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

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Исследовательский проект на тему
"Методы обучения с подкреплением для ответов
на вопросы на основе графов знаний"

Выполнил студент группы 171, 4 курса,
Шабалин Александр Михайлович

Руководитель ВКР:
Доцент,
Департамент больших данных и информационного поиска
Артемова Екатерина Леонидовна

Консультант:
Технический директор
Optimate AI
Свиридов Сергей Владимирович

Москва 2021

Содержание

Аннотация

В этой работе мы решаем задачу ответов на вопросы на основе графа знаний (ГЗ). Более конкретно, мы формулируем проблему в терминах обучения с подкреплением и обучаем агента находить путь в ГЗ, который ведет от начальной сущности к целевой, в соответствии с заданным отношением между ними. В качестве агента мы предлагаем модель на основе трансформера, обученную с помощью алгоритма REINFORCE выбирать наиболее перспективное ребро в ГЗ и переходить по нему. Во всех предыдущих работах, решающих задачу обучением с подкреплением, использовались рекуррентные сети для генерации векторных представлений пройденных путей и простые полносвязные сети для предсказания отношений. Мы заметили, что пройденный путь в ГЗ может рассматриваться как последовательность сущностей и отношений. В таком случае задача может быть переформулирована как классификация последовательности, для решения которой отлично подходят трансформеры. Мы также предлагаем две модификации функции награды для улучшения работы алгоритма: штраф за длину пути и награда за неверные ответы. Наши эксперименты показали, что на наборе данных WN18RR предложенный алгоритм значительно превосходит все предыдущие алгоритмы, решающие данную задачу.

Ключевые слова

Граф знаний, ответы на запросы на основе графов знаний, обучение с подкреплением, трансформер

Abstract

In this paper, we are solving the problem of answering questions by walking around a knowledge graph (KG). More specifically, we formulate the problem in terms of reinforcement learning and train an agent to find a path in KG that leads

from starting entity to the target, according to the given relation between them. Our agent is a transformer-based model trained with REINFORCE algorithm to sample the most promising edge and walk over it. All prior reinforcement learning approaches used recurrent networks for generating embeddings and simple MLPs for an action prediction. We noticed that a trajectory in the knowledge graph can be seen as a sequence of entities and relations. In this case the problem can be formulated as a sequence labeling taks and transformer-based model is a perfect solution to it. We also propose two modifications of the reward function: path lenght penalty and positive reward for the wrong answer. Experimental results on knowledge graph dataset WN18RR show that our approach outperforms stateof-the-art methods by a large margin.

Key words

Knowledge graph, Knowledge graph-based question answering, reinforcement learning, transformer

1 Введение

Мы решаем задачу нахождения в графе знаний пути из одной сущности в другую, связанную с ней заданным отношением. Графы знаний представляют из себя способ хранения больших объемов структурированной информации. Они могут состоять из миллионов сущностей и отношений между ними, однако большинство отношений между сущностями отсутствует, так как граф разрежен. Наша задача - восстановить отсутствующие отношения, пользуясь имеющейся информацией об устройстве графа. Граф знаний $\mathcal{G}$ можно представить в виде набора триплетов $(e_1, r, e_2) \in (\mathcal{E}, \mathcal{R}, \mathcal{E})$, где $\mathcal{E}$ - множество сущностей, а $\mathcal{R}$ - множество отношений. Тогда задачу можно переформулировать как поиск сущности e_o на основе имеющихся e_s и r_q для восстановления триплета (e_s, r_q, e_o). Для решения поставленной задачи мы используем аген-

та, обученного с помощью обучения с подкреплением, который, стартуя их сущности e_s итеративно переходит в другие сущности по имеющимся отношениям в поиске сущности e_o. В процессе обучения агент должен выучить политику переходов, чтобы после окончания обучения он был способен по не виденной им паре (e_s, r_q) найти путь к e_o. При этом во время перемещений по графу агент никак не может определить, правильно ли он действует. Агент получает информацию в виде ненулевой награды только в конце эпизода. Он получает награду 1, если в конце эпизода оказался в верной сущности e_o, и 0 иначе. По этой причине мы формулируем задачу в виде Марковского процесса принятия решений и обучаем агента с помощью обущения с подкреплением.

