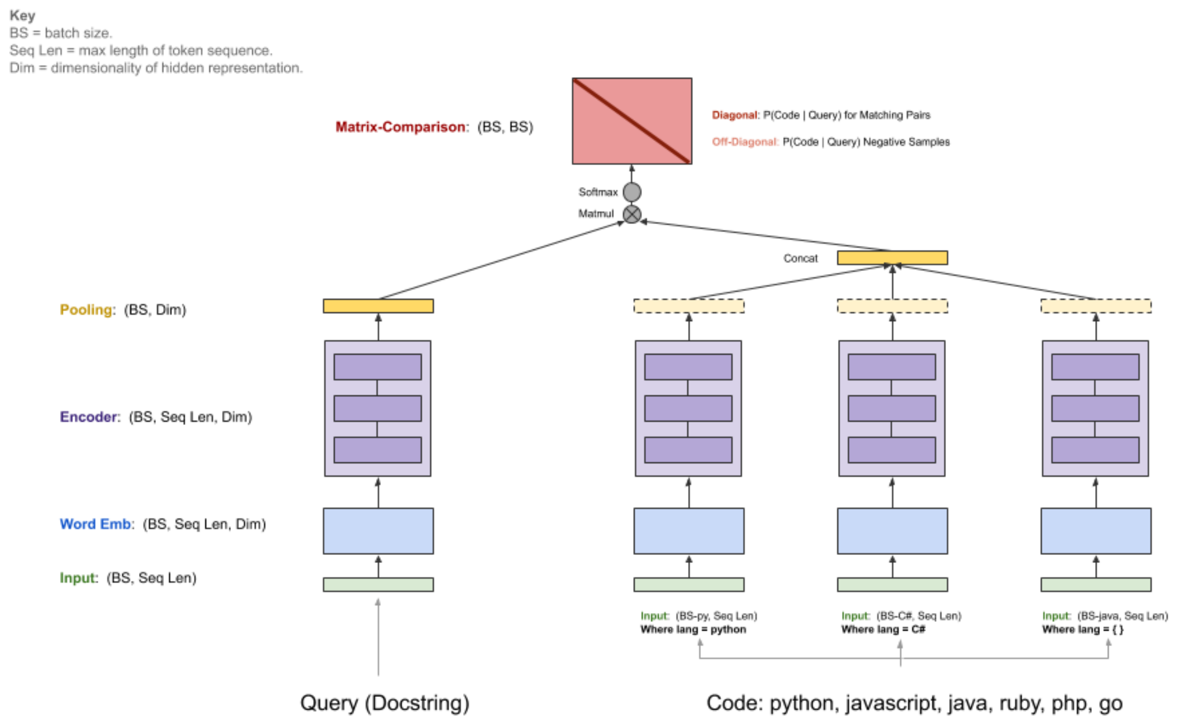
Semantic Code Search Report

Introduction

In Semantic Code Search, I will conduct a code search for a specific programming language: Python. Before starting to introduce this project, we first need to introduce and analyze the background of the Semantic Code Search challenge.

- **The task of the challenge**: The purpose of this project is to establish a dataset that can provide a quantitative evaluation baseline for code search engine quality.
- **The challenge dataset**: A large number of related code documents were collected in the open source project of Github, and the dataset of this challenge was formed after data preprocessing. Among them, the comments collected by the challenge are top-level function and method comments while the code is an entire function or method. They will be feed into our model in the format of (comment, code) pairs.
- **The basic model of the challenge**:


Key
BS = batch size.
Seq Len = max length of token sequence.
Dim = dimensionality of hidden representation.
Matrix-Comparison: (BS, BS)
Diagonal: P(Code | Query) for Matching Pairs
Off-Diagonal: P(Code | Query) Negative Samples
Softmax
Matmul
Concat
Pooling: (BS, Dim)
Encoder: (BS, Seq Len, Dim)
Word Emb: (BS, Seq Len, Dim)
Input: (BS, Seq Len)
Input: (BS-py, Seq Len) Where lang = python
Input: (BS-C#, Seq Len) Where lang = C#
Input: (BS-java, Seq Len) Where lang = { }
Query (Docstring)
Code: python, javascript, java, ruby, php, go

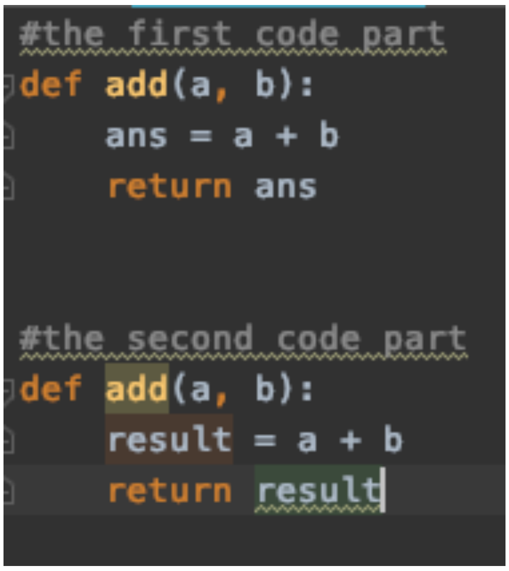
Explanation of Baseline Models

From the figure above, different baseline model has different encoders which includes(neural-bag-of-words, RNN, 1DCNN, BERT and so on). Here I only tried Neural-Bag-of-Words and BERT due tot computational costs. So in this part I will firstly introduce and explain these two models.

