Правительство Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет «Высшая школа
экономики»

Факультет компьютерных наук
Основная образовательная программа
01.03.02 «Прикладная математика и информатика»

ОТЧЕТ

ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Выполнил студент
группы БПМИ171
Гельван Кирилл Павлович

Руководитель практики

Лаборатория компании Самсунг, научный сотрудник
Чиркова Надежда Александровна

05.10.2020
Дата

10
(оценка)

Гельван К.П.

(подпись)

Москва 2020

Аннотация

В данной работе исследуются два подхода к решению задачи суммаризации исходного кода. В то время как один из них работает с программой, как с обычным текстом, другой опирается на структуру кода. Данная практика включает в себя изучение этих подходов и их совмещение в один, а также сравнение его с базовым решением.

Место прохождения практики

Производственная практика выполнена под руководством научного сотрудника лаборатории компании Самсунг.

Содержание

1 Введение **3**

2 Постановка задачи **4**

3 Связанные работы **4**

3.1 Transformer-based Source Code Summarization 5

3.1.1 Базовая структура модели трансформер 5

3.1.2 Self-Attention 5

3.1.3 Copy Attention 6

3.1.4 Position Representations 6

3.2 Structured Neural Summarization 7

3.2.1 Graph Neural Networks 7

3.2.2 Gated Graph Neural Networks 8

4 Предложенный подход **9**

5 Результаты **10**

1 Введение

Задача суммаризации исходного кода является задачей seq2seq и может быть сформулирована следующим образом: по заданной на языке программирования функции необходимо сгенерировать небольшой комментарий или docstring (краткое описание функционала данного программного кода).

Эта задача тесно связана с областью обработки естественного языка, в которой задача суммаризации текста исследуется достаточно давно. В последние годы появляется все больше и больше различных нейросетевых подходов как для решения данной задачи, так и применительно к коду. Часть из них в явном виде учитывает структуру программы Eriguchi et al. (2016), другая - использует классические модели обработки естественного языка Wei et al. (2019).

Наиболее популярно для первого подхода использование абстрактных синтаксических деревьев (AST) - представлений кода в виде ориентированного дерева, в котором внутренние вершины являются операторами языка программирования, а листья - переменными и константами. При втором же подходе важным аспектом является то, что в коде часто встречаются долгосрочные (между далекими токенами) зависимости. Именно поэтому с развиситем архитектуры трансформер с механизмом внимания Vaswani et al. (2017) данная область получила новый виток развития после рекуррентных сетей. Целью практики было совместить два подхода: реализовать архитектуру трансформера для задачи анализа исходного кода программ, учитывающую синтаксическую структуру кода (его AST представление).

Задачи:

1) Запустить базовое решение с трансформером для задачи суммаризации кода, используя код из статьи

2) Выполнить обработку данных, используя специальные библиотеки python, в частности работа с AST-деревьями

3) Реализовать архитектуру из выбранной статьи, используя библиотеку PyTorch

4) Сравнить реализованную и исходную архитектуры по качеству работы, проанализировать реализованную модель

2 Постановка задачи

Рассмотрим ожидаемый пример работы обученной модели. На вход ей подается пример обучающего объекта кода, которому соответствует описание на естественном языке:

Пример:

```
public static void _(String name,
                     int expected, MetricsRecordBuilder rb) {
    Assert.assertEquals("Bad value for metric " + name,
        expected,
        getIntCounter(name, rb));
}
```

Примеры целевого значения, соответствующего данной функции:

Правильный ответ: assert counter

Вариант предсказания: assert int counter

3 Связанные работы

За время прохождения практики были изучены: архитектура трансформера и ее особенности для работы с машинным кодом, Graph Neural Networks(GNN), а также новые подходы к обработке исходного кода программ на основе GNN и статьи Fernandes et al. (2018).

