NoCoDeR - Code generation from Natural Language

END Capstone Project

Y.Rajesh

TOC

Overview

NoCoDeR explores the magic of Transformer models to suggest Python code snippets based on English language input provided by the user.

Sample query:

```
Program to convert string to uppercase
```

Result:

```
st = "hello world"
upper_st = st.upper()
print("Upper Case", upper_st)
```

It is trained on a question-answer styled dataset that consists of ~4600 pairs of English Language questions and their equivalent Python programs.

Dataset Basic Cleanup

For Python Code

1. Alignment correction, formatting correction using pylint in VSCode
2. Convert all Python 2.x snips to Python3.x format.
3. Removal of comments and docstrings in code using tokenize library
4. Deletion of commented snippets
5. Removal of Ultra-long code pieces (E.g: complete Tic-Tac-Toe program!)
6. Removal of additional “driver” programs

For Question Prompts

1. Converting multi-line prompts to single line
2. Correction of typos and removal of blank lines or lines with only white spaces.

Preprocessing Pipeline

Basic Text File

Base Dataframe

Create a pandas dataframe consisting of question prompt, cleaned_code, length of code and length of prompt

Tokenization

AutoTokenizer for question prompts.
Custom tokenizer for Code.

Vocab Creation

Vocab and Dictionary for code and question prompts

Final Dataset

Numericalize, SOS/EOS tag, padding and masking to generate dataset.

- Source Tensor
- Source Mask
- Target Tensor
- Target Mask
- Target Token Types

Source = Q Prompt
Target = Python Code

Tokenization Scheme

Notebook Link

For Python Code

1. Based on "CuBERT Tokenizer" paper, basic tokenization converts code to equivalent tokens i.e variable names, symbols/operators, spaces, newlines, keywords etc.
2. Each variable name is identified as camel/snake case and is further split to smaller words.
3. Special tokens such as INDENT, DEDENT, NEWLINE, ENDMARKER etc are prepended with respective strings. E.g: NEWLINE is retained as __NEWLINE__
4. INDENT markers are transformed to "__INDENT__ <size of indentation>"
5. The class or type of every token is also retained as a sequence.
6. Using code pieces above, code vocabulary is created and has ~5.8K tokens.
7. Token/type vocabulary is created from the token type array generated in #5.

For Question Prompts

1. AutoTokenizer from HuggingFace library was used which is basically a BPE tokenizer without any pre-training.
2. Vocab size is ~56K with no <UNK> tokens.

Tokenization Scheme

[Notebook Link]

For Python Code - Example

Code Snippet

```
def add_two_numbers(num1, num2):
    sum = num1 + num2
    return sum
```

Code Tokenizer

Token Array

```
['def', 'length', '(', 'lst', ')', ':', '   NEWLINE   ', '   INDENT __
', 'if', 'not', 'lst', ':', '   NEWLINE   ', '   INDENT          ',
'return', '0', '   NEWLINE   ', '   DEDENT   ', 'return', '1', '+',
'length', '(', 'lst', '[', '1', ':', ':', '2', ']', ')', '+', 'length',
'(', 'lst', '[', '2', ':', ':', '2', ']', ')', '   NEWLINE  _',
'   DEDENT   ', 'a', '=', '[', '1', ',', '2', ',', '3', ']',
'   NEWLINE   ', 'print', '(', '"^', 'Length^', ' ^', 'of^', ' ^',
'the^', ' ^', 'string^', ' ^', 'is^', ':^', ' ^', '"', ')',
 '___NEWLINE___', 'print', '(', 'a', ')', '___EOS___']
```

