

輪講

SparseCoder: Identifier-Aware Sparse Transformer for File-Level Code Summarization

Yanlin Wang, Yanxian Huang, Daya Guo,
Hongyu Zhang and Zibin Zheng

背景:ファイルレベルのコード要約

ファイル全体で高レベルの要約を生成することが目的

ファイルレベルの要約がなければ、開発者は手作業でファイルを開いて目を通さなければならない

既存の研究ではメソッドレベルの要約に重点が置かれている

既存手法:自己アテンションメカニズム

CodeBERTやUniXcoder,CodeT5などの自然言語処理モデルで用いられている手法

計算複雑度が入力の長さに対して二次関数的に増大する

各トークンは、それぞれすべてのトークンに対して関連性が計算されるため、つまり総当たり

トークン数の制約

最大トークン数が1024であるのに対して、GitHub上のPythonソースファイルの平均トークン数は2090

→ファイルレベルのコード要約には工夫が必要

SparseCoder

3種類のマスク行列を導入することで、自己アテンションメカニズムでは総当たりしていたトークン同士の関係を疎なパターンだけに絞り込む

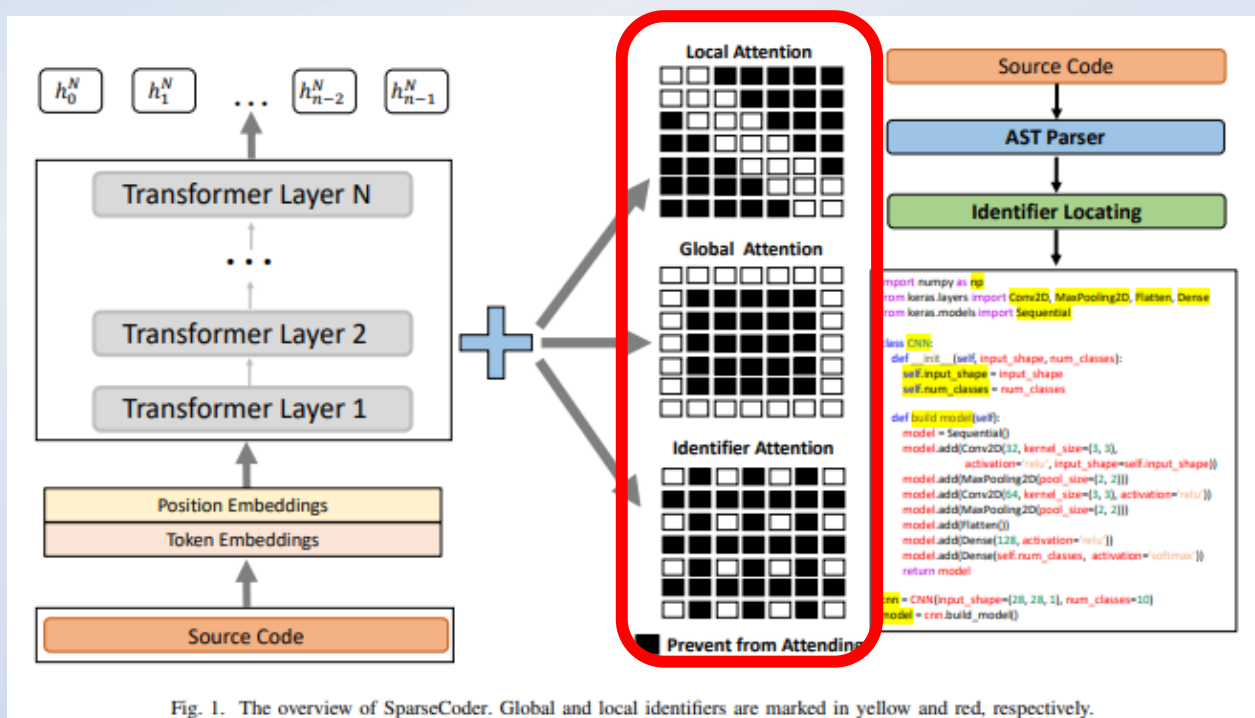


Fig. 1. The overview of SparseCoder. Global and local identifiers are marked in yellow and red, respectively.