Neural-Bag-Of-Words Model (BOW Model):

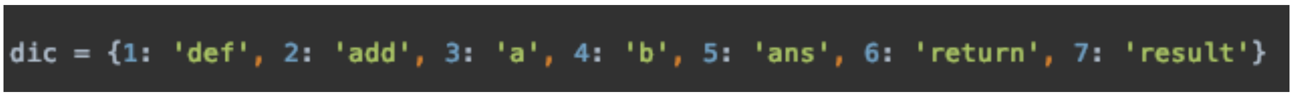
Generally speaking, the BOW model assumes that for a document, it ignores its word order, grammar, syntax, and other elements, and treats it as a collection of several vocabularies. The occurrence of each word in the document is independent and does not depend on whether other words appear. Meanwhile, the occurrence of each word in the document is also independent with word order. That is to say, any word that appears anywhere in the document is independently selected by the semantic meaning of the document. So what exactly does it mean? Then give specific examples.

Suppose that we have 2 python code here:



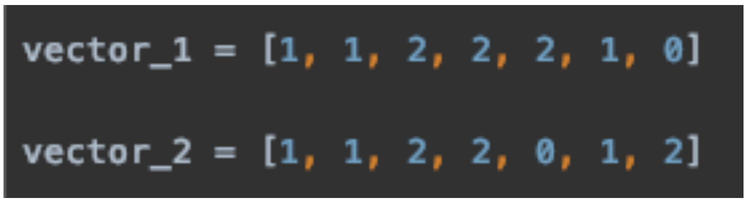
```python
#the first code part
def add(a, b):
    ans = a + b
    return ans


#the second code part
def add(a, b):
    result = a + b
    return result
```

Based on the above two python codes, we can construct a dictionary composed of keywords in the code. In order to explain the BOW method in this place, we ignore the operators in the code snippet and only deal with the letters and words in it.



```python
dic = {1: 'def', 2: 'add', 3: 'a', 4: 'b', 5: 'ans', 6: 'return', 7: 'result'}
```

This dictionary contains a total of 7 different words. Using the index number of the dictionary, each of the above two python code can be represented by a 7-dimensional vector (using integer numbers 0 ~ n to indicate the number of occurrences). In this way, according to the number of occurrences of keywords in each python code, the above two codes can be represented in the form of vectors:



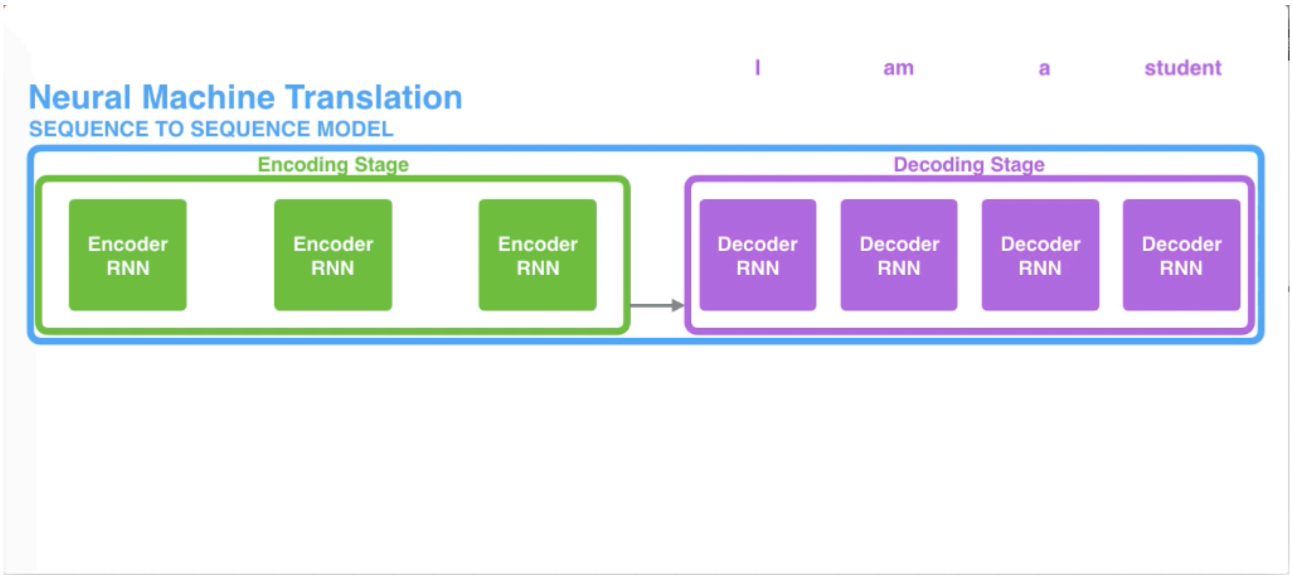
```python
vector_1 = [1, 1, 2, 2, 2, 1, 0]

vector_2 = [1, 1, 2, 2, 0, 1, 2]
```

