Оглавление

ВВЕДЕНИЕ

В последние годы наблюдается значительный рост интереса к графовым структурам и их применениям в различных областях науки и техники. Графы успешно используются для моделирования взаимосвязей между объектами, что делает их особенно полезными в таких областях, как биоинформатика, социальные сети, компьютерная визуализация и многие другие сферы, где данные могут быть представлены в виде сложной сети взаимосвязей. В частности, графовые нейронные сети (Graph Neural Networks, GNN) [1] стали важным инструментом в машинном обучении, обеспечивая новую парадигму для обработки и анализа графовых структур.

Поиск архитектуры нейронных сетей привлёк значительное внимание в научном сообществе вычислительного интеллекта и способствовал продвижению современных моделей нейронных сетей для решения задач, связанных с сетевидными данными, такими как тексты и изображения. Однако в области графовых нейронных сетей посвящено гораздо меньше исследований, что ограничивает их применение для неструктурированных сетевых данных. Учитывая огромное количество вариантов и комбинаций компонентов, таких как агрегаторы и функции активации, определение подходящей модели GNN для конкретной задачи обычно требует значительных знаний и трудоемких экспериментов. Кроме того, незначительные изменения гиперпараметров, таких как скорость обучения и уровень dropout, могут существенно повлиять на учебные способности модели GNN.

Несмотря на всеобщее принятие графовых нейронных сетей, задача поиска оптимальной их архитектуры остаётся актуальной и сложной. Неправильно настроенные гиперпараметры или неподходящая архитектура могут снизить производительность модели и привести к плохим результатам. В этом контексте поиск оптимальной архитектуры GNN с использованием методов автоматизированного подбора гиперпараметров, таких как генетические алгоритмы, становится важной темой для исследования.

Генетический алгоритм (ГА) [2,3], вдохновлённый процессами естественного отбора и эволюции, предлагает мощный подход к задаче оптимизации архитектур GNN. Он основан на принципах эволюционной биологии, что позволяет исследовать большое пространство гиперпараметров, комбинируя существующие архитектуры и вводя случайные мутации, что помогает находить более эффективные и результативные комбинации для решения конкретных задач.

Цель настоящей курсовой работы заключается в исследовании и применении генетического алгоритма к поиску оптимальной архитектуры графовых нейронных сетей на сложных наборах данных. В работе рассмотрен процесс обучения GNN, использование различных методов для построения индивидуумов в генетическом алгоритме, а также анализ полученных результатов на нескольких графовых датасетах. Основное внимание уделено тому, как GNN могут быть использованы для различных задач, и как подходы на основе ГА могут улучшить эти результаты.

Результатом работы будет: выявление наиболее эффективных комбинаций гиперпараметров для различных архитектур GNN; выявление характеристик и особенностей отдельных архитектур GNN на основе полученных результатов; сохранение и представление результатов работы в виде графиков, метрик и таблиц.

1. Обзор предметной области.

Графовые нейронные сети представляют собой класс нейронных сетей, специализированных для обработки данных, представленных в виде графов. В отличие от традиционных моделей глубокого обучения, таких как свёрточные нейронные сети (CNN), которые работают с данными фиксированной структуры (например, изображения), GNN способны эффективно обрабатывать графовые структуры с переменной топологией. Это создаёт широкие возможности для применения GNN в таких областях, как социальные сети, биоинформатика, анализ графов и обработка естественного языка.

1.1 Определение и основные концепции.

Графовые нейронные сети являются обобщением свёрточных нейронных сетей, применяемых для обработки данных, организованных в фиксированные решётки, таких как изображения. В отличие от них, GNN способны обрабатывать графы, которые могут иметь произвольную структуру и размер. Граф состоит из множества узлов (вершин) и рёбер (связей), которые могут представлять отношения между узлами. Ключевые концепции GNN включают:

- Агрегация: Узлы в графовой сети обновляют свои представления на основе информации, получаемой от соседних узлов. Этот процесс обрабатывается через функции агрегации, что позволяет моделировать взаимосвязи и зависимости между узлами.
- Передача сообщений и обновление: Основным механизмом работы GNN является передача сообщений между узлами, где каждый узел получает информацию от соседних узлов. На каждом шаге узлы комбинируют свои представления и представления соседей, что позволяет им адаптироваться и изменять свои знания о графе. Например, в одном шаге узел может воспользоваться информацией о своих непосредственных соседях, в то время как в следующих шагах узлы могут получать информацию от более удалённых узлов графа.

Эти концепции позволяют GNN эффективно работать с графовыми данными и успешно применять их для решения различных задач, таких как классификация узлов, классификация графов и предсказание связей.

1.2 Архитектуры GNN (GCN, SAGEConv, GATConv, GraphConv).

Существует несколько архитектур графовых нейронных сетей, каждая из которых имеет свои особенности и применяется для различных задач. Ниже перечислены наиболее популярные архитектуры GNN:

1.2.1 GCN (Graph Convolutional Network)

GCN [4] является одной из первых архитектур GNN, которая применяет концепцию свёртки на графах. Она обрабатывает информацию о соседях, обновляя представление каждого узла на основе их вкладов. GCN использует фиксированное количество соседей, что помогает контролировать объём вычислений, и подходит для задач как классификации узлов, так и классификации графов. Обновление узла h_v в GCN может быть записано как:

$$H^{l+1} = \sigma(\hat{A}H^{(l)}W^{(l)})$$

где $H^{(l)}$ – преставление узлов на l-ом слое,

$\hat{A}$ – нормализованная матрица смежности,

$W^{(l)}$ – обучаемые веса слоя,

σ – функция активации.

1.2.2 GAT (Graph Attention Network)

GAT [5] использует механизм внимания, который позволяет модели определять, насколько важны её соседи при агрегации информации. Это делает GAT более гибким, особенно в графах, где связи между узлами могут быть неоднородными. Обновление представления узла происходит следующим образом:

$$h'_v = \sigma(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} W h_u)$$

где α_{vu} – коэффициент внимания, который определяется на этапе обучения.

1.2.3 GraphSAGE (Graph Sample and Aggregation)

GraphSAGE [6] разработан для работы с большими графами и позволяет модели обучаться на подмножествах соседей. На каждом шаге он выбирает случайное подмножество соседей и применяет функции агрегации, такие как среднее или максимум. Эта архитектура подходит для задач индуктивного обучения, где модель может обрабатывать графы, неизвестные во время обучения. Обновление узла записывается как:

$$h_v^{(l+1)} = \sigma(W^{(l)} h_v^{(l)} + AGGREGATE^{(l)}(\{h_u^{(l)} : u \in \mathcal{N}(v)\}))$$

1.2.4 GraphConv (Graph Convolution)

GraphConv является улучшенной версией GCNConv, предложенной в работе Morris et al. в 2019 году. Этот метод также использует спектральное представление графа, но вводит дополнительные параметры для улучшения выразительности модели.

GraphConv использует более гибкую форму агрегации информации от соседних узлов. Основная формула для обновления представления узла i выглядит следующим образом:

$$h_i^{(l+1)} = \sigma(\sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{d_i d_j}} h_j^l W^{(l)} + b^{(l)})$$

где $b^{(l)}$- вектор смещения слоя l.

1.3 Методы обучения GNN.

Обучение графовых нейронных сетей включает несколько ключевых компонентов, таких как:

- Функция потерь: Обычно используется кросс-энтропийная функция потерь для задач классификации, а для регрессионных задач могут использоваться другие функции потерь, такие как среднеквадратичная ошибка.
- Передача сообщений: Обучение GNN базируется на механизме, называемом "обмен сообщениями", где узлы обмениваются и агрегируют информацию с соседями. Эта процедура повторяется

несколько раз, создавая представления узлов, учитывающие информацию от более удалённых узлов.