При решении поставленной задачи появляются две основные проблемы: первая – это необходимость очень точно выбирать действия, так как при нескольких неточностях подряд агент уходит в неверную сторону и ему становится сложно вернуться. Вторая – разреженность награды, из-за чего агент не может получить достаточно информации о том, верным ли было конкретное действие. На Рис. 1 изображен пример ГЗ. Из него видно, что верное действие невозможно выбрать на основе только текущей сущности. При нахождении в сущности $e_t =$ “Жёрдочка” нельзя уверенно сказать, что лучшим действием будет переход в сущность $e_{t+1} =$ “Дом”. Поэтому для нахождения верного пути, в каждом состоянии необходимо хратить весь пройденный путь, а так же начальную сущность e_s и целевое отношение r_q. Это значит, что для построения точной модели нам надо идеально классифицировать последовательность совершенных действий для выбора следующего, поэтому наш выбор - трансформер, который хорошо справляется с этой задачей и решает первую проблему. Мы решаем задачу с помощью алгоритма REINFORCE [15], поэтому вторая проблема имеет для нас ключевое значение и значительно замедляет обучение. Для борьбы с ней мы вводим три модификации функции награды:

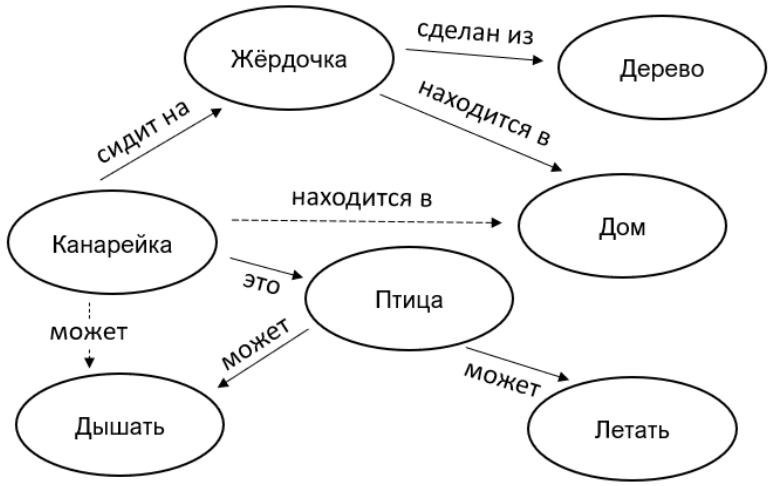

Рис. 1: Пример фрагмента графа знаний. Сплошными линиями показаны наблюдаемые отношения, а пунктирными – запросы.

1) Мы награждаем агента за выбор любой из верных сущностей наградой 1. На примере графа на Рис. 1 при запросе (“Канарейка”, “может”) верным ответами будут сущности “Летать” и “Дышать”. Так как по данному запросу из невозможно различить, нет никакого смысла награждать агента только за выбор одной из них.

2) Мы добавляем штраф за длину пути, чтобы поддталкивать агента к выбору более коротких путей.

3) За выбор неверного ответа мы награждаем агента с помощью модели, построенной на векторных представлениях, которая возвращает уверенность к том, что конкретный ответ верный. Благодаря такому подходу мы можем указывать агенту, насколько сильно он ошибся в своем выборе.

Эксперименты на наборах данных WN18RR [3] показали, что предложенный нами алгоритм превосходит все предыдущие со значительным отрывом, однако на наборах данных FB15k-237 [13] и Kinship подобного результата достичь не удалось.

Остальная часть статьи организована следующим образом: В разделе 2 описаны существующие методы решения задачи ответов на вопросы на основе графов знаний. В разделе 3.1 формально описывается поставленная задача. В разделах 3.2 и 3.3 описан процесс обучения. В разделе 3.4 представлены

модификации функции награды. Результаты работы алгоритма описаны в разделе 3.6. После него идет раздел с проведенными экспериментами, которые не дали улучшений в качестве 3.5. Наконец, в разделе 4 мы подводим итоги работы.

2 Обзор литературы

Задачи на основе графов знаний направлены на восстановления отсутствующих фактов и делятся на два типа: проверка существования отношения между двумя сущностями (запросы вида $(e_1, ?, e_2)$), и поиск ответа на вопрос (запросы вида $(e_1, r, ?)$). Оба типа задач и их решения имеют длинную историю развития, и в настоящее время данная область начала гораздо активнее развиваться. Причина этого явления заключается в том, что в виде ГЗ очень удобно хранить большое число фактов, а успешные алгоритмы их восстановления могут в разы увеличить объемы хранящейся информации. Также подобные алгоритмы черезвычайно востребованны во многих задачах, связанных с обработкой естественного языка, и могут внести значительный вклад в эту сферу. Несмотря на возросшую популярность, пока никому не уданость предложить state-of-the-art решение для задач на основе графов знаний. В данном обзоре мы будем рассматривать подходы к решению только второго типа задач: ответов на вопросы.

