Существует две основных задачи машинного обучения в химии: прогнозирование химических свойств и генерация молекул. Попытаемся разработать более совершенные методы моделирования на примере этих двух задач.

1.2 Непоследовательные данные в задачах прогнозирования

Прогнозирование химических свойств, таких как растворимость или молекулярная энергия – примеры многих задач прогнозирования, решаемых с помощью машинного обучения. Несмотря на то, что некоторые методы решения подобных задач уже были разработаны, обычно эти методы применимы только к одной конкретной задаче [41, 17]. На текущий момент, большинство способов решения подобных задач используют молекулярные отпечатки, которые в упрощенной форме подсчитывают, количество подструктур, существующих в молекуле [51]. Использование методов машинного обучения на графах вместо молекулярных отпечатков, может показать намного более применимые на практике результаты.

Одной из целей этой работы является разработка новых иерархичных методов прогнозирования. Молекулы – отличных пример данных для этой задачи. Они имеют разнообразные размеры и структуру. При прогнозировании свойств молекул хорошая модель может делать точные прогнозы как для сложных, так и для совсем небольших молекул, таких как аминокислоты с числом атомов меньше 20 или большие молекулы с сотнями атомов. Для того чтобы показать приемлемые для обоих случаев результаты, перед обучением необходимо иерархично кодировать исходные данные – несколько уровней представления для одного графа.

Генеративное моделирование показало успешное применение в обучении на изображениях и в распознавании речи. Генеративное

моделирование в обработке естественного языка основано на рекуррентных нейронных сетях (recurrent neural networks, RNN) [54]. Такие, достаточно новые, подходы уже показали отличные результаты при работе с изображениями [13, 28].

Предпринимались попытки по использованию этих достижений в генерации молекул [21].

Однако строение графа сильно отличается от последовательности, и не смогло быть представлено в подобном виде.

В главе 4 работы показан новый метод генерации графов на примере молекул. Это – подход последовательного разбиения и упрощения: представление атомов упрощено на первом шаге, а упрощение связей между атомами – последовательно на следующих шагах.

Этот метод нацелен на создание относительно небольших графов из, зачастую сложно упрощаемых, комплексных молекул с различными повторяющимися частями.

Одно из самых полезных применений машинного обучения в химии – разработка новых препаратов.

Фармакологические компании инвестируют огромные ресурсы в разработку, синтез и тестирование новых соединений веществ [56].

На данный момент химики используют свой профессиональный опыт для того, чтобы предложить новые соединения с правильной комбинацией функциональных групп, надеясь на то, что на испытаниях эти соединения покажут нужный эффект.

Этот процесс может быть кардинально оптимизирован.

У машинного обучения есть потенциал кардинально поменять работу фармацевтических компаний и их систем. Точный и быстрый способ определения химических свойств даст возможность оптимизировать и автоматизировать работу таких компаний с помощью использования новых информационных систем с применением машинного обучения.

Приведем понятия и методы, на которых основано дальнейшее исследование. Для начала введем понятие обучения представлениям (representation learning) и автокодировки (autoencoding). Затем опишем критерии, по которым выборка может считаться валидной, репрезентативной. В завершении приведем существующие методы машинного обучения на графах.

1.3 Кодировка данных

Обучение представлениям это процесс обучения когда исходные данные $x_i \in R^n$ преобразуются в массивы меньшей размерности $v_i \in R^d$, где d < n [6]. Такое более компактное представление обычно содержит в себе основные характеристики исходных данных. Это – обычное преобразование для уменьшения размерности [6, 22].

После того как данные преобразованы, они могут использоваться для решения задач классификации и генерации. Такие малоразмерные массивы, представляющие полные данные, принято называть кодировкой (encoding).

Автокодировка – процесс создания кодировки, из которой в последующем могут быть получены исходные данные [37, 54].

Автокодировщики состоят из двух частей: кодировщик, который вычисляет уменьшенное представление данных и декодировщик, восстанавливающий исходные данные из кодировки [6, 37].

Один из вариантов автокодировки – вариативная автокодировка, в которой размер под кодировку ограничен так чтобы удовлетворять распределению вероятностей (обычно распределению Гаусса). Это позволяет модели кроме создания образцов еще и воссоздавать исходные данные [2, 37].

Такой процесс осуществляется путем выборки наиболее удачных образцов из нормального распределения, исходя из заданного размера образца с использованием декодировщика для воссоздания исходных образцов [37, 47].

Кодировка и автокодировка (вариативная) широко и успешно применяются при работе с последовательностями [35, 55, 43].

При кодировке и декодировке моделей последовательных данных обычно используются рекуррентные нейронные сети [35, 55].

Это обусловлено тем, что в таких сетях существует скрытое состояние памяти (hidden memory state), где сохраняется информация о части последовательности, прочитанной на данный момент [43].

Исходя из этого, рекуррентные нейронные сети изначально оптимизированы для кодировки последовательных данных, так как они сохраняют данные о связях между текущей записью и предыдущей записью в последовательности [35, 43].

Рекуррентные нейронные сети могут использоваться как декодирующие сети ввиду того, что в скрытом состоянии хранится информация о текущей и предыдущей записях в последовательности [35, 55].

Однако, для графов и множеств пока нет общепринятых методов автокодировки [22, 17].

Создание таких методов – одна из целей этого исследования.

1.4 Выделение образцовых подмножеств

Следующая проблема, связанная с представлением множеств – выделение образцовых подмножеств.

Для данного исследования особенно важно чтобы выделенное подмножество максимально полно отражало характеристики всего множества.

Выделение образцовых подмножеств применяется в решении задач прогнозирования, обобщения, поиска и компьютерного зрения [19, 39].

Детерминантный точечный процесс (determinantal point process, DPP) – это вероятность распределения выбранного множества по всем возможным подмножествам исходного множества для получения подмножества с максимально несходными элементами.

DPP используется на множествах размера n путем создания симметричной $n \times n$ матрицы $K \in R^{n \times n}$. K ограничено тем, что все собственные значения не могу быть больше 1. $K_{i,j}$ показывает сходность между записями i и j [19].

Вероятность того, что подмножество $A \subset \{1, \ldots, n\}$ пропорциональна $\det(K_a)$, где K_a – минимальное значение K для строки и столбца с индексом элемента A. Другими словами, определитель подматрицы исходной матрицы, где строки и столбцы имеют индекс элемента A, пропорционален вероятности выбора заданного множества [19, 39].

Несмотря на то, что существует 2^n возможных подмножеств, выборка с использованием DTP занимает только $O(n^3)$ времени [26]. Это – очень эффективный алгоритм для подобных распределений. Но, все же, при больших значениях n необходимые ресурсы процессора возрастают критически.

Поэтому попытаемся найти альтернативный метод используя нейронные сети и DPP как исходную точку.

Один из способов борьбы со сложностью, добавляемой при использовании множеств в машинном обучении, это предположение того, что множество ведет себя также как последовательность.

Этого можно достичь если заранее закрепить определенную последовательность в элементах множества или менять последовательность при каждом обращении к множеству.

Таким образом негативные моменты, добавленные множеством можно нейтрализовать.

Это позволит использовать проверенные методы машинного обучения на последовательных данных, такие как RNN.

Эта идея была довольно хорошо изучена [58].

В конечном счете, экспериментально было показано, что независимо от того, переставлять ли элементы в множестве или установить определенный порядок, существующие фундаментальные проблемы затрудняют попытки

разработать точные модели для последовательностей, имитирующих множества [58].

Поэтому в данной работе проведены эксперименты над множествами, а не перевод их в последовательности.

1.5 Сети указателей и глубокие множества

На текущей момент, было предложено несколько методов работы с множествами [59, 31, 48]. Один из самых популярных подходов — это сети указателей (Pointer Networks). Он нацелен на решение задач получения подмножества или приведения исходного множества, что в свою очередь необходимо для задачи получения несходного подмножества.

Сети указателей используют алгоритм RNN для кодировки исходного множества и добавляют сверки содержимого исходного и кодированного множеств [9] чтобы удостовериться что полученный результат не зависит от перестановки элементов. Полученная кодировка запоминает свое состояние используя RNN [59]. Этот подход работает на практике, но очень требователен к вычислительным ресурсам – $O(n^2)$ для исходного множества с n элементов. Это происходит ввиду того, что RNN использует n шагов и каждый шаг требует $O(n)$ для проверок скрытого состояния.

Данная работа нацелена на получение эффективных алгоритмов с низкой стоимостью вычисления. Возможно, использование сетей указателей или подобных подходов приемлемо, когда доступно большое число вычислительной мощности, однако данная работа ставит своей задачей разработку методов, применимых, когда доступно $O(n)$ времени на получение прогноза, что чаще всего случается в решении задач прогнозирования на практике.

Функция множества – это функция, принимающая набор элементов и возвращающая нечувствительную к перестановкам функцию этих элементов [62]. Доказано что любая функция множества $s(X)$ имеет универсальный аппроксиматор вида $f(\sum_{x\in X} \emptyset(x))$ для функций $\emptyset(\cdot)$ и $f(\cdot)$ [62].

Эти выводы составляют основу для аппроксимации функций с помощью нейронных сетей. Совместно тренируя кодирующие функции для элементов $\widehat{\emptyset}$ и a, нейронная сеть проводит операции над их суммой $\hat{f}$. Имея достаточно большую выборку для каждой из этих аппроксимирующих функций, можно аппроксимировать любой набор функций.

В данной работе используется описанный выше подход. Исходя из того, что распределение вероятностей DPP – это набор функций, возможна аппроксимация общей функции вероятности используя полученную модель. Более того, время расчета при таком подходе равно $O(1)$, если известны все составляющие кодировки.

1.6 Обучение на графах

До текущего времени было предпринято много попыток по расширению методологии машинного обучения для использования графов. Ввиду того что графы имеют сложное строение, требуется иная методология для синтеза всей имеющейся в графе информации и генерации новых сущностей, несущих в себе часть этой информации.

Приведем два, наиболее распространенных, метода для приведения графов в более привычный формат и опишем сложности, возникающие при их применении. Затем затронем класс нейронных сетей, используемых для получения информации из графов. Также опишем генеративные модели для получения несходных образцов графов. В заключении, приведем существующие техники по получению общих повторяющихся элементов из набора графов, которые будем использовать при создании моделей.

Было предпринято много попыток по машинному обучению на данных из химической отрасли в виде графов. Чаще всего графы пытались привести к строкам, затем использовать технологии обработки естественного языка (NLP) для того, чтобы получить модели представления [41, 21]. В большинстве попыток использовалась упрощенная строковая молекулярная

система записи (simplified molecular-input line-entry system, SMILES). Эта система используется для привода химических структур к строкам [60, 21].

Однако, такие подходы, применимо к молекулам, не показали точных результатов по сравнению с общими подходами кодировки графов [17, 21, 41]. Неудовлетворительные результаты были получены, предположительно, ввиду того что использовались модели, не приспособленные к такому типу данных. Конвертирование графов в строковые последовательно и применение RNN к этим последовательностям, сильно усложняет исходную задачу прогнозирования на данных в виде графов.

Получение молекулярных отпечатков – это процесс получения свойств заранее известных типов из молекул для использования в качестве исходных данных при прогнозировании [29, 51]. Отпечатки могут сильно разниться, учитывая точность используемых дескрипторов и сколько итераций проводилось при получении этих отпечатков [29, 51].

В информатике химии было разработано несколько методов нацеленных на получение свойств из графов молекул [14, 51]. Существует способ записи MDL-166, иногда именуемый как система доступа к молекулам (Molecular Access System, MACCS). В этой записи используются 166 битные массивы, где каждый бит показывает наличие определенного свойства в молекуле [45,14]. Также существуют способы записи более высокого порядка: MDL-960 [14] и «PubChem CACTVS» [29], содержащие большее количество информации.

Однако все эти техники получения отпечатков молекул требуют существенных познаний в химии для того, чтобы определить нужные свойства [14, 29]. Поэтому метод получения молекулярных отпечатков в целом ограничен потребностью в специфичных знаниях об определении нужных свойств. Кроме того, специальные знания требуются чтобы выбрать только нужные свойства из большого списка всех возможных свойств.

Применимо к социальным сетям, например, подобные подходы, основанные на специфичных для отрасли знаниях, не могут быть

использованы ввиду того, что информация в виде графов в этом случае очень плохо структурирована. Поэтому в данной работе мы будем использовать общие, не зависимые от отрасли методы кодировки графов.

1.7 Передача сообщений по нейросети

Нейронный сети с возможностью передачи сообщений (Message passing neural networks, MPNN) часто применяются при кодировке информации в виде графов [22, 17]. MPNN получает кодировку в виде массива используя два последовательных шага: шаг отправки сообщения и шаг считывания.

На шаге передачи сообщения через каждое ребро графа отправляются сообщения. Эти сообщения сохраняются в каждой из вершин графа, используя скрытое состояние этой вершины в кодировщике. Этот шаг повторяется фиксированное число T раз [22].

Обозначим скрытое состояние в вершине v при проходе t как h_v^t.

Тогда сообщение из вершины w к вершине v при проходе t через ребро e_{wv} может быть получено как $M(h_w^t, h_v^t, e_{wv})$.

Функции сообщения M_t при проходе t могут отличаться.

Сообщения в вершине v сохраняются.

Так для прохода $t + 1$ сообщения будут равны:

$$\mathrm{m_v^{t+1}} = \sum_{\mathrm{w \in N(v)}} \mathrm{M(h_w^t, h_v^t, e_{wv})}, \quad (1)$$

Скрытые состояния вершин обновляются, используя функцию обновления:

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}), \quad (2)$$

где U_t – функция обновления, которая может изменяться при каждом проходе [22, 17].

Далее идет фаза считывания, в которой скрытые состояния всех вершин собираются в одну общую кодировку всего графа [17].

Обычно к конечным скрытым состояниям h_v^t каждой вершины применяется функция трансформации f, затем измененные состояния вершин собираются вместе, используя один из существующих алгоритмов (подвыборка, суммирование, усреднение) [22, 17].

Усреднение, например, может выглядеть как:

$$R(\{h_v^T | v \in V\}) = \sum_{v \in V} f(h_v^T), \quad (3)$$

где f – функция, параметризованная с помощью нейронной сети.

Существуют много различных методов, собранных под общей категорией алгоритмов MPNN.

Каждый из этих методов может быть представлен, используя запись выше и подставляя в нее нужные значения M_t, U_t и R [17].

В экспериментах данной работы используются алгоритмы MPNN для вычисления кодировок каждой вершины графа и графа в целом.

1.8 Алгоритмы SchNet

SchNet – набор алгоритмов получения кодировки графов, схожий с алгоритмами MPNN.

SchNet использует те же подходы что и MPNN, добавляя блоки взаимодействия при каждом шаге отправки сообщения. Эти блоки используются для того, чтобы сохранить информацию о том, каким образом вершины взаимодействуют между собой.

Ключевая разница состоит в том, что в SchNet подразумевается, что набор данных включает в себя информацию о расстоянии между вершинами графа, используемую в дальнейшем для подсчета взаимодействий.

В MPNN информация о расстоянии может использоваться, но она не обязательна.

Ключевым компонентом блока взаимодействия является процесс (слой) непрерывной фильтрации (cfconv).

На входе cfconv принимает расстояние между вершинами и свойства вершин $x_1, \dots, x_{|V|}$.

Процесс cfconv использует радиальную базисную функцию (radial basis function, RBF) для того, чтобы расширить расстояние между вершинами и получить 300-мерный вектор свойств для каждой вершины.

После того как все вершины соединены, используя метод расширения RBF, будут получены многомерные представления свойств вершин $e_1, \dots, e_{|E|}$.

В итоге новые расстояния между вершинами подсчитываются по формуле:

$$w_i = \sum_{j \sim i} e_{ij} \odot x_j, \quad (4)$$

где $\odot$ показывает умножение на уровне элемента.

В каждом блоке взаимодействия есть один слой cfconv, находящийся между различным количеством слоев, соединенных между собой.

Блоки взаимодействия соединены между собой используя вид соединения ResNet [25]: используя $x^{(i)}$ как вектор свойства вершины, предшествующий блоку взаимодействия i и $b^{(i)}(\cdot)$ как выходное значение блока взаимодействия i. Тогда:

$$x^{(i+1)} = x^{(i)} + b^{(i)}\left(x^{(i)}\right), \quad (5)$$

Таким образом, выходные значения каждого слоя cfconv или блока взаимодействия будут разницей между вершинами, а не между кодировками вершин.

В целом SchNet состоит из четырех основных компонентов:

1. Начальное положение вершины.
2. Блоки взаимодействия.
3. Соединенные между собой слои, проецируемые для каждого вектора как одиночная скалярная величина.

4. Сумма скаляров вершин, необходимая для получения конечного результата.

Будем использовать SchNet как основу для дальнейших экспериментов по прогнозированию ввиду того, что этот метод дает получить лучшие, на сегодняшний день, результаты в задачах прогнозирования [53], при проверке с использованием QM9 [52, 50].

1.9 Использование графонов

Графон – это вероятностная модель графа, широко используемая в теории графов и математике [18, 7, 44].

Эта модель представляет вершины графа как случайные переменные в распределении Бернулли, с вероятностью существования вершины, определяемой по меткам вершины $\in [0,1]$.

В простом режиме одноразмерных меток вершины, графон представляет собой функцию параметров $f : [0,1]^2 \to [0,1]$, которая показывает вероятность существования вершины между двумя узлами как аргументы функции f [7, 57].

Модель может быть легко расширена применительно к многомерным меткам вершин.

Предположим, что для каждого узла i в графе существует k-размерная метка x_i. Многомерный графон дает возможность вычислить вероятность $f(x_i x_j) \in [0,1]$ того, что эти два узла связаны.

