Введение в DL для работы с кодом

Сергей Лоптев

Содержание

Часть Сергея:

- Задачи и метрики
- Как готовить датасеты
- Abstract Syntax Tree

Часть Захара:

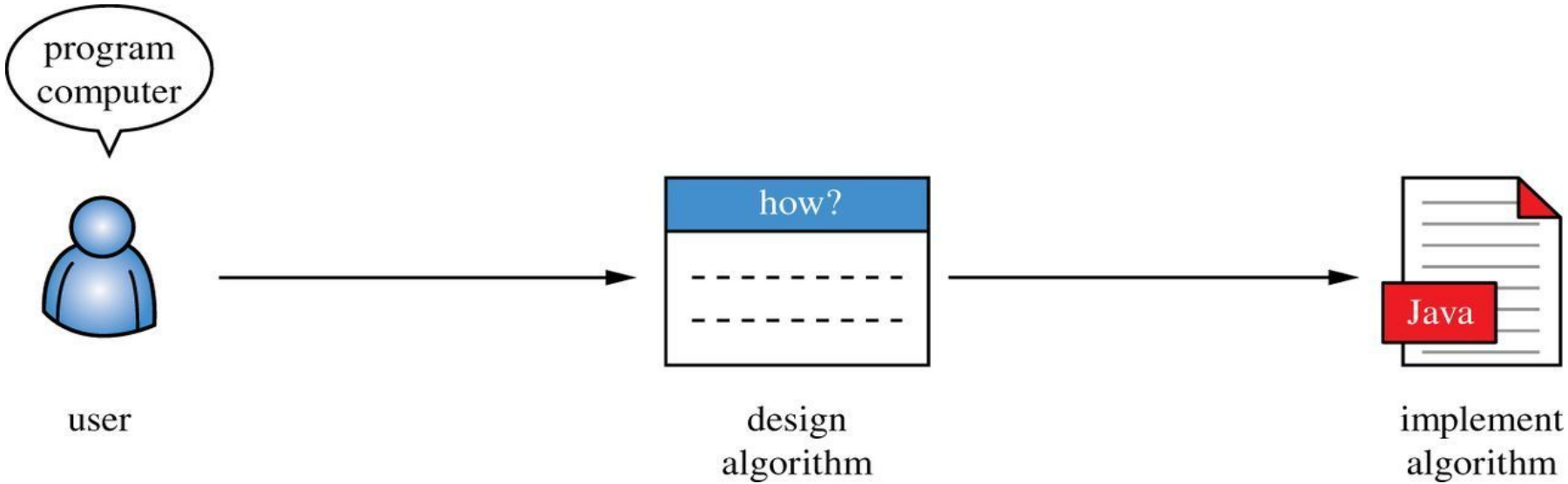
- BERT и GPT
- Codex
- GraphCodeBert

Задачи

- Program synthesis
- Code autocompletion
- Code search
- Clone detection
- Bug fixing

Program synthesis

- Общая задача: по некоторой спецификации (обычно представленной естественным языком) синтезировать программу, удовлетворяющую ей.
- Метрики: BLEU



BLEU

- Постановка задачи: есть кандидат, представленный машинным переводом, и референсы (переводы) людей, считающиеся верными. Нужно определить, насколько наш кандидат "похож" на переводы людей.

BLEU

- Кандидат: "the the the the the the the"
- Референс 1: "the cat is on the mat"
- Референс 2: "there is a cat on the mat"

BLEU

- Кандидат: "the the the the the the the"
- Референс 1: "the cat is on the mat"
- Референс 2: "there is a cat on the mat"
- Precision для униграмм?

$$Precision = \frac{\sum_{i=1}^{n} \left[\bigvee_{j=1}^{m} cand_i \in ref_j \right]}{n} = \frac{7}{7} = 1$$

BLEU

- Кандидат: "the the the the the the the"
- Референс 1: "the cat is on the mat"
- Референс 2: "there is a cat on the mat"
- Precision для униграмм?

$$Precision = \frac{\sum_{i=1}^{n} \left[\bigvee_{j=1}^{m} cand_i \in ref_j \right]}{n} = \frac{7}{7} = 1$$