Первые методы решения задачи опирались на предсказания на основе только векторных представлений сущностей и отношений [2; 3]. Такие модели показывают хорошие результаты, однако при предсказании они не используют информацию о связях в графе и поэтому им не удается превзойти другие подходы.

Методы на основе перемещений по графу оказались превосходящими по качеству и интерпретируемыми, поскольку они напрямую используют связи между сущностями и строят свои предсказания, перемещаясь между ними, опираясь на информацию о пройденном пути. Данные методы можно разбить

на два вида: использующие обученные логические правила для получения вероятностей переходов [14; 18] и использующие обучение с подкреплением [6; 9; 16]. Несмотря на то, что подход с логическими правилами является более интерпретируемыми, обучение с подкреплением его вытесняет.

Задача поиска путей имеет две проблемы, требующие решения. Первая заключается в необходимости находить векторные представления сущностей и отношений, а вторая – в разреженности награды. Из-за отсутствия в обучении с подкреплением лучшего алгоритма, способного обучаться в таких условиях, авторы статей используют различные алгоритмы. Для решения первой проблемы в статье [9] используется GRU-RNN для получения векторных представлений состояний и полносвязные сети для возможных действий. Аналогичное решение используется в статье [16]. Они применяют LSTM к действиям на пройденной траектории, текущей сущности и целевому отношению. После этого они используют две полносвязных сети для выбора следующего действия.

Вторая проблема более сложная и имеет больше различных решений. В статье [16] авторы обучают модель с помощью алгоритма REINFORCE, добавляя ряд изменений функции награды для ускорения обучения. В статье [9] используется алгоритм дерева поиска Монте-Карло (MCTS) совместно с Q-learning. Они выбирают выбирают действия в соответствии с текущей политикой, а затем обновляют веса Q-функции с помощью MCTS. Подход с предобучением описан в статье [17]. Ее авторы составляют набор данных, обходя граф знаний с помощью BFS. Так они получают кратчайшие пути от начальной сущности к целевой, а затем обучают агента воспроизводить действия на каждом из путей.

3 Основная часть

В этой части мы подробно опишем наш подход на основе обучения с подкреплением для нахождения ответов на вопросы. Сперва мы сформулируем

задачу и введем необходимые определения. Затем мы покажем, что данная задача может быть перефразирована в терминах классификации последовательности, для решения которой мы предложим архитектуру агента на основе трансформера [1] и разберем процесс обучения.

3.1 Постановка задачи в терминах обучения с подкреплением

В терминах обучения с подкреплением задача ответов на вопросы на основе графа знаний состоит из двух частей: окружения и агента, действующего в данном окружении.

Окружение представляет собой Марковский процесс принятия решений, состоящий из множества состояний, действий, вероятностей переходов и функции награды.

Агент представляет из себя политику в виде нейронной сети $\pi_\theta(s, a) = p(a \mid s, \theta)$, где θ - параметры нейронной сети. Политика задает вероятность выбора действия a при нахождении в состоянии s.

Окружение в данной задаче задается графом знаний. *Граф знаний* - это ориентированный граф $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{E})$, в котором $\mathcal{E}$ - простанство сущностей, $\mathcal{R}$ - пространство отношений. Таким образом, граф состоит из трилетов (e_1, r, e_2). Из каждой сущности выходят несколько отношений, каждое из которых ведет к нескольким сущностям. В нашей реализации мы добавляем обратные ко всем ребрам, делая граф неориентированным. То есть для каждого триплета (e_1, r, e_2) добавляем триплет $(e_2, r^{-1}e_1)$. Это не меняет задачи, но дает агенту больше возможностей для перемещения по графу.

Состояния

В каждый момент времени состояние агента состоит из кортежа $s_t = ((e_s, r_q), h_t)$, где h_t - история эпизода в момент времени t, хранящая весь пройденный путь (посещенные сущности и отношения). Таким образом, каждое

состояние содержит глобальную информацию о запросе (e_s, r_q) и локальную информацию к расположении агента в графе h_t.

Действия

Набор возможных действий в момент времени t $A_t \in \mathcal{A}$ состоит из всех рёбер, исходящих из e_{t-1}. Чтобы дать возможость агенту закончить поиск, мы также добавляем специальное действие "СТОП", при выборе которого эпизод завершается. Более формально:

$$A_t = \{(r', e') \mid (e_{t-1}, r', e') \in \mathcal{G}\} \cup (\text{“СТОП”}, e_{t-1}) \tag{1}$$

Вместо действия "СТОП" мы могли добавить пустое действие, оставляющее агента в той же сущности, однако мы заметили, что такой подход уменьшает точность агента и увеличивает время эпизода из-за невозможности прервать его раньше заданного числа шагов.