- Оптимизация: Обычно используются методы оптимизации, такие как Adam или SGD (Stochastic Gradient Descent), для минимизации функции потерь. Эта оптимизация может быть затруднённой из-за проблемы затухающего градиента, особенно в глубоких архитектурах.
- Регуляризация: Использование методов регуляризации, таких как Dropout и Weight Decay, помогает избежать переобучения и увеличивает обобщающую способность модели.
- Фазовое обучение: GNN можно обучать поэтапно, сначала заточив на местных структурах графов, а затем обучая на более глобальных задачах. Это может включать в себя различные стратегии для обработки графовых структур, как для трансдуктивных, так и индуктивных задач.

1.4 Генетический алгоритм.

Генетический алгоритм (ГА) представляет собой класс адаптивных стохастических оптимизационных методов, основанных на концепциях естественного отбора и генетической эволюции. Он используется для решения сложных оптимизационных задач, где традиционные методы могут быть неэффективными. ГА имитирует эволюционные процессы, такие как отбор, кроссинговер и мутация, что позволяет находить оптимальные или близкие к оптимальным решения в большом пространстве возможных вариантов.

1.4.1 Принципы работы ГА.

Принципы работы генетического алгоритма основаны на эволюционных концепциях. Основные этапы работы ГА включают:

- Инициализация популяции: Генетический алгоритм начинается с создания начальной популяции возможных решений, называемых индивидами. Эти индивиды представляют собой возможные решения оптимизационной задачи и могут быть

закодированы в виде строк, массивов или объектов, в зависимости от проблемы.

- Оценка фитнеса: Каждый индивид оценивается с помощью функции фитнеса, которая определяет, насколько хорошо он решает задачу. Функция фитнеса может быть любой метрикой, соответствующей контексту задачи, например, точностью модели в машинном обучении.
- Отбор: На этом этапе выбираются наиболее подходящие индивиды для дальнейшего размножения. Существуют различные методы отбора, такие как отбор по турнирам, рулеточный отбор и др. Более высокие значения фитнеса приводят к большей вероятности выбора данного индивида для создания потомства.
- Кроссинговер (скрещивание): Пара индивидов (родители) комбинируется для создания нового индивида (потомка). Это может быть реализовано различными способами, например, путем обмена сегментами кодов между двумя родителями. Кроссинговер позволяет комбинировать успешные характеристики двух или более решений.
- Мутация: Для введения генетического разнообразия в популяцию применяется мутация, которая вносит случайные изменения в отдельных индивидов. Это помогает избежать преждевременной сходимости популяции к локальным оптимумам.
- Завершение: Процесс продолжается, пока не будет достигнуто заданное количество итераций или пока не будет найдено удовлетворительное решение.

Такой подход обеспечивает эффективный поиск решений, которые могут быть недоступны более традиционными методами оптимизации.

1.4.2 Применение ГА для оптимизации гиперпараметров.

В области машинного обучения существует множество гиперпараметров, которые могут сильно повлиять на производительность модели. Генетический алгоритм предоставляют эффективный метод для поиска оптимальных значений этих гиперпараметров, автоматизируя процесс настройки.

С точки зрения графовых нейронных сетей (GNN), гиперпараметрами могут быть:

- Количество скрытых слоёв и их размерность: Эти параметры определяют архитектуру модели и могут значительно повлиять на её способность извлекать представления из графовых данных.
- Тип свёрточного слоя: Выбор между типами сверточных слоев приводит к различной производительности на разных задачах. Генетические алгоритмы могут помочь в оптимизации выбора слоёв в зависимости от данных.
- Функции активации: Функции активации, такие как ReLU, Tanh, Sigmoid или SiLU могут повлиять на способность передачи градиентов и общее обучение модели.
- Скорость обучения и параметры регуляризации: Скорость обучения определяет, как быстро модель будет адаптироваться к новым данным. Регуляризационные параметры, такие как уровень dropout, помогают уменьшить переобучение.

Процесс оптимизации гиперпараметров с помощью ГА включает следующие этапы:

1) На каждом этапе создаются индивидуумы, которые представляют различные комбинации гиперпараметров.
2) Каждый индивидуум (комбинация гиперпараметров) обучается на валидационном наборе данных, и его производительность оценивается (например, точность классификации). Эта оценка служит функцией фитнеса.

3) На основе фитнеса осуществляется выбор лучших индивидов для создания потомства. Операции кроссинговера и мутации помогают создавать новые индивидуумы с комбинацией успешных гиперпараметров и некоторыми изменениями для поиска лучших решений.
4) Итерация процесса: Процесс повторяется как минимум до достижения заданного количества поколений или пока не будет достигнута оценка, которая считается удовлетворительной.

1.5 Обзор графовых данных.

1.5.1 Наборы данных для тестирования

1.5.1.1 ENZYMES

TUDataset (ENZYMES) [7] — это набор данных, содержащий графы, представляющие структуры ферментов. Каждый граф соответствует одному ферменту, а узлы и ребра графа представляют атомы и химические связи между ними.

Этот набор данных используется для задачи классификации графов. Цель состоит в том, чтобы предсказать класс фермента на основе его структуры.

Характеристики:

- Количество графов: 600
- Количество классов: 6
- Среднее количество узлов в графе: 32.63
- Среднее количество ребер в графе: 62.14

Причина выбора:

- Сложность задачи: Классификация ферментов на основе их структуры является сложной задачей, требующей учета как локальных, так и глобальных структурных особенностей графа.
- Релевантность: Ферменты играют важную роль в биохимии и биоинформатике, и их классификация имеет практическое значение.

Визуализация одного из графов из датасета ENZYMES представлена на рисунке 1.

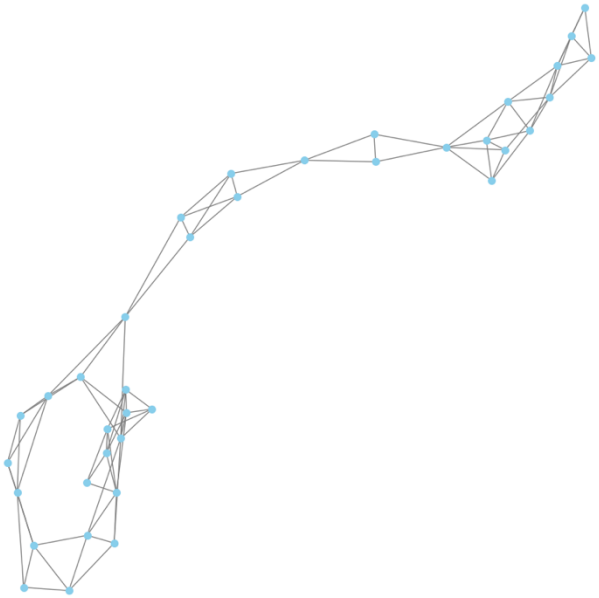

Рисунок 1 – Визуализация одного графа из датасета ENZYMES.

1.5.1.2 PROTEINS

TUDataset (PROTEINS) [8] — это набор данных, содержащий графы, представляющие структуры белков. Каждый граф соответствует одному белку, а узлы и ребра графа представляют аминокислоты и их взаимодействия.

Этот набор данных используется для задачи классификации графов. Цель состоит в том, чтобы предсказать класс белка на основе его структуры. Белки классифицируются по двум классам: ферменты и неферменты. Каждый граф имеет метку, указывающую на один из двух классов белков.