TokenType Array

```
['KEYWORD', 'IDENTIFIER', 'PUNCTUATION', 'IDENTIFIER', 'PUNCTUATION',
 'PUNCTUATION', 'NEWLINE', 'INDENT', 'KEYWORD', 'KEYWORD', 'IDENTIFIER',
 'PUNCTUATION', 'NEWLINE', 'INDENT', 'KEYWORD', 'NUMBER', 'NEWLINE', 'DEDENT',
 'KEYWORD', 'NUMBER', 'PUNCTUATION', 'IDENTIFIER', 'PUNCTUATION', 'IDENTIFIER',
 'PUNCTUATION', 'NUMBER', 'PUNCTUATION', 'PUNCTUATION', 'NUMBER', 'PUNCTUATION',
 'PUNCTUATION', 'PUNCTUATION', 'IDENTIFIER', 'PUNCTUATION', 'IDENTIFIER',
 'PUNCTUATION', 'NUMBER', 'PUNCTUATION', 'PUNCTUATION', 'NUMBER', 'PUNCTUATION',
 'PUNCTUATION', 'NEWLINE', 'DEDENT', 'IDENTIFIER', 'PUNCTUATION', 'PUNCTUATION',
 'NUMBER', 'PUNCTUATION', 'NUMBER', 'PUNCTUATION', 'NUMBER', 'PUNCTUATION',
 'NEWLINE', 'IDENTIFIER', 'PUNCTUATION', 'STRING', 'STRING', 'STRING', 'STRING',
 'STRING', 'STRING', 'STRING', 'STRING', 'STRING', 'STRING', 'STRING', 'STRING',
 'STRING', 'PUNCTUATION', 'NEWLINE', 'IDENTIFIER', 'PUNCTUATION', 'IDENTIFIER',
 'PUNCTUATION', 'ENDMARKER']
```

Tokenization Scheme

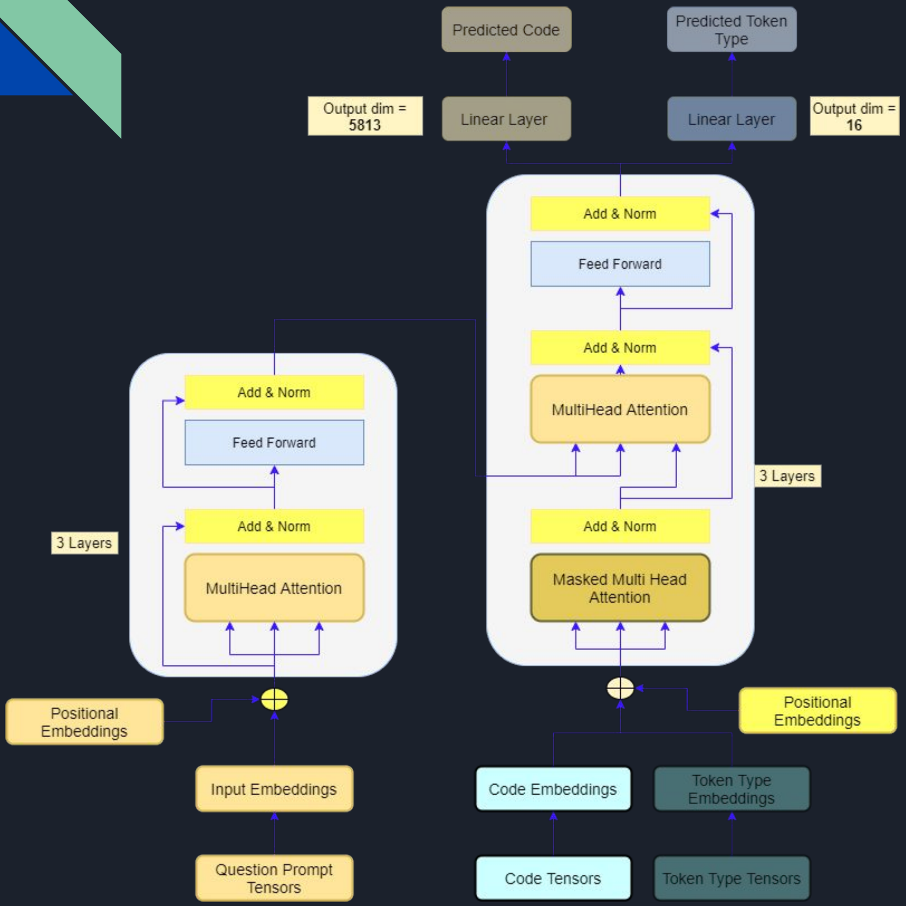
Notebook Link

Dataset and Iterator Creation

1. For a pre-specified max sequence length and for each pair of question prompt and code snippet following were generated:
    a. Array of NL tokens with <pad> markers if length < max sequence length
    b. Array of Mask values for NL tokens (0 for <pad> markers and 1 otherwise)
    c. Array of Code Tokens with <pad> markers if length < max sequence length
    d. Array of Code Token Types with <pad> markers if length < max sequence length
    e. Array of Mask values for Code tokens (0 for <pad> markers and 1 otherwise)
2. `TensorDataset` was created based on above arrays over all the selected samples.
3. Validation(20%) and Training(80%) Iterators were created using PyTorch's `SubsetRandomSampler`

