Contextual Token Prediction in Python

Scott Kovach dskovach@stanford.edu
CS229 Fall 2019

Overview

Neural networks have been recently applied to program autocompletion and program synthesis tasks. In this project, we apply an attention-based neural network to the problem of predicting source code tokens from (bidirectional) context and measure its accuracy on a dataset of real-world Python programs.

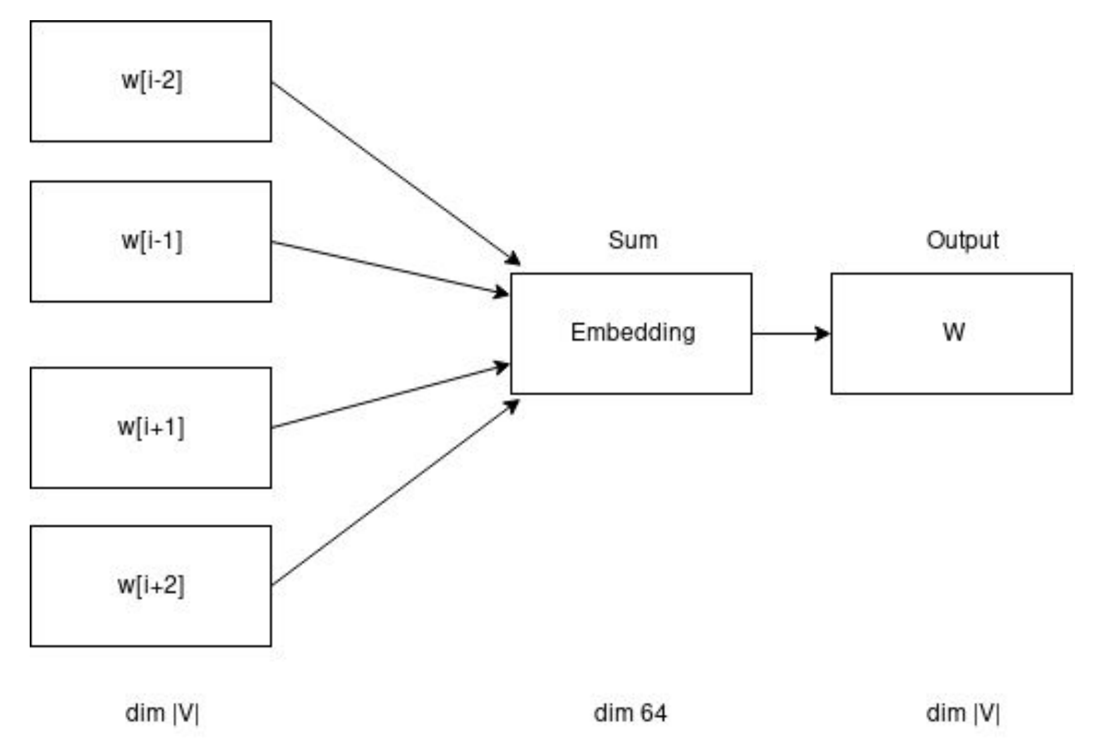
Dataset

We use a dataset of 150,000 Python source files[1]. The subset of files used for development contain approximately 113 million tokens and over 1 million unique tokens, most of which occur infrequently. To obtain a manageable vocabulary (of size 2048), we include a subset of the most frequently occurring tokens, and for each training/test example map other tokens to locally unique identifiers[2]. In the current work, we ignore the content of string literals by mapping them to a single unique identifier, but include all other token types.

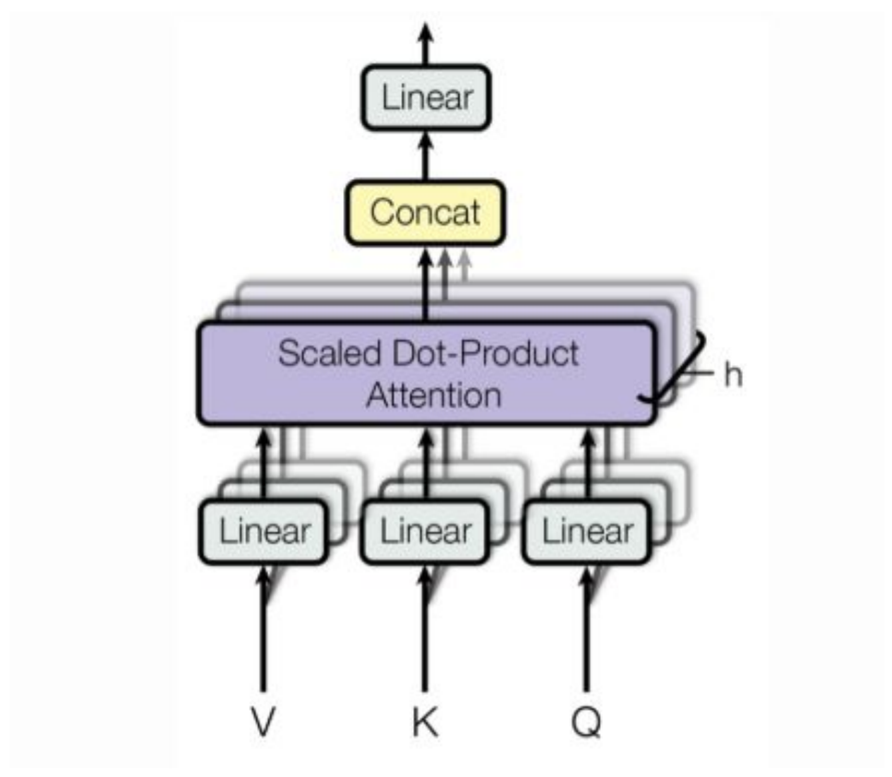
```
> tokenize('foo = foo + bar()')

[x1, '=', x1, '+', x2, '(', ')']
```

Models

As a baseline, we use the continuous bag-of-words model[3]. CBOW is a linear mapping of the vocabulary into a smaller embedding space. Predictions ignore token order.



We train a model (**Tr**) with two transformer layers, two attention heads per layer, shared weights[4], and hidden dimension 64. The training procedure samples windows of 32 tokens at a time and creates training examples using the masked language model loss. The model contains 337216 total learnable parameters.



We show the top-1 and top-3 prediction accuracy for CBOW with a context of eight tokens, the transformer with a full context of 32 tokens, and the transformer with a smaller context for a closer comparison with the baseline.

Results

	top-1 accuracy	top-3 accuracy
1-Gram	0.14	0.28
CBOW-8	0.42	0.67
Tr-8	0.56	0.73
Tr-32	**0.68**	**0.81**

Future Work

- experiment with deeper models, a larger vocabulary, alternative pre-processing strategies; better engineer the training procedure to handle these larger cases
- evaluate usefulness as an autocompletion tool or error predictor
- evaluate fine-tuning within a single repository for more context-sensitive results

References

[1]https://www.sri.inf.ethz.ch/py150
[2]https://web.stanford.edu/~chshah/files/contextual-code-completion.pdf
[3]https://arxiv.org/abs/1301.3781
[4]https://arxiv.org/pdf/1909.11942.pdf