Характеристики:

- Количество графов: 1113
- Количество классов: 2
- Среднее количество узлов в графе: 39.06
- Среднее количество ребер в графе: 72.82

Причина выбора:

- Сложность задачи: Классификация белков на основе их структуры является сложной задачей, требующей учета как локальных, так и глобальных структурных особенностей графа.

- Релевантность: Белки играют важную роль в биологии и медицине, и их классификация имеет практическое значение.

Визуализация одного из графов из датасета PROTEINS представлена на рисунке 2.

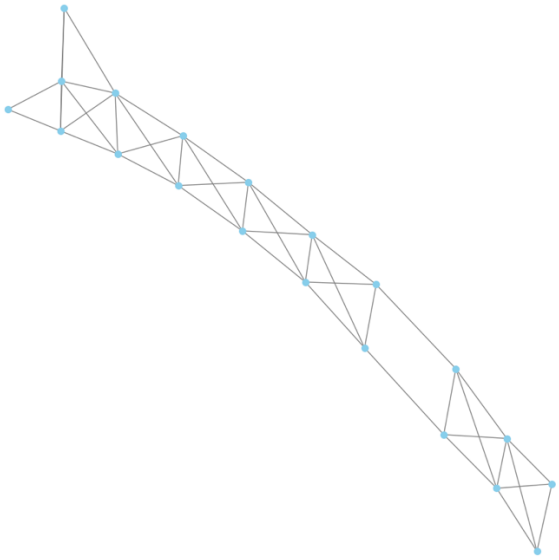

Рисунок 2 – Визуализация одного графа из датасета PROTEINS.

1.5.1.3 Cora

Planetoid (Cora) [9] — это набор данных, содержащий цитатный граф научных публикаций. Узлы графа представляют научные статьи, а ребра — цитаты между ними. Каждый узел имеет атрибуты, представляющие собой слова из текста статьи, и метку, указывающую на категорию статьи.

Этот набор данных используется для задачи классификации узлов. Цель состоит в том, чтобы предсказать категорию статьи на основе её атрибутов и структуры графа. Статьи классифицируются по семи различным категориям, таким как "Neural Networks", "Rule Learning", "Reinforcement Learning", "Probabilistic Methods", "Theory", "Genetic Algorithms", и "Case Based".

Характеристики:

- Количество узлов: 2708
- Количество ребер: 5429
- Количество классов: 7
- Количество атрибутов узла: 1433

Причина выбора, также является сложность и релевантность. Классификация научных статей имеет практическое значение для организации и поиска научной литературы.

Визуализация графа из датасета Cora представлена на рисунке 3.

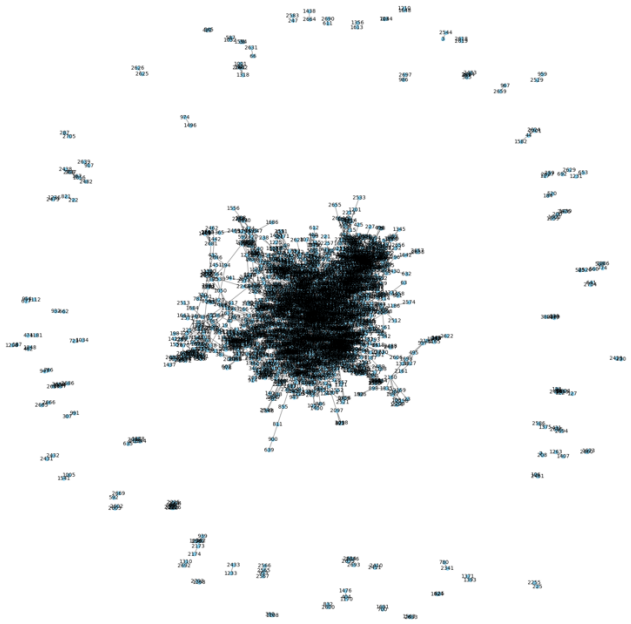

Рисунок 3 – Визуализация одного графа из датасета Cora.

1.5.1.4 Pubmed

Planetoid (Pubmed) [10] — это набор данных, содержащий цитатный граф научных публикаций из Pubmed. Узлы графа представляют научные статьи, а ребра — цитаты между ними. Каждый узел имеет атрибуты, представляющие собой слова из текста статьи, и метку, указывающую на категорию статьи. Статьи классифицируются по четырем различным категориям, таким как "Diabetes Mellitus Experimental", "Diabetes Mellitus Type 1", и "Diabetes Mellitus Type 2".

Этот набор данных используется для задачи классификации узлов. Цель состоит в том, чтобы предсказать категорию статьи на основе её атрибутов и структуры графа.

Характеристики:

- Количество узлов: 19717

- Количество ребер: 44338

- Количество классов: 4

- Количество атрибутов узла: 500

Эти датасеты предоставляют разнообразные и сложные задачи для графовых нейронных сетей, включая классификацию графов и узлов. Использование этих датасетов позволяет оценить эффективность различных архитектур GNN и методов оптимизации, таких как генетический алгоритм. Выбор этих датасетов обусловлен их сложностью и релевантностью для практических задач в биоинформатике и организации научной литературы.

Визуализация графа из датасета Pubmed представлена на рисунке 4.

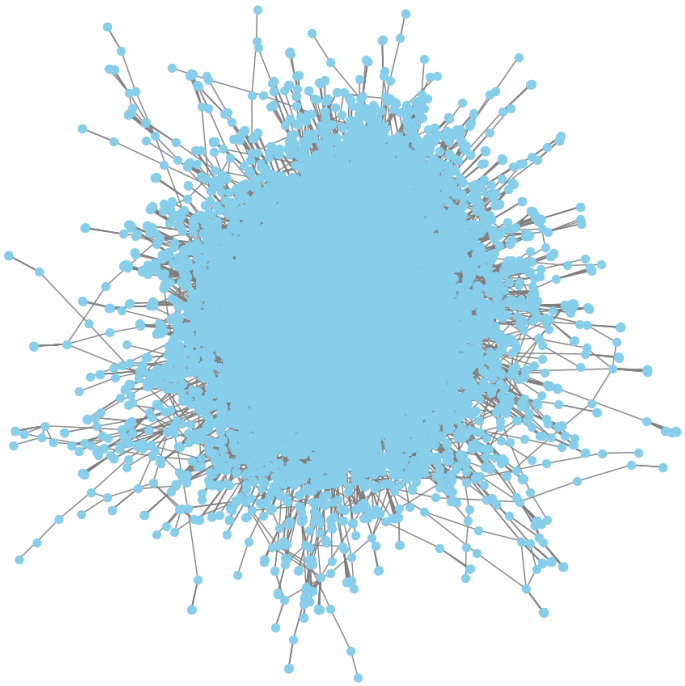

Рисунок 4 – Визуализация одного графа из датасета Pubmed.

1.5.2 Задачи анализа графов.

В данной курсовой работе рассматриваются две основные задачи анализа графов: классификация графов и классификация узлов.

1.5.2.1 Классификация графов (Graph Classification)

Задача классификации графов заключается в предсказании метки (класса) для целого графа на основе его структуры и атрибутов узлов и ребер. В этой задаче каждый граф рассматривается как единое целое.

1.5.2.2 Классификация узлов (Node Classification)

Задача классификации узлов заключается в предсказании метки (класса) для каждого узла в графе на основе его атрибутов и структуры графа. В этой задаче каждый узел рассматривается отдельно.

1.5.3 Формирование тренировочных, валидационных и тестовых наборов данных.

Для успешного обучения модели необходимо правильно организовать данные, разделив их на три основных набора:

1. Обучающий набор данных (Training Set) — главный набор, на основе которого происходит обучение модели. Он используется для оптимизации весов и параметров алгоритма.