Графоны хорошо применимы для плотных графов [18, 7, 57]. Предположим, что метки вершин распределены по вероятностной характеристике p от R^k, не зависимо для каждой вершины, тогда результирующее число ребер в порождённом подграфе возрастает вместе с вероятностью существования ребра как $\binom{n}{2} = O(n^2)$.

Однако, молекулы представляют собой разреженные графы, так как атомы обычно имеют ограниченное количество соединений, и не один атом не может иметь более 8 связей [4]. Б

олее того, молекулы имеют жестко фиксированное строение, и изменение даже одной связи может сделать молекулу нестабильной [4].

Поэтому для графов, таких как молекулы, необходимо разработать дополнительную методику для применения моделей, основанных на графонах.

1.10 Определение повторяющихся подграфов

Определение повторяющихся подграфов – фундаментальная задача, которая изучается более десяти лет [61, 63, 32]. Цель этой задачи – найти набор подграфов, повторяющихся в исходном наборе графов. Обычно в задаче нахождения повторяющихся подграфов известен набор графов A и минимально необходимое количество повторений s. Тогда результатом решения задачи будет набор графов B где:

$$\forall \mathrm{b} \in \mathrm{B}, \frac{1}{|\mathrm{A}|} \sum_{\mathrm{G} \epsilon \mathrm{A}} [\mathrm{b} \epsilon \mathrm{G}] \geq \mathrm{s}, \quad (6)$$

Другими словами, будут получены все подграфы, повторяющиеся более $s * |A|$ раз, в исходном наборе графов A [32]. Определение повторяющихся подграфов в нашем случае важно ввиду того, что существует тенденция повторения многих частей в тысячах различных молекул.

За десятилетия изучения химиками были выделены различные функциональные группы, являющиеся группами атомов, часто составляющими более сложные молекулы. Такие функциональные группы зачастую влияют определенным образом на свойства молекул [4].

Однако, выделение функциональных подгрупп – сложная задача, так как не существует одной большой базы, хранящей все такие подгруппы.

В данной работе используются методы определения повторяющихся подграфов для выделения и упрощения часто повторяющихся частей молекул, без необходимости использовании знаний в химии. Более того нам интересны части любых размеров, и поэтому задачей стоит выявление методов, достаточно быстрых для работы с большими графами и выявлению больших подграфов, без нарушения структуры исходных графов.

Одна из самых популярных техник для решения этой задачи – gSpan (Graph-based Substructure Pattern Mining). Это – довольно быстрый метод определения всех часто повторяющихся подграфов в определенном наборе графов [61]. gSpan находит все подграфы, которые повторяются в более чем k процентах графов исходного набора, где k – входной параметр алгоритма. Кроме того, этот алгоритм не нарушает изоморфность графов [61]. gSpan выстраивает иерархичную структуру из подграфов относительно других графов или подграфов [61]. В данной работе использован gSpan, ввиду того что он точен и достаточно быстр. В частности, в экспериментах использовался gBolt [63] – реализация алгоритма gSpan на C++ с поддержкой параллельных вычислений.

Выводы по главе 1

В ходе анализа текущей ситуации в направлении работы с непоследовательными данными, было показано что существующие методы и алгоритмы использования непоследовательных данных для машинного обучения требуют дальнейшей проработки для того чтобы сделать их более общеприменимыми и быстродействующими. Это относится как методам и алгоритмам работы с множествами, так и с графами. Были приведены уже существующие подходы, их преимущества и недостатки, и отмечено какие из подходов и технологий будут использоваться в дальнейших экспериментах для достижения целей данной работы.

Глава 2 Выборка подмножеств

В данной главе описана проделанная работа по совершенствованию алгоритмов определения повторяющихся подмножеств с использованием приближенной выборки и нейронных сетей. Целью является выявление менее затратных методов аппроксимации выборки по сравнению с DPP и совершенствованию функций DPP. Для этого будет произведена эмуляция нескольких уже известных методов выборки используя модель, выработанную в результате экспериментов.

Выборка подмножеств используя DPP занимает $O(n^3)$ времени, где n – размерность исходного множества [26]. Для очень больших множеств процесс выборки занимает непозволительно большое время. Поэтому разработка быстрых и точных методов выборки подмножеств представляет существенную ценность в практическом плане.

Уже существуют некоторые методы максимизации детерминантной функции DPP, так чтобы процесс занимал меньше n^3 времени. В этой части работы будем использовать и улучшать методы, использующие нейронные сети для совершения приближенной выборки с высокой точностью. Для начала будет создана последовательная модель, которая совершает жадный отбор элементов множества и поэтому является хорошим кандидатом для симуляции последовательных детерминантных алгоритмов. Далее будет создана непоследовательная модель, в которой выходные данные не зависят от перестановки входных данных. Эта модель лучше других подходит для симуляции реальных результатов применения DPP и симуляции других не последовательных, чаще всего жадных, алгоритмов. В заключении эти алгоритмы будут проверены на подходящих проверки наборах данных.

2.1 Определение последовательной модели

Задачу данной части работы можно сформулировать как: создать модель, которая для каждого элемента множества задаст определенный

порядок относительно других элементов, затем определит добавить или нет этот элемент в «активное множество» элементов и, в конечном итоге, выдаст активное множество как результирующее подмножество. Этот подход не является до конца обобщенным: он позволяет использовать только методы выбора, которые по своей природе являются инкрементными и жадными. Поэтому модель была обучена на данных, полученных с использованием жадных алгоритмов.

Используемая модель состоит из трех основных компонентов:

1) кодировщик для кодировки каждого элемента;

2) объединяющая функция для сбора кодировок подмножества;

3) сеть прогнозирования для оценки качества разделения на подмножества.

Опишем каждый из этих компонентов.

Кодировка каждого элемента была произведена с помощью нейронной сети. Существует большое количество используемых в практике архитектур сетей для кодировки. Было решено использовать четыре способа кодирования:

– Последовательные полностью связанные слои. Это – наиболее часто используемая архитектура для нейронных сетей.

– Кодировщик ResNet [25]. Сети в подобной архитектуре используют пропускающие связи между слоями и часто используются в обработке изображений.

– Кодировщик ConcatNet. В этой архитектуре каждый слой подается в объединенные выходы каждого из предыдущих слоев вместе с исходными данными.

– Кодировщик DenseNet [27]. Этот метод похож на предыдущий, с единственным отличием в том, что окончательное кодирование — это объединение результатов всех слоев, а не только результатов последнего слоя.

Эти четыре метода охватывают большее число возможных моделей и включают современные архитектуры, позволяющие получить максимально корректные результаты [25, 27]. При проведении экспериментов варьировались параметры сети, такие как размер получаемой кодировки и число слоев.

В ходе экспериментов были использованы два метода объединения: суммирование и усреднение. Исходная формулировка глубоких сетей говорит, что суммирование, при условии достаточного количества функций кодировки, позволяет аппроксимировать любой набор функций. Кроме этого, было проверено усреднение, так как это – процесс похожий на суммирование, но, в дополнение, позволяющий провести нормализацию, что может быть полезно для решения поставленной задачи.

Разделы выше описывают метод, использованный для объединения и кодировки подмножества. Теперь опишем метод, который использовался для определения, является ли подходящим то или иное подмножество.

Сеть прогнозирования имеет два входных параметра:

1) e_a, общая кодировка текущего активного множества;
2) e_b, общая кодировка пока не проверенных элементов.

Обозначим сеть прогнозирования как $f_\theta(e_a, e_b)$, параметризованную через параметры нейронной сети θ. Выходным результатом этой сети будет являться мера того, насколько подходящим является заданное разбиение.

Кроме того, было проведено тестирование сети прогнозирования, где для модели задано только e_a, вместо передачи еще и общей кодировки пока не проверенных элементов. Однако основная представленная в работе модель имеет более общий характер и больше возможностей чем такая альтернативная модель. Модель e_b позволяет учитывать, в каком порядке расположены оставшиеся элементы. Это, в свою очередь, позволяет модели решать, что включать в активное множество, основываясь на том, какие элементы остались не обработанными.

2.2 Алгоритм последовательного выбора

Основным способом использования полученной модели является алгоритм последовательного выбора. Сначала распределим элементы множества в каком-то определенном порядке, затем пошагово проверим нужно ли добавить каждый из элементов в активное множество. В завершении вернем активное множество как выбранное подмножество. По сути, это – жадный алгоритм последовательного выбора, и поэтому он является хорошим кандидатом для воспроизведения результатов на других жадных алгоритмах последовательного выбора.

Предположим, что изначально множество на входе в алгоритм было S. Рассмотрим, как производилось решение о включении элемента e, учитывая, что текущие активное множество $A \subset S$ и оставшиеся не распределенные элементы $B \subset S$, где $A \cap B = 0$ и $e \notin B$. Пусть e_a – это кодировка множества A, а $e_{a'}$ – это кодировка множества $A \cup \{e\}$. Вычислим вероятность того, что элемент включен в активный набор как максимальное значение $[f_\theta(e_{a'}, e_b), f_\theta(e_a, e_b)]$ используя распределение Бернулли. Обозначим вероятность того, что модель выберет текущую характеристику элемента y как $p_\theta(y)$.

Каждый элемент оценивается для включения описанным выше образом, в заранее заданной последовательности. В завершении процесса, после того как все элементы прошли проверку, возвращается активное множество A.

Требовалось определить, как полученная модель покажет себя по сравнению с более традиционной моделью – рекуррентной нейронной сетью. Ожидается что предложенная методология будет способна более точно запечатлеть строение подмножества чем обычная последовательная модель, ввиду использования глубоких множеств.

Для того чтобы подтвердить это базисное утверждение использовался подход Gated Recurrent Unit (GRU). Он необходим для объединения кодировок элементов текущего множества [10]. Входные данные модели —

это последовательность элементарных данных (записей). GRU показал отличные результаты и эффективность при выполнении задач кодирования и декодирования последовательностей [11, 10, 34]. Таким образом, была поставлена задача проверки предложенной гипотезы того, что использование глубоких множеств при моделировании выборки подмножеств приведет к созданию моделей, показывающих лучшие результаты по сравнению с моделями, просто конвертирующими подмножества в последовательности.

Условимся сначала производить кодирование поэлементно чтобы получить кодировку каждого элемента в последовательности. Затем передадим последовательность в многослойный двунаправленный алгоритм GRU [46]. Двунаправленный GRU состоит из 2х GRU: первый для приема исходных данных в определенном заранее порядке, второй для приема тех же данных, но в обратном порядке. Затем выходные данные для каждого отдельного элемента из обоих GRU объединяются и создается двунаправленный GRU [5]. Результаты применения двунаправленного GRU подаются в конечный слой, который непосредственно прогнозирует вероятность включения элемента, поэлементно.

При таком подходе не существует активного множества для хранения отобранных элементов в процессе работы, поэтому нет зависимости результатов от промежуточного состояния работы алгоритма. Однако, модель сохраняет возможность оценки отдельно взятого элемента относительно предыдущего и последующего элементов, ввиду двунаправленности применяемого GRU. Исходя из этого, предполагается использовать этот подход как базисный для сравнения моделей глубоких множеств.

2.3 Определение нелинейной модели

Для выработки специального нелинейного подхода к определению аппроксимирующих функций алгоритма DPP использовалась похожая методика. Для этой модели не использовался жадный алгоритм для отбора

элементов, а оценивался отбор каждого элемента как отдельно стоящее событие. Кроме того, не было добавлено никаких свойств для отражения последовательности для того, чтобы модель получилась инвариантной, не чувствительной к перестановке элементов.

Такая модель лучше подходит для алгоритмов аппроксимирования, которые по своей природе непоследовательны и не зависят от порядка исходных элементов. Это в корне отличает нелинейную модель от предыдущих созданных моделей, которые работают с элементами в линейном порядке.

Существует две причины для использования такой модели. Во-первых, требуется проверить верно ли то, что использование данных, полученных комбинацией нормального распределения и распределения Бернулли достаточно для эмуляции разброса данных при использовании DPP. Тезис состоит в том, что полученный случайный массив будет достаточен для использования в алгоритмах DPP.

Во-вторых, эта модель позволит определить верно ли то, что объединенная кодировка всех элементов множества позволит определить находится ли элемент в этом множестве. Если данное утверждение верно, тогда комбинации из кодировки конкретного элемента и объединенной кодировки всех элементов будет достаточно для определения нахождения любого элемента в конкретном подмножестве.

Для работы сети прогнозирования необходимо три типа входных данных:

1) z: случайный массив $\in R^d$;
2) e_i: кодировка текущего поверяемого элемента;
3) s: объединенная кодировка всех проверяемых элементов.

Результатом на выходе из модели будет вероятность того, что проверяемый элемент входит в выбранное подмножество. В проведенных экспериментах была использована та же самая архитектура кодировки как в разделе 2.1 и та же самая объединяющая функция как в разделе 2.2.

Для того чтобы получить образец подмножества используя эту модель, необходимо задать массив $z \sim N(0, I_d)$. Затем для каждого элемента e_i в исходном множестве вычислить вероятность p_i того, что он входит в выбранное подмножество. В заключении, по распределению Бернулли, параметризованному по p_i проверить должен ли элемент быть отобран.

2.4 Генерация данных

Был сгенерирован простейший набор данных, используя нормальное распределение и подход Gaussian Mixture Model. Такой упрощенный подход был применен для того, чтобы получить начальное понимание того, возможно ли использование полученной модели для определения аппроксимирующих функций DPP, которые будут применимы к реальным наборам данных, применяемым на практике при прогнозировании. Набор данных, сгенерированный через Gaussian Mixture, имеет очень простое, по сравнению с реальными данными, строение.

Набор данных был сгенерирован следующим образом:

1) Случайным образом сгенерировано распределение P используя Gaussian Mixture для d размерностей, состоящее из k независимых Гауссовых функций.

2) Для $i = 1, \dots, n$ образцов создано m независимых и нормально распределенных точек $X^{(i)} = x_1^{(i)}, \dots, x_m^{(i)}$ используя P, таким образом чтобы $X^{(i)} \sim P^m$. Набором точек $X^{(i)}$ характеризуется одно множество.

3) Установлено $y_j^{(i)} = 0$ или 1, используя любую аппроксимирующую функцию DPP. Значение 1 показывает, что элемент входит в отобранное подмножество, 0 показывает, что элемент не был отобран.

Установим матрицу для DPP как матрицу, заданную радиальной функцией определителя Грама для данных в $X^{(i)}$. В итоге получается оценка, негативно характеризующая рядом лежащие точки. Подход проще – с использованием простого линейного определителя Грама не сработал для

нашего случая, так как детерминанта определителя Грама равна нулю, если набор векторов, который она конвертирует линейно независим. Исходя из этого детерминанта транспонированной матрицы с $A \subset [n]$ будет равна нулю если $|A| \geq d$. Использование радиальной функции для определителя Грама гарантирует что каждый вектор бесконечен, и не существует набора линейно зависимых векторов.

Таким образом, корневой набор данных определен и состоит из m элементов. Получается, если набор данных определен описанным образом, тогда результат применения DPP к его элементам скорее всего выдаст множество, распределенное несколькими различными функциями Гаусса, нежели подмножество, где все элементы распределены по одному нормальному (Гауссову) распределению. Для Гауссовых распределений с относительно небольшим радиусом, два образца, взятых с использованием одной Гауссовой функции скорее все будут содержать одни и те же элементы в корневой матрице.

Поставленной задачей было точное моделирование нахождения точки максимума функции DPP. Таким образом функцией максимизации будет являться детерминанта минора матрицы $K^{(i)}$ из $X^{(i)}$ индексированной по множеству из $A^{(i)}$ элементов чьи значения y равны 1. В ходе эксперимента были заданы значения $y^{(i)}$ для нахождения максимумов.

Были проверены два различных алгоритма выборки, которые использовались для отбора нужных подмножеств из, сгенерированных ранее, множеств.

Первым тестировался простой жадный алгоритм, показанный в листинге 1. В начале проверяется добавление (если оно необходимо) какого элемента больше всего увеличит результат характеризующей функции подмножества. Если включение ни одного из элементов не увеличит характеризующую функцию, или все элементы данных были добавлены, возвращается выбранное подмножество.

Это – очень прямолинейная процедура выбора подмножества, и ввиду ее простоты, изначально предполагалось что такую процедуру будет проще всего воссоздать с помощью нейронной сети. Кроме того, порядок элементов на входе в алгоритм не влияет на результат. Однако, это достигается ценой того, что алгоритм отрабатывает за $O(n^2)$ времени для множеств с n элементами.

Исходя из того, что этот алгоритм, по своей сути, асинхронный, мы предполагаем, что непоследовательная нейронная модель сможет хорошо аппроксимировать его. С другой стороны, последовательная нейронная модель, предположительно, должна лучше подойти для аппроксимирования последовательного алгоритма.

Второй использованный алгоритм выборки — это алгоритм случайной двойной жадной максимизации, созданный для максимизации сабмодульных функций [8]. Он оперирует над множеством в порядке ввода в алгоритм, и поддерживает текущее активное множество, в которое добавляет выбранные элементы. В момент проверки включения элемента e_i в активное множество A, производится проверка того, насколько e_i увеличивает $f(A)$ при добавлении в A, по сравнению с тем насколько удаление этого элемента из не добавленных элементов B уменьшает $f(B)$. Затем алгоритм недетерминированно добавляет e_i в множество A с вероятностью, основанной на полученных значениях [8].