Переходы

Наше окружение постоено таким образом, что случайность при переходах отсутствует. То есть $P(e' \mid s_t, (r', e')) = 1$.

Функция награды

По умолчанию агент получает награду 1, если при выборе действия "СТОП" он находится в сущности e_o и награду 0 в любом другом случае.

$$R(s_t \mid e_o) = \mathbb{1}\big\{(r_t = \text{“СТОП”}) \;\&\; (e_t = e_o)\big\} \tag{2}$$

3.2 Политика

Политика задает правила переходов из одного состояние в другое. Она принимает в себя запрос и историю переходов (предыдущих действий) и возвращает вероятностное распределение на возможные действия. Так как гра-

фы знаний обычно хранят в себе десятки тысяч сущностей, мы не можем работать с ними напрямую, для работы с ними нужно перевести их в векторное представление. Мы переводим каждую сущность и отношение в векторы одной размерности $\mathbf{e} \in \mathbb{R}^d$ и $\mathbf{r} \in \mathbb{R}^d$. Тогда каждое действие представляется в виде конкатенации отношения и сущности $\mathbf{a} = [\mathbf{e}, \mathbf{r}] \in \mathbb{R}^{2d}$.

История $h_t = (e_s, r_1, e_1, \ldots, r_t, e_t)$ состоит из набора последовательных действий и может быть записана в виде $h_t = (e_s, a_1, \ldots, a_t)$. Для удобства векторизации мы не включаем стартовую сущность e_s в историю. Тогда, история в векторизованом виде имеет форму матрицы $\mathbf{h}_t = [\mathbf{a}_1, \ldots, \mathbf{a}_t] \in \mathbb{R}^{2d \times t}$.

Политика принимает в себя состояние: запрос (e_s, r_q) и историю h_t. Мы заметили, что каждое состояние можно представить в виде последовательности действий. Тогда итоговоя задача становится обычной классификацией последовательности.

Так как для достижения единственно верной сущности необходимо очень грамотно выбирать действия на протяжении всего пути, модель политики должна быть достаточно сложной и хорошо настроенной. Мы решили, что отличным выбором будет модель трансформера, хорошо зарекомендовавшая себя в задачах обработки естественного языка.

Пространство возможных действий A_t переводится в векторное представление путем конкатенации векторных представлений всех действий в A_t: $\mathbf{A}_t \in \mathbb{R}^{|A_t| \times 2d}$.

Тогда модель политики задается как:

$$\pi_\theta(a_t \mid s_t) = \sigma(\mathbf{A}_t \times \text{TRANSFORMER}([(\mathbf{r}_q, \mathbf{e}_s), \mathbf{h}_t])), \tag{3}$$

где σ - оператор softmax.

3.3 Обучение

Обучение модели политики происходит с помощью максимизации ожидаемой награды за все запросы:

$$J(\theta) = \mathbb{E}_{(e_s, r_q, e_o) \in \mathcal{G}} \Big[\mathbb{E}_{a_1, \ldots, a_T \sim \pi_\theta} R(s_t \mid e_o) \Big] \tag{4}$$

Мы оптимизируем ожидаемую награду с помощью алгоритма REINFORCE. Он итерируется по всем триплетам (e_s, r_q, e_o) в графе знаний и на каждой итерации обновляет веса с помощью стохастического градиента:

$$\begin{aligned} \nabla_\theta J(\theta) = & \sum_t \sum_{a \in A_t} R(s_t \mid e_o) \pi\left(a \mid s_t; \theta\right) \nabla_\theta \log \pi\left(a \mid s_t; \theta\right) \approx \\ \approx & \nabla_\theta \sum_t R(s_t \mid e_o) \log \pi\left(a_t \mid s_t; \theta\right) \end{aligned} \tag{5}$$

На практике вместо обычных наград подставляются дисконтированные награды G_t.

$$G(s_t \mid e_0) = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k} \mid e_0) \tag{6}$$

Получаем:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_t G(s_t \mid e_o) \log \pi\left(a_t \mid s_t; \theta\right) \tag{7}$$

Выбор такого алгоритма обосновывается тем, что с помощью него мы можем напрямую оптимизировать политику. При этом нам не требуется обучать модель для оценки награды в каждом состоянии, как подходах в Q-learning, DQN [11], MCTS и других. Задача оценки награды в каждом состоянии всегда гораздо сложнее задачи выбора действия, поэтому, используя REINFORCE, мы облегчаем задачу агенту.