2. Валидационный набор данных (Validation Set) — используется для периодической оценки производительности модели в процессе ее обучения. Он позволяет отслеживать, как хорошо модель обобщает данные и уменьшает вероятность переобучения.

3. Тестовый набор данных (Test Set) — набор, который используется для окончательной оценки модели после её обучения и настройки. Он помогает проверить, насколько уверенно модель может работать с новыми, невидимыми данными.

В данной курсовой работе данные разделены следующим образом: 60% на обучающие данные, 20% на валидационный набор и 20% на тестовый набор данных.

2. Разработка.

Поиск оптимальной архитектуры графовой нейронной сети является важной задачей в области машинного обучения и анализа данных. Архитектура GNN включает в себя выбор типа слоев, количество слоев, размерность скрытых слоев, функции активации и другие гиперпараметры. Оптимальная архитектура должна обеспечивать высокую точность и эффективность на заданной задаче.

2.1 Формулировка задачи поиска архитектуры GNN.

Задача поиска оптимальной архитектуры GNN может быть сформулирована как задача оптимизации, где целью является минимизация функции потерь или максимизация метрики качества (например, точности) на валидационном наборе данных. Формально, задача может быть представлена следующим образом:

$$maximize\ f(A)$$

где $f(A)$— функция фитнеса, оценивающая качество архитектуры A на валидационном наборе данных. Функция фитнеса может быть определена как точность классификации, F1-score или другая метрика качества.

Для оптимизации архитектуры GNN необходимо определить набор гиперпараметров, которые будут подлежать оптимизации. Основные гиперпараметры включают:

- Тип слоев (например, GCNConv, GraphConv, SAGEConv).
- Количество слоев.
- Размерность скрытых слоев.
- Функции активации (например, ReLU, Tanh, Sigmoid, SiLU).
- Методы агрегации (например, Global Mean Pooling, Global Max Pooling).
- Скорость обучения (learning rate).
- Коэффициент dropout.

2.2 Подход к оптимизации GNN с использованием ГА.

Генетические алгоритмы (ГА) являются эффективным методом оптимизации, основанным на принципах естественного отбора и генетики. ГА позволяют эффективно исследовать пространство гиперпараметров и находить оптимальные решения для сложных задач оптимизации.

Генетический алгоритм включает в себя следующие основные компоненты:

1) Формирование исходной популяции.

Задается номер популяции $t = 0$, максимальное количество популяций Np, номер итерации цикла $k = 1$, размер популяции Mp.

Случайным образом выбирается начальная точка x^0 - исходная хромосома. Она может быть выбрана как внутренняя точка гиперкуба области допустимых значений D. Из этой точки формируется исходная популяция. Для этого с помощью равномерного распределения на единичном отрезке [0,1] Mp раз генерируется последовательность из n случайных точек $\{P_i^{0k}\}_{i=1,n}^{k-1,Mp}, i = 1,...,n; k = 1,...,Mp$. Используя линейное преобразование, каждая точка отображается на соответствующий ей промежуток $[\alpha,\beta]: P_i^k = (\beta_i - \alpha_i)P_i^{0k} + \alpha_i$. Составляя векторы из точек последовательности $\{P_i^k\}$ при фиксированных k, получаем Mp начальных векторов $x^k = (x_1^k,...,x_n^k)^T, x_i^k = P_i^k, i = 1,...,n$, координаты которых x_i имеют равномерное распределение на отрезках $[\alpha_i,\beta_i], i = 1,...,n$. Таким образом может быть сформирована начальная популяция $I_0 = \{x^k, k = 1,...,Mp \,|\, x^k = (x_1^k, x_2^k,..., x_n^k) \in D\}$.

Вычисляется значение функции фитнеса для каждой особи $x^k \in I_0 : \mu_k = \mu(x^k), k = 1,...,Mp$ и популяции I_0 в целом $\mu = \sum_{k=1}^{Mp} \mu_i$.

2) Отбор (селекция).

Селекция – это операция, которая осуществляет отбор особь (хромосом) x^k в соответствии со значениями функции фитнеса $\mu(x^k)$ для последующего их скрещивания.

Необходимо, вычислить кумулятивную вероятность $q_i = \sum_{j=1}^{i} \mu_j(x^j), i = 1,2,...,Mp$. Затем, сформировать случайное действительное число r в интервале $(0, Mp]$. Выбрать i - ю хромосому x^i $(1 \le i \le Mp)$ так, чтобы $q_{i-1} < r \le q_i$. Перейти на шаг с формированием случайного действительного числа r в интервале $(0, Mp]$ до тех пор, пока не будет сформирована новая популяция ($while(i \le Mp)$).

3) Кроссинговер (скрещивание).

Скрещивание – это операция, при которой из нескольких, обычно двух хромосом (особей), называемых родителями, порождается одна или несколько новых, называемых потомками.

Определяется параметр $Pc \in (0,1]$ как вероятность кроссинговера. Эта вероятность дает ожидаемое число $Pc \cdot Mp$ хромосом, подвергаемых операции кроссинговера.

Для операции кроссинговера выполняется процесс, повторяющийся от $i = 1$ до $Pc \cdot Mp$: формируется случайное действительное число r из сегмента $[0,1]$, при этом, если $r < Pc$, то хромосома x^i выбирается как родительская.

Отбираются пары родительских хромосом (x^i, x^j), где $i \neq j$. Действие оператора кроссинговера осуществляется следующим образом:

Формируется случайное число $c \in (0,1)$, затем оператор кроссинговера, действующий на исходные пары (x^i, x^j), производит две хромосомы потомки X и Y:

$$X = c \cdot x^i + (1-c) \cdot x^j, \quad Y = (1-c) \cdot x^i + c \cdot x^j.$$

Если допустимое множество является выпуклым, то кроссинговер обеспечивает допустимость обоих потомков, в случае если допустимы оба родителя. Следует проверить допустимость каждого потомка перед тем, как он будет включен в новую популяцию. Если оба потомка являются допустимыми, тогда родители заменяются этими потомками. Если это не так, сохраняется допустимый потомок, если он существует, а затем вновь выполняется оператор кроссинговера с новым значением случайного числа c до тех пор, пока не будут получены два новых допустимых потомка или не будет превышено заданное число циклов. В этом случае осуществляется замена родителей только теми (сохраненными ранее) потомками, которые оказались допустимыми.

4) Мутация.

Мутация – это преобразование хромосомы, случайно изменяющее один или несколько из её генов. Оператор мутации предназначен для того, чтобы поддерживать разнообразие особей в популяции.

Определим параметр $Pm \in (0,1]$ как вероятность мутации. Эта вероятность дает ожидаемое число $Pm \cdot Mp$ хромосом, подвергаемых операции мутации.

Для операции мутации выполняется процесс, повторяющийся от $i = 1$ до $Pm \cdot Mp$: формируется случайное действительное число r из сегмента $[0,1]$, при этом, если $r < Pm$, то хромосома x^i выбирается как родительская для операции мутации. Для каждой выбранной родительской хромосомы x^i, обозначенной как $Z = (x_1, x_2, ..., x_n)$, производится мутация.

Поочередно рассматривается каждый потомок из ожидаемого числа $Pm \cdot Mp$ хромосом. Среди генов выбранной родительской хромосомы $Z = (x_1, x_2, ..., x_n)$ случайно (с вероятностью $1/n$) выбирается один с номером $p \in (1,2,...,n)$ подлежащий замене. Его новое значение x_p^M случайным образом выбирается из промежутка $[\alpha_p, \beta_p]$ изменения выбранной координаты x_p.

5) Формирование новой популяции.