Листинг 1. Алгоритм жадного аппроксимирования целевой функции множества

Require: X как список элементов данных, f как функцию выбранного подмножества

```
1: function Greedy(X, f)
2:     n ← len(X)
3:     A ← {}
4:     while True do
5:         bestF ← f (A)
```

```
    bestIdx ← None
    for i ← 1 to n do
        if i ∉ A and f (A ∪ {i}) > bestF  then
            bestF ← f (A ∪ {i})
            bestIdx ← i
    if bestIdx == None then
        break
    A ← A ∪ {curIdx}
return A
```

В отличие от жадной аппроксимации, которая не имеет четких гарантий оптимальности, этот алгоритм является 1/2 – аппроксимацией: объективная функция, проверяемая на выбранном подмножестве, должна иметь значение равное хотя бы половине значения, полученного проверкой объективной функции на действительном оптимуме [8]. Ввиду того что детерминанта основного минора матрицы как функция от индексов отобранных строк или столбцов, является сабмодульной функцией, использование такой аппроксимации наряду с объективной функцией DPP дает нужные гарантии результата.

В качестве реализации этого алгоритма, использовался быстрый детерминированный алгоритм [36], позволивший быстро определить, как добавление или удаление элемента из множества скажется на детерминанте основного минора, содержащего индексы этого множества.

В дополнение к тому, что двойной жадный алгоритм гарантирует хорошую оптимальность результатов, он занимает всего $O(n)$ времени на вычисление детерминанты, что намного быстрее чем квадратичная характеристика количества времени, требуемого на вычисление детерминанты, необходимая обычному жадному алгоритму.

2.5 Проведенные эксперименты

Распределение, из которого выбирались использованные точки является однородной Гауссовой смесью (Uniform Gaussian Mixture) из 50 различных Гауссовых функций с 10 размерностями. Среднее значение каждой такой функции взято из однородного распределения $[0, 20]^{10}$, а среднеквадратичное отклонение было взято равномерно из $[0{,}5, 4]^{10}$ с нулевой ковариацией.

Каждый набор данных является множеством с 50 различными элементами, взятых из этой смеси. Набор данных для обучения содержит 10000 таких множеств. Наборы данных для валидации и тестирования содержат 2000 таких множества каждый. Данные были отобраны используя жадный либо двойной жадный алгоритмы, описанные выше.

Была проведена оценка данных, выбранных нелинейной моделью с использованием жадного алгоритма, и оценка данных, выбранных линейном моделью с использованием двойного жадного алгоритма. Такое разделение было выбрано ввиду различного поведения алгоритмов относительно порядка, в котором подаются данные. Двойной жадный алгоритм маркирует данные входного множества, опираясь на порядок, в котором элементы подавались в алгоритм. Линейная модель ведет себя похожим образом – решает выбирать или нет каждый элемент в строгой последовательности. С другой стороны, жадный алгоритм маркирует элементы входного множества не зависимо от их порядка, и нелинейная модель никак не оперирует порядком ввода при прогнозировании.

Результаты были получены после обучения на 40 эпохах с периодом обучения $1e-4$, используя оптимизатор Adam. Во всех экспериментах использовался PyTorch [46] – фреймворк глубокого обучения для Python.

Для обучения линейной модели элементы брались в том же порядке, в котором они передавались изначально в модель. Для каждого элемента использовалась модель для определения вероятности выбора этого элемента.

Затем подсчитывалась сумма негативных значений логарифмических функций правдоподобия относительно верного выбора для каждого элемента

и это значение добавлялось к функции потерь. Таким образом, минимизируемые будут являться отрицательными значениями функций правдоподобия выбора всего полностью верного подмножества.

Для обеих моделей проводилось обучение на 40 эпохах с размером пакетов равным 10, и использовалось проверочное множество для оценки правильности выборки моделей. Только минимальные изменения были зафиксированы в обучении, валидации и результатах тестирования после 40 эпох.

Для оценки результатов использовались две метрики. Первая – точность (accuracy) – какая доля элементов была промаркирована правильно при прогнозировании с помощью использованной модели. Точность выражается как:

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m} 1\left[\hat{y}_j^{(i)} = y_j^{(i)}\right], \quad (7)$$

Эта метрика использует верные проверочные метки только в самом конце - для сверки выбранного подмножества с проверочным. Таким образом, при использовании такой метрики для линейной модели не используется проверочное множество для выбора верного действия для каждого отдельного элемента.

Вторая метрика - средняя отрицательная логарифмическая вероятность (mean negative log likelihood, MNLL) для проверки успешности выбора для каждого элемента:

$$MNLL = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m} -\log \hat{p}_\theta\left(y_j^{(i)}\right), \quad (8)$$

Эта метка очень похожа на метрику оценки потерь при обучении, но не с учетом суммы значений логарифмических вероятностей, а с учетом их среднего значения. Таким образом эта оценочная метрика находит

вероятность применения верного действия для каждого элемента последовательно, затем совершает *верное* действие и переходит к следующему элементу. MNLL может быть представлена как логарифм от геометрической средней всех элементов для вероятности совершения верного действия для каждого элемента.

В отличие от точности, MNLL дает лучшее понимание того, насколько вероятно то, что модель выберет в точности верное множество. Точность, с другой стороны, имеет больший разброс, так как ошибка в начале может повлиять на всю последующую выборку. Поэтому, несмотря на то что точность проще интерпретируема, будем полагаться в большей степени на MNLL. В таблице 2.1 показаны полученные результаты точности для жадного аппроксимирования.

Таблица 2.1 – Точность жадного аппроксимирования

Тип кодировщика	Точность обучения	Точность тестирования
Последовательный	0,717	0,702
DenseNet	0,719	0,707
ResNet	0,717	0,707
ConcatNet	0,716	0,702

2.6 Результаты экспериментов

Были сравнены точности, полученные при использовании четырех разных типов архитектуры кодировки, путем использования кодировщиков для каждого элемента. Остальные архитектурные свойства каждой использованной модели были приведены к одному виду – размер полученной кодировки равен 256 и каждый кодировщик состоит из трех соединенных линейных слоев. Архитектура конечной сети прогнозирования также была приведена к виду – 3 соединенных слоя с 256 характеристиками в каждом слое с использованием активаций ReLU. Такой набор параметров показал лучшие результаты для каждого из кодировщиков. Полученные результаты показаны в таблице 2.2.

Было зафиксировано что тип архитектуры кодировки практически не влияет на *точность* модели на проверочном множестве. Также не было зафиксировано постоянства в порядке каждой из четырех схем кодировки на каждой из двух проверяемых задач. Однако был замечен довольно постоянный разрыв между точностями, полученными на тренировочном и тестовом множествах. Это показатель небольшого переобучения модели на тренировочном множестве, однако такой разрыв не критичен для подобной модели.

Таблица 2.2 – Точность аппроксимирования двойного жадного алгоритма

Тип кодировщика	Точность обучения	Точность тестирования
Последовательный	0,831	0,783
DenseNet	0,837	0,789
ResNet	0,830	0,784
ConcatNet	0,839	0,791

Метрика MNLL, с другой стороны, показала себя сильно зависимой от выбранной архитектуры кодировки. В таблице 2.3 показаны значения MNLL при использовании разных кодировщиков и последовательной модели. Данная модель использовалась для прогнозирования выборки с использованием двойного жадного алгоритма. Для этой кодировки число характеристик установлено равным 256, а число соединенных слоев равным 3 с 256 скрытыми нейронами в каждом.

Таблица 2.3 – MNLL аппроксимирования двойного жадного алгоритма

Тип кодировщика	Потери обучения	Потери тестирования
Последовательный	0,131	0,160
DenseNet	0,163	0,186
ResNet	0,151	0,182
ConcatNet	0,157	0,175

При использовании простейшего кодировщика – последовательной сети прямой связи были достигнуты минимальные потери обучения и тестирования, относительно других моделей. Значение MNLL, полученное на при тестировании показывает геометрическую среднюю вероятности правильной маркировки относительно сделанных прогнозов (вычисленную как $exp(-MNLL)$ и равную 0,85). Полученный результат отличается от результата кодировщика DenseNet, который достиг меньшего значения геометрической средней точности равного 0,83.

Было измерено как при неизменном типе кодировщика изменяется производительность модели при использовании различных архитектур сети прогнозирования. Для этих экспериментов использовалась последовательная модель для прогнозирования на наборе данных с использованием двойного жадного алгоритма. Изменяемыми параметрами архитектуры прогнозирования были количество слоев и размерность каждого скрытого слоя. Размерность кодировщика была фиксирована и равна размерности скрытых слоев. Таким образом, первый слой сети прогнозирования был просто проекцией. Для каждой из использованных моделей было продолжено использование сети прогнозирования из полностью соединенных слоев с активацией ReLU.

Как показано в таблице 2.4, изменение этих параметров сети почти не влияло на точность модели. Для сбора характеристик, приведенных в таблице, была использована архитектура ResNet с 5 слоями и количеством характеристик кодировки на выходе равным 64. Не было получено четкой зависимости между увеличением количества слоев или количества характеристик сети проектирования и MNLL модели.

Таблица 2.4 – Параметры архитектуры сети

Количество слоев	Количество характеристик	MNLL обучения	MNLL тестирования
3	32	0,266	0,265
3	64	0,233	0,235

3	256	0,131	0,160
5	32	0,248	0,248
5	64	0,221	0,222
5	256	0,153	0,179

На рисунке 1 показано как, в отличие от точности, изменяется значение MNLL при изменении размерности кодировки. В каждом из показанных измерений количество слоев сети прогнозирования было также фиксировано и равно размерности кодировки. Количество слоев в архитектуре кодировки было выставлено равным 5, а количество слоев сети прогнозирования равным 3. Как видно из графика, существует общая тенденция – при увеличении размерности кодировок, уменьшается значение MNLL тестирования. Это показывает то, что при добавлении дополнительных характеристик (до 256) качество модели улучшается. Однако при очень больших размерах кодировок (512 и 1024) было замечено серьезное переобучение и значение MNLL было намного больше, чем даже значение для модели с кодировкой из 32 характеристик. Эти замеры не представляют ценности и не показаны на графике.

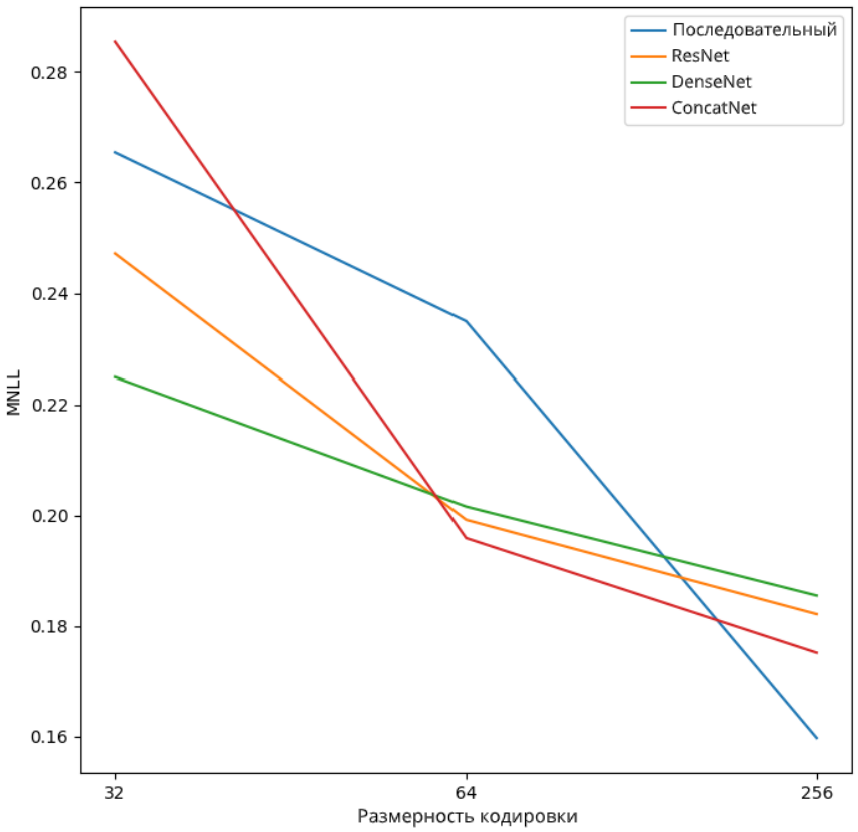

Рисунок 1 – Влияние размерности кодировки на значение MNLL при тестировании для использованных типов архитектур кодировки

Для того чтобы оценить результаты, полученные с использованием последовательной модели для прогнозирования на наборе данных двойного жадного алгоритма, был использован базисный GRU, описанный в разделе 2.2. Эти результаты показаны в таблице 2.5. В данном эксперименте варьировалось количество слоев двунаправленного GRU и размерность элементов кодировки. Было измерено какое влияние это окажет на точность и MNLL. Использовался последовательный кодировщик с 5 слоями, который показал лучшую производительность в предыдущих экспериментах. Модели GRU были обучены на 40 эпохах для согласованности с подходом, использованным для последовательных глубоких множеств.

Результаты проведенных экспериментов показаны в таблице 2.5. Использование модели базисного GRU показало намного лучшую точность

чем использование глубоких множеств. Однако, модель GRU показала MNLL хуже, чем последовательная модель. Как показано в таблице 2.4, последовательная модель с последовательным кодировщиком и сетью прогнозирования, состоящей из 5 256-размерных слоев показала значение MNLL тестирования равным 0,160. Этот результат лучше, чем любой из результатов, полученных с помощью GRU – самая эффективная модель, основанная на GRU, показала значение MNLL равное 0,181.

Таблица 2.5 – Базисный GRU для набора данных, полученных с использованием двойного жадного алгоритма

Количество слоев	Количество характеристик	Точность тестирования	MNLL тестирования
3	32	0,900	0,236
3	64	0,922	0,189
3	256	0,925	0,181
5	32	0,897	0,244
5	64	0,900	0,229
5	256	0,924	0,186

В таблице 2.1 показана точность, полученная на непоследовательных моделях при прогнозировании того, какие элементы будут выбраны жадным алгоритмом максимизации с использованием каждой из четырех архитектур кодировки. Полученная точность не очень высока: для каждого элемента верная маркировка была спрогнозирована только в 70% случаев, что отражено в результирующем MNLL. Ввиду недостаточной результативности этой модели, она не была использована в последующих экспериментах.

Стоит отметить, что точность, выражаемая через MNLL и подсчитанная на последовательной модели намного выше, чем точность, подсчитанная на проверочном множестве. Такое случается ввиду наличия активного состояния в модели: одна ошибка модели при выборе одного из первых элементов может привести к совершенно иным результатам при принятии решения по остальным элементам. Точность, которую показывает

эта модель намного ниже геометрической средней вероятности совершения верного действия над каждым элементом ввиду сверки с предыдущими действиями. Такое поведение делает данную модель не желательной в использовании для выборки *точного* верного подмножества, полученного двойным жадным алгоритмом, но, возможно, сможет помочь при выборке несходных подмножеств.

Выводы по главе 2

Как было описано выше, использовались два различных типа моделей – последовательная и непоследовательная. Обе предложенные модели работали по алгоритму с $O(n)$. Время выполнения выборки обеих моделей хуже, чем время подсчета действительной максимизации детерминанты объективной функции и выборки с помощью DPP.

Использование подхода глубоких множеств к построению моделей, основанных на суммах или средних кодировок каждого элемента, не привело к созданию модели, способной аппроксимировать детерминантные методы для выбора подмножеств с точностью выше, чем у моделей GRU. Несмотря на то, что значение MNLL получилось лучше по сравнению с моделями, основанными на GRU, улучшение получилось незначительным. Возможно, для получения еще более различных между собой моделей, нужно использовать другой набор данных.

Вопреки выводам, описанным в статьях по глубоким множествам, для нашего случая получилось, что такие модели либо сложно обучаемы при наличии различных локальных минимумов, которых нужно избегать, либо не имеют достаточных возможностей для работы над требуемыми в нашем случае функциями.

В случае недостаточных возможностей, вероятно, необходимо использовать более сложные кодировки элементов. Это может означать что необходима разработка более сложной архитектуры сети, или необходимо использовать массив характеристик с большей размерностью для каждого

элемента. Также можно предположить, что функция, оперирующая суммами кодировок элементов, могла не обладать достаточными возможностями для использованной модели. Подход глубоких множеств позволяет функции f оперировать суммами кодировок любой сложности для нахождения необходимой функции. Однако в нашем случае использована относительно простая нейронная сеть, которой может быть недостаточно для нахождения необходимой функции.

Для последующих исследований стоит рассмотреть какое-то отдельное свойство полученного подмножества, а не проверять получено ли в точности верное подмножество. Например, стоит проверить значения DPP выбранного подмножества так как эти значения варьируются между моделью с использованием глубоких множеств и моделью, основанной на GRU.

Глава 3 Вариативный автокодировщик с использованием графона

Автокодировка графов – довольно изученная проблема, но пока не было сформировано общепринятых оптимальных решений данной проблемы. Основной целью систем вариативной автокодировки является обучение на наборе данных графа тому, как сгенерировать новые графы, схожие с исходным набором данных, без того чтобы в итоге получился исходный граф.

Было предложено подойти к рассмотрению этой проблемы с точки зрения графона. Целью является использование представлений x_v для каждой вершины v для определения существования вершин графа. Такая задача легко решается для плотных слабо упорядоченных графов, как показано в разделе 1.3. Однако, подобная кодировка сложно выполнима для разреженных сильно упорядоченных графов.

3.1 Определение модели

Предположим, что входные и выходные значения модели – это ненаправленные сложные графы с маркированными вершинами, но без маркированных ребер, представленные как $G = (V, E)$. Данная модель подчиняется правилам для всех вариативных автокодировщиков, описанным в разделе 1.4 и имеет кодировщик и декодировщик.