3.1 Transformer-based Source Code Summarization

В качестве базовой модели была выбрана работа Ahmad et al. (2020). Авторы предлагают подход к суммаризации кода, основанный на архитектуре трансформер, и достигают с ним выскоих результатов относительно конкурентных работ. Также модель имеет авторскую реализацию в открытом доступе на языке PyTorch, что удовлетворяет задачам практики. Авторы показывают превосходство их модели над ближайшими конкурентами, а также демонстрируют улучшение большинства приведенных метрик относительно базовой версии трансформера и других подходов.

3.1.1 Базовая структура модели трансформер

Архитектура трансформер была представлена еще в 2017 году и только набирает популярность с тех пор. Модель изначально решала задачу машинного перевода, то есть на входе и на выходе имела последовательность некоторых токенов(seq2seq). Архитектура представляет из себя поочередные линейные слои и слои multi-head attention для энкодера и декодера. На каждом слое используется h голов внимания и механизм self-attention.

3.1.2 Self-Attention

Идея описывается еще в оригинальной статье по трансформеру: В каждой голове имеется последовательность входных векторов $x = (x_1, \ldots, x_n)$, а на выход подается послетовательность векторов $o = (o_1, \ldots, o_n) \in \mathbb{R}^{d_k}$:

$$e_{ij} = \frac{x_i W^Q (x_j W^K)^T}{\sqrt{d_k}}$$

$$o_i = \sum_{j=1}^{n} \text{softmax}(e_{ij})(x_j W^V)$$

Матрицы W^Q, W^K и W^V это обучаемые матрицы параметров queries, keys и values соответственно. Они уникальны для каждого слоя и головы.

3.1.3 Copy Attention

Важным дополнением работы Ahmad et al. (2020) к базовой архитектуре транфсорфмера является механизм копирования copy attention. Он изначально был представлен в работе See et al. (2017). Дополнительный слой механизма копирования вставляется после всех декодеров и позволяет модели копировать редкие токены (напр. названия функций, перменных).

Для каждого токена на основе механизма внимания высчитывается вероятность $p_{gen} \in [0, 1]$, которая отвечает за генерацию токенов из словаря (против копирования токенов из исходного текста). Затем p_{gen} используется как мягкий переключатель между сэмплированием из словаря и копированием токена из источника текста:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t$$

где вектор a^t это вектор распределения внимания в слое copy attention на шаге t.

Также стоит отметить, что авторы статьи упоминают об использовании AST в контексте дополнения copy attention. Представление кода в виде абстрактного синтаксического дерева использовалось ими в качестве маски для внимания: внутренние вершины дерева не могли быть скопированы в данном слое.

3.1.4 Position Representations

Еще одним дополнением от авторов стала проделанная работа над улучшением position representations (представлений позиций). Проведенные эксперименты показывают, что кодирование абсолютных позиций токенов не помогает при решении задачи суммаризации кода. Однако, механизм self-attention был изменен для кодирования относительных (попарных) представ-

лений между токенами:

$$e_{ij} = \frac{x_i W^Q (x_j W^K + \mathbf{a_{ij}^K})^T}{\sqrt{d_k}}$$

$$o_i = \sum_{j=1}^{n} \text{softmax}(e_{ij})(x_j W^V + \mathbf{a_{ij}^K})$$

a_{ij}^V и a_{ij}^K отвечают за относительные представления двух позиций i и j. Нововведением стало игнорирование направления при использовании попарных представлений позиций.

3.2 Structured Neural Summarization

Другой же ключевой работой в данном исследовании можно назвать Fernandes et al. (2018). Авторы предлагают решать проблему длинных зависимостей в текстах при помощи графовых нейронных сетей (GNN). Делается это при помощи совмещения sequence encoders и GNN. Сначала используется стандартный энкодер для последовательности(напр. двунаправленная рекуррентная нейронная сеть), затем представления токенов прогоняются через GNN (будет обсуждаться далее), а полученные представления передаются в декодеры. Авторы проводят эксперименты в том числе и на более структурированных данных, чем тексты на естественном языке. Одна из рассмотренных задач (MethodDoc) и является задачей суммаризации кода.

3.2.1 Graph Neural Networks

Для более понятных дальнейших переходов приведем краткое описание графовых нейронных сетей.