Мы также не использовали вариант алгоритма со смещением награды по двум причинам. Во-первых, такие типы алгоритмов борятся с дисперсией значений наград, которая мешает обучению. В нашем случае дисперсия очень

низкая и причин для ее уменьшения нет. Во-вторых, все подобные алгоритмы требуют нахождения оценки итоговой награды в каждом из состояний, но это слишком сложная задача по описанным выше причинам.

Для подтверждения данных гипотез мы реализовали алгоритмы REINFORCE-baseline и A2C [8], а также опробовали различные нормализации наград. Подробнее об этом секции Эксперименты.

3.4 Модификации функции награды

3.4.1 Добавление верных сущностей

Особенность алгоритма REINFORCE заключается в том, что для обновления весов ему необходимо сыграть эпизод от начала до конца. Как следует из определения функции награды 2, агент получает награду 1, только когда останавливается в верной сущности. Из-за того, что награды разреженные и агент получает ненулевую награду редко, такое обучение занимает очень много времени.

Мы изменили награду таким образом, что агент получает +1, если при завершении поиска существует переход (e_s, r_q, e_t).

$$R(s_t \mid e_o) = \mathbb{1}\big\{(r_t = \text{“СТОП”}) \;\&\; (e_s, r_q, e_t) \in \mathcal{G}\big\} \tag{8}$$

Данное изменение оправдано, так как с точки зрения имеющейся у агента информации все сущности, связанные с e_s отношением r_q равновероятны, поэтому справедливо считать все пути, которые привели к одной из них, верными.

3.4.2 Штраф за длину пути

Так как мы ввели обратные ребра, агент получает больше возможностей для передвижений по графу и, следовательно, больше различных путей. При этом с точки зрения награды нет никакой разницы между длинным и ко-

ротким путем, несмотря на то, что короткий путь предпочтительнее из-за меньшего объема потраченного на него времени, а также из-за того, что с увеличением длинны пути количество различных путей растет экспоненциально. Это значит, что вероятность попасть в верную сущность значительно снижается.

Для того, чтобы способствовать уменьшению длины пути, мы ввели штраф

$$\mathcal{L}_{len\_penalty}(T) = \mu \times \arctan(\nu T), \tag{9}$$

где μ и ν – гиперпараметры.

Данный штраф за счет формы функции arctan мотивирует агента выбирать короткие пути, но вместе с этим не так существенно снижает награду за длинные пути. Таким образом, если длина двух путей мала, агент будет выбирать меньший, однако, если длина пути велика, агент все равно будет получать положительную награду, так как остановка в правильной сущности приоритетнее длины пути до этой сущности.

3.4.3 Награда за неверные сущности

В соответствии с уравнением 8 агент получает награду 0 за остановку в неверной сущности независимо от того, насколько эта сущность похожа на верный ответ. Отличное решение к этой проблеме было представлено в статье [16]. Мы можем обучить модель, предсказывающую верную сущность, опираясь только на векторные представления e_s и r_q. Формально, модель будет переводить запрос (e_s, r_q) в векторное пространство и оценивать правдоподобие фактов (e_s, r_q, e) для любого $e \in \mathcal{E}$ с помощью некоторой функции $f(e_s, r_q, e)$. Например, возьмем в качестве такой модели ComplEx [2] или ConvE [3].

Изменим функцию награды следующим образом:

$$\begin{aligned} R_b(s_t \mid e_o) &= \mathbb{1}\big\{(r_t = \text{“СТОП”}) \;\&\; (e_s, r_q, e_t) \in \mathcal{G}\big\} \\ R(s_t \mid e_o) &= R_b(s_t \mid e_o) + \alpha\big(1 - R_b(s_t \mid e_o)\big) f(e_s, r_q, e_t) \end{aligned} \tag{10}$$

Таким образом, если агент останавливается в правильной сущности, он по-прежнему получает награду 1. Однако, если он останавливается в неправильной, то получает вероятность этой сущности быть правильным ответом, основываясь на предсказании обученной модели. Такой подход дает значительное увеличение качества, так как помогает лучше корректировать поведение агента.

3.5 Результаты

В этом разделе приведены результаты работы нашего алгоритма на следующих наборах данных: WN18RR, FB15k-237 и Kinship. WN18RR является подмножеством WN18 с удаленными обратными отношениями. WN18 состоит из лексических связей между английскими словами из WordNet [10]. FB15k-237 является подмножеством FB15k с удаленными обратными отношениями. FB15k построен на основе Freebase [4], огромного набора общечеловеческих фактов. Kinship – набор данных с родственными связями между людьми из двух семей. Информация о характеристиках этих наборов данных представлена в Таблице 1.

