《CodeBERT:A Pre-Trained Model for Programming and Natural Languages》阅读报告

研究现状

现存各种预训练模型：ELMo、GPT、BERT、XLNet
各种多模态的预训练模型：ViLBERT、VideoBERT

本文认为自然语言NL与编程语言PL也是一种跨模态的领域，提出CodeBERT，该模型能够处理NL-PL的普遍问题，譬如用自然语言搜索代码、自动代码生成等

研究方法

- 本文模型仍使用多层双向Transformer框架，与RoBERTa-base相同，模型总参数为125M。
- 在预训练阶段，模型输入为一个NL-PL对：一个是自然语言文本，视为单词的序列，拆分为WordPiece。另一部分为程序代码，看做Token的序列。拼接成Bert要求的方式：[CLS], w1, w2, w3, ..., wn, [SEP], c1, c2, c3, ..., cm, [EOS]（w为自然语言单词，c为程序token）。

例子：

```
def _parse_memory(s):
    """
    Parse a memory string in the format supported by Java (e.g. 1g, 200m) and
    return the value in MiB

    >>> _parse_memory("256m")
    256
    >>> _parse_memory("2g")
    2048
    """
    units = {'g': 1024, 'm': 1, 't': 1 << 20, 'k': 1.0 / 1024}
    if s[-1].lower() not in units:
        raise ValueError("invalid format: " + s)
    return int(float(s[:-1]) * units[s[-1].lower()])
```

Figure 1: An example of the NL-PL pair, where NL is the first paragraph (filled in red) from the documentation (dashed line in black) of a function.

其中红色框为自然语言文本部分。

模型输出包括两部分：
（1）对于自然语言和程序代码的每个Token的有标记的上下文向量（contextual vector）
（2）聚合序列表示（[CLS]对应的向量）

- 在模型训练的设计上，包括两个目标：

(1) 掩码语言模型（Masked Language Modeling, MLM）。将NL-PL对作为输入，随机为NL和PL选择位置进行掩码，然后用特殊的掩码Token进行替换。注意，掩码语言建模的任务是预测出被掩码的原始Token。其损失函数为：

$$L_{MLM}(\theta) = \sum_{i \epsilon m^w \cup m^c} -logP^{D_1}(x_i|w^{masked}, c^{masked})$$

其中m^w和m^c是被mask掉的位置.

(2) 替换Token检测（Replaced Token Detection, RTD）。在这部分有两个数据生成器，分别是NL生成器和PL生成器，这两个生成器都用于随机掩码位置集（randomly masked positions）生成合理的备选方案。另外，还有一个学习生成器用来检测一个词是否为原词，其背后原理是一个二分类器，这里与GAN不同的是，如果生成器碰巧产生正确的Token，则该Token的标签是“real”而不是“fake”。损失函数为：

$$L_{RTD}(\theta) = \sum_{i=1}^{|w|+|c|} (\delta(i) logp^{D_2}(x^{corrupt}, i) + (1-\delta(i))(1 - logp^{D_2}(x^{corrupt}, i)))$$

$$\delta(i) = \begin{cases} 1 & , \text{if } x_i^{corrupt} = x_i \\ 0 & , \text{if } otherwise \end{cases}$$

那么，总的损失函数为：

$$\min_{\theta} L_{MLML}(\theta) + L_{RTD}(\theta)$$

- 模型的微调：例如在自然语言代码搜索中，会使用与预训练阶段相同的输入方式，并且使用[CLS]的表示向量来衡量代码跟自然语言直接的相关性。

而在代码到文本的生成中，使用编码器-解码器框架，并使用CodeBERT初始化生成模型的编码器。

研究结论

本文作者做了四个实验：

（1）将CodeBERT应用到自然语言代码搜索任务上，并与传统方法进行对比。

给定一段自然语言作为输入，代码搜索的目标是从一组代码中找到语义上最相关的代码，如：

Query

create file and write something

Search Results (top2)

https://github.com/darknessomi/musicbox/blob/master/NEMbox/utils.py#L37-L40

```
def create_file(path, default="\n"):
        if not os.path.exists(path):
            with open(path, "w") as f:
                f.write(default)
```