Each of these two vectors contains 7 elements, where the i-th element represents the number of times the i-th word in the dictionary appears in the sentence. Therefore, the BOW model can be considered as a statistical histogram. So the word frequency can be easily calculated by this model. Now imagine that in a huge code set D, there are a total of M code parts in D, and all the words in each code part are extracted to form a dictionary of N words. Using the BOW model, each code part can be represented as an N-dimensional vector. After becoming an N-dimensional vector, many problems become very easy to solve. The computer is very good at dealing with numerical vectors. We can use cosine distance to find the similarity between two code parts, or we can use this vector as a feature vector.

Bidirectional Encoder Representations from Transformers (BERT) :

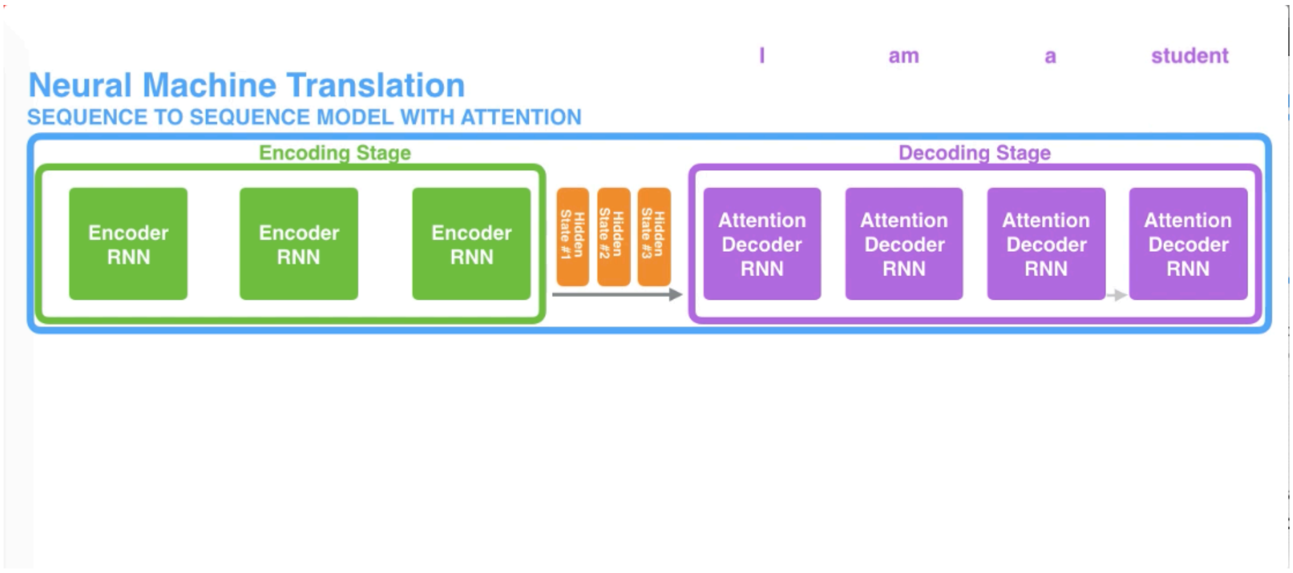
In this section, I will introduce the BERT model which impressed all researchers in 2018. Before we introduce BERT, we need to introduce the basic knowledge of BERT which includes Attention mechanism and Transformer model.

1. Attention: In the NLP field, we often design RNN as a black box that can encode and decode tokens. The traditional encoder and decoder are designed based on RNN's own network structure. As shown below： Each token will be sent to the encoder for