Для каждого из наборов данных мы сравниваем полученные значения метрик с другими алгоритмами. Для сравнения мы выбрали две модели на основе векторных представлений: ComplEx [2] и ConvE [3], модель на основе логических правил NeuralLP [18] и модель, использующую обучение с подкреплением Multi-Hop [16].

Набор данных	#Сущностей	#Отношений	#Фактов	Средняя степень
WN18RR	40,945	11	86,835	2.19
FB15k-237	14,505	237	272,115	19.74
Kinship	104	25	8,544	85.15

Таблица 1: Наборы данных, используемые в исследовании.

Перед рассмотрением полученных результатов опишем процесс оценки качества.

3.5.1 Алгоритм оценки качества

Для каждого триплета (e_s, r_q, e_o) из тестового набора данных мы составляем запрос (e_s, r_q), для которого получаем список сущностей $E_o = \{e_1, \ldots, e_l\}$, отсортированный по убыванию уверенности модели в каждой из них. Индекс r_{e_o} сущности e_o в этом списке называется ее рангом. Для получения значения уверенности мы используем алгоритм beam search. Начиная из сущности e_s, мы предсказываем вероятности всех возможных действий и выбираем из них k наиболее вероятных, где k - размер луча. После чего для всех полученных сущностей находим вероятности возможных действий и снова выбираем k наиболее вероятных. Данную процедуру мы повторяем заданное количество раз. При посещении сущности мы считаем логарифм вероятности остановки в ней по формуле:

$$\log\big(P(e_t \mid \theta)\big) = \log\big(P_{stop}(e_t \mid \theta)\big) + \sum_{i=1}^{t} \log\big(P(a_i \mid \theta)\big), \tag{11}$$

где a_i - i-ое действие, на пути от e_s до e_t. При повторном посещении сущности мы обновляем логарифм вероятности, если его новое значение больше текущего.

Полученный список логарифмов вероятностей и есть список уверенностей модели в каждой из сущностей. В конце из отсортированного списка сущностей мы удаляем все, являющиеся правильными ответами, кроме e_o.

После этого мы считаем значения ранговых метрик HITS@k (доля примеров, в которых $r_{e_o} \leq k$) и MRR (среднее значение $1/r_{e_o}$ по всем примерам).

3.5.2 Гиперпараметры

Наша модель обучалась с одинаковым набором гиперпараметров для всех датасетов и одинаковой архитектурой.

Архитектура

Модель представляет собой стандартную версию трансформера с 3-мя

слоями внимания и 8-ю головами у каждого слоя. Внутренняя размерность векторов всех слоев равна 128, в слое прямого распространения, идущем после каждого слоя внимания размерность повышается до 256. Так же в этом слое используется swish функция активации [12]. Мы добавляем dropout со значением 0.1 ко всем векторным представлениям, а также после слоя внимания и после слоя прямого распространения. Все веса модели инициализируются при помощи Xavier инициализации [5].

Обучение

Агент обучается с помощью метода REINFORCE в течение 5000 эпох с отпимизатором Adam [7] и скоростью обучения 0.0001, значение batch size выставлено на 512. Для всех сущностей и отношений размерность векторных представлений равна 128. Максимальное число шагов в одном эпизоде – 20. Для получения награды за неверные сущности мы используем модель ComplEx и применяем сигмоиду к ее выходам. Полученное значение умножается на коэффициент 0.3. В процессе обучения для набора данных WN18RR мы добавляем регуляризацию с помощью энтропии с коэффициентом регуляризации 0.03.

При оценке качества мы выставляем значение размера луча равным 128, число шагов – 10.

Для подбора гиперпараметров использовался метод подбора по сетке.

Алгоритм подсчета значений HITS@1, HITS@10 и MRR для нашего подхода описан в разделе 3.5.1. Для остальных подходов значения метрик скопированы из соответствующих статей.

Из результатов в Таблице 2 видно, что предложенный нами алгоритм превзошел остальные только на наборе данных WN18RR. Такое качество на WN18RR объясняется устройством набора данных. Его набор тестовых триплетов на 34% состоит из отношений “однокоренные”. Это симметричное отношение, однако, так как триплеты в тренировочной выборке и тестовой

Алгоритм	Kinship			FB15k-237			WN18RR		
	@1	@10	MRR	@1	@10	MRR	@1	@10	MRR
ComplEx	**81.8**	98.1	**88.4**	32.8	61.6	42.5	41.8	48.0	43.7
ConvE	79.7	98.1	87.1	**34.1**	**62.2**	**43.5**	40.3	54.0	44.9
NeuralLP	47.5	91.2	61.9	16.6	34.8	22.7	37.6	65.7	46.3
Multi-Hop	81.1	**98.2**	87.8	32.9	56.4	40.7	43.7	54.2	47.2
Ours(Test)	33.5	83.0	51.0	27.9	44.2	32.6	**50.7**	**67.3**	**56.5**
Ours(Train)	90.5	95.0	92.4	54.5	80.7	65.6	86.3	91.5	89.9

Таблица 2: Результаты алгоритмов на наборах данных Kinship, FB15k-237 и WN18RR.