- Надо чинить. Введём вспомогательные величины:

$$Count(unigram) = \sum_{i=1}^{n} [cand_i = unigram] \qquad Count_{clip}(unigram) = \min(Count(unigram), \max_{j=1..m} Count_{ref_j}(unigram))$$

BLEU

- Кандидат: "the the the the the the the"
- Референс 1: "the cat is on the mat"
- Референс 2: "there is a cat on the mat"
- Modified Precision для униграмм:

$$p_1 = \frac{\sum_{unigram \in C} Count_{clip}(unigram)}{\sum_{unigram \in C} Count(unigram)} = 2/7$$

BLEU

- Кандидат: "the the the the the the the"
- Референс 1: "the cat is on the mat"
- Референс 2: "there is a cat on the mat"
- Modified Precision для униграмм:

$$p_1 = \frac{\sum_{unigram \in C} Count_{clip}(unigram)}{\sum_{unigram \in C} Count(unigram)} = 2/7$$

- Общая формула для n-грамм:

$$p_n = \frac{\sum_{ngram \in C} Count_{clip}(ngram)}{\sum_{ngram \in C} Count(ngram)}$$

BLEU

- Идея: будем использовать не только униграммы, но и n-граммы
- Для объединения используем экспоненту от взвешенного среднего логарифмов p_n:

$$\exp\left(\sum_{i=1}^{n} w_i \cdot \log p_i\right)$$

Figure 1: Distinguishing Human from Machine

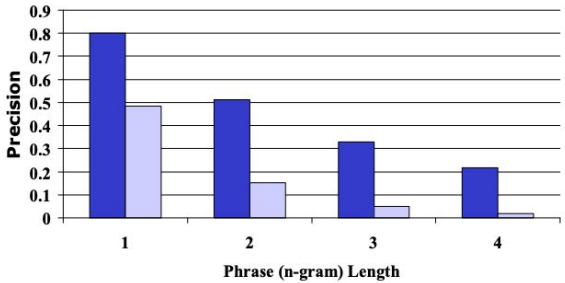
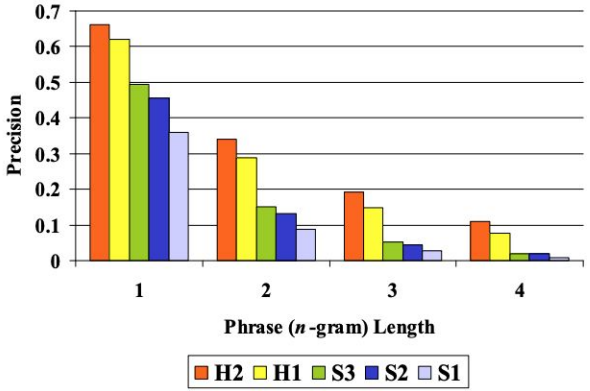


Figure 2: Machine and Human Translations



BLEU

- Идея: будем использовать не только униграммы, но и n-граммы
- Для объединения используем экспоненту от взвешенного среднего логарифмов p_n:

$$\exp\left(\sum_{i=1}^{n} w_i \cdot \log p_i\right)$$

- В оригинальной метрике BLEU:

$$\exp\left(\sum_{i=1}^{4} w_i \cdot \log p_i\right), \quad w_i = \frac{1}{4}$$

BLEU

- Осталась проблема с короткими кандидатами, введём штраф.
- Пусть c -- сумма длин слов в кандидате
- Пусть r -- сумма длин слов в ближайшем по длине референсе

BLEU

- Осталась проблема с короткими кандидатами, введём штраф.
- Пусть c -- сумма длин слов в кандидате
- Пусть r -- сумма длин слов в ближайшем по длине референсе
- Brevity penalty:

$$BP = \begin{cases} 1, & c \geq r \\ \exp(1 - r/c), & c < r \end{cases}$$

BLEU

- Итоговая формула:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

BLEU

- Итоговая формула:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

- И её логарифм:

$$\log BLEU = \min(1 - r/c,\ 0) + \sum_{n=1}^{N} w_n \log p_n$$

BLEU

- Evaluation:

Table 1: BLEU on 500 sentences

S1	S2	S3	H1	H2
0.0527	0.0829	0.0930	0.1934	0.2571

Table 2: Paired t-statistics on 20 blocks

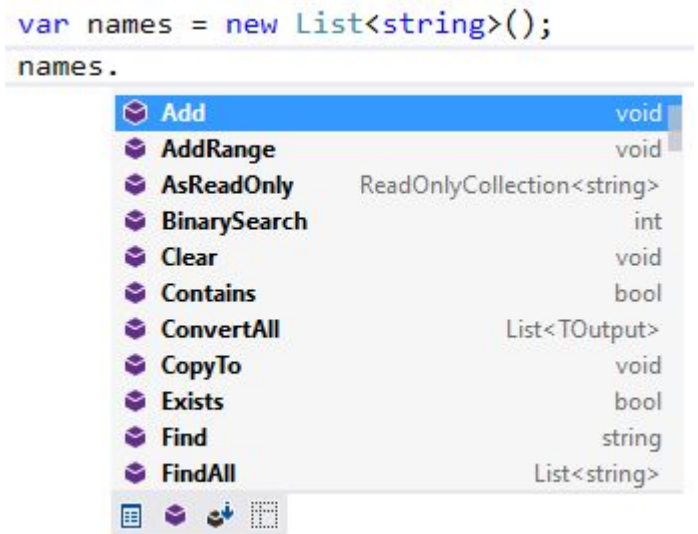
	S1	S2	S3	H1	H2
Mean	0.051	0.081	0.090	0.192	0.256
StdDev	0.017	0.025	0.020	0.030	0.039
t	—	6	3.4	24	11

Задачи

- Program synthesis
- Code autocompletion
- Code search
- Clone detection
- Bug fixing

Code autocompletion

- Общая задача: по контексту и началу введённого токена предсказать окончание этого токена.
- Метрики: F-measure, AUC-ROC

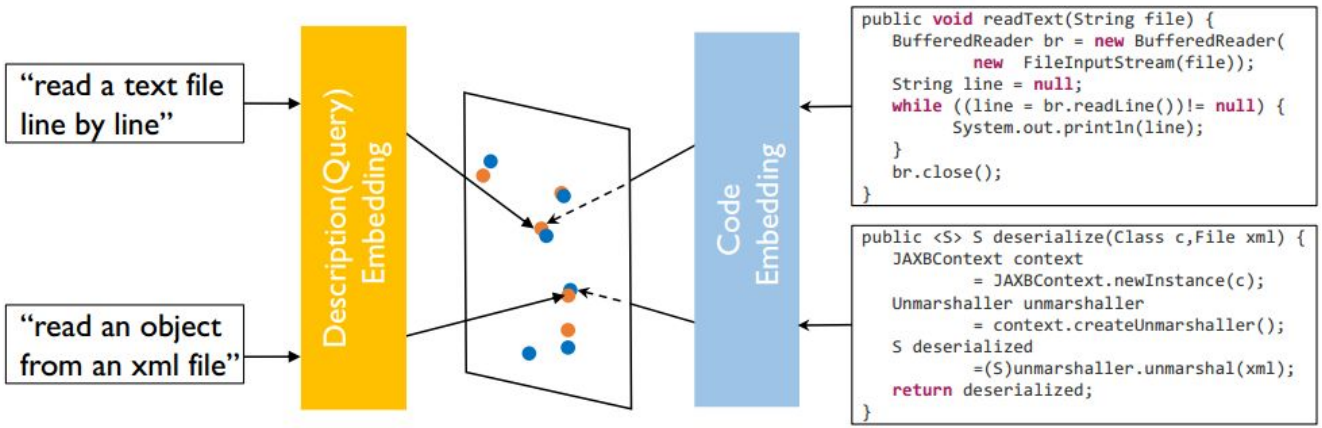


Задачи

- Program synthesis
- Code autocompletion
- Code search
- Clone detection
- Bug fixing

Code search

- Общая задача: по интенту разработчика, выраженному в естественном языке, и имея большую кодовую базу, найти в этой кодовой базе куски кода, соответствующие интенту
- Метрики: DCG, MRR



DCG

- Постановка задачи: мы выдали по запросу p документов, для которых затем как-то оценили их релевантность. Теперь хотим оценить какой-то общий score выдачи, причём мы бы хотели, чтобы более релевантные документы были в ней выше.

DCG

- Обозначим релевантность i-го документа выдачи за rel_i.