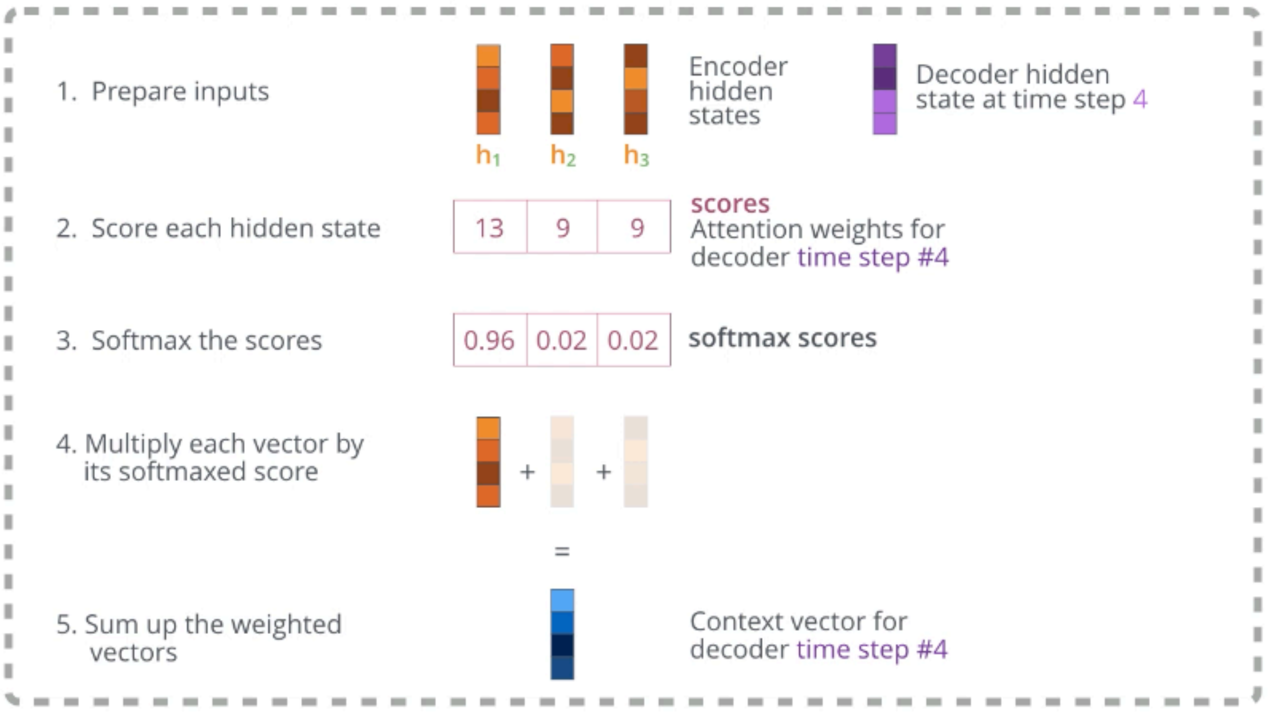


encoding using the RNN status parameters after the last token encoding is completed, and at the same time changing the RNN network status parameters. The next token is also executed when another token is sent to the encoder. In this way, all the input tokens will be converted into context vectors that the computer can calculate. Subsequently, these word vectors are sent to the decoder to be decoded and converted into the language we are familiar with.
But if the sentences we process are too long, it will have a huge impact on the performance of the model. Because the model cannot focus on every element of such a long sentence at the same time. Just like we see a picture in life, we cannot observe every object in the picture in a balanced manner at the same time, but focus on the objects we are interested in according to our attention mechanism. Therefore, the attention mechanism was born. But if the sentences we process are too long, it will

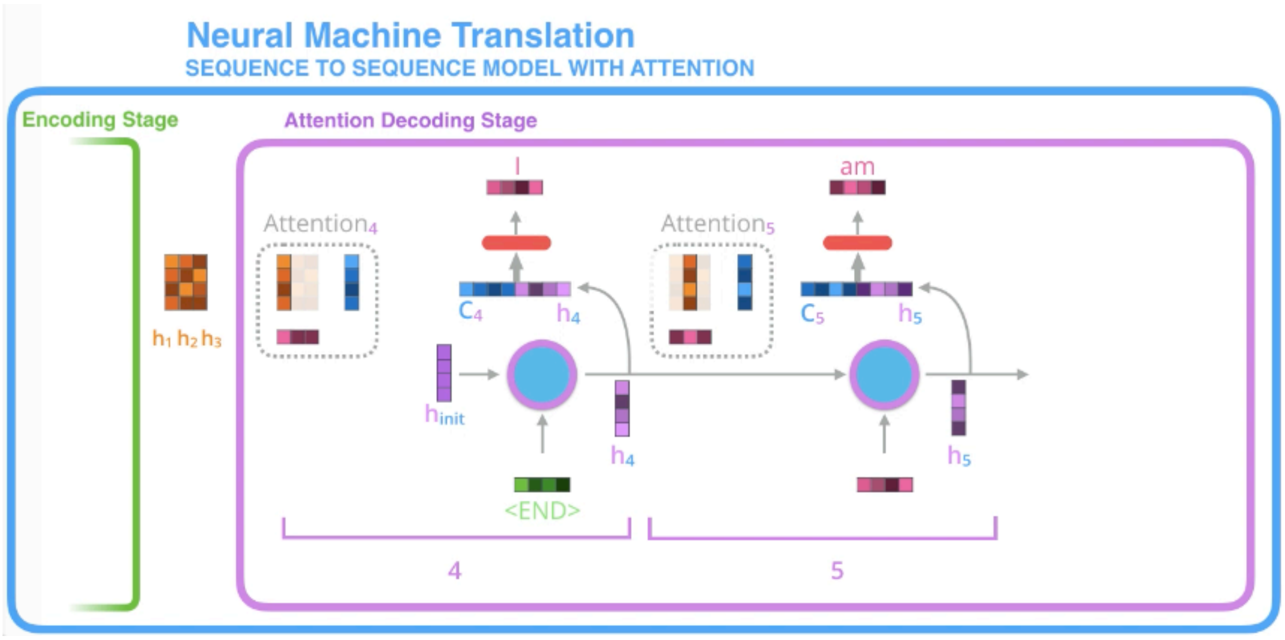
have a huge impact on the performance of the model. Because the model cannot focus on every element of such a long sentence at the same time. Just like we see a picture in life, we cannot observe every object in the picture in a balanced manner at the same time, but focus on the objects we are interested in according to our attention mechanism. Therefore, the attention mechanism was born. The picture below shows how the attention mechanism differs from the traditional encoder decoder. We will pass the RNN network state parameters that encode different tokens to the decoder as a set of weights, so that the decoder can focus on some tokens with high weights. In other words, we gave the decoder attention.