Кодировщик преобразует исходный граф G в его представление, состоящее из кодировок каждой вершины графа: $\{e_{vi} | v_i \in V\}$. Декодировщик принимает на вход это представление графа и создает образцовый граф используя распределения вероятностей, основанного на представлениях вершин. Для того чтобы сгенерировать новый граф необходимо выбрать образцы представлений и предать их в декодировщик для генерации образцового графа.

В этой части эксперимента не было использовано представление графа в виде вектора для того, чтобы модель была расширяема под графы различных размеров. Модель, производящая кодировку фиксированного размера для небольших графов, например, не может показать удовлетворительный результат для огромных графов.

Таким образом, было принято, что процесс кодировки будет выдавать кодировку фиксированного размера для каждой *вершины* исходного графа. Когда необходимо получить кодировку всего графа можно использовать методы объединения кодировок вершин для получения кодировки графа. Однако, в данном эксперименте не было необходимости в использовании объединения графа.

Использованный кодировщик применяет алгоритмы MPNN, описанные в разделе 1.4, для передачи информации между вершинами исходного графа. После нескольких циклов передачи сообщений, финальное скрытое состояние каждого узла подается на выход из кодировщика.

В модели для кодировки был использован один из вариантов MPNN – свертка графа (graph convolution) [38]. Если существуют кодировки вершин e_v и e_w для вершин v и w функция сообщения $v \to w$ может быть представлена как:

$$M_t(h_w^t, h_v^t, e_{vw}) = h_v^t \quad (9)$$

Функция обновления задается как:

$$h_v^{t+1} = U_t(h_v^t, m_m^{t+1}) = ReLU(m_v^{t+1}) \quad (10)$$

Таким образом, на каждом шаге каждая вершина графа собирает скрытые состояния всех соседних вершин и применяет сглаживающую функцию (ReLU).

Вместо использования считывающей функции для кодировки всего графа, для его представления был собран набор кодировок всех вершин этого графа. При необходимости получения вектора фиксированного размера для

прогнозирования на уровне всего графа, для объединения кодировок вершин была использована суммирующая функция.

Для получения образцов кодировок вершин использовалась выборка из равномерного случайного распределения $[0,1]^d$. Для определения финальных скрытых состояний использовались 4 цикла передачи сообщений. Для вычисления значений финальных скрытых состояний была применена сигмоидальная функция для того, чтобы размерность кодировки была в пределах (0,1).

В начале эксперимента была разработана методология декодировки плотных графов, а затем эта методология модифицирована для разреженных графов. Декодидировщик плотных графов – более простая модель и естественный предшественник декодировщика разреженных графов. Как было показано в разделе 1.5, графоны, по своей природе, лучше подходят для плотных графов. Таким образом, для декодировщика плотных графов лучше использовать подход, основанный на графонах.

Используя небольшие модификации, была разработана модель, подходящая под разреженные графы. В частности, позволяя функции вероятности ребер зависеть от того, какие ребра уже входят в каждую вершину, мы можем расширить фундаментальную идею графонов для работы с сильно структурированными, разреженными графами. В экспериментах этого раздела был протестирован только разреженный декодировщик, так как это новый, не проверенный, компонент нашей системы.

На вход декодировщик получает набор кодировок каждой вершины. Одна из задач декодировщика – прогнозирование того, какие ребра лежат между какими вершинами.

Плотный декодировщик

В нашем случае декодировщик для плотных графов – это простой графон. Обозначим кодировку вершины v как $e_v \in R^d$. Тогда функция

графона, вычисляющая вероятность существования ребра между двумя вершинами может быть записана как:

$$f_\theta(e_v, e_u), \quad (11)$$

где θ – параметр функции. Если принять что кодировки $\{e_v | \forall v \in V\}$, декодировщик проверяет каждую пару вершин u, v и вычисляет $p(u, v) = f_\theta(e_v, e_u)$. Затем выбирается событие из распределения Бернулли, равное 1 с вероятностью $p(u, v)$ и, если значения события равно 1, создается ребро между этими двумя вершинами.

Такая техника декодирования слишком общая для плотных графов без сложных структурных свойств, таких как фиксированная или максимальная степень. Таким образом, данная модель лучше всего подходит для плотных слабо упорядоченных графов, таких как социальные сети или сети цитирования. С другой стороны, когда существование двух разных ребер в графе не является условно независимым – данная модель показывает не лучшие результаты.

Модель разреженного декодировщика была создана специально для графов, в которых вероятности существования двух отдельных ребер могут быть условно зависимы друг от друга. Предпосылкой к созданию этой модели служат молекулярные графы, описанные в разделе 1.5. Молекулы – сильно упорядоченные графы, в которых малейшие неточности могут подорвать стабильность всей молекулы [4]. Исходя из этого обычный графон не очень подходит для декодировки молекул, так как они представляют собой плотные графы без четкой структуры [44].

Разреженный декодировщик – это графон, дополненный для того, чтобы вероятность выборки ребра (u, v) зависела от того, какие ребра уже были отобраны. В частности, ребра, уже попадающие на u или v в момент выборки, влияют на вероятность выбора проверяемого ребра. В этом случае функция графона будет выглядеть как:

$$f_\theta(e_v, s_v, e_u, s_u), \quad (12)$$

где e_v и e_u – ранее упомянутые кодировки вершин v и u. Дополнительные параметры s_v и s_u – это суммы кодировок каждой вершины, уже соединенной с вершинами v и u:

$$s_v = \sum_{v_i \in N(v)} e_{v_i}, \quad (13)$$

где $N(v)$ – текущее множество соседних узлов для узла v. Если v и w имеют два различных узла, то w будет в $N(v)$ два раза.

Такое определение модели позволяет декодировщику поддерживать многие химические свойства:

– фиксированная степень для каждой вершины, в зависимости от метки вершины;

– существование ребер зависит от того, какие ребра находятся в проверяемой вершине;

– повышенная вероятность нахождения повторяющихся подграфов.

В нашем случае декодировщик – это нейронная сеть из 3-х слоев с активацией ReLU. Конечной величиной является скаляр, чья сигмоидальная функция показывает вероятность выбора проверяемого ребра. Число скрытых нейронов в каждом слое варьировалось в качестве гиперпараметров этой модели.

Одной из целей вариативного автокодирования является создание модели для получения новых образцовых сущностей [2]. В экспериментах этой части работы такая модель использовалась для генерации новых молекул. На вход генеративная модель получала n – количество атомов новой молекулы.

Данная генеративная модель использует два шага:

1) Выборка n кодировок $e_1, \dots, e_n$;
2) Выборка графа из полученных кодировок.

Для того чтобы отобрать кодировки вершин, кодировки были смоделированы как $v_i \in R^d$, отобранные независимо из $U[0,1]^d$. Этот подход был применен в обучении путем применения *генеративно-состязательной сети* – добавления состязательных потерь к общим потерям процесса обучения для кодировки.

Такие состязательные потери характеризуют насколько вероятно то, что дискриминатор считает, что проверяемая кодировка сгенерирована кодировщиком, а не взята путем генерации из $P_d = U[0,1]^d$. Дискриминатор обучен параллельно с моделью путем обучения на действительных случайных примерах, сгенерированных из целевого P_d и не случайных примеров, сгенерированных получением кодировок вершин графа. Дискриминатор обучен различать эти два типа входных данных. Модель обучена обманывать дискриминатора, понижая параметры примеров, которые дискриминатор признает неслучайными.

Такой метод принуждения модели приблизиться к желаемому ограничению может быть характеризован как генеративно-состязательная сеть (generative adversarial network, GAN) [20, 3]. Генеративно-состязательные сети чаще всего используются для получения данных, которые выглядят так, как будто они сгенерированы из исходного источника данных [20]. В нашем случае задачей является генерация кодировок вершин, которые выглядят как будто они были сгенерированы из P_d, хотя на самом деле они сгенерированы кодировщиком для передачи характеристик исходной вершины.

В данном случае был использован относительно простой дискриминатор – нейронная сеть из 2-х слоев с активацией ReLU. На вход дискриминатора подавался вектор, который был либо кодировкой вершины, либо случайным вектором. К вектору была применена сигмоидальная функция для вычисления вероятности того, что выбранный дискриминатором результат является не случайным. Потери дискриминатора – это двоичные перекрестно-энтропийные потери правильного предсказания

случайных векторов и кодировок. Эти дискриминантные потери добавляются к общим потерям модели и сумма представляет собой двоичные перекрестно-энтропийные потери предсказания дискриминатором того, что кодировки модели являются действительно случайными.

Метод, описанный выше больше подходит для генерации небольших графов. Это происходит ввиду того, что алгоритм проверяет все возможные вершины, которые могут быть сгенерированы. Время работы такого алгоритма составляет $O(n^2)$ для графа с n вершин. Кроме того, модель принимает во внимание только низкоуровневое строение графа, но для обеспечения стабильности больших графов требуется учитывать также высокоуровневое строение. Для принятия точных решений и для генерации молекул, например, требуется более стабильный результат на больших графах.

Один из альтернативных путей генерации графов – ограничение набора соседних вершин. Если вершина ограничена тем, что она может быть соединена с максимум c других вершин, то модель будет намного более эффективнее генерировать возможные варианты молекул. Это позволит обойти вычислительную сложность генерации и позволит влиять на то, какие соседи могут быть у конкретного атома.

Для того чтобы генерировать верные молекулы больших размеров, декодировщик может быть «прогрет» путем подачи на вход набора существующих ребер между вершинами. Например, если требуется чтобы какой-то набор функциональных групп был включен в генерируемую молекулу, объединение этих функциональных групп может быть подано в декодировщик в дополнение к набору еще не соединенных вершин. Декодировщик может просто *продолжить* генерацию с этого места, прогнозируя какие *дополнительные* вершины должны быть добавлены к уже введенным вершинам.

3.2 Проведенные эксперименты

Как упоминалось ранее, для проверки полученной вероятностной модели в экспериментах используется изначально заданный набор молекул. В частности, используется набор данных QM9 [52,50] – набор небольших молекул, содержащих атомы C, N, O, F и H. Каждая молекула может содержать максимум 9 атомов отличных от водорода, что делает этот набор более простым в представлении нежели наборы из различных больших молекул [50].

Целью является использование только самой базовой информации о молекулах для проведения автокодировки этих графов. Таким образом, изначальный набор свойств каждого атома – это только тип химического элемента. Граф конвертируется в мультиграф путем представления двойных и тройных связей, двумя и тремя ненаправленными ребрами, соответственно. Это – единственные специфичные для молекул характеристики, использованные для совершения автокодировки.

Эксперименты проводились для проверки того, насколько успешно полученная модель сможет автокодировать молекулярные графы из набора данных QM9. Для разработки и обучения модели был использован Pytorch [46]. Для обучения был использован оптимизатор Adam с начальным временем обучения $1e-2$ и уменьшение веса равным $5e-4$. Для обучения модели и дискриминатора были использованы два различных оптимизатора: оптимизатор с постоянным временем обучения и оптимизатор с параметрами угасания, соответственно.

Потери для модели состоят из 2-х частей: потери восстановления и потери дискриминатора. Потери дискриминатора описаны в разделе 3.1 и показывают, насколько плохо модель способна обмануть дискриминатора.

Набор данных QM9, использованный для обучения состоит из относительно небольших графов. Ввиду этого, для каждого графа удалось оценить вероятность того, что каждая из вершин действительно существует. Согласно определению модели декодировщика разреженных графов: если два ребра не имеют общих вершин, то выбор одного ребра условно не

зависим от того, выбрано или нет другое ребро. Таким образом, для того чтобы вычислить потери при реконструкции, было произведено разбиение набора вершин исходного графа на набор соответствий. Соответствия – набор ребер, не имеющих общих вершин. Исходя из этого, при каждом цикле добавления ребер в декодировщик, рассчитывается двоичная перекрестно-энтропийная потеря (binary cross-entropy, BCE) [46] выбора верного действия для каждого, пока не отобранного ребра, которая вычисляется как:

$$BCE = -\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}\log\left(\hat{p}_\theta\left(y_{ij}\right)\right) + \left(1 - y_{ij}\right)\log\left(\hat{p}_\theta\left(1 - y_{ij}\right)\right), \qquad (14)$$

где $y_{ij} = 1$ если между вершинами i и j существует ребро, $y_{ij} = 0$ в противном случае.

Затем суммируются действительно верные соответствия и процесс повторяется до тех пор, пока соответствия не закончатся. Сумма перекрестно-энтропийных потерь от выбора верного действия перед добавлением каждого соответствия показывает потери при восстановлении.

Набор данных QM9 был разделен на тренировочную, валидационную и тестовую части. Тренировочные данные – случайное множество из 110000 молекул. Валидационные данные были использованы для выбора моделей, а тестовые данные для оценки полученных моделей. Все показатели, описанные ниже были вычислены с использованием тестовых данных.

Для оценки результатов обучения для обеих моделей был использован показатель точности при выборе ребра. Этот показатель вычисляется как средняя для всех ребер вероятность совершения верного выбора. Точность рассчитывается пошагово, используя подход, описанный в разделе 3.1. Перед каждым шагом проверки соответствия, подсчитывается средняя вероятность совершения верного выбора, затем среднее по всем соответствиям. Точность для графа $G = (V, E)$ можно записать как:

$$\mathrm{accuracy_G} = \frac{1}{\mathrm{M}} \sum_{\mathrm{m=1}}^{\mathrm{M}} \frac{1}{|\mathrm{V}|^2} \sum_{\mathrm{i=1}}^{|\mathrm{V}|} \sum_{\mathrm{j=1}}^{|\mathrm{V}|} \hat{\mathrm{p}}_\theta \left(\mathrm{y}_{\mathrm{ij}}^{(\mathrm{m})} \right), \quad (15)$$

где $y_{ij}^{(m)}$ равно 1 если между вершинами i, j имеется ребро после первых $m-1$ соответствий, и 0 если ребра не существует. Тогда общая точность будет средней точностью всех графов тестовых данных:

$$accuracy = \frac{1}{n} \sum_{i=1}^{n} accuracy_{G_i} \quad (16)$$

Кроме того, как оценочные показатели отображающие, насколько хорошо модель обучилась принимать решение какие вершины соединять, использовались потери – сумма перекрестно-энтропийных потерь и потерь дискриминатора.

Для того чтобы проверить эффективные потери дискриминатора для d-размерностей, полученные данные были визуализированы для небольших размерностей используя гиперкубы.

В заключении, было визуально оценено насколько верно кодировки вершин показывают их маркировку – насколько верно сгруппированы кодировки вершин, относительно их химического элемента.

3.3 Результаты экспериментов

В таблице 3.1 показаны значения точности, достигнутые полученной моделью. В ходе работы варьировались размерности различных параметров модели и проверялось как ведет себя каждый из оценочных параметров.

Таблица 3.1 – Точность вариативной автокодировки модели на наборе данных QM9

Размерности кодировки	Скрытые размерности	Точность	Потери
1	16	0,7385	82,7716
16	16	0,7313	79,838

16	32	0,562	83,693
32	32	0,6944	81,554
64	64	0,7915	87,546

Рисунок 3 показывает скрытые кодировки для каждого атома в тестовом наборе данных. Атомы разделены по цветам, согласно легенде рисунка. На рисунке показаны четыре различных стадии процесса обучения модели: 10, 20, 30 и 40 эпох. Модель была обучена с соотношением потерь восстановления к потерям дискриминатора равным 7 : 3 и конечной размерностью для каждой вершины равной 2 для лучшей наглядности результатов.

На рисунке 2 показан пример распределения кодировок вершин, созданных моделью. В частности, рисунок показывает вероятность того, что две вершины, не имеющие в данный момент соединения между собой, будут соединены. Для удобства восприятия была использована кодировка вершины с одной размерностью. Ввиду этого упрощения кодировки на рисунках 2 и 3 отличаются. Рисунок 2 включен для графической демонстрации диапазона различных кодировок и их относительных вероятностей начального связывания вершин. На рисунке видно, что процедура обучения обеспечивает перестановочную инвариативность между двумя наборами входных данных, и присутствует почти полный диапазон вероятностей (0, 1).

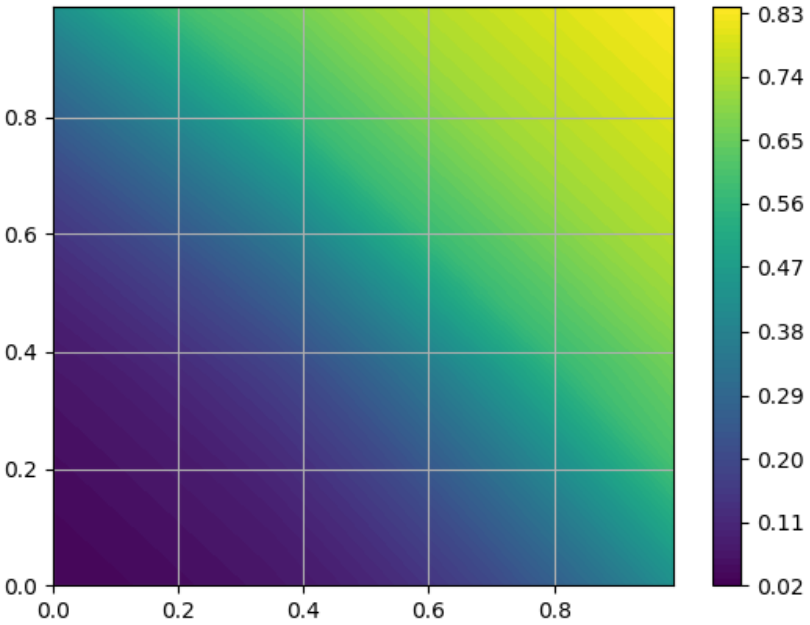

Рисунок 2 – Распределение вероятностей для пар кодировок вершин

Из полученных результатов видно, что потери дискриминатора не оказали предполагаемого воздействия: вместо того чтобы вынудить модель равномерно распределить кодировки, кодировки были сильно сгруппированы в некоторых местах кодировочного пространства. Исходя из этого, потери дискриминатора не гарантируют достаточной выборки, при использовании кодировок с выборкой из $U[0,1]^d$.