$$M = \max(M_{ij}^{local}, M_{ij}^{global}, M_{ij}^{identifier})$$

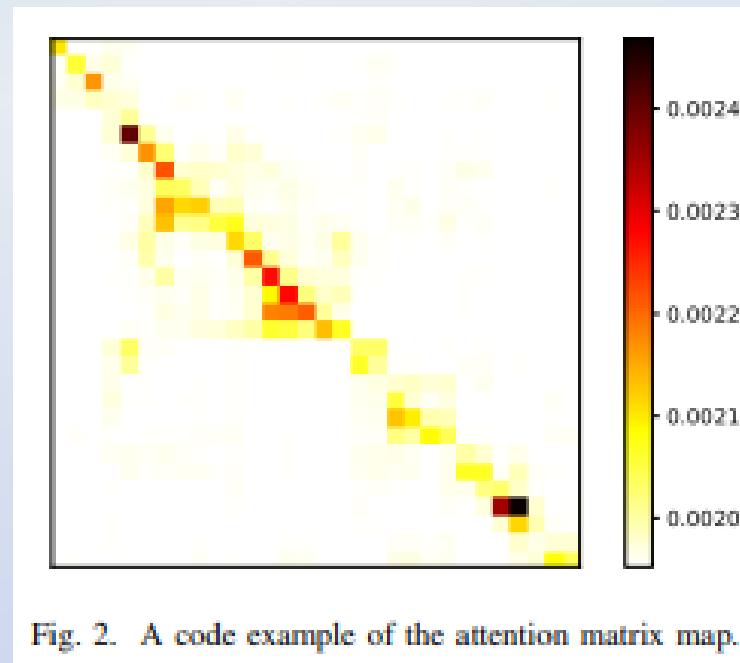
Local Attention

依存するトークンは互いに近い位置に現れやすい

サンプルコードの各トークン間のアテンションマトリクスを示すヒートマップ

対角線で濃くなっている

→近いトークン同士であるほどスコアが高い



マスク行列の式

$$M_{ij}^{local} = \begin{cases} 0 & \text{if } |i - j| \leq \lfloor \frac{w}{2} \rfloor \\ -\infty & \text{otherwise} \end{cases}$$

Global Attention

グローバルな識別子はファイルのどこからでも呼び出せる

サンプルコードで、クラスの定義はファイルの先頭に、変数の作成はファイルの最後にある

→長距離の依存関係

静的解析でグローバルな識別子を見つけ、その位置をGとする

マスク行列の式

$$M_{ij}^{global} = \begin{cases} 0 & \text{if } i \text{ or } j \in G \\ -\infty & \text{otherwise} \end{cases}$$

```
import numpy as np
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.models import Sequential

class CNN:
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes

    def build_model(self):
        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3, 3),
                        activation='relu', input_shape=self.input_shape))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(self.num_classes, activation='softmax'))
        return model

cnn = CNN(input_shape=(28, 28, 1), num_classes=10)
model = cnn.build_model()
```

Identifier Attention

大域的な識別子

Input_shapeなどはウィンドウサイズを超えるような依存関係を持ち得る

コード内すべての識別子をIとする

グローバルな識別子はすでにアテンションされているため除外する

マスク行列の式

$$M_{ij}^{identifier} = \begin{cases} 0 & \text{if } i \text{ and } j \in I - G \\ -\infty & \text{otherwise} \end{cases}$$

```
import numpy as np
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.models import Sequential

class CNN:
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes

    def build_model(self):
        model = Sequential()
        model.add(Conv2D(32, kernel_size=(3, 3),
                        activation='relu', input_shape=self.input_shape))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(self.num_classes, activation='softmax'))
        return model

cnn = CNN(input_shape=(28, 28, 1), num_classes=10)
model = cnn.build_model()
```

検証

◆データセット

FILE-CS: ファイルレベルのコード要約のためのデータセットを作成
github-code2からコードファイルを収集
コメントなどから, ファイルの要約情報を抽出

TABLE I
DATA STATISTICS ABOUT FILE-CS.

Length	Avg	25%	50%	75%
Source Code	1295.0	564	890	1563
Summary	11.0	6	8	12

◆検証内容

- A. SparseCodeとベースラインを比較する
- B. コンポーネントと解析
- C. メモリコストの分析
- D. 人間による評価
- E. ケーススタディ

SparseCodeとベースラインを比較

スパースモデル: 入力と出力の接続を制限したモデル
表上部5つは非スパースモデル, 残りはスパースモデル

BLEU: 参照文と生成文の平均n-gram精度

Rouge-L: 最長共通部分列(LCS)に基づいた精度と再現率の調和平均であるFスコア

Meteor: 参考文献からモデルがどの程度内容を捉えているかを示す指標

非スパースモデルは入力長による切り捨てにより重要な情報が失われている

SparseCoderは他のスパースモデルと比べて高いスコアを出した

TABLE II
PERFORMANCE OF DIFFERENT MODELS ON FILE-CS.

Model	BLEU	Rouge-L	Meteor
GraphCodeBERT	16.9	26.2	21.8
CodeBERT	17.4	26.7	22.2
PLBART	18.6	28.0	25.2
CodeT5	19.1	28.7	25.2
UniXcoder	19.3	29.3	25.0
BigBird	19.7	29.3	25.2
SASA	19.9	29.7	25.4
LongFormer	20.1	30.4	26.1
LongT5	18.6	25.3	24.3
Unix-LongT5*	20.6	29.3	26.3
LSG Attention	20.8	30.5	26.6
SparseCoder	21.4	32.2	27.6

*LongT5 initialized from UniXcoder.

人間による評価

SparseCoderと性能の良いベースライン(UniXcoder, LSG Attention)

FILE-CSから無作為に100個のpythonファイルを選択し、要約

正しさ, 自然さ, 情報量の側面について, 4人の開発経験3年以上のボランティアが0~4点で評価し平均スコアを算出

TABLE IV
RESULTS OF HUMAN EVALUATION (STANDARD DEVIATION IN PARENTHESES).

Model	Informativeness	Naturalness	Similarity
SparseCoder	2.085 (0.98)	3.175 (0.60)	2.353 (1.19)
LSG Attention	2.065(1.02)	3.094(0.70)	2.160(0.93)
UniXcoder	2.020(1.00)	3.075(0.70)	2.133(0.93)

SparseCoderはすべての側面でベースラインを上回った

結論

SparseCoderは3種類の疎なアテンションパターン,
Local Attention, Global Attention, Identifier Attention
を導入し, ファイルレベルのコード要約において先行研究を上回ることが示された

所感

ファイルレベルの要約はコードドキュメントで済みそうだが, それを書くことの自動化にはなりそう

総当たりしているような先行研究から, その計算を有用そうな条件で絞るという手法は使えそう