С равной вероятностью из потомков мутантов предыдущего шага выбирается один $x^M = (x_1, x_2, ..., x_p^M, ..., x_n)$.

Выбранный потомок добавляется в популяцию вместо хромосомы, которой соответствует наименьшее значение функции фитнеса (наихудшее из допустимых значений).

Вычисляется значение функции фитнеса для мутантного потомка $\mu_M = \mu(x^M)$. Затем, проверка условий:

- Если $k < Mp$, то $k = k + 1$ и переход на шаг с селекцией.
- Если $k = Mp$, то $t = t + 1$ и переход на проверку условия остановки генетического алгоритма.

6) Проверка условия остановки генетического алгоритма.

Условием окончания работы генетического алгоритма является формирование заданного количества популяций $t = Np$. Если условие не выполнено, то полагаем $k = 1$ и переход на шаг с селекцией. Если условие окончания работы выполнено, то в качестве решения (приближенного) задачи $\mu(x_\mu^*) = \max_{x \in D} \mu(x)$ выбирается особь с лучшим значением функции фитнеса из текущей популяции: $x_\mu^* \cong x_\mu^e = Arg\max \mu(x^k)$, а по нему определяется приближенное решение поставленной задачи $f(x^*) = \min f(x) : x^* = x_\mu^*$.

3. Реализация

В этом разделе будет представлена реализация графовой нейронной сети, в соответствии с архитектурой, указанной в предыдущих разделах, а также интеграция генетического алгоритма (ГА) для оптимизации гиперпараметров.

3.1 Архитектура модели

Архитектура модели GNN (представлена в листинге 1), реализованной в классе `GNN`, построена на основе набора свёрточных слоёв, которые предназначены для обработки графовых структур данных. Модель была спроектирована с целью обеспечения гибкости и адаптивности, позволяя автоматически настраивать количество слоёв, их тип и другие гиперпараметры в зависимости от решаемой задачи node или graph классификация.

Основные компоненты архитектуры:

- Входные данные: Входные данные представляют собой атрибуты узлов графа `data.x` и информацию о структуре графа `data.edge_index`, которые подаются на первый свёрточный слой модели.
- Слои свёртки: Модель включает несколько свёрточных слоёв, определённых параметром `conv_type`, что позволяет выбирать между различными архитектурами GNN, такими как GCNConv, GraphConv, SAGEConv и GATConv. Каждый слой свёртки отвечает за агрегацию

информации от соседних узлов и обновление представления текущего узла.

- Функции активации: После каждого свёрточного слоя применяется функция активации, определяемая параметром `activation` модели. Это вводит нелинейность в модель, что позволяет ей обучаться более сложным паттернам в данных.
- Пуллинг: Для задач графовой классификации используется метод агрегации информации от всех узлов графа через слой пула `pooling`. В зависимости от настроек, может использоваться `global_mean_pool` или `global_max_pool`, которые агрегируют информацию путем вычисления среднего или максимального значения признаков узлов соответственно.
- Выходной слой: После агрегации информации на уровне графа или узла, представление передается в полностью связанный (линейный) слой `self.linear`, который преобразует его в конечные классы. На выходе используется логарифмически нормализованный софтмакс `LogSoftmax` для получения логарифмических вероятностей принадлежности к классам.

Архитектура модели позволяет легко адаптироваться к различным типам задач и данным, обеспечивая высокую гибкость и модульность. Возможность выбора различных типов свёрточных слоёв и функций активации делает модель универсальной и способной справляться с широким спектром графовых задач.

Листинг 1 – Класс GNN.

```
class GNN(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim=32,
hidden_num=2, conv_type='GCNConv',
                 activation='relu', pooling='global_mean_pool',
task='node'):
        super(GNN, self).__init__()
        self.convs = nn.ModuleList()
        self.sm = nn.LogSoftmax(dim=1)
        self.convs.append(getattr(pyg_nn, conv_type)(input_dim,
hidden_dim))
        for _ in range(hidden_num - 1):
```

```
            self.convs.append(getattr(pyg_nn,
conv_type)(hidden_dim, hidden_dim))
        self.linear = nn.Linear(hidden_dim, output_dim)
        self.activation = getattr(F, activation)
        self.pooling = pooling
        self.task = task

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index,
data.batch
        for conv in self.convs:
            x = conv(x, edge_index)
            x = self.activation(x)

        if self.task == 'graph':
            pooling_fn = getattr(pyg_nn, self.pooling) if
self.pooling else pyg_nn.global_mean_pool
            x = pooling_fn(x, batch)
        elif self.task == 'node':
            pass
        x = self.linear(x)
        x = self.sm(x)
        return x

    def loss(self, pred, label):
        return F.nll_loss(pred, label)
```

3.2 Использование Генетического Алгоритма для Оптимизации GNN.

3.2.1 Формирование популяции индивидуумов

В рамках генетического алгоритма популяция индивидуумов представляет собой множество различных конфигураций гиперпараметров, которые определяют архитектуру GNN. Каждый индивидуум может быть описан как набор параметров, таких как:

- Количество скрытых слоёв `hidden_num:` Параметр, указывающий, сколько свёрточных слоёв будет использоваться в модели.
- Размерность скрытых слоёв `hidden_dim`: Определяет количество нейронов в каждом скрытом слое, что влияет на способность модели учиться.

- Тип свёрточного слоя `conv_type`: Выбор между различными архитектурами свёртки, такими как GCNConv, GraphConv, SAGEConv и GATConv.
- Функция активации `activation`: Выбор функции активации для извлечения нелинейности.
- Метод агрегации `pooling`: Определение того, как информация агрегируется между узлами графа.
- Скорость обучения `lr`: Параметр, контролирующий скорость обновления весов модели.
- Уровень регуляризации `dropout`: Степень, до которой нейроны отключаются во время обучения, направленная на предотвращение переобучения.

Каждый индивидуум формируется случайным образом при инициализации популяции, что делает первый набор индивидуумов разнообразным.

3.2.2 Кроссинговер

Кроссинговер — это генетическая операция, которая сочетает характеристики двух родителей для создания нового потомка. Этот процесс позволяет комбинировать удачные гиперпараметры, унаследованные от обоих родителей. В контексте GNN кроссинговер можно реализовать через случайный выбор параметров от двух родителей: изменение отдельных гиперпараметров, таких как количество скрытых слоёв, размер скрытых слоёв или тип свёрточного слоя.

Создание нового индивидуума (потомка), содержащего качества обоих родителей, улучшает вероятность нахождения более эффективной архитектуры.

Реализация функции кроссинговера представлена в листинге 2.

Листинг 2 – Кроссинговер.

```
def crossover(parent1, parent2):
   hidden_num = random.choice([parent1.hidden_num,
parent2.hidden_num])
```

```
    hidden_dim = random.choice([parent1.hidden_dim,
parent2.hidden_dim])
    conv_type = random.choice([parent1.conv_type,
parent2.conv_type])
    activation = random.choice([parent1.activation,
parent2.activation])
    pooling = random.choice([parent1.pooling, parent2.pooling])
    lr = random.choice([parent1.lr, parent2.lr])
    return Individual(hidden_num, hidden_dim, conv_type,
activation, pooling, lr)
```

3.2.3 Мутация

Мутация является важным этапом генетического алгоритма, обеспечивающим разнообразие популяции. Она включает случайное изменение одного или нескольких гиперпараметров индивидов, что позволяет избежать преждевременной сходимости.

Реализация мутации представлена в листинге 3.