Граф $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ состоит из множества вершин $\mathcal{V}$, множества признаков (векторных представлений) вершин $\mathcal{X}$ и списка направленных ребер $\mathcal{E} = (\varepsilon_1, \ldots, \varepsilon_K)$, где ε_i - ребро, представленное парой вершин. Каждой вершине $v \in \mathcal{V}$ ставится в соответствие вещественный вектор представления x_v (напр.

эмбеддинг). Будем также следить за представлениями вершины графа на каждом этапе обучения, на нулевом: $h_v^0 = x_v$

Обучение графовой нейронной сети можно разделить на 2 последовательных этапа: проход по графу и применение модели.

На первом этапе происходит распространение по графу (propagation). Часто (в том числе и авторы рассматриваемой работы) под этим подразумевают отправку сообщений (neural message passing). Для этого каждая вершина v отправляет сообщение своим соседям при помощи трансформации своего представления $h_v^{(i)}$ некоторой функцией f. Все сообщения вычисляются и выполняются одновременно. Затем происходит аггрегация:

$m_v^{(i)} = g(\{f(h_u^{(i)}) \mid \exists l : \varepsilon_l = (u, v)\})$

где g - функция аггрегации.

На втором этапе каждая вершина имеет полученное сагреггированное сообщение $m_v^{(i)}$ и собственное представление $h_v^{(i)}$. На их основе вычисляется новое собственное представление для каждой вершины:

$h_v^{(i+1)} = q(m_v^{(i)}, h_v^{(i)})$

Количество итераций может меняться в зависимости от сходимости или выполняться фиксированное количество раз.

3.2.2 Gated Graph Neural Networks

Данный подход был представлен в работе Li et al. (2017)

Основные его идеи заключаются в конкретном применении GNN:

Функция f - линейный слой нейросети. g - поэлементная сумма. q - GRU, где GRU это управляемый рекуррентный нейрон. Фиксируется также количество шагов T, как выход модели используются финальные представления вершин:

$\mathsf{GGNN}((\mathcal{V}, \mathcal{E}, \mathcal{X})) = \{h_v^{(T)}\}_{v \in \mathcal{V}}$

Авторы статьи Fernandes et al. (2018) применяют Gated Graph Neural Networks (GGNN) вместе с sequence encoders для получения представленных результатов следующим образом:

Для начала каждый токен в последовательнотсти будем считать верши-

ной $s_i \in S$. Ребра (список пар вершин) обозрачим за R (способ его получения зависит от задачи). После получения представлений с помощью sequence encoder $[e_1 \dots e_n]$ строится GGNN:

$$[e'_1 \dots e'_N] = \mathsf{GGNN}((S, R, [e_1 \dots e_N]))$$

В декодер подаются полученные представления $[e'_1 \dots e'_N]$.

Авторы рассмотрели большое количество комбинаций различных энкодеров и декодеров. В том числе и с использованием self-attention, однако, использовалась лишь базовая часть архитектуры трансформер. Добавление GGNN в трансформер показало неоднозначные результаты как на разных задачах, так и на разных метриках.

4 Предложенный подход

В качестве модели для реализации было решено совместить модель суммаризации кода на основе трансформера и GGNN, а именно добавить вышеописанные слои между энкодерами и декодерами. Для построения ребер было выбрано AST представление программного кода.

Базовая модель трансформера, дополненная графовым слоем, давала прирост в качетсве, поэтому можно было ожидать успеха и в данном исследовании. Так как работа выполнялась на языке PyTorch для работы с графами была выбрана библиотека pytorch-geometric.

Использовался уже готовый AST-парсер. Он и построение ребер были вынесены в общий препроцессинг для модели. Ребра сохраняются в отдельных файлах и загружаются в модель вместе с остальными данными с использованием специфических для фреймворка и модели структур данных. Также стоит отметить, что зачастую графы хранятся в разряженном виде, поэтому был применен трюк описанный в Allamanis et al. (2018). Для этого также использовались встроенные инструменты библиотеки pytorch-geometric.