DCG

- Обозначим релевантность i-го документа выдачи за rel_i.
- Формула DCG@p:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

DCG

- Обозначим релевантность i-го документа выдачи за rel_i.
- Формула DCG@p:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

- Альтернативная формула DCG@p:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Normalized DCG

- Введём ideal DCG (с отсортированными по убыванию релевантности документами):

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Normalized DCG

- Введём ideal DCG (с отсортированными по убыванию релевантности документами):

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

- Тогда nDCG вычисляется так:

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

MRR

- Постановка задачи: мы выдали по каждому из Q запросов по n документов, для которых оценили их релевантность бинарно. Теперь хотим оценить какой-то общий score всей модели, причём мы бы хотели, чтобы релевантные документы были в выдаче как можно выше.

MRR

- У нас есть Q запросов и по n документов на каждый
- Обозначим за rank_i позицию первого релевантного документа для i-го запроса (нумерация с 1)

MRR

- У нас есть Q запросов и по n документов на каждый
- Обозначим за rank_i позицию первого релевантного документа для i-го запроса (нумерация с 1)
- Формула для MRR:

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i}$$

MRR

- У нас есть Q запросов и по n документов на каждый
- Обозначим за rank_i позицию первого релевантного документа для i-го запроса (нумерация с 1)
- Формула для MRR:

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i}$$

- Если для запроса не нашлось релевантного документа, то соответствующее слагаемое устанавливаем равным нулю

Задачи

- Program synthesis
- Code autocompletion
- Code search
- Clone detection
- Bug fixing

Clone detection

- Общая задача: имея большую кодовую базу, находить куски кода, дублирующие друг друга
- Используемые метрики: F-measure, AUC-ROC

Задачи

- Program synthesis
- Code autocompletion
- Code search
- Clone detection
- Bug fixing

Bug fixing

- Внутри направления бывают разные задачи: обнаружить возможный баг в программе, предложить варианты исправления бага
- Используемые метрики: F-measure, AUC-ROC

Подготовка данных для тестирования и обучения

- Разберём на примере датасета CodeSearchNet.

CodeSearchNet

- CodeSearchNet -- датасет, подготовленный для соревнования CodeSearchNetChallenge.
- Содержит 6 млн. функций на 6 языках (Go, Java, JavaScript, Python, PHP, Ruby)
- 2 млн. функций из них задокументированы, то есть имеют объект для подачи в модель
- Используемая метрика -- nDCG.

	Number of Functions	
	w/ documentation	All
Go	347 789	726 768
Java	542 991	1 569 889
JavaScript	157 988	1 857 835
PHP	717 313	977 821
Python	503 502	1 156 085
Ruby	57 393	164 048
All	2 326 976	6 452 446

Table 1: Dataset Size Statistics

CodeSearchNet

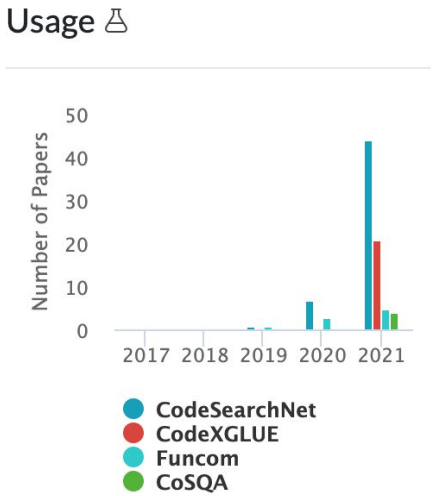
- Опишем процесс построения датасета
- В качестве сырых данных взяли все no-fork проекты с GitHub
- Удостоверились с помощью libraries.io, что каждый проект хоть где-то используется
- Отсортировали все функции по "популярности" -- звёздам и форкам на GitHub
- Удалили проекты, лицензия которых не разрешает использовать код в подобных целях
- Токенизировали код утилитой Tree Sitter от GitHub
- Извлекли и токенизировали документацию регулярным выражением

CodeSearchNet

- Теперь отфильтруем датасет
- У нас есть 2 категории функций -- задокументированные и нет. Первые обозначим общим набором пар (c_i, d_i), где c_i -- функция, d_i -- документация
- Оставим в каждом d_i только первый абзац
- Выкинем все пары, где d_i состоит не более чем из трёх токенов
- Выкинем все пары, где c_i состоит не более чем из трёх строк
- Выкинем все функции, где название содержит слово test или является стандартным (__len__, toString)
- С помощью clone detection и других утилит находим дубликаты функций и оставляем только по одному