Листинг 3 – Мутация.

```
def mutate(individual, mutation_rate, task):
    if random.random() < mutation_rate:
        individual.hidden_num = random.randint(2, 10)
    if random.random() < mutation_rate:
        individual.hidden_dim = random.randint(16, 128)
    if random.random() < mutation_rate:
        individual.conv_type = random.choice(['GCNConv',
'GraphConv', 'SAGEConv', 'GATConv'])
    if random.random() < mutation_rate:
        individual.activation = random.choice(['relu', 'tanh',
'sigmoid', 'silu'])
    if random.random() < mutation_rate:
        if task == 'graph':
            individual.pooling =
random.choice(['global_mean_pool', 'global_max_pool'])
        else:
            individual.pooling = random.choice([None,
'global_mean_pool', 'global_max_pool'])
    if random.random() < mutation_rate:
        individual.lr = round(random.uniform(0.0001, 0.01), 4)
    return individual
```

Эти операции кроссинговера и мутации обеспечивают гибкость в генетическом алгоритме, позволяя динамически исследовать пространство

гиперпараметров и находить оптимальные конфигурации для архитектур GNN.

Всё это позволяет генетическому алгоритму адаптивно искать оптимальную комбинацию гиперпараметров, что приводит к значительному улучшению точности графовой нейронной сети на сложных наборах данных. Генетический алгоритм последовательно генерирует популяцию индивидов, оценивает их производительность, отбирает лучших, а затем производит кроссинговер и мутацию для создания новой популяции, тем самым исследуя пространство гиперпараметров.

Процесс продолжается, пока не будут достигнуты предельные условия, такие как максимальное количество поколений или достижение удовлетворительной точности. Это позволяет эффективно находить высокоэффективные архитектуры GNN, что является важным этапом в проектировании и разработке решений для анализа графовых данных.

3.2.4 Триггеры для ранней остановки.

Ранняя остановка — это метод, который прерывает обучение, когда модель начинает ухудшать свою производительность на валидационном наборе данных. Это особенно полезно, когда данные разделены на обучающий, валидационный и тестовый наборы. Если производительность модели не улучшается на валидационном наборе на протяжении заданного количества эпох (параметр, называемый `patience`), обучение прекращается. Это позволяет сохранять лучшую модель, достигнутую до начала ухудшения производительности. Реализация ранней остановки представлена в листинге 4.

Листинг 4 – Ранняя остановка.

```
if val_acc > best_val_acc:
    best_val_acc = val_acc
    best_model_state = copy.deepcopy(model.state_dict())
    epochs_without_improvement = 0
else:
    epochs_without_improvement += 1
    if epochs_without_improvement >= patience:
        print(f"Early stopping triggered after {epoch + 1}
epochs.")
        break
```

Здесь `epochs_without_improvement` отслеживает количество эпох, когда валидационная точность не улучшалась. При достижении заданного порога (patience) происходит остановка. Эти триггеры могут быть адаптированы и настроены в зависимости от конкретных задач и структур данных.

Использование триггеров для остановки итераций во время обучения позволяет значительно оптимизировать процесс, снижая вероятность переобучения и уменьшив время вычислений.

3.2.5 Сохранение и обработка результатов

В процессе выполнения генетического алгоритма все результаты, включая лучших индивидуумов и их гиперпараметры, сохраняются для последующего анализа. Результаты сохраняются в формате JSON, что позволяет легко извлекать и обрабатывать данные для дальнейшего анализа производительности моделей.

Каждый индивидуум имеет атрибуты, которые включают значение его фитнеса, параметры гиперпараметров и информацию о том, как они были получены. Результаты также включают точности на валидационных и тестовых наборах, а также конкретные архитектурные параметры (количество слоёв, размерность и типы активации).

4. Тестирование.

В этом разделе приведен анализ полученных данных о результатах поиска оптимальной архитектуры графовых нейронных сетей на различных датасетах, а также о лучших гиперпараметрах, найденных с помощью генетического алгоритма.

Вывод лучших архитектур для всех датасетов представлен в таблице 1.

Dataset	hidden_num	hidden_dim	conv_type	activation	pooling	lr
Cora	8	74	SAGEConv	silu	global_max_pool	0.007
Pubmed	5	107	GATConv	silu	global_max_pool	0.0033
Enzymes	2	44	SAGEConv	relu	global_max_pool	0.0084
Proteins	3	35	SAGEConv	silu	global_mean_pool	0.0078

Таблица 1 – Лучшие архитектуры для датасетов.

4.1 Результаты на наборе данных ENZYMES

Лучший индивид, полученный в процессе, имеет следующие гиперпараметры:

- Количество скрытых слоёв: 8
- Размерность скрытых слоёв: 74
- Тип сверточного слоя: SAGEConv
- Функция активации: Silu
- Метод пуллинга: Global Max Pool
- Скорость обучения: 0.007
- Точность: 0.4417

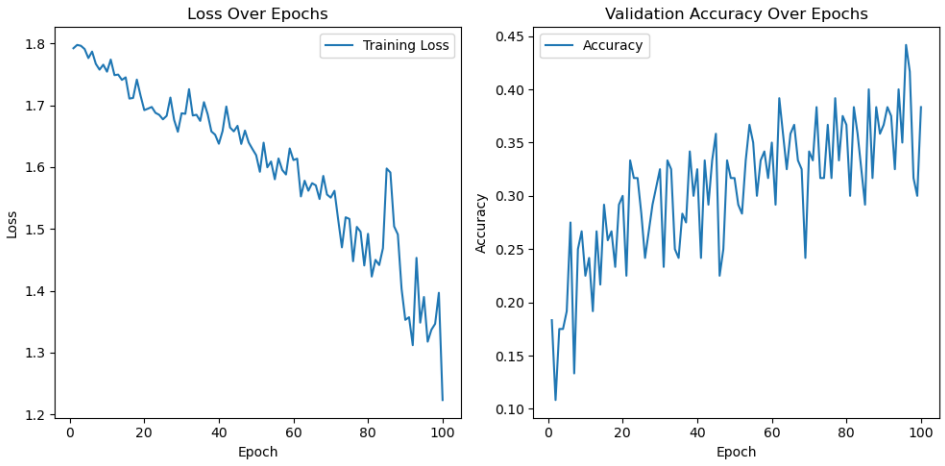

Рисунок 5 – График зависимости эпох от точности и потерь.

Значительно более низкая точность по сравнению с датасетом протеинов. Использование SAGEConv может быть недостаточно эффективным для данного типа графов. Возможно, требуется дополнительная оптимизация архитектуры или применение других методов обработки графовых структур.

Генетический алгоритм не смог достигнуть значительных результатов на данном наборе данных, так как большая часть индивидуумов демонстрировала низкие значения точности как на валидационных, так и на тестовых данных.

В первые эпохи обучения, точность моделей была очень низкой, и, несмотря на некоторые колебания, не удалось достичь заметного улучшения производительности. Модели на различных этапах сталкиваются с проблемами в обучении, то есть потери уменьшаются медленно, а точность остается на низком уровне. Также результаты нестабильны, похоже, архитектуры и гиперпараметры, которые были выбраны, не смогли адекватно отразить распределение классов в данных.

Низкие точности могут быть связаны с высокой сложностью задачи.

4.2 Результаты на наборе данных PROTEINS.

