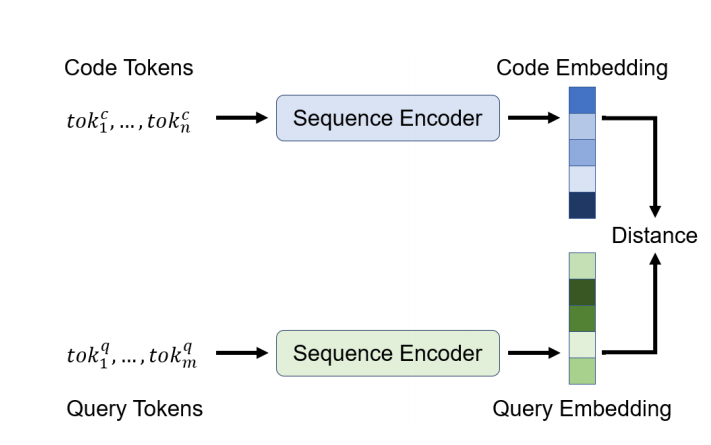
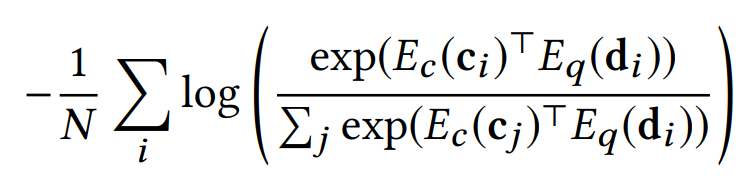
Project descriptions

The CodeSearchNet challenge requires a model to find out the most relevant code to given natural language among several candidate code samples. The original paper proposed an baseline technic to solve this problem: Joint Vector Representations for Code Search. Their architecture employs one encoder per input (natural or programming) language and trains them to map inputs into a single, joint vector space. The training objective is to map code and the corresponding language onto vectors that are near to each other, as we can then implement a search method by embedding the query and then returning the set of code snippets that are “near” in embedding space.



The loss are calculated as this. Given a set of N pairs (ci , di) of code and natural language descriptions and have instantiated a code encoder Ec and a query encoder Eq. The model was trained by minimizing the loss:



In the evaluation phase, the model will be tested on 4026 annotations across six programming languages and prioritized coverage over multiple annotations per query-snippet pair. NDCG score will be calculated to indicate how the model performs.

Many encoders can used for this challenge for example, Neural Bag of Words, Bidirectional RNN models, 1D Convolutional Neural Network, Self-Attention and so on. We tested the Neural Bag of Words and 1D Convolutional Neural Network and noticed that Neural Bag of Words preforms far more better than the other one. The original paper also mentions that Neural Bag of Words works better than other method listed. So our experiments are focused on Neural Bag of Words model. This model use simple embedding technics which each (sub)token is embedded to a learnable embedding (vector representation). After that the token sequence are combined into a sequence embedding using a pooling function which map the token sequence in to a fix dimension：128. Mean/max-pooling and an attention-like weighted sum mechanism can be implemented for this step.

Experiment Results

**Baseline result: (Neuralbow)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Mean NDCG | go | java | javascript | php | python | ruby |
| 0.189848815 | 0.1456 | 0.2031 | 0.1516 | 0.1547 | 0.2336 | 0.2506 |

Note this base line result was acquired from a submit in the challenge leaderboard

<https://app.wandb.ai/github/codesearchnet/runs/1oje6ykg/overview>

We have also run the base line model by ourselves but we cannot submit it to the leader board(the project owner closed our pull request)

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/1l4swj2a?workspace=user-xinchi>

We noticed that the baseline model was trained with six different language and then tested on six different language. It is intuitive to come up with the idea like this, If we use data of single language to train the model and test the model on the same language, will this lead to a improvement to the model performance. So we do the following experiment. Train models for each language with single language only. The results are listed below.