CodeSearchNet

- Датасет получился вполне успешным



CodeSearchNet

- Но не без проблем:
- Документация -- это всё-таки не запрос пользователя
- Неизвестна точность описания функций документациями
- Известно, что некоторые документации написаны не на английском языке

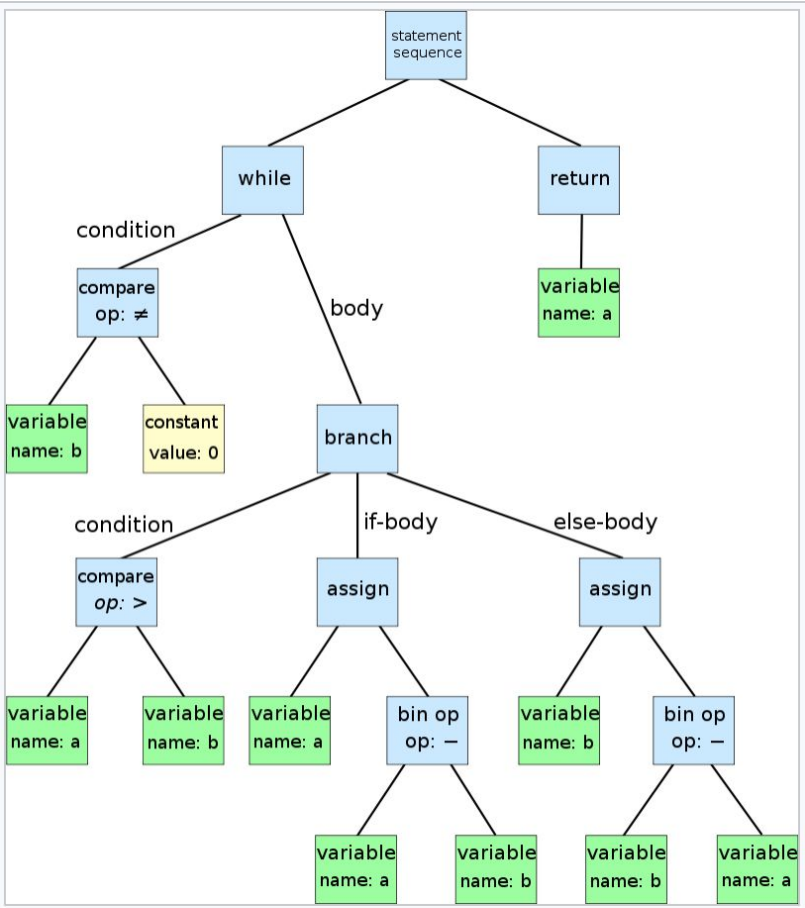
Abstract Syntax Tree

- AST -- дерево, хранящее в себе достаточно абстрактную (не касающуюся синтаксиса) информацию об исходном коде программы

Abstract Syntax Tree

- Пример:

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```



An abstract syntax tree for the following code for the Euclidean algorithm:

Abstract Syntax Tree

- Зачем это надо
- Позволяет компилятору проверить "грамматическую" верность написанного кода
- Может хранить в вершинах дополнительную информацию
- Используется для кодогенерации следующих итераций кода
- Применяется в DL (пример -- code2vec)

Источники

- https://en.wikipedia.org/wiki/Program_synthesis
- https://www.youtube.com/watch?v=DejHQYAGb7Q
- https://en.wikipedia.org/wiki/BLEU
- https://www.researchgate.net/profile/Salim-Roukos/publication/2588204_BLEU_a_Method_for_Automatic_Evaluation_of_Machine_Translation/links/54dca00c0cf25b09b9126f9a/BLEU-a-Method-for-Automatic-Evaluation-of-Machine-Translation.pdf
- https://en.wikipedia.org/wiki/Discounted_cumulative_gain
- https://en.wikipedia.org/wiki/Intelligent_code_completion
- https://ru.wikipedia.org/wiki/Среднеобратный_ранг
- https://arxiv.org/pdf/1909.09436v3.pdf
- https://en.wikipedia.org/wiki/Abstract_syntax_tree