Лучший индивид, полученный в процессе, имеет следующие гиперпараметры:

- Количество скрытых слоёв: 5
- Размерность скрытых слоёв: 107
- Тип сверточного слоя: GATConv
- Функция активации: Silu
- Метод пуллинга: Global Max Pool
- Скорость обучения: 0.0033
- Точность: 0.7937

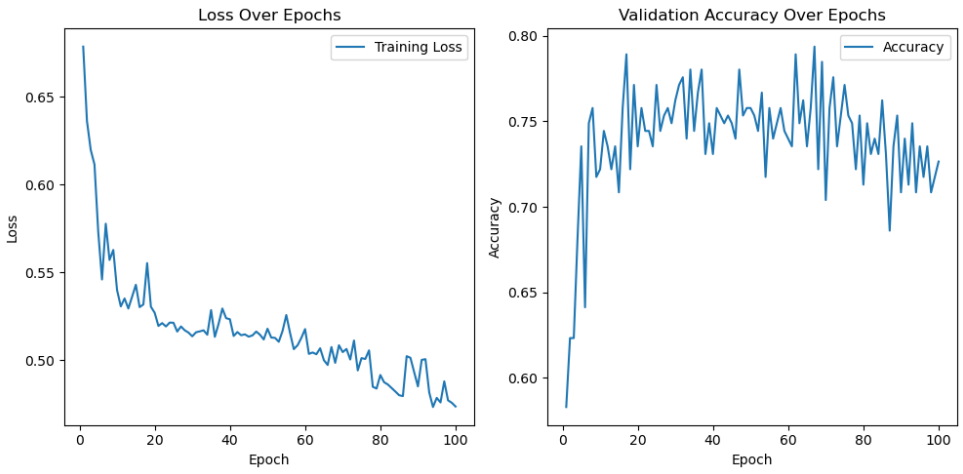

Рисунок 2 – График зависимости эпох от точности и потерь.

Модель с использованием GATConv демонстрирует высокую точность. Это может указывать на эффективность внимания к локальной структуре графов белковых взаимодействий. Также модели с использованием GATConv в начале алгоритма показывают высокую стабильность при изменении гиперпараметров по сравнению с другими, менее удачными комбинациями. Архитектура с 5 скрытыми слоями позволяет извлекать сложные нелинейные признаки.

Результаты показывают, что использование GATConv с выбранной комбинацией параметров дает лучшие результаты, чем другие методы свертки. Это может означать, что графовые сети хорошо справляются с рассматриваемыми задачами, особенно когда в модели есть внимательные механизмы.

Влияние гиперпараметров: Например, выбор silu и global_mean_pool в сочетании с GATConv предоставляет отличные возможности для улучшения производительности, указывая на важность правильного выбора функций активации и Pooling.

Хорошая валидационная точность (ближе к 0.7937) свидетельствует о хорошей способности модели к обобщению, подтверждая, что алгоритм успешен в оптимизации гиперпараметров.

4.3 Результаты на наборе данных Cora.

Лучший индивид, полученный в процессе, имеет следующие гиперпараметры:

- Количество скрытых слоёв: 3
- Размерность скрытых слоёв: 35
- Тип сверточного слоя: SAGEConv
- Функция активации: Silu
- Метод пуллинга: Global Mean Pool
- Скорость обучения: 0.0078
- Точность: 0.984

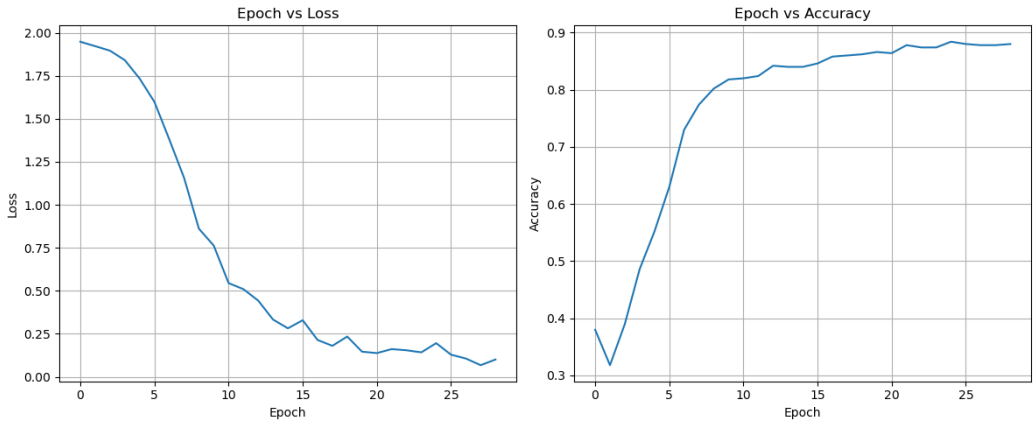

Рисунок 1 – График зависимости эпох от точности и потерь.

Исключительно высокая точность указывает на отличную способность модели к обобщению на этом датасете. Относительно небольшое количество скрытых слоёв и использование SAGEConv позволило эффективно извлекать признаки из графовой структуры научных цитирований.

Сработал триггер на раннюю остановку на 4 поколении. В последней итерации в 4-ом последнем поколении, в результате мутации число слоев увеличилось с 3 до 8 и увеличился learning rate, в результате чего точность упала.

Значения количества скрытых слоев варьируется от 2 до 10. Наиболее часто встречаются 3 скрытых слоя. Размерность скрытых слоев варьируется от 6 до 50. Распределение разнообразное, с акцентом на размерности 35 и 44.

Типы свертки SAGEConv и GCNConv доминируют в большинстве архитектур. В активационной функции доминирует silu, особенно в поколениях 2-9, за которым следует relu.

В пулинге преобладает global_mean_pool, но global_max_pool также широко используется, особенно в более поздних поколениях.

Значения скорости обучения (lr) варьируются от 0.0006 до 0.0095. Наиболее часто встречаются значения 0.0048 и 0.0078.

Значения метрики производительности (fitness) варьируются от 0.142 до 0.98. Большинство значений выше 0.8, что указывает на хорошую производительность моделей.

Если рассмотреть тренд по поколениям в 1-3 заметно улучшение значений fitness, устойчивое доминирование SAGEConv и GCNConv с активацией silu.И в последнем поколении стабилизация на высоких значениях fitness (до 0.98). Постепенный отход от разнообразия типов сверток в пользу SAGEConv.

4.4 Результаты на наборе данных PubMed.

Лучший индивид, полученный в процессе, имеет следующие гиперпараметры:

- Количество скрытых слоёв: 7
- Размерность скрытых слоёв: 51
- Тип сверточного слоя: GraphConv
- Функция активации: Tanh
- Метод пуллинга: Global Max Pool
- Скорость обучения: 0.0017
- Точность: 0.789

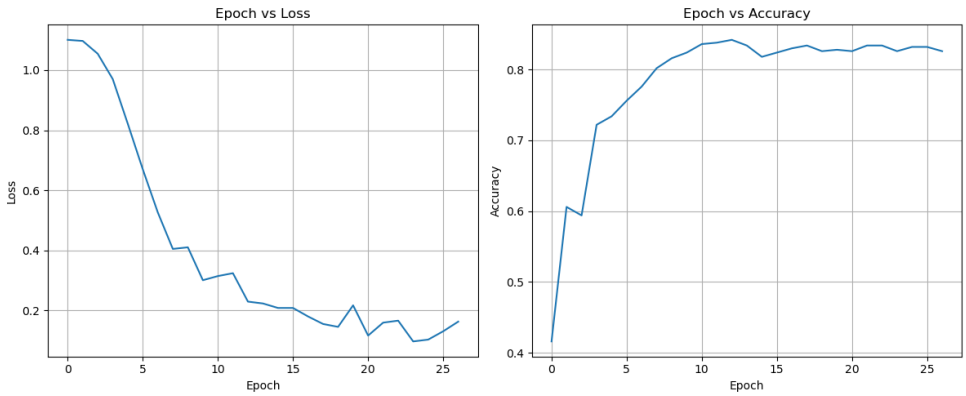

Рисунок 4 – График зависимости эпох от точности и потерь.