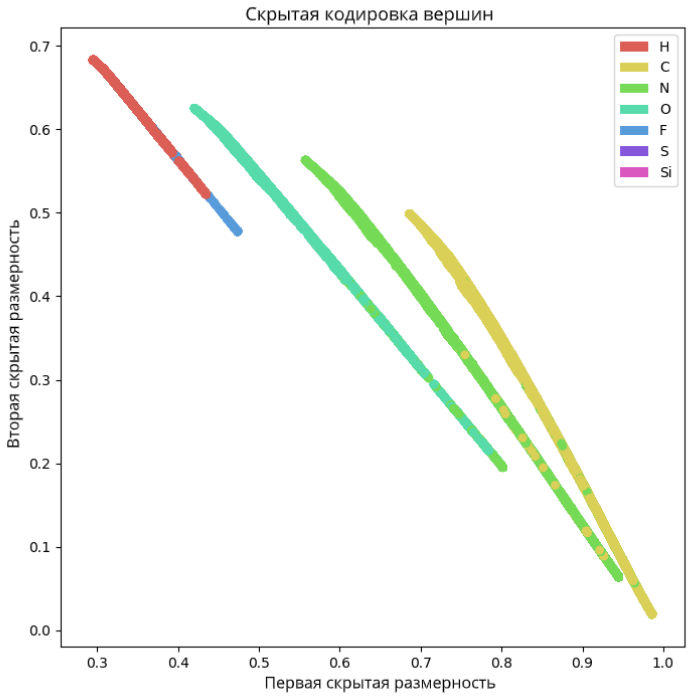

Рисунок 3 – Скрытая кодировка вершин: 10 эпох

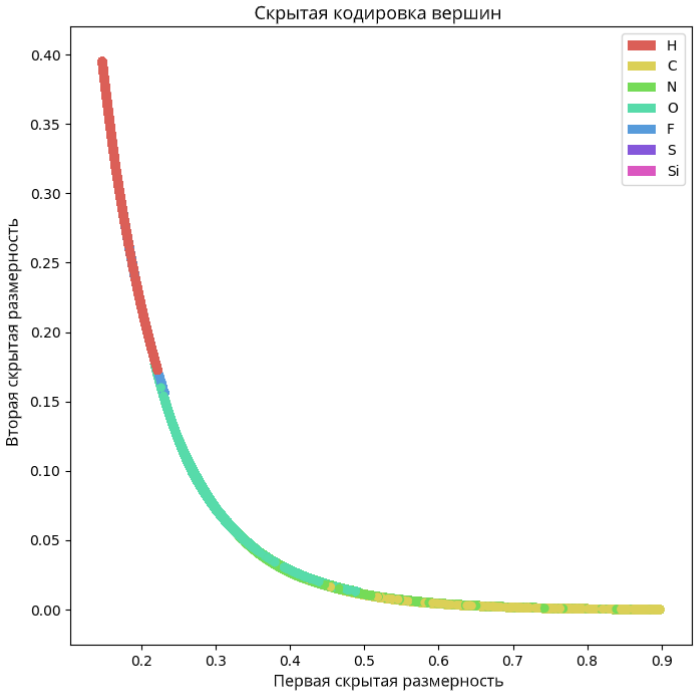

Рисунок 4 – Скрытая кодировка вершин: 20 эпох

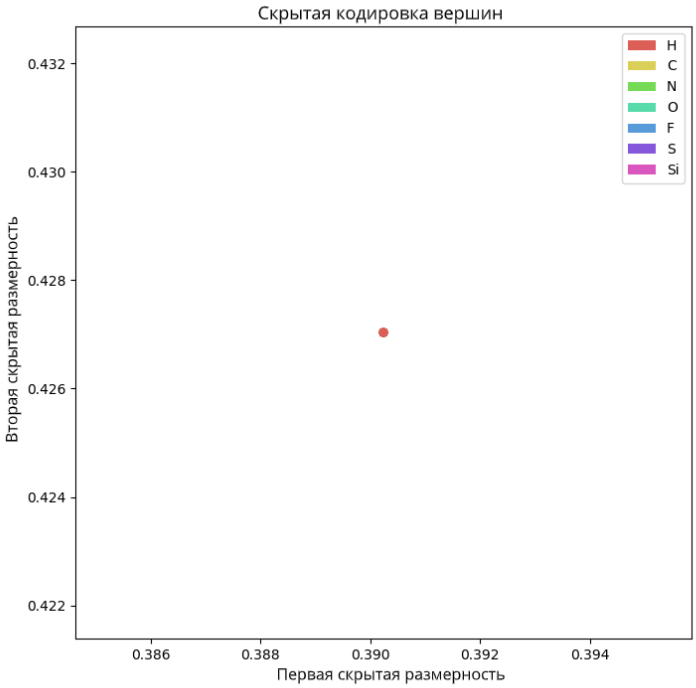

Рисунок 5 – Скрытая кодировка вершин: 30 эпох

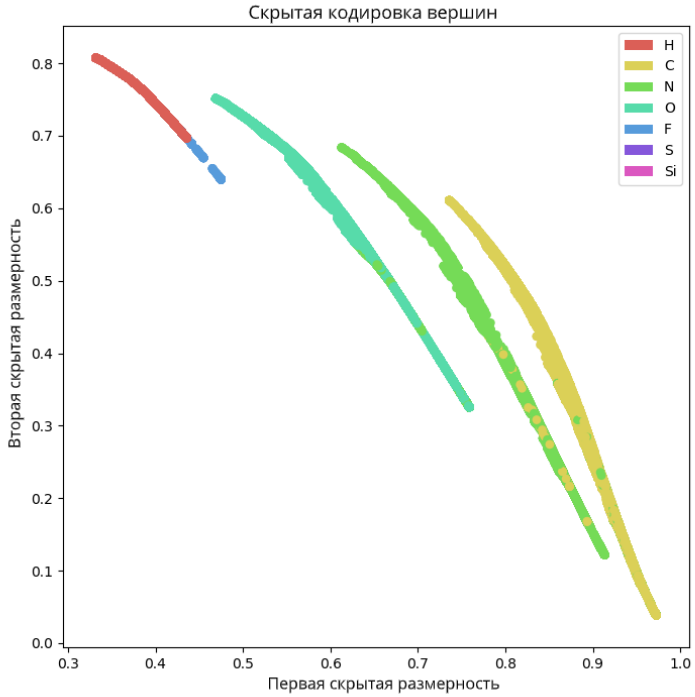

Рисунок 6 – Скрытая кодировка вершин: 40 эпох

То, как изменяются скрытые кодировки для вершин, показывает, как модель пыталась отвлечь направление хода вычислений дискриминатора, пытающегося распознать результаты работы модели. В частности, модель просто изменяла места, в которых в которых происходила кодировка вершин определенных типов. Между 10 и 20 эпохами (рисунки 3 и 4) можно увидеть, что модель просто сдвинула места кластеров в скрытом пространстве. На рисунке 5 наблюдается вырожденный случай схлопывания в точке скрытого пространства, которой, вероятно, дискриминатор на предыдущем шаге назначил высокую вероятность принадлежности истинному равномерному случайному распределению. В заключении, на рисунке 6 можно видеть, как модель сходится обратно к распределению, схожему со скрытым распределением 3, после того как как дискриминатор начал выявлять вероятно случайные точки.

Таким образом, использования данного дискриминатора было недостаточно для получения равномерного распределения кодировок. Исходя из этого, создание новых графов путем выборки кодировок вершин из целевого равномерного распределения не представляется возможным.

В целом, получилось, что кодировки хорошо передают тип элемента, который представляет каждая вершина. На рисунках 3 и 6 видно, что вершины определенных элементов сгруппированы. Единственный случай, когда вершины одного элемента находятся в одной группе с вершинами другого элемента, имеют место, когда два элемента имеют одинаковое общее количество связей. В частности, наблюдается пересечение между кодировками атомов H и F ввиду того, что эти атомы всегда имеют только одну связь. Также существует небольшое пересечение между атомами N и C. Это случается, когда N имеет 4 связи или когда C имеет только 3 связи, что нетипично, но встречается в используемом наборе данных.

На рисунке 4 наблюдается похожая группировка на скрытом пространстве. Однако видно, что распределение непрерывно, без существенных разрывов между группами. Это – явное отличие от

распределений 3 и 6. Таким образом, в этих случаях модель придает меньшее значение элементам вершин.

Наконец, на рисунке 5 виден вырожденный случай, описанный ранее. Такое поведение характерно для случаев, когда потери дискриминатора сильно преобладают над потерями восстановления.

В таблице 3.1 показаны значения точности, достигнутые моделью вариативной автокодировки при различных значениях параметров архитектуры кодировки и декодировщике в виде графона. В частности, изменялось количество скрытых слоев в кодировщике и декодировщике совместно и размерность кодировок, подаваемых в графон. Точность вычислялась на тестовом наборе данных.

Потери модели не изменялись последовательным образом при изменении размерности кодировки или изменении размерности скрытых слоев. Потери существенно возросли при использовании 64-размерных скрытых слоев и кодировок, хотя при предыдущих значениях потери понижались.

Точность модели также не изменялась последовательным образом при изменении размерности кодировки или изменении размерности скрытых слоев. Максимальная точность была достигнута при установке обоих параметров равными 64. Такой набор параметров характеризует максимально сложную из проверенных архитектур. Однако, наименее точной была не модель с минимальным количеством параметров.

При использовании значений параметров равными 16 и 32, соответственно, модель показала серьезное переобучение при проверке на тестовых данных. Полученной модели удалось достичь сравнимую с другими моделями тренировочную точность, однако тестовая точность оказалась существенно ниже других моделей, примерно на 0,17. Более того, такое поведение аномально – ни одна из других моделей с большим количеством параметров не показали такого сильного переобучения. Это может быть связано с особенностью методов обучения с дискриминатором.

Дискриминатор мог, например, привести модель к вырожденному состоянию в текущем или около него моменте процесса обучения.

Выводы по главе 3

После проведения экспериментов над разреженным декодировщиком путем использования набора данных из химической отрасли, были выявлены некоторые свойства, специфичные для такой модели. Во-первых, нет способа обновить вероятность существования ребра, основываясь на том, что ребро *не было* выбрано. Такое поведение позволяет модели генерировать неверные молекулы с очень высокой вероятностью. Например, если каждое ребро было выбрано и определено, что не существует ребер кроме одного, то модель не сможет обработать событие того, что ребра не существуют, и назначить более высокую вероятность выбранному ребру.

Во-вторых, модель дает не точные вероятности, когда существует множество одинаковых и симметричных ребер. Возьмем, например, циклогексан – молекулу, показанную на рисунке 7. Ввиду высокой симметричности этой молекулы, кодировка каждого атома углерода будет равна значению e_C, а кодировка каждого атома водорода будет равна значению e_H. Таким образом, после обучения, модель будет обучена тому, что вероятность $f(e_H, 0, e_C, 0)$ должна быть равна 1/6, потому что каждый атом углерода соединен только с двумя атомами водорода. Это может привести к тому, что модель будет формировать совершенно другие структуры, даже если использовать кодировку, в точности передающую циклогексан.

В ходе экспериментов было выявлено, что перечисленные выше недостатки приводят к тому, что полученная модель показывает не удовлетворительные результаты при автокодировании. Модель, целью которой является достаточно точная автокодировка структур данных в виде графов должная уметь справляться с подобными ситуациями. Один из способов, который, возможно, сможет исправить ситуацию, состоит в том,

чтобы разделить вероятность существования ребра на две составляющие. Первой составляющей будет вероятность того, что у вершины остаются не выбранные пока ребра, а второй будет функция softmax относительно других вершин, которые могут соединяться с этой вершиной. Это позволить дать более надежные гарантии того, что у вершины будут существовать еще не выбранные ребра, и даст возможность модели работать в случае сильной симметрии.

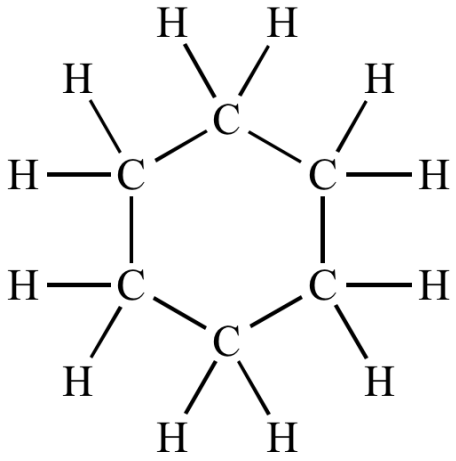

Рисунок 7 – Молекулярная структура циклогексана (C_6H_{12}) [24]

Кроме того, способ, который, как предполагалось, обеспечит равномерное распределение вершин по пространству кодирования, был в корне ошибочным. Использование такого способа привело к схлопыванию и цикличности модели, что является известной проблемой генеративно-состязательных сетей [49]. Возможно, стоит использовать более надежный метод, который работает с группами кодировок вершин. Предполагается что такой метод сможет гарантировать то, что кодировки случайны и равномерно распределены.

Глава 4 Иерархический кодировщик графов

В этой главе описана методика, разработанная для иерархического кодирования графов и предназначенная для того, чтобы кодирование графа могло распознавать структуру и взаимодействия более высокого порядка, чем при использовании обычных методов. В отличие от предыдущей части, работа в этой части сфокусирована на графах со свойствами, подобными химическим молекулам – часто встречающиеся подграфы и разреженные связи.

На высоком уровне фаза кодирования применяемой модели состоит из замены каждого распознанного подграфа, который назовем *компонентом*, одним узлом, представляющим весь этот компонент. Затем происходит фаза передачи сообщений на полученном графе, возможно объединенном с исходным графом. На заключительном шаге скрытые состояния компонентов собираются вместе для того, чтобы получить представление всего графа.

Идея такого подхода состоит в том, что путем передачи сообщений на графе более высокого уровня, модель будет отслеживать взаимодействия между более крупными компонентами графа. Такие виды химических взаимодействий между функциональными компонентами в молекулах широко используются в химической индустрии для определения свойств молекул [4, 16].

4.1 Интеллектуальный анализ подграфов

Вместо того чтобы использовать уже собранную вручную базу данных функциональных компонентов, попытаемся выявить функциональные компоненты путем определения подграфов. В частности, попытаемся разработать метод, позволяющий выявить какие подграфы, часто встречаются в молекулах исходного набора данных.

Алгоритм gSpan, описанный в разделе 1.4 имеет нужные для решения этой задачи возможности. На выход этот алгоритм выдает подграфы в виде

леса (forest), где компонент A является родителем компонента B если A – подграф B, ввиду того что если A присутствует в k графов, то каждый подграф A также присутствует как минимум в k графов. Таким образом, представление в виде леса является эффективным методом представления компонентов. Пример одного из деревьев компонентов показан на рисунке 8.

Для экспериментов использовался набор данных QM9, описанный в разделе 3.2. Для того чтобы собрать базу данных подграфов для использования на этапе кодирования ко всему набору данных применялась реализация gSpan gBolt [63] с минимальной значением поддержки 10%. На выходе получились 131 различных подграфа, которые использовались в дальнейшем для сегментации молекул.

4.2 Алгоритм сегментации

Решение об отделении замены циклов от замены остальных компонентов было сделано по двум причинам. Во-первых, циклические структуры зачастую выделяются как отдельные функциональные группы [4]. Во-вторых, некоторые компоненты, выявленные с помощью gSpan, могут быть подграфами циклов, поэтому схлопывание циклов перед заменой не цикличных компонентов приводит к более желаемой сегментации.

Таким образом, первый шаг сегментации – удаление каждого цикла и замена на компонент цикла. Затем производится замена всех оставшихся подграфов. Данный алгоритм сегментации приведен листинге 2.

Порядок словаря V, передаваемого в алгоритм определяет какие компоненты будут заменены первыми. Отдельная молекула может быть сегментирована совершенно разными способами, в зависимости от входного словаря. Таким образом, выбор того в каком порядке расположить компоненты в словаре сильно влияет на граф компонентов.

Листинг 2. Алгоритм сегментации графа молекулы

Require: *G* исходный граф молекулы, *V* список подграфов

Require: *cycles*(·) возвращает все циклы исходного графа

Require: *replace*(·,·) заменяет каждый экземпляр подграфа на узел

```
1: function Segment(G,V)
2:     C = cycles(G)
3:     for c ∈ C do
4:         G ← replace(G,c)
5:     for v ∈ V do
6:         if v ∈ G then
7:             G ← replace(G,c)
8:     return G
```

Для конвертации представлений молекул в виде строк в графы и получения химический свойств использовался программный продукт RDKit [40]. Для получения списка циклов, присутствующего в каждой молекуле, использовался функционал Symmetric Smallest Set of Smallest Rings (SymmSSSR) из RDKit.

Важно отметить, что в нашем случае допускается, чтобы узлы циклов перекрывались атомами, но не позволяется никаким узлам нециклических компонентов перекрываться. Это сделано в том числе для того, чтобы множественные «сплавленные» кольца можно было разделить на несколько дискретных циклов. Это приводит к избыточной сегментации для частей молекул, которые представляют собой кольца.

Молекулярный граф можно сегментировать различными способами, и точный способ сегментации зависит от порядка входного словаря возможных подграфов. Было решено разбивать молекулы на максимально большие сегменты. По сути, это означает проверку наличия листьев дерева компонентов перед проверкой внутренних узлов. Таким образом, входной словарь представляет собой сбор результатов обратного обхода каждого из деревьев в лесу подграфов сгенерированного gSpan. Использование обратного обхода гарантирует что наличие потомков данного компонента

проверяется прежде, чем происходит поиск самого компонента. Это – жадный подход, гарантирующий сегментирование графа сначала на большие компоненты.