The following picture describes calculation process of the Attention mechanism:

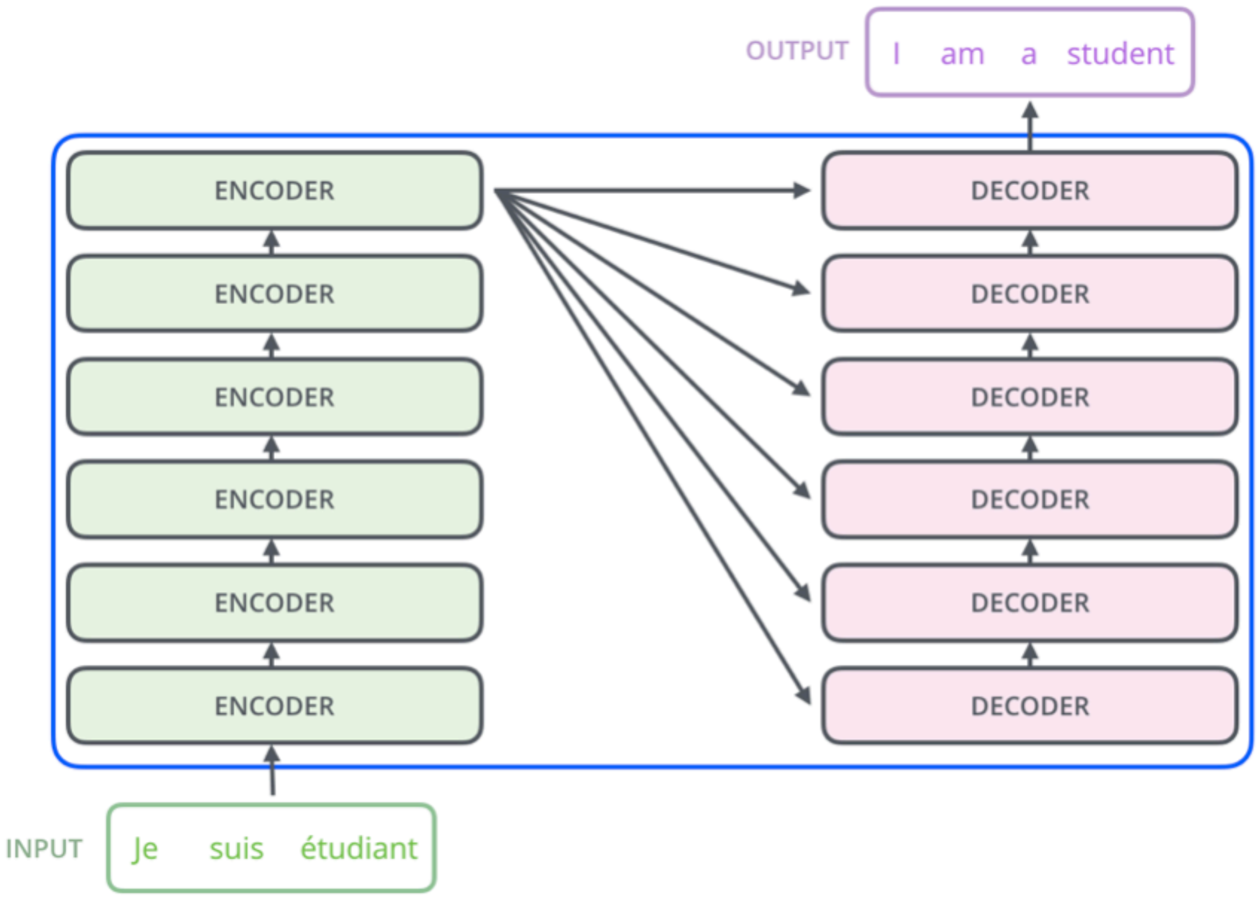

1. Prepare inputs
Encoder hidden states
Decoder hidden state at time step 4
h1 h2 h3
2. Score each hidden state
13 9 9
scores Attention weights for decoder time step #4
3. Softmax the scores
0.96 0.02 0.02
softmax scores
4. Multiply each vector by its softmaxed score
+ +
=
5. Sum up the weighted vectors
Context vector for decoder time step #4

Now let’s combine them together:

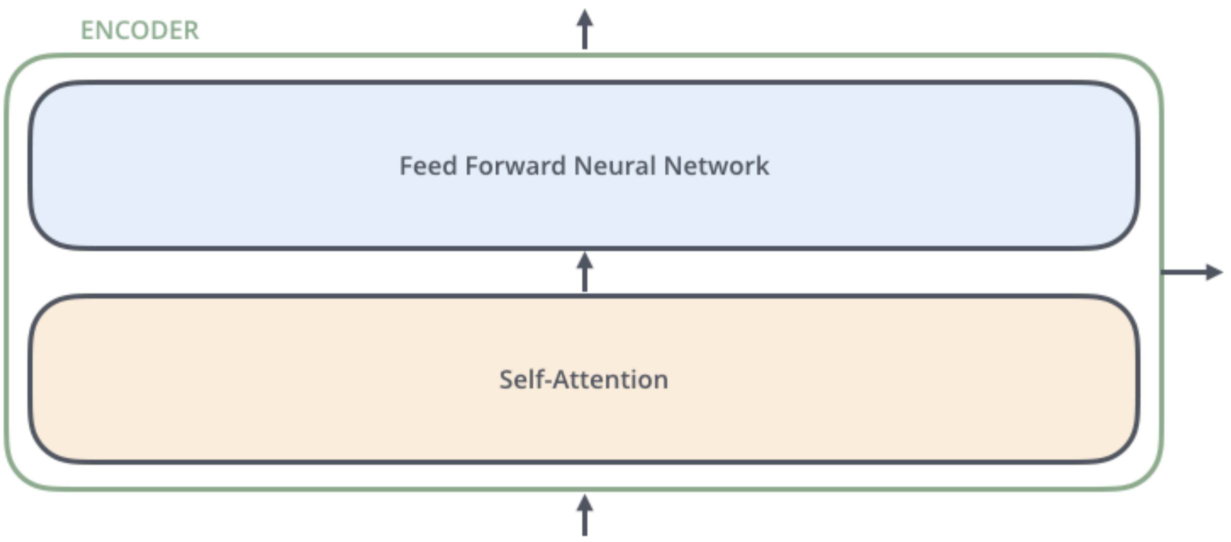


First, the encoder receives the token "END", realizes that this is the last token, and then initializes the decoder's RNN state parameter (h_init). After the encoding of "END" is completed, the decoder's RNN status parameter will be updated (h4). Next, we perform the concatenate operation on the context vector and the RNN state parameter of the current decoder, and send the generated new vector to the feed-forward network to generate the decoding result. Then update the decoder's RNN state parameter (h5), and loop this process until all context vectors are decoded.

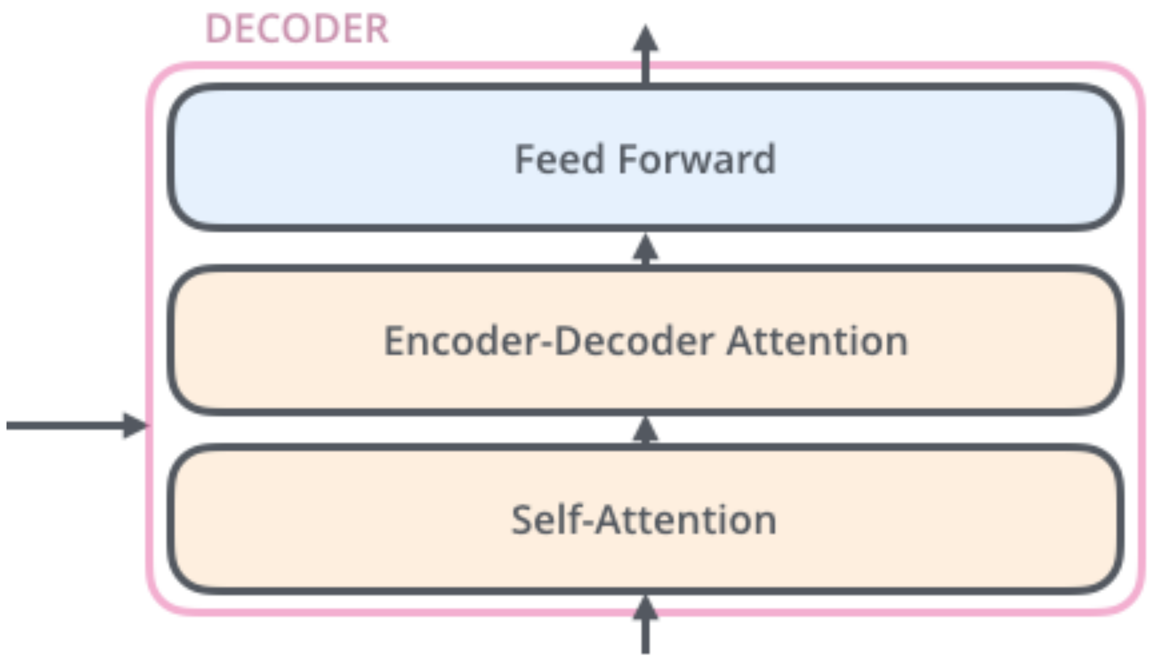
2. Transformer: There are two problems with the Attention mechanism. The first is that the calculation of time t depends on the calculation result at time t-1, which limits the parallelism of the model. The second problem is that the RNN network is difficult to deal with the long-term memory relationship. Transformer solves the above two problems. First, it uses the Attention mechanism to reduce the distance between any two words in the sequence to a constant; second, it is not an RNN-like sequential structure, so it has better parallel ability. As set in the paper, the encoder is composed of 6 encoding blocks, and the decoder is also composed of 6 decoding blocks. As with all generative models, the output of the encoder will be used as the input of the decoder, as shown in figure below:



In Transformer's encoder, the data will first go through a module called 'self-attention' which is introduced above to get a weighted feature vector Z (context vector). Then this Z vector will be sent to the encoder's next feedforward network to get the encoder output. Here is the structure of the encoder module.



The structure of the Decoder is shown in the following figure.
The difference between it and the encoder is that the Decoder has an additional



Encoder-Decoder Attention. The two Attentions are used to calculate the weights of the input and output, respectively:
- Self-Attention: the relationship between the current token and the decoded tokens;
- Encoder-Decoder Attention: The relationship between the current token and the encoded feature vector.

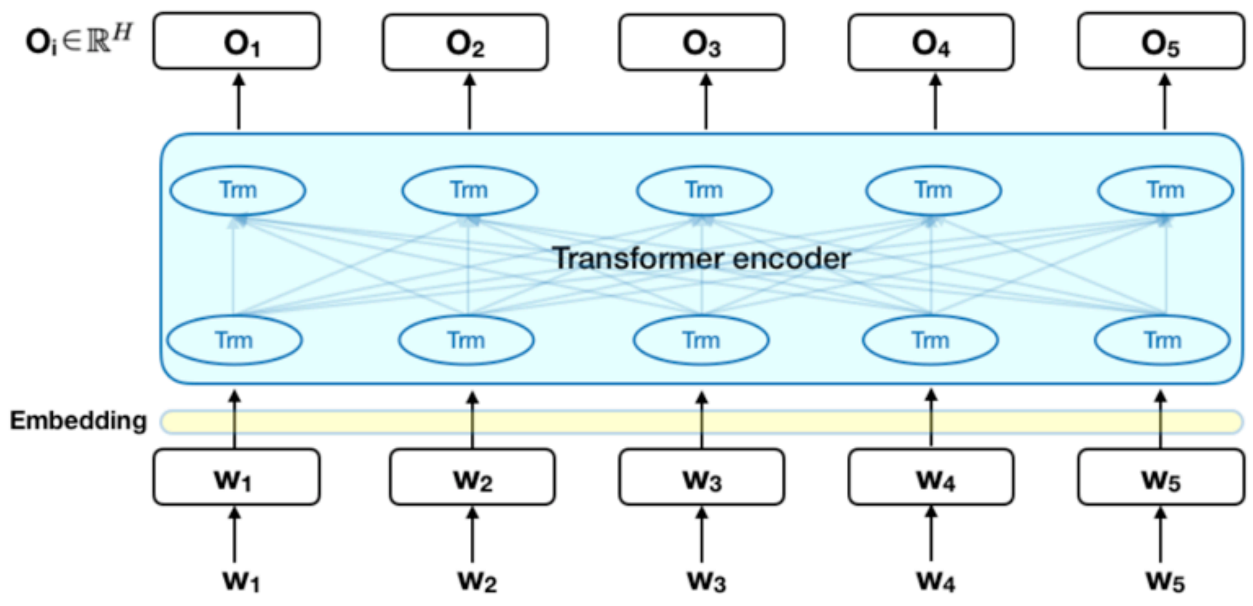
In order to solve the problem of order dependence, the position encoding (Position Embedding) is introduced when encoding word vectors. Specifically, the position coding will add the position information of the word to the word vector, so that the Transformer can distinguish the words in different positions.

$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

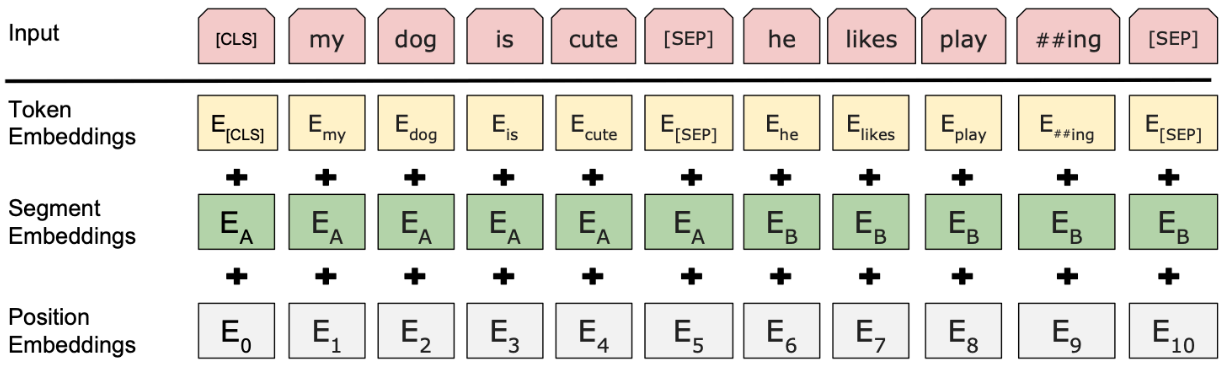
The author uses the above two formulas for position coding design. ‘pos’ indicates the position of a word and I indicates the dimension of a word.