не пересекаются, то в тренировочной выборке оказывается триплет (e_1, "однокоренные", e_2), а в тестовой (e_2, "однокоренные", e_1). Добавляя обратные отношения при построении графа знаний по тренировочной выборке, мы также восполняем все отсутствующие триплеты с отношением "однокоренные". Поэтому, когда агент получает такой триплет из тестовой выборки, он сразу же переходит по обратному отношению в нужную сущность и останавливается. Таким образом, большая часть верных ответов (34% из 50.7%) получается благодаря утечке.

Также видно, что на остальных наборах данных агент смог хорошо выучить переходы в графе знаний, однако он плохо справился с определением значений сущностей. Мы уверены, что эту проблему можно исправить, настроив гиперпараметры агента более тщательно. В данной работе мы больше были сосредоточены на исследование обучения с подкреплением как подхода к решению задачи, поэтому оставили настройку на будущее.

На наборах данных Kinship и FB15k-237 лучшее качество показывают модели, обученные на векторных представлениях сущностей и отношений. Объяснить это можно тем, что такие модели сопоставляют все сущности векторам из одного пространства. В этом пространстве они неявно строят связи между всеми сущностями графа. Методы на основе анализа пути в свою очередь при построении траектории видят лишь небольшой сегмент графа, упуская часть глобальной информации.

3.6 Эксперименты

Для экспериментов мы использовали набор данных WN18RR [3].

Для каждого из проверяемых подходов мы тщательно подбирали гиперпараметры и обучали модель до сходимости.

3.6.1 Уменьшение дисперсии наград

Часто при использовании алгоритма REINFORCE возникает проблема высокой дисперсии наград. В нашей задаче дисперсия не бывает высокой, так как все награды строго лежат на отрезке [0, 1], однако мы все равно исследовали ряд подходов уменьшающих ее и ускоряющих сходимость.

Нормализация награды

Нормализация награды осуществляется с помощью вычитания среднего из дисконтированных наград и деления на дисперсию. В наших экспериментах нормализация привела к тому, что агент перестал учиться. Такое поведение можно объяснить следующим образом. Из-за того, что ненулевую награду агент получает только в конце эпизода, в нашей задаче дисконтированные награды монотонно неубывают. Это значит, что при нормализации награда, соответствующая первому действию всегда будет не больше нуля. Из-за этого любое первое действие агент считает неверным, и единственное первое действие, при котором он гарантированно получает максимальную (нулевую) награду - это "СТОП". Таким образом, агент учится останавливаться на первой итерации и алгоритм перестает работать.

Вычитание функции

Для уменьшения дисперсии наград также используется вычитание функции. Этот метод заменяет градиент ожидаемой награды из уравнения 7 на

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_t \left[G(s_t \mid e_o) - b(s_t)\right] \log \pi \left(a_t \mid s_t; \theta\right), \tag{12}$$

где $b(s_t)$ - любая функция, зависящая только от состояния.

Такой подход, в отличие от нормализации, не заставляет агента сразу завершать эпизод и может помочь при обучении. Лучше всего в качестве $b(s_t)$ выбирать функцию ожидаемой награды для данного состояния $V(s_t)$, параметры которой подбираются во время тренировки параллельно с параметрами политики. Так получается алгоритм A2C. В наших экспериментах из-за сложности в оптимизации функции $V(s_t)$ агент начал вести себя нестабильно и стал гораздо хуже обучаться.

Из-за полученных результатов мы не стали добавлять описанные подходы в итоговый алгоритм и оставили оптимизируемую функцию ожидаемой награды без изменений.

3.6.2 Предобучение модели

В задаче ответов на вопросы на основе графа знаний функция награды устроена таким образом, что агент получает ненулевую награду очень редко. Из-за этого на ранних стадиях тренировки он обучается очень медленно. Эту проблему можно решить, обучив агента выбирать правильные действия заранее с помощью обучения с учителем.

Для предобучения мы составляем набор данных из случайных действий, приводящих к верной сущности, включающий 30000 путей. Из каждого найденного пути мы убираем все циклы. Для набора данных WN18RR средняя длина кратчайшего пути между сущностями без использования прямого отношения равна 2.3. В сгенерированном нами наборе данных средняя длина пути равна 3.8.