**The result of models trained for specific language(ie trained on python, tested on python**)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| go | java | javascript | php | python | ruby |
| 0.115372674 | 0.196455293 | 0.128024427 | 0.196455293 | 0.277846501 | 0.153800662 |

Weight&bias link:

Python

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/17znu4sm/overview?workspace=user-xinchi>

Go

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/3vq9oqee/overview?workspace=user-xinchi>

Javascript

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/2edi1vq2?workspace=user-xinchi>

Java

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/1r2nueh7?workspace=user-xinchi>

PHP

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/nwzx7kyh/overview?workspace=user-xinchi>

Ruby

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/10nxneiv/overview?workspace=user-xinchi>

Form the result we find out that for python and PHP the NDCG score have significant improvement. But the NDCG score of javascript, ruby and go have significant decrease.

After that, our works are mainly focus on Python.

We noticed that the Elastic Search method works better than any other machine learning models which trying to get the inner relationship between tokens. And any programming language has some stop words and some special characters. These words and characters appears frequently and appears in almost every possible results. So we implement a mechanism to remove or change these special tokens before the model load them.

The performance of neuralbow model trained with data which remove special tokens are listed below:

|  |  |
| --- | --- |
| Removed tokens | NDCG(python) |
| 1.Remove all reserved words and special characters | 0.160823206 |
| 2.Remove reserved words only | 0.275305682 |
| 3.Remove special characters only | 0.201 |

Links to experiment results:

1: <https://app.wandb.ai/xinchi/CodeSearchNet/runs/3u5ye95x?workspace=user-xinchi>

2: <https://app.wandb.ai/xinchi/CodeSearchNet/runs/2y836jl1?workspace=user-xinchi>

3: <https://app.wandb.ai/xinchi/CodeSearchNet/runs/2rqvwioa?workspace=user-xinchi>

**Reserved words:**

'False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def',

'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'

**Special characters:**

'(', ')', '[', ']', '{', '}', ',', '\'', '\"', ':', '+', '-', '\*', '/', '%', '=', '.', '+=', '-=', '==','!=', '.','"\\n"', '"\\t"', '>', '<'

Base on the experiment results, we found out that simply remove these special word lead to a drop in NDCG scores. This means these special words is somehow necessary for code search, even though they are not distinct features for code search. We then developed another strategies to group these special words and change all words in each group to a special token.

The grouping strategies are listed below:

**All special tokens to one token:**

'False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def',

'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield''(', ')', '[', ']', '{', '}', ',', '\'', '\"', ':', '+', '-', '\*', '/', '%', '=', '.', '+=', '-=', '==','!=', '.','"\\n"', '"\\t"', '>', '<'

TO ‘[s]’

**Categorized by reserved words and special characters:**

'False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def',

'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'

TO ‘[w]’

'(', ')', '[', ']', '{', '}', ',', '\'', '\"', ':', '+', '-', '\*', '/', '%', '=', '.', '+=', '-=', '==','!=', '.','"\\n"', '"\\t"', '>', '<'

TO ‘[c]‘

**Categorized by function:**

'False', 'None', 'True', 'as', 'assert', 'async', 'await', 'class', 'continue', 'def', 'del','finally', 'from', 'global', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'with', 'yield'

TO ‘[w]’ (note that ’for’,’while’,’if’,’else’,’try’,’’except’,’elif’,’return’ are not considered as special word in this case)

',', '\'', '\"', ':', '"\\n"', '"\\t"' TO ‘[c]’

'(', ')', '[', ']', '{', '}'TO ’[q]‘

'+', '-', '\*', '/', '%', '=', '.', '+=', '-=', '==', '!=', '>', '<' TO ‘[O]’

**The model performance with these three grouping strategies are listed below:**

|  |  |
| --- | --- |
| Strategies | NDCG(python) |
| All special tokens to one token | 0.248670152 |
| Categorized by reserved words and special characters | 0.263144392 |
| Categorized by function | **0.281984870** |
| Baseline result(python only) | 0.277846501 |
| Baseline result(original) | 0.2336 |

Links to experiment results:

All special tokens to one token

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/25z4bgqs/overview?workspace=user-xinchi>

Categorized by reserved words and special characters

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/2rybmbx4?workspace=user-xinchi>

Categorized by function

<https://app.wandb.ai/xinchi/CodeSearchNet/runs/2m3dx3s2?workspace=user-xinchi>

Conclusions

The experiment results shows that for python and PHP the model’s NDCG score have significant improvement. Some special words in codes are necessary for code search. But if we can carefully group these special word and change them to one token for each group, we can also boost up the model performance.