---

https://github.com/datakortet/yamldirs/blob/master/yamldirs/filemaker.py#L114-L118

```
def make_file(self, filename, content):
        """Create a new file with name ``filename`` and content ``content``.
        """
        with open(filename, 'w') as fp:
            fp.write(content)
```

各模型的MRR(Mean Reciprocal Rank)对比如下：

MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	MA-AVG
NBOW	0.4285	0.4607	0.6409	0.5809	0.5140	0.4835	0.5181
CNN	0.2450	0.3523	0.6274	0.5708	0.5270	0.5294	0.4753
BIRNN	0.0835	0.1530	0.4524	0.3213	0.2865	0.2512	0.2580
SELFATT	0.3651	0.4506	0.6809	0.6922	0.5866	0.6011	0.5628
ROBERTA	0.6245	0.6060	0.8204	0.8087	0.6659	0.6576	0.6972
PT W/ CODE ONLY (INIT=S)	0.5712	0.5557	0.7929	0.7855	0.6567	0.6172	0.6632
PT W/ CODE ONLY (INIT=R)	0.6612	0.6402	0.8191	0.8438	0.7213	0.6706	0.7260
CODEBERT (MLM, INIT=S)	0.5695	0.6029	0.8304	0.8261	0.7142	0.6556	0.6998
CODEBERT (MLM, INIT=R)	0.6898	0.6997	0.8383	0.8647	0.7476	0.6893	0.7549
CODEBERT (RTD, INIT=R)	0.6414	0.6512	0.8285	0.8263	0.7150	0.6774	0.7233
CODEBERT (MLM+RTD, INIT=R)	**0.6926**	**0.7059**	**0.8400**	**0.8685**	**0.7484**	**0.7062**	**0.7603**

（2）进行NL-PL Probing实验，考察CodeBERT在预训练阶段到底学习了什么知识。

这部分实验主要研究在不更改参数的的情况下，Code BERT能够学习哪些类型的知识。目前学界还没有针对NL-PLProbing的工作，所以在这部分实验中，作者自行创建了数据集。

给定NL-PL对，NL-PL Probing的目标是测试模型的正确预测能力。模型精确度(正确预测实例的数量与全部实例数量的比例)比较结果如下：

	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	ALL
NUMBER OF DATAPOINTS FOR PROBING							
PL (2 CHOICES)	38	272	152	1,264	482	407	2,615
NL (4 CHOICES)	20	65	159	216	323	73	856
PL PROBING							
ROBERTA	73.68	63.97	72.37	59.18	59.96	69.78	62.45
PRE-TRAIN W/ CODE ONLY	71.05	77.94	89.47	70.41	70.12	82.31	74.11
CODEBERT (MLM)	**86.84**	**86.40**	**90.79**	**82.20**	**90.46**	**88.21**	**85.66**
PL PROBING WITH PRECEDING CONTEXT ONLY							
ROBERTA	**73.68**	**53.31**	51.32	55.14	42.32	52.58	52.24
PRE-TRAIN W/ CODE ONLY	63.16	48.53	**61.84**	56.25	**58.51**	58.97	56.71
CODEBERT (MLM)	65.79	50.74	59.21	**62.03**	54.98	**59.95**	**59.12**
NL PROBING							
ROBERTA	50.00	72.31	54.72	61.57	61.61	65.75	61.21
PRE-TRAIN W/ CODE ONLY	55.00	67.69	60.38	68.06	65.02	68.49	65.19
CODEBERT (MLM)	**65.00**	**89.23**	**66.67**	**76.85**	**73.37**	**79.45**	**74.53**

Table 3: Statistics of the data for NL-PL probing and the performance of different pre-trained models. Accuracies (%) are reported. Best results in each group are in bold.