3. Bidirectional Encoder Representation from Transformers (BERT): The structure of the BERT model is shown below:



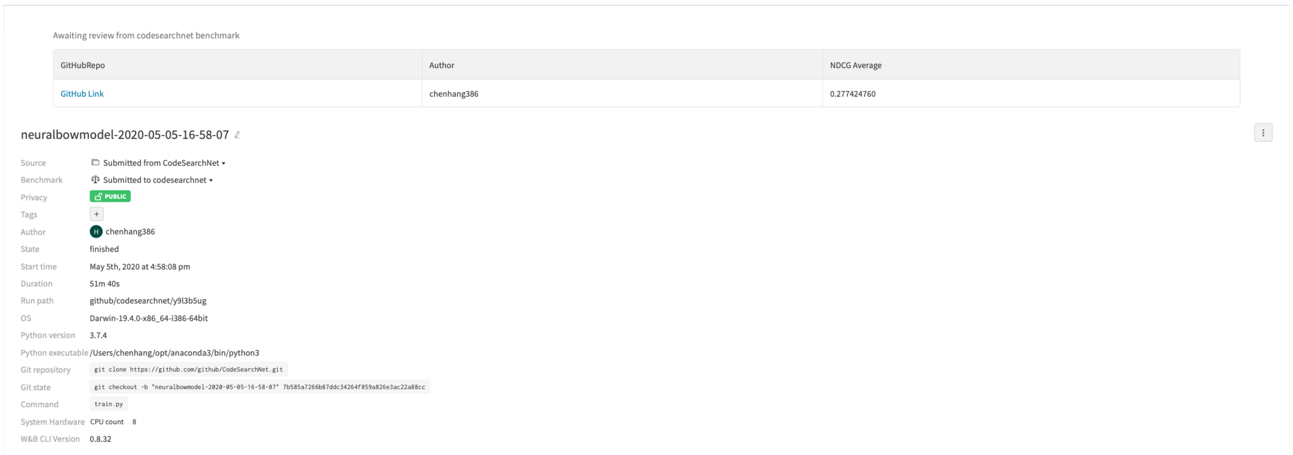
There are some techniques that BERT used:
- Masked LM: Before entering the word sequence into BERT, some of the words will be randomly selected (the data given in the paper is 15%) and labeled with MASK. In other words, these masked word cannot be seen by the model. The model then attempts to predict the original value of the masked word based on the context provided by other unmasked words in the sequence.

- Next Sentence Prediction: During the BERT training process, the model receives pairs of sentences as input and predicts whether the second sentence in the original document is also a subsequent sentence. During training, 50% of the input pairs are contextual in the original document, and the other 50% are randomly composed from the corpus and are disconnected from the first sentence. To help the model distinguish between the two sentences in training, the input is processed in the following ways before entering the model:
1. Insert the [CLS] tag at the beginning of the first sentence, and the [SEP] tag at the end of each sentence.
2. Add a sentence embedding representing sentence A or sentence B to each token.
3. Add a position embedding to each token to indicate its position in the sequence.
The below figure show how it works:


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Experiment and Result

In Semantic Code Search I have tried BOW model and BERT. But I only finished training BOW model and submit it to the leaderboard because training a BERT model for 1 Epoch on only Python language will cost my computer 12 hours. So I have to shut it down. Here I post my BOW result with NDCG Average Score 0.277


Awaiting review from codesearchnet benchmark

GitHubRepo	Author	NDCG Average
GitHub Link	chenhang386	0.277424760

neuralbowmodel-2020-05-05-16-58-07

Source	Submitted from CodeSearchNet
Benchmark	Submitted to codesearchnet
Privacy	PUBLIC
Tags	+
Author	chenhang386
State	finished
Start time	May 5th, 2020 at 4:58:08 pm
Duration	51m 40s
Run path	github/codesearchnet/y9l3b5ug
OS	Darwin-19.4.0-x86_64-i386-64bit
Python version	3.7.4
Python executable	/Users/chenhang/opt/anaconda3/bin/python3
Git repository	`git clone https://github.com/github/CodeSearchNet.git`
Git state	`git checkout -b "neuralbowmodel-2020-05-05-16-58-07" 7b585a7266b87ddc34264f859a826e3ac22a88cc`
Command	`train.py`
System Hardware	CPU count 8
W&B CLI Version	0.8.32