Пример того, как производится такой обход показан на рисунке 8. Показанная фигура – дерево леса сгенерированного gSpan на наборе данных QM9 со значением поддержки равным 10%. Запись SMILES в каждом узле показывает химическую структуру, представляемую данным узлом, а число показывает порядок, в котором данный подграф будет представлен в конечном словаре. Предок каждого узла – это подграф данной молекулы. Таким образом, сначала происходит разбиение на максимально большие компоненты, а поиск внутреннему узлу производится только после того, как прошел поиск по всем его потомкам.

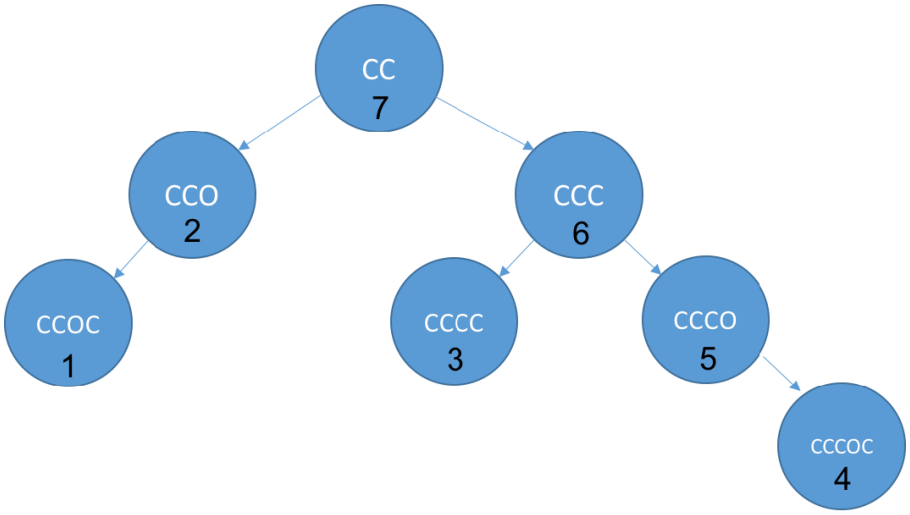

Рисунок 8 – Обратный обход дерева, сгенерированного gSpan

Интеллектуальный анализ подграфов производился на наборе данных молекул после удаления всех атомов водорода. Количество и тип исходящих из какого-либо компонента ребер характеризуют число неявных атомов водорода, существующих в нем, и, следовательно, включение атомов водорода в компоненты не является необходимым. Кроме того, включение атомов водорода приводит к тому, что многие подграфы, которые в химии считаются, по существу, одинаковыми, будут представлены раздельно на выходе из gSpan. Геометрия тяжелых атомов чаще всего лучше представляет

химические свойства [4]. Для проверки существования подграфа в молекуле использовался функционал сравнения подструктур из RDKit [40].

При создании графа компонентов выделялись два типа графов: с вершинами, состоящими только из компонентов, и с вершинами, состоящими из компонентов и атомов. Включение атомов как вершин позволяет компонентам собирать информацию из атомов этих компонентов при передаче сообщений между атомами.

В результате сегментации получился граф, состоящий из узлов, представляющих цикл, не циклический подграф или отдельный атом. Отдельный атом представляет собой атом в исходной молекуле, не вошедший ни в один из выбранных подграфов.

При использовании не циклических подграфов для каждого атома исходной молекулы дополнительно добавлялись вершины в виде атомов. Эти вершины отличны от вершин в виде компонентов.

Все типы ребер, существующих в графе молекулы, (одиночная, двойная, тройная и ароматическая связи) существуют и между компонентами. Однако, был добавлен новых тип ребра, существующий только между компонентами в виде циклов, представляющий количество атомов, общих для этих циклов. В органической химии часто встречаются циклы, имеющие один, два, три или более общих атомов. Примеры таких типов связей показаны на рисунке 9. Исходя из этого, были добавлены три новых вида ребер, применимых для связей графов второго уровня.

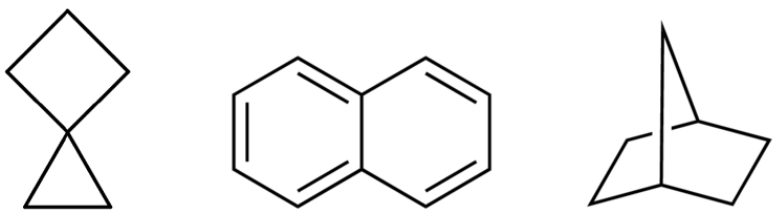

Рисунок 9 – Молекулы со сплавленными кольцами, имеющие один (слева), два (в центре) и 3+ (справа) общих атома

Все описанные ребра существуют между компонентами и, ввиду этого, присутствуют в обоих типах графов компонентов.

Для второго типа графа компонентов, который содержит отдельные атомы, были добавлены два дополнительных типа ребер:

– ребро «компонент→атом», направленное от какого-либо компонента к любому атому этого компонента;

– ребро «атом→компонент», направленное от какого-либо атома к компоненту или компонентам, содержащим этот атом.

Разделение на два типа ребер необходимо ввиду того, что в нашем случае все ребра графа считаются направленными. Все ребра компонент-компонент симметричны, а два дополнительных типа вершин для связей типа компонент-атом добавлены для представления двух, различных между собой, отношений.

4.3 Определение модели

В этом разделе описано как с помощью иерархического подхода можно улучшить любой существующий метод кодировки. Затем приведена модель, использованная для проверки такого подхода.

Предлагаемый иерархический подход для кодировки графов может быть применен к различным схемам кодировки. Использование такого подхода для получения кодировки на уровне компонентов на примере графов молекул состоит из следующих шагов:

1) использование gSpan для получения и сохранения подграфов;

2) сегментация графов молекул на циклы и полученные ранее компонентов;

3) сборка нового графа, состоящего из компонентов, и присвоение вектора свойств для каждой вершины и ребра этого графа;

4) отправка полученного графа на кодировку выбранным способом для получения высокоуровневой кодировки.

После получения высокоуровневой кодировки, эта кодировка может быть использована как есть или объединена с кодировкой, полученной из низкоуровневого графа. Объединение позволяет результирующей кодировке содержать информацию, собранную на нескольких уровнях графа. Такой подход использовался в последующих экспериментах ввиду того, что он расширяет возможности модели.

Для определения конкретного варианта кодирования более высокого уровня необходимо решить три базовых задачи: выбор между двумя возможными графами компонентов, выбор метода установки векторов свойств вершин и ребер нового графа компонентов и выбор метода кодировки графов. Опишем проверенные варианты решения этих задач.

Первая проверяемая модель использовала MPNN, описанный в разделе 1.5, для кодировки графов атомов и компонентов. Если принять кодировки узлов $h_w^{(t)}$ для вершин v и кодировки ребер e_{wv} для ребра $w \to v$, тогда сообщение $w \to v$ для времени t можно представить как:

$$M_t\left(h_w^{(t)}, h_v^{(t)}, e_{wv}\right) = \tau\left(W_1 e_{wv} + W_2 h_w^{(t)}\right) \quad (17)$$

Сообщения объединяются простым суммированием вида $m_v^{(t+1)}$, а скрытые состояния обновляются в виде:

$$h_v^{(t+1)} = m_v^{(t+1)} \quad (18)$$

Таким образом, и функция сообщения, и функция обновления используют весовую привязку на всех временных шагах t. В итоге, функция считывания представляет собой сумму линейных преобразований:

$$R(\{h_v^T | v \in V\}) = \sum_{v \in V} \tau(W_3 h_v^T) \quad (19)$$

В приведенной формуле, $\tau(\cdot)$ представляет собой выпрямляющую функцию (rectified linear unit, ReLU). Эта модель передачи сообщений

использует относительно простые преобразования и похожа на зацикленное распространение доверий [33, 13].

Такой метод передачи сообщений не является самым современным [17]. Однако он был выбран для проверки ввиду того, что данный метод достаточно прост и, потенциально, может помочь добавить иерархичные свойства к используемым моделям.

Для задания свойств ребер e_{wv} использовались конвертация типа связи ребра (одиночная, двойная, тройная или ароматическая) в вектор свойств и добавление в вектор свойств расстояния между атомами. Вектор свойств атома включает всю базовую химическую информацию, доступную в RDKit [40]:

1) тип атома;
2) гибридизация;
3) донор или акцептор;
4) количество связанных атомов водорода.

Это – обычные свойства, используемые для решения задач прогнозирования в химии [17]. После проведения экспериментов с явным и неявным представлением атомов водорода в графе, оказалось, что явное представление существенно повышает точность. Эти результаты подтверждают результаты экспериментов других работ [17].

Для добавления иерархичного подхода к моделям MPNN создадим отдельную модель MPNN для работы исключительно с высокоуровневом графом, затем объединим высокоуровневую и низкоуровневую кодировки для передачи в сеть прогнозирования. Такой метод полностью разделяет процессы иерархичной кодировки и кодировки уровня атомов до момента прогнозирования. Кроме того, в данном случае, не использовался второй тип графов компонентов, включающий атомы, а использовались только компоненты без вершин в виде атомов.

Свойства компонентных узлов определяются в зависимости от типа компонента. Набор свойств вершин определяются путем привода 7

возможных типов связей компонент-компонент к вектору свойств фиксированного размера.

Вторым видом нейронной сети для кодировки графов для изучаемой модели является SchNet, описанный в разделе 1.8. Реализация этой сети имеет открытый код, который использовался в экспериментах этого раздела [53].

Основой для не иерархической модели SchNet служит обычная модель SchNet [53]. В нашем случае использовались 6 блоков взаимодействия для агрегации информации о каждой вершине графа и усредненное объединение для агрегирования кодировок узлов в кодировку графа.

В работе, определяющей SchNet, для создания векторного элемента для каждого ребра, используются расстояния между вершинами в графе - расширение радиальной базисной функции (RBF) расстояния до ребра [53]. Вместо использования подобного подхода для графа компонентов, в экспериментах этой части работы использовались приведения ребер для каждого типа ребра между компонентами. В дальнейшем, полученные приведения использовались вместо RBF для передачи в слои cfconv, как описано в разделе 1.5. Кроме того, ребра создавались только между компонентами, которые имеют один из перечисленных выше типов ребер, то есть либо связаны напрямую, либо имеют общие атомы. Перечисленные модификации отличают использованный подход от оригинального SchNet, в котором все атомы в пределах определенного порогового расстояния друг от друга соединены ребром [53].

Для создания изначальных векторов свойств для каждого компонента, использовались приведения, полученные путем привода вектора унитарных кодов, описывающих компонент к k-размерному вектору свойств. Такие приведения активно использовались в процессе обучения. Для получения кодировок каждого атома на низком уровне для базисной и иерархичной модели использовался обычный процесс SchNet.

Были проверены несколько методов объединения кодировок графов высокого и низкого уровней:

– отдельное прогнозирование, используя каждую из кодировок, и нахождение среднего результатов прогнозирования;

– объединение кодировок компонентов отдельно от кодировок атомов, затем соединение полученных суммарных кодировок и прогнозирование с использованием другой сети;

– объединение кодировок компонентов и атомов и выполнение прогнозирования.

В результатах экспериментов приведены данные лучшей из приведенных моделей.

Кроме того, был проверен второй тип графов компонентов, включающий и компоненты, и атомы. Для этой модели кодировки атомов были вычислены с помощью низкоуровневого кодировщика SchNet. Затем эти кодировки использовались для создания высокоуровневого графа. Кодировки отдельных компонентов задавались прежним методом – использованием приведений, основанных на типе компонента. Высокоуровневый граф проходил через несколько циклов передачи сообщений, после чего были сформированы окончательные кодировки компонентов и атомов. После этого кодировки компонентов могут быть объединены с кодировками атомов или использоваться раздельно.

Для данной модели для получения кодировок атомов использовались 6 блоков взаимодействия на графе низкого уровня, затем 6 блоков взаимодействия на графе компонентов. Для получения финального результата использовалось агрегирование по среднему значению относительно компонентов и атомов.

4.4 Проведенные эксперименты

Существует несколько общедоступных вариантов набора данных QM9 [52]. В существующих работах по машинному обучению чаще всего

использовалось подмножество из 130 462 молекулы исходного набора данных из 133 855 молекул [17]. В данной работе также использовался этот набор данных ввиду того, что он был собран, используя самые современные, на данный момент, методы. Также на этом наборе данных уже были обучены различные модели, результаты применения которых могут быть сравнены с результатом полученной в данной работе модели.

В наборе данных QM9 [52] для каждой молекулы имеется двенадцать различных целей прогнозирования. Полученная модель прогнозирования была проверена на этих целях. Перед сравнением каждой цели для этой цели была обучена отдельная модель, что является общепринятым подходом решения подобных задач прогнозирования [53, 17]. Однако, в дополнение к этому, была проверена эффективность MPNN при совместной тренировке по всем 12 целям одновременно.

Цели для набора данных QM9 были изначально разработаны с использованием теории функционала плотности – обычного метода симуляции для прогнозирования химических свойств [12]. Поэтому было проведено сравнение, насколько хорошо полученная модель способна спрогнозировать свойства, полученные с помощью использования теории функционала плотности, для каждой из молекул в наборе данных QM9.

В ходе экспериментов для обучения были использованы 110 тысяч молекул и по 10 тысяч для валидации и тестирования. Для каждого из экспериментов были использованы наборы данных, одинаковые не только по количеству, но и по содержанию, для того чтобы полученные значения точностей были сравнимы.

Для оценки модели использовалось значение средней абсолютной ошибки (mean absolute error, MAE), что является общепринятым подходом оценки в существующих работах [53, 17, 15]. Эта оценочная характеристика находится как среднее значение абсолютных разностей между целевым и прогнозируемым значениями по всем молекулам в тестовом наборе данных:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (20)$$

Стоит отметить, что обучение проводилось на основе потерь среднеквадратичной ошибки (mean squared error, MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (21)$$

Отклонения от целей в наборе данных QM9 для экспериментальных данных и аппроксимаций уже известны [15]. Используем эти значения для определения того, насколько хорошо полученная модель способна смоделировать симуляцию теории функционала плотности, использованную изначально для сбора данных QM9. Кроме того, проведем сравнение с уже полученными результатами из других работ [53, 17].

При обучении совместно на всех целях использовалось значение MAE для нормализованных целей для того, чтобы значения целей были сравнимы между собой. Однако, при прогнозировании одиночных целей всегда использовалось не нормализованное значение MAE для возможности сравнения результатов с результатами, ранее полученными в других работах.

Для оценки влияния иерархичности вместе с каждой иерархичной моделью была обучена модель без иерархичности, но с той же архитектурой. Были проведены сравнения моделей: иерархичная SchNet и SchNet, иерархичная MPNN и MPNN. Цель этих сравнений состоит в том, чтобы точно оценить, насколько иерархичность компонентов повышает точность исходной модели.

Кроме того, результаты, полученные с помощью иерархичной модели SchNet, были сравнены с результатами прогнозирующих моделей из других работ [17].

Для экспериментов, использующих SchNet, был использован оптимизатор Adam со скоростью обучения 1e-3 и весом фактора затухания

0,96 для каждых 100 000 шагов. Эти оптимизационные гиперпараметры показали себя лучше всего для обоих типов моделей. Модель была реализована с помощью PyTorch [46] и основана на моделях общедоступной реализации MPNN [33].

4.5 Результаты экспериментов

Сначала была проверена эффективность иерархичной модели MPNN по сравнению с исходной моделью MPNN. Для этого эксперимента все цели были нормализованы, чтобы иметь среднее значение 0 и среднеквадратическое отклонение 1. Была обучена одна модель для прогнозирования всех 12 целей набора данных QM9 одновременно. Потерями будет являться сумма MSE нормализованных целей. Нормализация была необходима для того, чтобы модель не придавала больший вес целям с большими значениями. В остальных экспериментах нормализация не использовалась.

Таблица 4.1 – Нормализованное MAE для иерархической сети MPNN

Цель	MPNN	Иерархичная MPNN
μ	0,3324	0,3274
α	0,1677	0,0702
HOMO	0,2095	0,1768
LUMO	0,1112	0,0952
gap	0,1416	0,1202
R2	0,1947	0,1215
ZPVE	0,0963	0,0375
U0	0,1555	0,0381
U	0,1556	0,0382
H	0,1555	0,0382
G	0,1557	0,0383
Cv	0,1531	0,0569
Mean	0,1691	0,0965

Результаты эксперимента показаны в таблице 4.1. Из результатов видно, что для каждой цели иерархичная модель показала результаты лучше, чем изначальная модель MPNN. Однако, ни одна из моделей не смогла показать точности, описанной в работе «Neural message passing for quantum chemistry» [17], использующей новый, специальный подход к передаче сообщений.

Такое отклонение, вероятно, связано с использованным вариантом передачи сообщений. Использованная система передачи сообщений представляет собой простой алгоритм распространения доверия, и единственными элементами, использованными в функциях передачи, обновления и считывания являются ReLU и линейные трансформации. Полученная модель не имеет таких обширных возможностей как модель MPNN, основанная на GRU, описанная работе 17. Тот факт, что модель имеет малую емкость, подтверждается тем, что показанные выше точности тестирования были чрезвычайно близки к точности обучения, обеих моделей.

Таким образом, этот эксперимент показывает, что добавление иерархичности, может улучшить модели с низкими возможностями передачи сообщений, путем использования данных о взаимодействии более высокого порядка между крупными компонентами графа.