5 Результаты

Реализован предложенный подход. Проведены эксперименты на вычислительном кластере НИУ ВШЭ с использованием системы управления заданиями Slurm. Были рассмотрены разные гиперпараметры для слоя GGNN(табл. 5.1). Всего были проведены эксперименты с 5 разными моделями:

Таблица 5.1: Валидация на данных языка Python

	BLEU	ROGUE − L	parametres
baseline	36.49	43.34	84M
GRU size=1	28.89	36.23	85.8M
GRU size=1, num of GRU=2	10.02	15.12	87.7M
GRU size=1, Message Passing=2	10.04	14.95	85.8M
GRU size=2	6.92	10.81	86.1M

1. Базовое решение. Без использования GGNN слоев (baseline)
2. Добавление 1 GGNN слоя с глубиной GRU 1 (GRU size=1)
3. Добавление 2 последовательных GGNN слоев с глубиной GRU 1 каждый (GRU size=1, num of GRU=2)
4. Добавление 1 GGNN слоя с глубиной GRU 2 (GRU size=2)
5. Добавление 1 GGNN слой с двумя последовательными передачами ошибки по графу (GRU size=1, Message Passing=2)

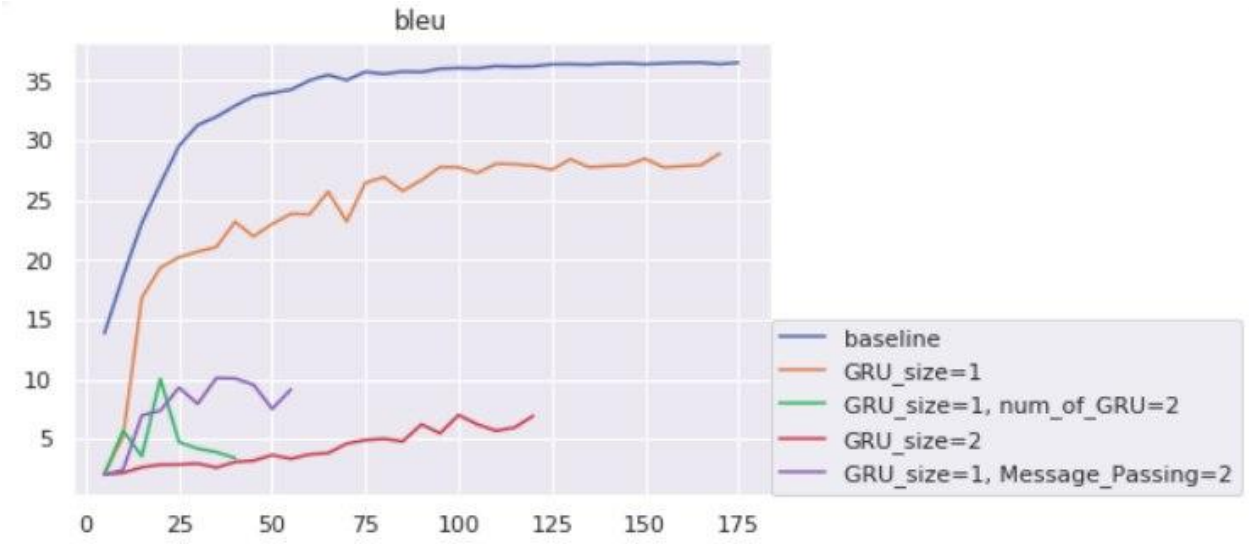

Рис. 5.1: График зависимости метрики BLEU от номера эпохи

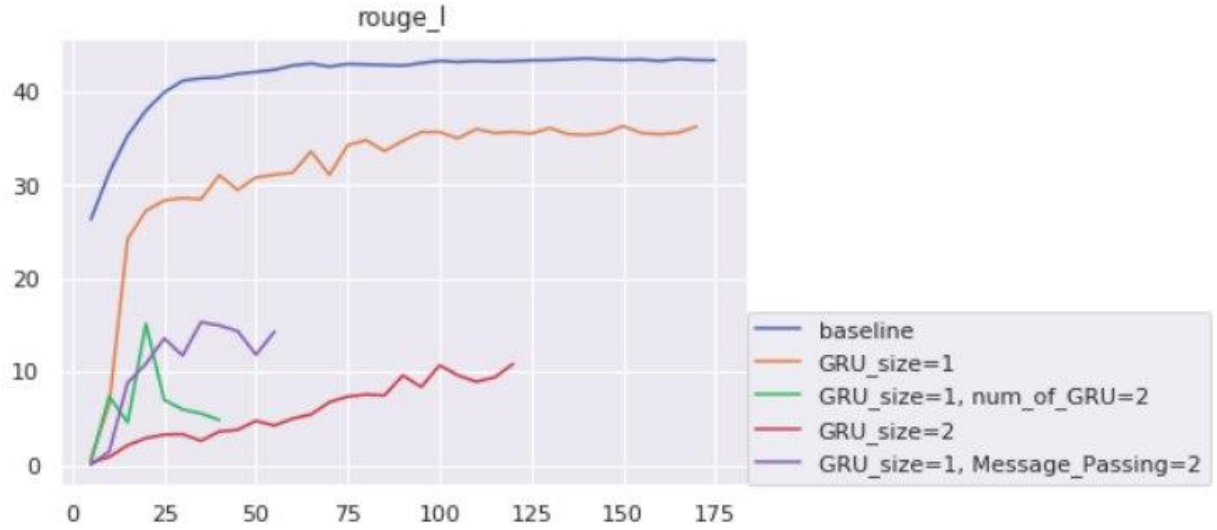

Рис. 5.2: График зависимости метрики ROGUE-L от номера эпохи

К сожалению, построенная модель не смогла по качеству превзойти базовую. Возможно, это связано с увеличением количества параметров в модели, в связи с чем могло бы помочь изменение гиперпараметров базовой модели. В Fernandes et al. (2018) не использовался датасет языка программирования Python (который считается более высокоуровневым яыком программирования нежели рассматриваемые Java и C), что тоже могло отразиться на изменении метрик.