也可以用一个具体的案例来对比下。下图案例中分别掩盖了NL和PL中的“min”:

masked NL token

"Transforms a vector np.arange(-N, M, dx) to np.arange(min (|vec|), max(N,M),dx)]"

```
def vec_to_halfvec(vec):

    d = vec[1:] - vec[:-1]
    if ((d/d.mean()).std() > 1e-14) or (d.mean() < 0):
        raise ValueError('vec must be np.arange() in increasing order')
    dx = d.mean()
    lowest = np.abs(vec).min()
    highest = np.abs(vec).max()
    return np.arange(lowest, highest + 0.1*dx, dx).astype(vec.dtype)
```

masked PL token

		max	*min*	*less*	*greater*
NL	Roberta	96.24%	3.73%	0.02%	0.01%
	CodeBERT (MLM)	39.38%	60.60%	0.02%	0.0003%
PL	Roberta	95.85%	4.15%	-	-
	CodeBERT (MLM)	0.001%	99.999%	-	-

Figure 3: Case study on python language. Masked tokens in NL (in blue) and PL (in yellow) are separately applied. Predicted probabilities of RoBERTa and CodeBERT are given.

对比RoBerta与CodeBert预测该位置为“min”的精确率有显著提升：

（3）将CodeBERT应用到生成任务当中。

这部分研究代码到文档的生成问题，并在六种编程语言中对比了生成任务在CodeSearchNet数据集上的结果。作者采用了各种预训练的模型作为编码器，并保持了超参数的一致性。各模型的BLEU-4分属于如下：

MODEL	RUBY	JAVASCRIPT	GO	PYTHON	JAVA	PHP	OVERALL
SEQ2SEQ	9.64	10.21	13.98	15.93	15.09	21.08	14.32
TRANSFORMER	11.18	11.59	16.38	15.81	16.26	22.12	15.56
ROBERTA	11.17	11.90	17.72	18.14	16.47	24.02	16.57
PRE-TRAIN W/ CODE ONLY	11.91	13.99	17.78	18.58	17.50	24.34	17.35
CODEBERT (RTD)	11.42	13.27	17.53	18.29	17.35	24.10	17.00
CODEBERT (MLM)	11.57	14.41	17.78	18.77	17.38	24.85	17.46
CODEBERT (RTD+MLM)	**12.16**	**14.90**	**18.07**	**19.06**	**17.65**	**25.16**	**17.83**

Table 4: Results on Code-to-Documentation generation, evaluated with smoothed BLEU-4 score.

（4）考察CodeBERT预训练模型的泛化能力。

本文的CodeBert只在Python、JavaScript、Java、Ruby、PHP、Go这些语言上做预训练，而本实验的在C#语言上（不是上面6中预训练用的编程语言）进行摘要生成，BLEU-4分数如下：

MODEL	BLEU
MOSES (KOEHN ET AL., 2007)	11.57
IR	13.66
SUM-NN (RUSH ET AL., 2015)	19.31
2-LAYER BILSTM	19.78
TRANSFORMER (VASWANI ET AL., 2017)	19.68
TREELSTM (TAI ET AL., 2015)	20.11
CODENN (IYER ET AL., 2016)	20.53
CODE2SEQ (ALON ET AL., 2019)	**23.04**
ROBERTA	19.81
PRE-TRAIN W/ CODE ONLY	20.65
CODEBERT (RTD)	22.14
CODEBERT (MLM)	22.32
CODEBERT (MLM+RTD)	**22.36**

Table 5: Code-to-NL generation on C# language.

说明CodeBERT能够更好地推广到其他编程语言。但模型的效果略低于code2seq，作者认为原因可能是code2seq融入了抽象语法树AST中的表示，而CodeBERT仅将原始代码作为输入，所以CodeBERT在组合AST的语义上还有待研究。

附：

- 关于Zero-shot：zero show learning就是希望我们的模型能够对其从没见过的类别进行分类，让机器具有推理能力，实现真正的智能。其中零次（Zero-shot）是指对于要分类的类别对象，一次也不学习。
- 论文地址：https://arxiv.org/pdf/2002.08155.pdf
- 代码：https://github.com/microsoft/CodeBERT
- 数据集：CodeSearchNet