Архитектура с GraphConv показала стабильную производительность. Использование tanh в качестве функции активации и глобального максимального пуллинга позволило эффективно обрабатывать графовые структуры научных публикаций.

Архитектуры с размерностью 44 и 50 часто давали наивысшие значения фитнеса. Однако стоит обратить внимание на производительность архитектур с меньшими размерностями (например, 8 и 14).

4.5 Сравнительный анализ датасетов, использующихся для классификации графов.

1. Количество скрытых слоев:

PROTEINS: 5 слоев

ENZYMES: 8 слоев

Для PROTEINS оптимальное количество слоев меньше, что может указывать на менее сложную структуру графов в этом наборе данных. Для ENZYMES потребовалось больше слоев для извлечения информативных признаков, что говорит о более сложной внутренней структуре графов.

2. Размерность скрытых слоев

PROTEINS: 107 нейронов

ENZYMES: 74 нейрона

PROTEINS требует большей размерности для представления признаков, что может означать более богатое внутреннее представление графов. ENZYMES с меньшей размерностью слоев показывает сложность в извлечении признаков.

3. Типы графовых сверточных сетей.

PROTEINS: GATConv (Graph Attention Convolution)

ENZYMES: SAGEConv (GraphSAGE Convolution)

GATConv использует механизм внимания,позволяет динамически взвешивать вклад соседей, он эффективен для графов с неоднородной структурой связей

SAGEConv использует агрегацию признаков соседей, работает через усреднение или суммирование признаков соседних узлов и он подходит для графов с более однородной структурой

4. Различия в производительности:

На PROTEINS GATConv показал высокую эффективность, что может указывать на наличие сложных, нелинейных связей между узлами.

На ENZYMES SAGEConv не смог эффективно извлечь признаки, возможно, из-за более сложной топологии графов.

4. Активационная функция

Обе архитектуры: silu (Sigmoid Linear Unit)

Характеристики silu:

- Гладкая нелинейность
- Помогает в преодолении затухания градиентов

5. Метод пуллинга

Обе архитектуры: global_max_pool

Преимущества max-пуллинга:

- Извлекает наиболее значимые признаки
- Подавляет шум
- Работает хорошо для выделения ключевых характеристик графа

6. Сложность графов

PROTEINS: Более простая, более предсказуемая структура графов

ENZYMES: Более сложная, менее упорядоченная структура графов

7. Эффективность архитектуры

На PROTEINS генетический алгоритм нашел более оптимальную конфигурацию. На ENZYMES алгоритм испытывал существенные трудности в подборе архитектуры

Различия в архитектурах между PROTEINS и ENZYMES демонстрируют, насколько важен индивидуальный подход к каждому набору данных. Генетический алгоритм показал свою эффективность для PROTEINS, но потребовал дальнейшей настройки для ENZYMES.

4.6 Вывод.

Различные датасеты требуют уникальных подходов к архитектуре графовых нейронных сетей. PROTEINS и Cora достигли высоких результатов благодаря оптимальным конфигурациям, в то время как ENZYMES потребовал дополнительной оптимизации из-за большей сложности графов.

Наиболее универсальными типами свёрточных слоёв оказались SAGEConv и GCNConv, демонстрируя стабильные и высокие значения fitness в большинстве случаев

GCNConv и GraphConv появляются наиболее часто и, похоже, обеспечивают стабильные результаты, включая некоторые из самых высоких значений fitness.

SAGEConv также показывает хорошие результаты, особенно в сочетании с активацией relu и методом объединения global_max_pool.

GATConv иногда показывает хорошие результаты, но наблюдается большая вариативность, возможно, из-за чувствительности к другим гиперпараметрам.

Функции активации Silu и ReLU показали себя эффективными, способствуя хорошей производительности моделей и стабильному обучению. Sigmoid и Tanh показывают менее стабильные результаты.

Методы пуллинга Global Mean Pooling и Global Max Pooling оказались наиболее эффективными, при этом Global Mean Pooling был более универсальным выбором, особенно в сочетании с SAGEConv.

Большинство моделей используют скорость обучения около 0,007-0,0084. Более низкие значения, такие как 0,002, иногда приводят к снижению производительности, возможно, из-за слишком медленной сходимости.

Архитектуры с 3 скрытыми слоями и размерностью от 44 до 50 нейронов часто достигают высоких значений fitness. Более глубокие сети (до 11 слоев) не обязательно показывают лучшие результаты, что может указывать на переобучение или затруднённую тренировку глубоких моделей. Глубина сети и размерность скрытых слоёв играют значительную роль в производительности моделей, при этом оптимальные конфигурации варьировались в зависимости от сложности графов.

ЗАКЛЮЧЕНИЕ

В данной работе был проведен поиск оптимальной архитектуры графовых нейронных сетей на основе генетического алгоритма на четырех различных наборах данных: PROTEINS, ENZYMES, Cora, Pubmed. Исследование включало в себя разработку и выбор архитектур моделей, обучение, анализ точности на валидационных и тестовых наборах, а также выявление лучших конфигураций для каждой из задач.

Был проведен всесторонний анализ полученных результатов, который показал, что производительность графовых нейронных сетей варьируется в зависимости от конфигурации архитектуры, типа свертки и характеристик задаваемых данных. Выявленные узкие места и недостатки алгоритмов обучения подчеркивают необходимость более тщательной настройки гиперпараметров и выбора архитектур для достижения лучших результатов.

Выбор типа свёрточного слоя, функции активации, метода пуллинга и гиперпараметров существенно влияют на производительность модели. Наиболее успешными оказались архитектуры с использованием SAGEConv и GCNConv, функциями активации Silu и ReLU, а также методами пуллинга Global Mean Pooling и Global Max Pooling. Важно учитывать специфику каждого датасета

Таким образом, можно сделать вывод, что не существует универсальной архитектуры графовой нейронной сети, подходящей для всех типов графов. Каждый набор данных требует индивидуального подхода и тщательной настройки гиперпараметров для достижения наилучших результатов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Графовые нейронные сети.

url: https://neerc.ifmo.ru/wiki/index.php?title=Графовые_нейронные_сети

[2] Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой : учебное пособие / Карпенко А. П. - 3-е изд. - М. : Изд-во МГТУ им. Н. Э. Баумана, 2021. - 446 с. : рис., табл., граф. - Библиогр. ISBN 978-5-7038-5563-8.

url: https://ibooks.ru/bookshelf/386338/reading

[3] Goldberg, D. E. (1989). "Genetic Algorithms in Search, Optimization, and Machine Learning."

[4] Supervised Classification with Graph Convolutional Networks. Thomas N. Kipf, Max Welling. International Conference on Learning Representations (ICLR), 2017

url: https://arxiv.org/abs/1609.02907

[5] Graph Attention Networks. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, Yoshua Bengio. International Conference on Learning Representations (ICLR), 2018

url: https://doi.org/10.48550/arXiv.1710.1090

[6] GraphSAGE

Inductive Representation Learning on Large Graphs. William L. Hamilton, Rex Ying, Jure Leskovec. Neural Information Processing Systems (NeurIPS), 2017

url: https://doi.org/10.48550/arXiv.1706.02216

[7] ENZYMES dataset.

url: https://paperswithcode.com/dataset/enzymes

[8] PROTEINS dataset.

url: https://paperswithcode.com/dataset/proteins

[9] Cora dataset.

url: https://paperswithcode.com/dataset/cora

[10] Pubmed dataset.

url: https://www.google.com/search?client=safari&rls=en&q=pubmed+datset&ie=UTF-8&oe=UTF-8