Model Overview

[Notebook Link](#)



1. Standard Transformer model with multi-head, masked multi-head and encoder-decoder attention is used
2. Encoder input - question prompt tensor and source mask.
3. Decoder has 3 inputs - code tensors, token-type tensors and causal mask.
4. Independent embedding layers are used to convert code tensors and token-type tensors to hidden dimensions.
5. Self-attention in the decoder is calculated over the sum of code-embedding, token-type and positional embedding.
6. Output of final enc-dec attention is passed through 2 different linear layers to predict code and token-type.

	Encoder	Decoder
d_model	256	256
Layers	3	3
Attention Heads	8	8
Pointwise FFN	1024	1024
Dropout	0.2	0.2
Input Vocab	50625	5813, 16

Model Parameter count = `21,649,861`

Loss Function and Metrics

Notebook Link

1. Total Loss = *a* * (Cross-Entropy Loss on Code Pred) + (*1-a*)*(Cross-Entropy Loss on Token Types)
   a. The value a is between 0 and 0.5 called the “mixing ratio”
2. Loss function has two components:
   a. Cross-Entropy loss on Code Predictions
   b. Cross-Entropy loss on Token-Type Predictions
3. Loss for both components was calculated only on the non-padded predictions.
4. During the initial training, *a* was set to 0.2 to give higher weightage to allow network to learn sequence of token-types better i.e allow the network to understand syntactical structure
5. In the later stages, *a* was gradually increased till 0.5 to give equal weightage for code and syntax learning.
6. Adam Optimizer with default settings was used, however learning rate was varied at various stages.
7. For evaluation, Rouge-Lsum and Rouge-L scores were calculated on predicted and original source code.

Training Strategy

[Notebook Link]

Stage-1

1. Samples with *code length* < ***128*** and *prompt len* < ***256***
2. 30 Epochs with lr=5e-4 and mix ratio = 0.8
3. Followed by ~200 epochs with LR scheduling with min_lr = 1e-9 and mix ratio = 0.5

Stage-2

1. Samples with *code length* < ***256*** and *prompt len* < ***512***
2. Followed by ~200 epochs with LR scheduling with min_lr = 1e-9 and mix ratio = 0.5

Stage-3

1. Samples with *code length* < ***512*** and *prompt len* < ***1024***
2. Followed by ~200 epochs with LR scheduling with min_lr = 1e-9 and mix ratio = 0.5

1. Vocab length for question prompts, code and token types is constant across all 3 stages
2. Max Seq Len generated by the model is fixed at 512.

Results

1. Lowest Validation Loss achieved was ~0.44
2. Sample code and attention visualization can be found in [this notebook]
3. Below is the Evaluation metric based on Rouge metrics:
    a. `R1_precision 0.762056`
    b. `R1_recall 0.786593`
    c. `R1_f 0.766437`
    d. `RL_precision 0.744741`
    e. `RL_recall 0.766019`
    f. `RL_f 0.748387`

Alternate Models and Results

Model Details	**Results**	**Model Link**
Use AutoTokenizer for both code and query prompt	1. Large model size because of output vocab also was ~56K 2. Misalignment and wrong indentation was significantly higher	https://github.com/rajy4683/END_NLP/blob/master/END_Capstone/END_NoCoDeR_AutoTokenize.ipynb
Pre-training on CodeSearchNet samples and fine-tune for capstone dataset	1. CodeSearchNet code samples were usually of larger size and were much more complicated than capstone dataset. 2. Docstrings were more descriptive of functions use rather than query/response style of capstone dataset.	N/A
Simple BPE tokenizers for code **without** token type output	1. Highly unstable during training probably due to some error in tokenizing so couldn't pursue further.	https://github.com/rajy4683/END_NLP/blob/master/END_Capstone/END_NoCoDeR_BPETokenize.ipynb

Credits and References

1. END Course Material and Lectures
2. END and EPAi batchmates for Capstone DataSet
3. CodeBert and GraphCodeBert papers and Repos
4. Nokia's Neural Code Search paper
5. CuBert Paper and Repos
6. **CodeSearchNet** Paper and Repos

A BIG THANKS TO ROHAN AND ZOHEB for great course content and the awesome capstone project!!

Thank you!