Using Deep Learning to Generate Automated Code Documentation Across Multiple Programming Languages

-- A survey of assessing CodeBERT's performance on different PLs

May 9th, 2022

Drew H. Klaubert (kdrew17@vt.edu)
V N S Rama Krishna Pinnimty (ramapinnimty@vt.edu)

Problem Statement

We are going to use deep learning to generate code documentation for different Programming languages.

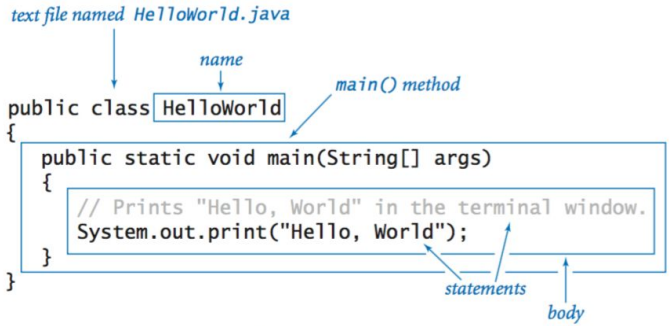
- The goal is to compare the models with each other to see if certain languages perform better than others.
- We are using 4 different languages for this project:
  - Python
  - Java
  - JavaScript
  - Ruby

Hypothesis

Since a language like Python has more free-form and contains natural language qualities over something like Java which has lots of cumbersome syntax, we hypothesize that it will perform better for the task at hand.

```
>>> greeting = "Hello World"
>>> name = "Zenon"
>>> planet = "X"
>>> intro = (
...     f"{greeting}. "
...     f"I am {name}. "
...     f"I come from {planet}."
... )
>>> print(intro)
"Hello World. I am Zenon. I come from X."
```

https://www.udacity.com/blog/2021/03/python-string-formatting-overview-and-real-examples.html



https://introcs.cs.princeton.edu/java/11cheatsheet/

Data Description

The data we are using is from the CodeSearchNet Dataset.

- This dataset contains code and natural language document pairs
- 6 different programming languages are represented in this dataset

Programming Language	Training	Dev	Test
Python	251,820	13,914	14,918
PHP	241,241	12,982	14,014
Go	167,288	7,325	8,122
Java	164,923	5,183	10,955
JavaScript	58,025	3,885	3,291
Ruby	24,927	1,400	1,261

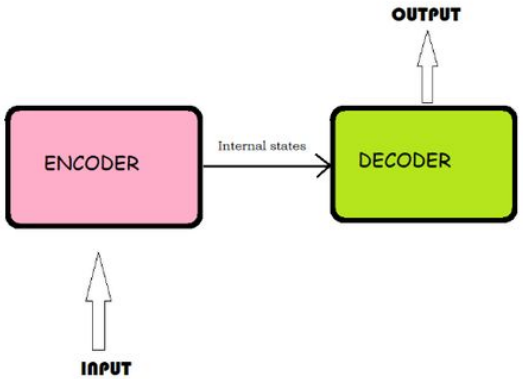
Data Pre-processing

We began pre-processing the data using the methods provided in the CodeXGLUE GitHub repository.

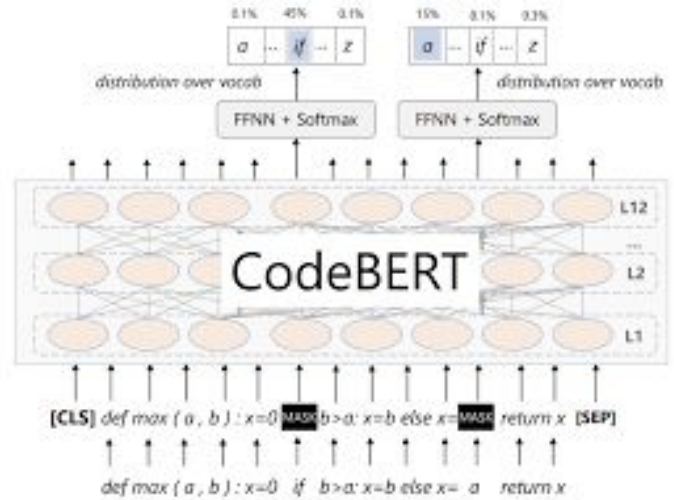
- Once the data was obtained we tried to train a few models
- Knowing how many models we wanted to train, we knew we needed to train a little faster
- Required that we truncate the data to cut down on training time
- Was possible given the nature of the hypothesis

The Models

For testing our hypothesis, we will be using an Encoder-Decoder architecture with CodeBERT or RoBERTa as the encoder and the TransformerDecoder in the PyTorch Library as the decoder.



Source



Source

Model Building Workflow

We have 4 different languages to test.

- Train a model for each of them and get evaluation score using BLEU metric
- Compare the results of the scores

Conduct follow-up experiments if necessary.

- Would require to train 4 more models (would be the same as the first experiment but with different data size)
- Original experiment has possibility of not having conclusive results due to the imbalance in the amount of data for each language

Evaluation

The first round of experiments yielded some interesting results, they are presented in the table below-

Language	Training Size	Bleu Score
Python	100,000	15.43
Java	100,000	16.0
Ruby	24,927	10.32
JavaScript	58,025	12.41

- Hypothesis not looking to be correct
- Need to do further testing to see if the difference in the scores is due to the model itself or the amount of training data that was used.

Evaluation (contd.)

The next round of experiments required that we make all of the training sizes equal.

- We needed to use a low number so that we could still be fair to the languages with smaller sizes (20,000)

Language	Training Size	Bleu Score
Python	20,000	11.7
Java	20,000	12.82
Ruby	20,000	9.68
JavaScript	20,000	8.04

- Hypothesis still does not seem to be correct, we can make this conclusion now
- BLEU scores are showing that the opposite might be true, Java performed better in our experiments

Google Drive with materials

Lessons Learned

- We have learned how to deploy and fine-tune some advanced models.
- Doing experiments like this has given us insight to learning about other models on a granular level and testing our own future models, should we develop them.
- Learned to perform computing tasks on a schedule due to the amount of time it took for our models to train.

Broader Impacts

- While we have not proven our hypothesis, we have proven the robustness of the state-of-the-art models.
- Future researchers could use a similar methodology to us to evaluate their models in multiple settings.
- Proving the robustness and utility of CodeBERT for this downstream task could support its use for many other applications.