Таблица 4.2 – MAE для сетей SchNet

Цель	SchNet	Иерархичная SchNet
μ	0,033	0,033
α	0,079	0,080

Было проведено сравнение эффективности модели, использующей SchNet и дополненной иерархическим графом, по сравнению с простой моделью SchNet. Эксперимент показал, что после 5 000 000 шагов обучения показатели обеих моделей были примерно равны.

В таблице 4.2 показано сравнение не нормализованного значения MAE для двух из 12 целей набора данных QM9. Оба метода способны сходиться при практически одинаковых значениях MAE к концу обучения; однако, как показано в разделе 4.2, скорости сходимости двух моделей сильно различаются.

В ходе процесса обучения было замечено что обе модели показали серьезное переобучение на тренировочных данных. При прогнозировании μ изначальная и иерархичная модели SchNet показали тренировочное MAE около 0,1, что является примерно третьей частью от достигнутого тестового MAE. Можно предположить, что ввиду тенденции к переобучению исходной модели SchNet, добавление иерархичности не может улучшить конечные показатели модели.

Было проверено насколько быстро иерархические модели могут достигнуть оптимальных, по сравнению с изначальной архитектурой SchNet, значений.

Несмотря на то, что было замечено что большинство иерархических моделей имеют сравнимое и исходными моделями время схождения, одна из иерархических моделей сильно обгоняла исходную модель SchNet.

В данном случае был использован комбинированный граф компонентов, в котором граф высокого уровня состоит из компонентов и атомов. Конечные состояния атомов и компонентов собирались отдельно, а для прогнозирования использовалась их средняя.

Результаты обучения обеих моделей на μ показаны на рисунке 10.

Здесь показано значение валидационного MAE на первых 500 000 шагах обучения.

Иерархическая модель сходится намного быстрее чем обычная модель SchNet, достигая, на примере обучения для μ, значения MAE ниже 0,1 менее чем за 13 000 шагов обучения. Т

акое значение равно примерно одной четвертой процента от количества шагов, требуемого для достижения подобной сходимости для современных опубликованных методов [53, 17].

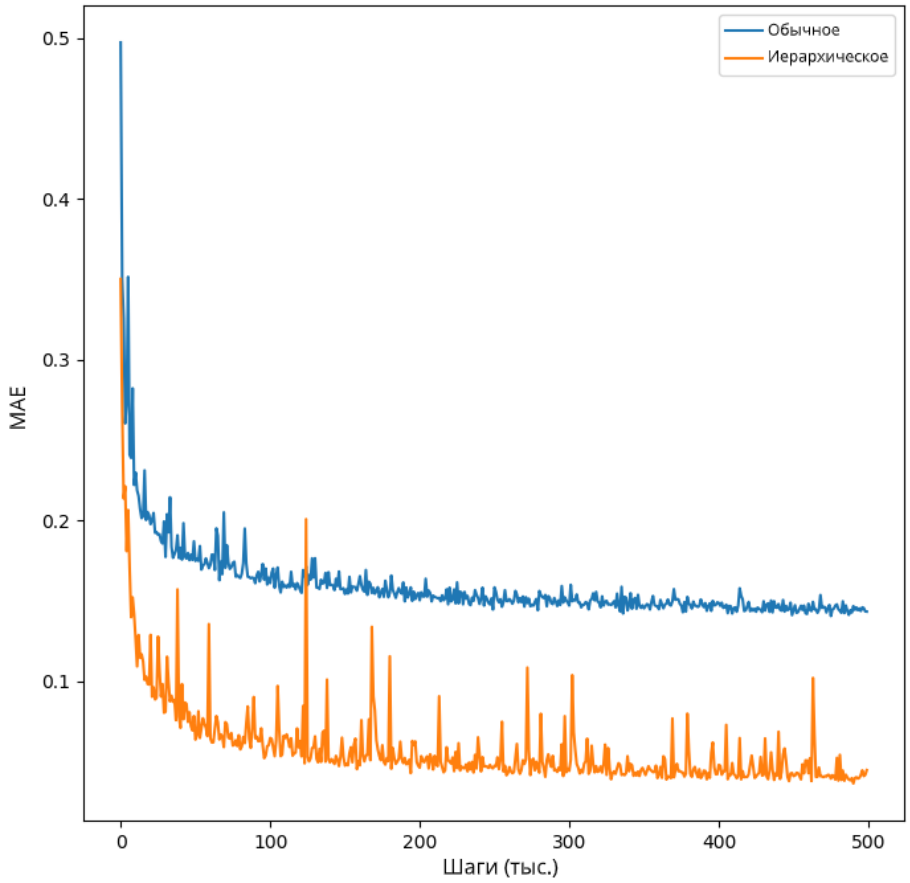

Рисунок 10 – Сходимость моделей SchNet для μ

Полученный результат намного лучше результата модели SchNet, которая не показала точности лучше 0,15 после первых 500 000 шагов. Обе модели используют идентичный оптимизатор.

Выводы по главе 4

В результате проведенных экспериментов видно, что можно серьезно увеличить точность прогнозирования при использовании подхода с иерархической кодировкой для моделей с небольшими возможностями передачи сообщений. При применении этого подхода на одной из самых

современных, на данный момент, модели – SchNet, иерархическая модель не дала значимого улучшения финальной точности.

Однако, использование высокоуровневого графа с включением атомов влияет на скорость схождения модели. Полученная модель показала способность достичь приемлемых уровней сходимости используя небольшое, в сравнении с подобными моделями, число шагов. Этот результат показывает, что иерархическая структура может быть полезна для моделей, использующих графы.

Как было замечено ранее, предполагается что иерархическая модель не может серьезно повлиять на результат SchNet ввиду того, что эта модель уже имеет добавляемые иерархичностью возможности, ввиду продвинутой систеые передачи сообщений. Кроме того, влияние оказывает то, что в нашем случае модель SchNet сильно переобучается и дополнительные возможности, добавляемые иерархичностью, не дают желаемого эффекта. Исходя из этого, добавление иерархичности больше подходит для моделей с небольшими возможностями.

Можно предположить, что иерархические модели также покажут лучшую эффективность для более сложных наборов данных, в частности для больших молекул. Молекулы в наборе данных QM9 достаточно небольшие, и, поэтому, преимущество использования информации о более крупных компонентах молекулярного графа может быть не таким полезным для задач прогнозирования свойств.

ZINC [30] – часто используемый набор данных, состоящий из органических соединений, с весом формул, доходящим до 700 и более обширным набором возможных компонентов. Использование иерархического подхода для решения задач прогнозирования на этом наборе данных может быть более эффективным чем на наборе данных QM9. Это направление может быть выбрано для дальнейших исследований.

На данном этапе представляет интерес рекурсивный иерархический подход к кодировке. В этом подходе группы соединенных между собой

компонентов могут быть заменены на компонент более высокого уровня. В этом случае будет реализована полная иерархия компонентов. Этот тип рекурсивной модели показал бы гораздо больше универсальности и устойчивости для молекул различных размеров.

Также может быть полезным использование метода «policy gradient» для нахождения компонентов в молекулах. Вместо интеллектуального анализа часто встречающихся подграфов можно использовать *сеть отсечения* для решения какое ребро нужно отсечь. Тогда после всех отсечений оставшиеся соединенными части можно использовать как компоненты. Возможно, стоит использовать «policy gradient» на функции потерь для обучения сети отсечения отсекать ребра чтобы получить наиболее подходящие для прогнозирования компоненты.

В заключении, стоит упомянуть, что существует много способов для расширения описанной модели до полной автокодирующей модели с поддержкой декодировки, нежели модели только с возможностями кодировки и прогнозирования. В литературе описаны различные методы создания деревьев автокодирования [33]. Обрезая определенные ребра между циклическими компонентами, чтобы исключить все циклы в графе компонентов, можно использовать любой из этих методов деревьев автокодирования для выполнения автоматического кодирования дерева компонентов. После этого возможно восстановление ребер в декодированном дереве для восстановления исходного графа.

Глава 5 Методика внедрения работы с непоследовательными данными в системы erp

5.1 Особенности применения машинного обучения в ERP

Одно из самых перспективных и быстро развивающихся направлений в совершенствовании информационных систем – внедрение достижений машинного обучения и искусственного интеллекта. Использование таких современных подходов позволяет ИС иметь, до этого не достижимые, возможности.

Перспективнее всего внедрение машинного обучения выглядит для корпоративных информационных систем, которые имеют сложную многомодульную структуру и работают с огромным количеством разнообразных данных. Самым многофункциональными и, поэтому, самыми комплексными являются системы класса ERP (Enterprise Resource Planning). Системы данного класса выполняют огромное количество разнообразных задач, помогают в принятии решений, от которых зависит будущее организаций и сотрудников. Ввиду этого, оптимизация и автоматизация процессов ERP с помощью машинного обучения позволит принести бизнесу существенную выгоду и, потенциально, вывести его на новый технологический уровень.

ERP системы, успешно использующие возможности машинного обучения и искусственного интеллекта иногда называют «интеллектуальными ERP» (i-ERP) [65]. Такие интеллектуальные системы используют машинное обучение для обработки огромных массивов данных, что позволяет создать инновационные продукты и услуги, повысить производительность труда и максимизировать прибыль на информационные активы.

Из-за высоких требований к инфраструктуре, необходимой для управления огромными, разнородными наборами данных, и для обеспечения принятия решений в узких окнах возможностей системы i-ERP зачастую

развертываются в облаке. Благодаря машинному обучению и прогнозной аналитике системы i-ERP могут учиться на исключениях и соответствующим образом адаптировать бизнес-правила. Опыт взаимодействия пользователей с системой (User eXperience, UX) включает услужливые стили общения (с использованием, в первую очередь, мобильных устройств); этот новый опыт появился благодаря достижениям в области обработки естественного языка (natural language processing, NLP) и машинного обучения.

Внедрение машинного обучения в системы ERP обеспечат компаниям получение следующих значительных преимуществ:

- оптимизация ресурсов,
- снижение эксплуатационных расходов,
- анализ в реальном времени и принятие решений.

Используя технологии вычислений в памяти (in-memory computing, IMC), приложения i-ERP могут анализировать и обрабатывать большие наборы данных в режиме реального времени.

Посредством взаимодействия человек-машина система ERP может определять, какие процессы следует автоматизировать, а какие – развивать, что позволит сотрудникам становиться более эффективными при выполнении своих обязанностей.

Кроме того, учитывая различные стили поведения персонала, пользовательский интерфейс проектируется так, чтобы он мог подстраиваться под эти стили и быть удобным для каждого работника.

Интерфейсы систем ERP должны быть способны в любое время и в любом месте обеспечивать передачу важной информации именно тем сотрудникам, которым она в данный момент необходима, благодаря чему руководство компании сможет принимать наиболее эффективные решения. Поддержка такого типа дает компаниям конкурентное преимущество в цифровой экономике, поскольку они способны быстро и гибко адаптироваться к возникающим препятствиям в режиме реального времени.

Кроме того, приложения являются масштабируемыми, поэтому компании всех размеров имеют возможность воспользоваться повышенной гибкостью [66].

Достижения машинного обучения могут внедряться в корпоративные ИС тремя основными путями:

– привлечением сторонних компаний, специализирующихся на внедрении таких решений;

– наем специалистов по Data Science;

– использование решений, предлагаемых вендорами ИС.

Одним из лидеров среди вендоров ИС является корпорация «SAP SE», успешно внедряющая машинное обучение и искусственный интеллект в свои продукты, в том числе и в SAP ERP.

Внедрение в ИС осуществляется с помощью решений:

– SAP Leonardo,

– SAP Intelligent Robotic Process Automation,

– SAP Conversational AI,

– SAP Data Intelligence.

Перечисленные решения успешно применяются в ERP системах, основанных на технологиях SAP [64].

Следует отметить, что модели машинного обучения в ИС чаще всего применяются к данным, хранящимся в последовательном виде – как массивы или векторы.

Работа с данными в виде множеств или графов затруднена или невозможна.

Однако, модели SAP Leonardo могут быть улучшены с помощью методик, выработанных в данном исследовании. Это позволит моделям выбирать несходные подмножества, эффективнее автокодировать данные в виде графов и работать с информацией, добавленной иерархичным подходом.

5.2 Общая методика внедрения на примере SAP ERP

Опишем методику внедрения результатов исследования на примере SAP ERP, использующей систему SAP Leonardo. SAP Leonardo – система, добавляющая возможности машинного обучения, интернета вещей, больших данных используя платформу SAP Cloud.

SAP Leonardo реализован как модифицируемые модели, работа с которыми возможна через обращение к облачному API (Application Programming Interface) или через использование отдельных микросервисов.

Для интеграции методик в SAP ERP необходимо выполнить два шага:

1) Модифицировать поток SAP HANA Smart Data Streaming (SDS) – систему передачи и обработки данных в режиме реального времени,

2) Расширить модифицируемые модели SAP Leonardo необходимым функционалом.

Остановимся подробнее на добавление потока SDS путем надстройки над шагом преобразования данных.

На рисунке 11 показан пример высокоуровневой архитектуры платформы SAP Hana для задачи по распознаванию изображений.

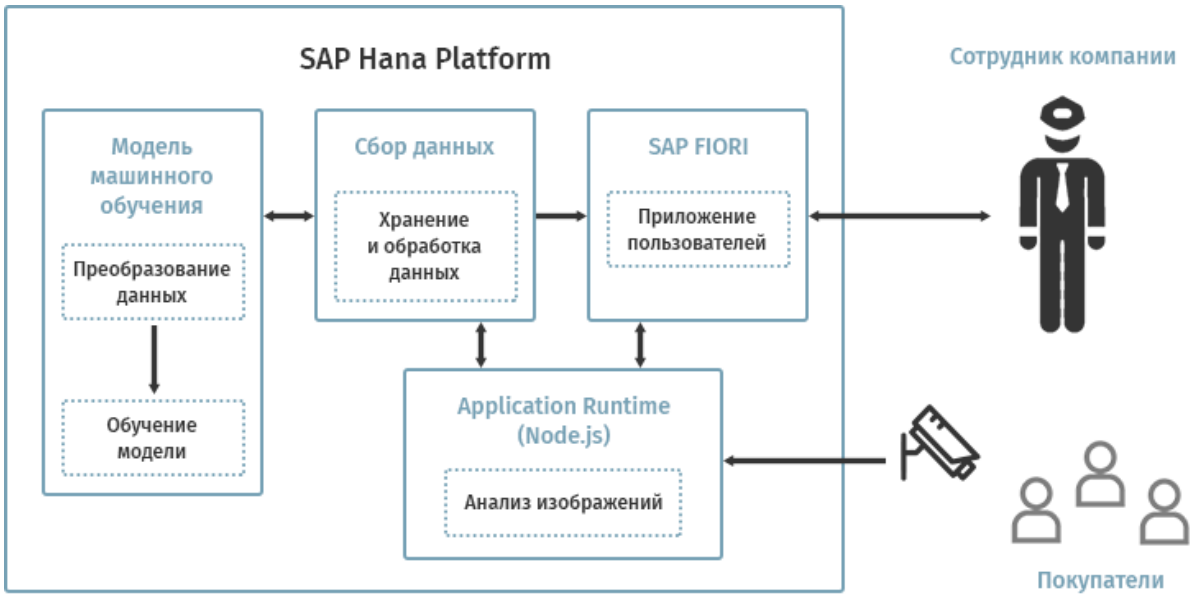

Рисунок 11 – Пример архитектуры платформы SAP Hana

Данные, в непоследовательном виде, полученные через API Big Data SAP Leonardo, необходимо автокодировать наиболее оптимальным вариантом автокодировки.

Пример ответа API SAP Leonardo для распознавания изображений приведен в листинге 3.

Листинг 3. Пример ответа API SAP Leonardo

```
{
  "id": "f09a0cfd-e1bc-4740-5e56-919a94a56d5b",
  "predictions": [
    {
      "name": "singleface.jpeg",
      "numberOfFaces": 1,
      "faces": {
        "faceLocation": [
          {
            "top": 242,
            "bottom": 464,
            "left": 563,
            "right": 756
          }
        ],
        "faceFeature": [
          -0.09335256367921829,
          0.04137273132801056,
          0.08437961339950562,
          -0.06715819239616394,
          -0.13240352272987366
        ]
      }
    }
  ],
  "processed_time": "2018-01-16T10:21:33.575961",
  "status": "DONE"
}
```

Для автокодировки следует применить генеративно-состязательную сеть с одной из архитектур, описанной в разделе 3.1 в зависимости от того плотные данные или разреженные.

Далее, необходимо модифицировать и переобучить модель методикой для множеств или графов.

Общая схема методики автоматизации анализа непоследовательных данных путем внедрения машинного обучения на непоследовательных данных показана на рисунке 12.

