CS 230

Robot Apocalypse: Generating Code Snippets from Natural Language Descriptions

Jinhie Skarda, Janette Cheng, Priyanka Sekhar, Stanford University, Dept. of Computer Science
jskarda@stanford.edu | jancheng@stanford.edu | psekhar@stanford.edu

Problem and Motivation

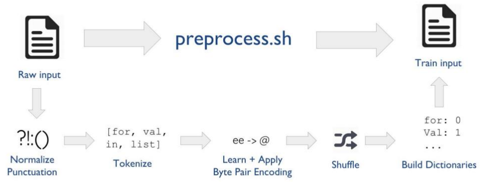
- Automated code generation from natural language can lower the barrier to building software and enhance engineering efficiency
- Our approach: **Sequence to Sequence** model that takes **pseudocode** snippets and translates them to **python code** snippets.

```
import numpy as np

Write parser for csv which takes input and
output files as parameters |
    def writecsv(infile, outfile):
      ...
```

Data

- Oda et al. [1] dataset
- **18,805** Django code to pseudocode snippets
- **80/10/10** train/dev/test split
- **BLEU** scores to compare models



Pseudocode	Code
`zip together new_keys and keys, convert it to dictionary, assign it to m.`	`m = dict ( zip ( new_keys , keys ) )`
`derive the class DisallowedHost from the SuspiciousOperation base class.`	`class DisallowedHost ( SuspiciousOperation ) :`

Hyperparameter Tuning

Learning Rate	Depth	Dropout Rate	Train Loss	Train BLEU	Dev BLEU
0.47943	8	0.26	2.36E+07	0.00	0.00
0.00568	1	0.78	21.19	0.00	0.00
0.00022	3	0.09	0.3668	9.70	6.09

- **Learning rate below 0.001** most important
- BLEU and loss correlate well

Word-Level Model

Best Model BLEU Score

Learning Rate = 0.0002, Depth = 4, Dropout Rate = 0.66

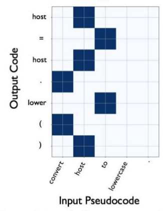
Train BLEU	Dev BLEU	Test BLEU
46.20	33.53	31.53

Previous Code Generation Model BLEU Scores [4] [5]

Retrieval	RL	SEQ2SEQ Code Gen	SEQ2TREE
18.6	24.94	35.9	44.6

- Outperforms probabilistic NLP approaches like Retrieval
- Comparable to other sequence to sequence code generation models
- Outperformed by sequence to tree methods

Attention Weights Analysis



Model consistently learns mappings e.g 'to' → 'lower' ; 'convert' → '.'

Model

We use an **LSTM** with **Luong** attention adapted from an open source Tensorflow Seq2Seq model [2] using **Softmax Cross Entropy Loss**.

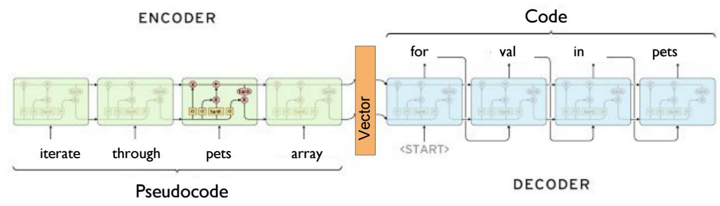


Figure 1: Encoder/Decoder Model Architecture [3]

Character-level Model

Individual characters are significant in code, so we trained some models with a **character-level target vocabulary**.

Target Output	With Normal Loss	With Weighted Loss
`class BaseDatabaseCache(BaseCache):`	`self.cache(self):`	`classe Cache(BaseCache(BaseCache(BaseCache`
`def __iter__ (self):`	`def clear(self):`	`def __int__(self, params)`
`def __init__(self, *args, **kwargs):`	`default(self, key, delta = 1, table, timeout = DEFAULT_TIMOUT):`	`def _det__(self, *args, ***kwargs):`

Output biased towards **frequent** characters → Tried **character-frequency-weighted** cross entropy loss (results above)

Future Work

- Use of **Abstract Syntax Tree** (AST) representations as suggested by Yin and Neubig [4]
- Investigation of higher **beam widths**, longer **train times**, and a much **larger dataset** to improve our model's generalizability

[1] https://ahcweb01.naist.jp/pseudogen/
[2] https://github.com/JayParks/tf-seq2seq
[3] http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/
[4] https://arxiv.org/pdf/1704.01696.pdf
[5] https://arxiv.org/pdf/1707.07402.pdf