Список литературы

1. Akiko Eriguchi and Kazuma Hashimoto and Yoshimasa Tsuruoka. Tree-to-Sequence Attentional Neural Machine Translation. In ACL 2016.

2. Bolin Wei and Ge Li and Xin Xia and Zhiyi Fu and Zhi Jin. Code Generation as a Dual Task of Code Summarization. In NeurIPS 2019.

3. Ashish Vaswani and Noam Shazeer and Niki Parmar and Jakob Uszkoreit and Llion Jones and Aidan N. Gomez and Lukasz Kaiser and Illia Polosukhin. Attention Is All You Need.

4. Wasi Uddin Ahmad and Saikat Chakraborty and Baishakhi Ray and Kai-Wei Chang. A Transformer-based Approach for Source Code Summarization. In ACL 2020.

5. Patrick Fernandes and Miltiadis Allamanis and Marc Brockschmidt. Structured Neural Summarization. In ICLR 2019.

6. Abigail See and Peter J. Liu and Christopher D. Manning. Get To The Point: Summarization with Pointer-Generator Networks.

7. Yujia Li and Daniel Tarlow and Marc Brockschmidt and Richard Zemel. Gated Graph Sequence Neural Networks. In ICLR 2016.

8. Miltiadis Allamanis and Marc Brockschmidt and Mahmoud Khademi. Learning to Represent Programs with Graphs. In ICLR 2018.

Рабочий план-график	
Организационное собрание и инструктаж	Выполнено, 01.07.2020
Построение базовоего решения и выбор статьи	Выполнено, 03.07.2020
Изучение статьи и предобработка данных	Выполнено, 08.07.2020
Построение решения и отладка	Выполнено, 11.07.2020
Сравнение результатов и написание отчета	Выполнено, 12.07.2020
Представление результатов и отчета	Выполнено, 13.07.2020