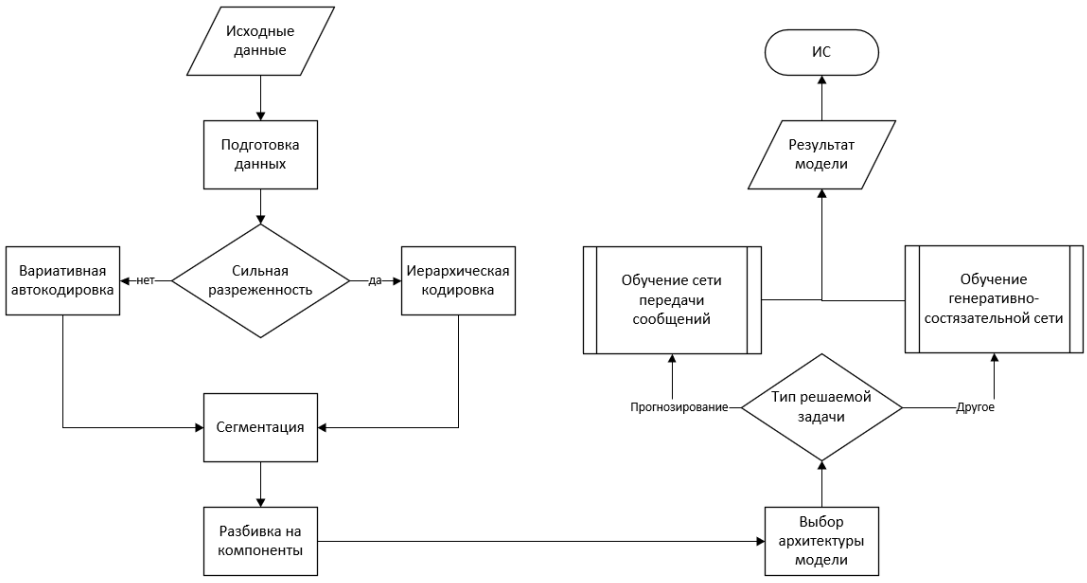

Рисунок 12 – Общая схема внедрения методики в ИС

Следует обратить внимание на разреженность исходных данных. Использование иерархического кодировщика серьезно повысит эффективность обучения для сильно разреженных данных. Генеративно-состязательная сеть может быть заменена простой сетью MPNN для всех задач прогнозирования.

5.3 Работа с множествами для моделей SAP Leonardo

Для добавления или улучшения работы с множествами SAP ERP, использующую набор моделей из SAP Leonardo необходимо переобучить одну или несколько моделей. Для этого потребуются следующие библиотеки:

- Scikit-learn,
- SAP Cloud Foundry Account,
- SAP CP Cloud Foundry CLI,
- плагин SAP Leonardo Ml Foundation CLI.

Для начала, необходимо клонировать сервис ML-Foundation из SAP Cloud Foundry. К модели, в данном случае, система будет обращаться с помощью Cloud Foundry и Ml Foundation. Для добавления новой модели следует создать новую сущность в Cloud Foundry. Архитектура модели и кодировщика должна быть выбрана по схеме, описанной в разделе 2.6.

Модель может быть зарегистрирована с помощью команды «sapml» плагина SAP Leonardo Ml Foundation CLI. Для обучения или переобучения модели необходимо зарегистрировать «работу» обучения. Для этого нужно создать файл «model_training.py», пример которого показан в листинге 4.

Листинг 4. Пример конфигурации работы для модели Ml Foundation

```
job:
  name: "model-training"
  execution:
    image: "ml-foundation/sklearn:0.19.1-py3"
    command: "pip3 install -r requirements.txt && python3 model_training.py"
    completionTime: "10000"
    resourcePlanId: "basic"
```

Параметры модели могут быть изменены с помощью редактирования конфигурационных файлов файловой системы, автоматически созданной Ml Foundation. При сильном переобучении модели следует изменить количество слоев кодировщика или отказаться от активаций ReLU.

Созданная и обученная модель может быть скачана с помощью интерфейса «Minio» SAP Leonardo ML Foundation. Следует отметить, что часть описанных выше действий может быть, при желании, автоматизирована через дальнейшую модификацию работы ML Foundation.

5.4 Добавление иерархичности для моделей SAP Leonardo

Для добавления сохранения информации об иерархичности к моделям SAP Leonardo, работающим с данными в виде графов недостаточно простого

создания модели и регистрации работы. В данном случае потребуется создания отдельного сервиса и добавление API с помощью:

- SAP CP Cloud Foundry CLI;
- плагина SAP Leonardo Ml Foundation CLI plugin;
- Scikit-learn;
- Flask.

Для начала следует зарегистрировать сервис Cloud Foundry с помощью команды «cf» и манифест-файла нового сервиса. Дальнейшая работа с сервисом будет проходить в облаке. Работа с данными в виде графов SAP Leonardo может быть реализована с помощью включение «скалирования» данных через запись «GraphScaler» в файле определения модели.

Реализация алгоритма сегментации, описанного в разделе 4.2, может быть выполнена через скрипт-утилиту модели и файл получения настроек, пример которого приведен в листинге 4.

Листинг 4. Пример получения настроек

```
def get_mlf_conf_variables():
    vcap_services = json.loads(str(os.getenv("VCAP_SERVICES", {})))
    if MLF_NAME in vcap_services.keys():
        client_id                                                    =
vcap_services.get(MLF_NAME)[0].get(CREDENTIALS).get(CLIENT_ID)
        client_secret                                                =
vcap_services.get(MLF_NAME)[0].get(CREDENTIALS).get(CLIENT_SECRET)
        authentication_url                                           =
vcap_services.get(MLF_NAME)[0].get(CREDENTIALS).get(AUTHENTICATION_URL
)
        job_url                                                      =
vcap_services.get(MLF_NAME)[0].get(CREDENTIALS).get(SERVICE_URLS).get(
JOB_SUBMISSION_URL)
        return client_id, client_secret, authentication_url, job_url,
API_VERSION
    else:
        return None, None, None, None, None
```

Для реализации алгоритма gSpan и кодировки полученных деревьев, описанной в разделе 4.3, следует использовать тот же подход что и для алгоритма сегментации.

При использовании нестандартного сервиса SAP Leonardo для прогнозирования нужно зарегистрировать API «/predict». При реализации модели следует реализовать все шаги добавления работы с иерархичностью:

1) использование gSpan для получения и сохранения подграфов;

2) сегментацию графов молекул на циклы и полученные ранее компонентов;

3) сборку нового графа, состоящего из компонентов, и присвоение вектора свойств для каждой вершины и ребра этого графа;

4) отправка полученного графа на кодировку выбранным способом для получения высокоуровневой кодировки.

Кодировщик может быть реализован в виде простой модели, зарегистрированной в ML Foundation.

Выводы по главе 5

Реализация методик по модернизации работы со множествами и добавления информации об иерархичности моделям, работающим с данными в виде графов, зависит от архитектуры информационной системы, для которой производится модернизация. В случае случает использования ERP систем, уже имеющих основы работы с моделями, таких как SAP ERP (SAP S/4HANA) или «1С:ERP Управление предприятием» внедрение состоит в модернизации моделей и потоков данных для работы с непоследовательными данными. Пример методики внедрения для такой системы показан в данной главе.

В случае нестандартной ERP системы необходимо добавление получения и очистки данных, отдельное от ИС обучение моделей и встраивание работы с этими моделями в поток данных ИС.

ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена методика машинного обучения на двух нестандартных типах данных. Во второй главе описана новая методика выделения образцовых несходных подмножеств с использованием системы инвариативных перестановочных множеств, и сравнение ее с более простыми подходами без обязательной перестановочной вариативности. Результаты показывают, что предложенный метод смог воспроизвести существующий алгоритм с большей целостностью. Это является важным шагом на пути к аппроксимированию детерминантных точечных процессов и других несходных множеств.

Во третьей главе был представлен новый подход для декодирования сильно структурированных графов и было показано внедрение этого подхода в вариативную автокодирующую модель на примере наборов данных из химической отрасли. Результаты показали, что подход имеет существенные недостатки и требуется внесение изменений для создания эффективного и точного декодировщика графов.

В четвертой главе, была предложена новая система кодировки для графов, использующая иерархию компонентов для сбора информации на нескольких уровнях. Была проведена апробация этого метода на небольших молекулах, и установлено что предложенный подход может ускорить процесс обучения для современных моделей и добавить больше возможностей для более простых моделей.

В пятой главе описаны особенности внедрения достижений машинного обучения в информационные системы, в частности, в системы ERP. Методка внедрения работы с непоследовательными данными показана на примере ERP системы «SAP S/4HANA».

В направлении работы с непоследовательными данными все еще существует большое количество не решенных задач. Задача выбора подмножеств пока не решена, и требуется больше усилий чтобы лучше

понять как различные подходы, основанные, например, на сетях указателей и глубоких множествах, соотносятся между собой.

Задача автокодировки для разреженных структурированных графов пока до конца не решена. Были разработаны некоторые методы, работающие только с деревьями [33], но не существует методов, способных сгенерировать новые образцовые графы с приемлемой, для сильно структурированных данных, точностью. Кроме того, большинство методов кодировки графов не могут передать зависимости между большими компонентами графов, и фокусируются на передаче взаимосвязей низкого уровня. Предложенная в данной работе методика позволяет выявить и использовать взаимосвязи на высоком уровне. Однако, для того чтобы оценить весь спектр применений этого подхода требуется его дальнейшее изучение.

Список используемой литературы и используемых источников

1. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng: TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. URL: https://arxiv.org/abs/1603.04467

2. Graphvae: Towards generation of small graphs using variational autoencoders. 2018. URL: https://arxiv.org/abs/1802.03480

3. Циликов Н. С., Федосин С. А. Графовые нейронные сети. 2012. URL: https://cyberleninka.ru/article/n/grafovye-neyronnye-seti

4. Peter Atkins, Loretta Jones. Chemical principles: The quest for insight. Macmillan, 2007.

5. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

6. Yoshua Bengio, Aaron Courville, Pascal Vincent. Representation learning: A review and new perspectives. 2013. URL: https://arxiv.org/abs/1206.5538

7. Тиханычев О. В. Об информационном обеспечении поддержки принятия решений. 2018. URL: https://cyberleninka.ru/article/n/ob-informatsionnom-obespechenii-podderzhki-prinyatiya-resheniy

8. N. Buchbinder, M. Feldman, J. Naor, R. Schwartz. A tight linear time (1/2)- approximation for unconstrained submodular maximization. С. 649–658. 2012. URL: https://theory.epfl.ch/moranfe/Publications/FOCS2012.pdf

9. Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, Yoshua Bengio. Attention-based models for speech recognition. С. 577–585. 2015. URL: https://arxiv.org/abs/1506.07503

10. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

11. Петров А. Б. Корпоративные информационные системы: проблемы и перспективы. 2019. URL: https://cyberleninka.ru/article/n/korporativnye-informatsionnye-sistemy-problemy-i-perspektivy

12. Aron J Cohen, Paula Mori-Sánchez, Weitao Yang. Insights into current limitations of density functional theory. Science. С. 792–794, 2008.

13. Неделько В.М. Основы статистических методов машинного обучения. Новосибирский Государственный Технический Университет. 2010.

14. Joseph L Durant, Burton A Leland, Douglas R Henry, James G Nourse. Reoptimization of mdl keys for use in drug discovery. Journal of chemical information and computer sciences. 2002.

15. FA Faber, L Hutchison, B Huang, J Gilmer, SS Schoenholz, GE Dahl, O Vinyals, S Kearnes, PF Riley, O Anatole von Lilienfeld. Machine learning prediction errors better than dft accuracy. ArXiv170205532 Phys, 2017.

16. Макаев Флюр, Жунгиету Григоре. Химия лекарственных средств. Palmarium Academic Publishing. 2015.

17. Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, George E Dahl. Neural message passing for quantum chemistry. arXiv preprint arXiv:1704.01212, 2017.

18. D. Glasscock. What is a graphon? ArXiv e-prints, 2016.

19. Boqing Gong, Wei-Lun Chao, Kristen Grauman, Fei Sha. Diverse sequentialsubset selection for supervised video summarization. 2014. URL: https://papers.nips.cc/paper/5413-diverse-sequential-subset-selection-for-supervised-video-summarization.pdf

20. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David WardeFarley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative adversarial nets. 2014. URL: https://arxiv.org/abs/1406.2661

21. Rafael GÃşmez-Bombarelli, Jennifer N. Wei, David Duvenaud, JosÃľ Miguel HernÃąndez-Lobato, BenjamÃŋn SÃąnchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, AlÃąn AspuruGuzik. Automatic chemical design using a data-driven continuous representation of molecules. ACS Central Science. С. 268–276. 2018.

22. William L. Hamilton, Rex Ying, Jure Leskovec. Representation learning on graphs: Methods and applications. CoRR. 2017.

23. Christian Häne, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, Paul Furgale, Torsten Sattler, Marc Pollefeys. 3d visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection. Image and Vision Computing. С. 14–27, 2017.

24. Наталия Нам, Игорь Грандберг. Органическая химия. ISBN: 978-5-8114-3901-0. С. 243 – 246. Лань. 2019.

25. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition. CoRR. 2015.

26. Philipp Hennig, Roman Garnett. Exact sampling from determinantal pointprocesses. CoRR. 2016.

27. Gao Huang, Zhuang Liu, Kilian Q. Weinberger. Densely connected convolutional networks. CoRR. 2016.

28. Демидова Л. А., Кираковский В. В., Пылькин А. Н. Принятие решений в условиях неопределенности. Горячая линия-Телеком. 2012

29. Wolf D Ihlenfeldt, Evan E Bolton, Stephen H Bryant. The pubchem chemical structure sketcher. Journal of cheminformatics. С. 20. 2009.

30. John J Irwin, Brian K Shoichet. Zinc- a free database of commercially available compounds for virtual screening. Journal of chemical information and modeling. С. 177–182. 2005.

31. Окасаки К. Чисто функциональные структуры данных. ДМК Пресс. 2016.

32. Chuntao Jiang, Frans Coenen, Michele Zito. A survey of frequent subgraph mining algorithms. The Knowledge Engineering Review. С. 75–105. 2013.

33. Wengong Jin, Regina Barzilay, Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. arXiv preprint arXiv:1802.04364, 2018.

34. Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever. An empirical exploration of recurrent network architectures. 2015. URL: http://proceedings.mlr.press/v37/jozefowicz15.pdf

35. Робинсон Ян, Вебер Джим, Эифрем Эмиль. Графовые базы данных: новые возможности для работы со связанными данными. ДМК Пресс. 2016.

36. Byungkon Kang. Fast determinantal point process sampling with application to clustering. 2013. URL: https://papers.nips.cc/paper/5008-fast-determinantal-point-process-sampling-with-application-to-clustering

37. D. P Kingma, M. Welling. Auto-Encoding Variational Bayes. ArXiv e-prints. 2013.

38. Thomas N Kipf, Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907. 2016.

39. Alex Kulesza, Ben Taskar. Determinantal point processes for machine learning. С. 123–286. 2012.

40. Greg Landrum. Rdkit: Open-source cheminformatics. 2006. URL: https://github.com/greglandrum/RDKit_Paper/blob/master/Text.md

41. Y. Li, L. Zhang, Z. Liu. Multi-Objective De Novo Drug Design with Conditional Graph Generative Model. 2018. URL: https://arxiv.org/abs/1801.07299

42. Niina Mallat, Virpi Tuunainen, Kristina Wittkowski. Voice activated personal assistants–consumer use contexts and usage behavior. 2017.

43. LR Medsker, LC Jain. Recurrent neural networks. Design and Applications, 2001.

44. P. Orbanz, D. M. Roy. Bayesian models of graphs, arrays and other exchangeable random structures. C. 437–461. 2015.

45. John R Owen, Ian T Nabney, José L Medina-Franco, Fabian López-Vallejo. Visualization of molecular fingerprints. Journal of chemical information and modeling. 2011. URL: https://pubs.acs.org/doi/abs/10.1021/ci1004042

46. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, Adam Lerer. Automatic differentiation in pytorch. 2017. URL: https://openreview.net/pdf?id=BJJsrmfCZ

47. Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. 2016. URL: https://arxiv.org/abs/1609.08976

48. Charles R Qi, Hao Su, Kaichun Mo, Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. 2017. URL: https://arxiv.org/abs/1612.00593

49. Alec Radford, Luke Metz, Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434. 2015.

50. Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. Scientific Data. 2014.

51. R. Ramprasad, R. Batra, G. Pilania, A. Mannodi-Kanakkithodi, C. Kim. Machine learning in materials informatics: recent applications and prospects. Computational Mathematics. 2017.

52. Lars Ruddigkeit, Ruud van Deursen, Lorenz C. Blum, Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. Journal of Chemical Information and Modeling. C. 2864–2875, 2012.

53. Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. 2017. URL: https://arxiv.org/abs/1706.08566

54. Ilya Sutskever, James Martens, Geoffrey E Hinton. Generating text with recurrent neural networks. 2011. URL:
https://www.cs.utoronto.ca/~ilya/pubs/2011/LANG-RNN.pdf

55. Ilya Sutskever, Oriol Vinyals, Quoc V Le. Sequence to sequence learning with neural networks. 2014. URL: https://arxiv.org/abs/1409.3215

56. Ana Swanson. Big pharmaceutical companies are spending far more on marketing than research. 2015.

57. V. Veitch, D. M. Roy. The Class of Random Graphs Arising from Exchangeable Random Measures. ArXiv e-prints. 2015.

58. Oriol Vinyals, Samy Bengio, Manjunath Kudlur. Order matters: Sequence to sequence for sets. arXiv preprint arXiv:1511.06391. 2015.

59. Oriol Vinyals, Meire Fortunato, Navdeep Jaitly. Pointer networks. 2015. https://arxiv.org/abs/1506.03134

60. David Weininger. Smiles, a chemical language and information system. Journal of chemical information and computer sciences. С 31–36. 1988.

61. Xifeng Yan, Jiawei Han. gspan: graph-based substructure pattern mining. 2002. URL: https://dm.kaist.ac.kr/kse625/resources/Yan_2002.pdf

62. Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, Alexander J Smola. Deep sets. Curran Associates. 2017.

63. Keren Zhou. Jokeren/datamining-gspan. 2017. URL: https://ieeexplore.ieee.org/document/7395783

64. Stoil Jotev. Configuring SAP S/4HANA Finance. SAP PRESS. 2019

65. Marianne Bradford. Modern ERP: Select, Implement, and Use Today's Advanced Business Systems. 2016

66. Ильин В. Внедрение ERP-систем: управление экономической эффективностью. Интермедиатор. 2015.