Мы обучаем агента оптимизировать функцию потерь LabelSmoothingCrossEntropy. Для обучения используется оптимизатор Adam со скоростью обучения 0.001 и batch size 128.

В наших экспериментах мы заметили, что при использовании предобученного подобным образом агента итоговое качество становится хуже, несмотря на то, что на первых итерациях обучения предобученный агент обгоняет случайного. Мы объясняем этот эффект тем, что агент учится совершать

неоптимальные действия, что следует из сравнения средних длин путей. Поэтому при дальнейшем обучении выученная политика мешает ему обучиться совершать лучшие действия.

4 Заключение

В данной работе мы предложили подход к решению задачи ответов на вопросы на основе графов знаний, включающий в себя использование трансформера в качестве агента, а также ряд модификаций функции награды для ускорения обучения. В результате мы получили алгоритм, превосходящий все предыдущие алгоритмы на датасете WN18RR. Детальных анализ поведения агента показал, что он отлично запоминает переходы в графе, однако он не способен находить глобальные связи между сущностями на должном уровне. Из-за этого алгоритм не смог обойти решения на основе векторных представлений на двух оставшихся наборах данных Kinship и FB15k-237.

Не смотря на это, мы уверены, что наш метод имеет огромный потенциал и мы продолжим работать над ним. В качестве дальнейшей работы мы хотим улучшить модель политики, а также исследовать подходы к выключению некоторых действий из возможных на каждом шаге агента для избежания запоминания графа знаний.

Список литературы

1. Attention is all you need / A. Vaswani [и др.] // arXiv preprint arXiv:1706.03762. — 2017.
2. Complex Embeddings for Simple Link Prediction / T. Trouillon [и др.] //. — 10.2016.
3. Convolutional 2D Knowledge Graph Embeddings / T. Dettmers [и др.] // CoRR. — 2017. — т. abs/1707.01476. — arXiv: 1707.01476.
4. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge / K. Bollacker [и др.] // Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. — Vancouver, Canada : Association for Computing Machinery, 2008. — с. 1247—1250. — (SIGMOD '08). — ISBN 9781605581026.
5. *Glorot X.*, *Bengio Y.* Understanding the difficulty of training deep feedforward neural networks // Journal of Machine Learning Research - Proceedings Track. — 2010. — янв. — т. 9. — с. 249—256.
6. Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning / R. Das [и др.] // CoRR. — 2017. — т. abs/1711.05851. — arXiv: 1711.05851.
7. *Kingma D. P.*, *Ba. J. L.* ADAM: A method for stochastic optimization. // arxiv.org/1412.6980.pdf. — 2014.
8. *Konda V.*, *Tsitsiklis J.* Actor-Critic Algorithms // Society for Industrial and Applied Mathematics. — 2001. — апр. — т. 42.
9. M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search / Y. Shen [и др.] // Advances in Neural Information Processing Systems. т. 31 / под ред. S. Bengio [и др.]. — Curran Associates, Inc., 2018.

10. *Miller G. A.* WordNet: A Lexical Database for English // Commun. ACM. — New York, NY, USA, 1995. — нояб. — т. 38, № 11. — с. 39—41. — ISSN 0001-0782.

11. Playing Atari with Deep Reinforcement Learning / V. Mnih [и др.] // CoRR. — 2013. — т. abs/1312.5602. — arXiv: 1312.5602.

12. *Ramachandran P., Zoph B., Le Q. V.* Searching for Activation Functions // CoRR. — 2017. — т. abs/1710.05941. — arXiv: 1710.05941.

13. Representing Text for Joint Embedding of Text and Knowledge Bases / K. Toutanova [и др.] // Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. — Lisbon, Portugal : Association for Computational Linguistics, 09.2015. — с. 1499—1509.

14. *Rocktäschel T., Riedel S.* End-to-end Differentiable Proving // CoRR. — 2017. — т. abs/1705.11040. — arXiv: 1705.11040.

15. *Williams R. J.* Simple statistical gradient-following algorithms for connectionist reinforcement learning // Reinforcement Learning, pages 5–32. Springe. — 1992.

16. *X. Lin R. S., Xiong C.* Multi-hop knowledge graph reasoning with reward shaping // arXiv:1808.10568. — 2018.

17. *Xiong W., Hoang T., Wang W. Y.* DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning // Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. — Copenhagen, Denmark : Association for Computational Linguistics, 09.2017. — с. 564—573.

18. *Yang Y., Song L.* Learn to Explain Efficiently via Neural Logic Inductive Learning // International Conference on Learning Representations. — 2020.