Abstract

Code search is a foundational task in software development, which studies semantic similarity between natural language queries and program code. Recent years have witnessed great progress made in code search. However, researchers tend to pretrain a code representation model in different program languages, so that it can be used in code related downstream problems, and then fine-tune it in code search with a specific program language. Our empirical study shows that there may be performance damage when fine-tuning code search in multiple program languages. We believed that it is caused by the disentanglement between code identifier information and code sentiment information. Therefore, we proposed a search enhancement framework based on GAN, so as to reduce identifier information provided by pretrained models.

Introduction

Code search plays an important role in the software development process, which is an essential field of Software Engineering and studies the semantic similarity between natural language queries and program code. Recent years have witnessed a huge increment in source code. The statistic shows that more than 60 million new projects have been created only in 2020. Thus, code search engines can improve the development efficiency of program developers, enabling them to search for existing code or examples of some application programming interface (API), instead of “rebuilding wheels”.

As deep learning has grown by leaps and bounds in recent years, a number of methods have been proposed in code search. such as Recurrent Neural Network (RNN) based models (\citealp{DeepCS}), Convolutional Neural Network (CNN) based models (\citealp{CQIL, ShuaiX0Y0L20}), graph based models (\citealp{, GuCM21}), and Pre-trained Language Models (PLMs) based models (\citealp{CodeBERT, CoCLR, GuoLDW0022, GuoRLFT0ZDSFTDC21}).

From the view of PLMs, all of them have well performance in code search, with complex model architecture and advanced training techniques. However, PLMs often treat code search as a downstream task, which means researchers can pre-train a model with hybrid objectives and multiple program language code data, then fine-tune it in a specific program language for code search (CITECITE). Our empirical study shows that there might be a performance decline when fine-tuning with data in multiple languages. Table 1 shows MRR comparisons between single program language fine-tuned models and multiple program language fine-tuned models. Analogous to multilingual models, code information can be roughly divided into identifier information, which distinguishes code from another code written in different program languages, and sentiment information, which reveals its intention. In code search, code sentiment information is only needed, for it is matched with specific queries. Identifier information may confound model training and decrease performance when fine-tuning with multiple program language data.

Therefore, we want to reduce the identifier information of the embedding vector given by code search pre-trained models. We followed the idea of GAN (Generative adversarial neural network, CITE CITE) and propose a search enhancement framework. Our method consists of two additional networks: a generator, which aims to generate code search embedding vector with less identifier information, and a discriminator, of which the purpose is to identify the language feature. We treat the discriminator as a classification problem. When there is less identifier information, the discriminator may not classify the data well. Fig 1 illustrates our framework architecture.

Reveal the performance decline problem that appears when fine-tuning code search with multiple program languages.

We propose a search enhancement framework using GAN. It can reduce the identifier information of the embedding vector provided by code pre-trained networks, in order to focus the model on sentiment information.

Performance comparisons with different fine-tuning strategies

## Related work

Our work is related to the following two fields:

Code Search.

Recent years’ works adopt deep learning models in code search, of which the idea is to embed natural language queries and program code into vectors and then calculate their similarity score. Models in code search can be roughly divided into the following aspects:

(1) RNN based models. Et al. build two networks to embed queries and program code into vectors, respectively. Cosine similarity is used to compute the similarity between the vectors.

(2) CNN based models. Followed by (), Et al. construct “name-query” and “body-query” latex match matrix with fastText and use CNN (convolutional neural network) to extract features.

(3) PLMs. Et al. (CodeBERT) followed the idea of BERT (\citealp(BERT)) and proposed a pre-train model training on CSN (\citealp(CSN)) dataset with hybrid objective functions: MLM (Masked Language Modeling) and RTD (Replaced Token Detection). GraphCodeBERT leverage data flow to enhance code representation. For the code data flow, they specifically designed a series of pre-training tasks: MLM, Edge Prediction, and Node Alignment. (UniXcoder) tranfrom code AST (Abstract Syntex Tree) to a sequence structure, thus the pre-training can utilize multi-modal contents.

The challenge of code search is to effectively measure the

semantic similarity between natural language queries and program code.

More specifically, it is to find the semantically best matching

answer from several candidate codes when facing query statements

entered by developers. Code search methods can be divided into two categories:

information retrieval based models and deep learning based models.

The former is usually based on keyword matching.

Researchers have proposed code search models with diverse network architecture:

(1) RNN based models (\citep{DeepCS}). (2) CNN based models (\citep{CQIL, ShuaiX0Y0L20}). (3) PLMs (\citep{CodeBERT, CoCLR, GuoLDW0022}).

(4) Graph based models (\citep{GraphCodeBERT, GuCM21}).

In all of them, the core idea is to measure the similarity between queries and code.

Common to these approaches is the conversion of queries and

codes into high-dimensional embedding vectors.

Related work

Recent years’ works adopt deep learning models in code search, of which the idea is to embed natural language queries and program code into vectors and then calculate their similarity score.

Fig 2 shows the framework of code search: a code encoder, a natural language encoder, and a similarity calculator. The encoder transforms code or queries into high-dimensional vectors, which are considered semantic information. The calculator o computes the similarity between the two vectors.

Models in code search based on deep learning can be roughly divided into the following aspects:

Methodology

We propose two strategies to improve code search performance based on the assumption that identity information will confuse the model when fine-tuning multiple language data. Instead of changing the structure and pre-training the whole code search model, we focus on transforming the model output embedding vectors during the fine-tuning process, which is less computing complexity.

The first strategy is leveraging GAN to eliminate identity information. The second strategy is to distance the KL divergence (CITE) between identity embedding and semantic embedding.

Identity and semantic information. The identity information we consider as the syntax signal of different program languages, which distinguishes them from each other. We consider the identity information as the signal that the model can classify each program language data correctly. The semantic information is the intention of a code snippet, which describes the function of the code.

Strategy 1.

In this section, we introduce a GAN-based enhance network, which contains a generator and a discriminator. We start by introducing the basic idea and the model structure. Then we describe the training procedure in this paper in detail.

The GAN structure is used widely in image generation. Generally, it contains two sub-network: a generator, which aims at generating data of a specific distribution, and a discriminator, which intends to determine the generated data if it is in the ideal distribution. In this paper, we introduce an MLP (Multilayer Perceptron) as the generator, of which the purpose is to generate high-dimensional vectors free of identity information. We also use another MLP as the discriminator. The object of the discriminator is to determine whether the vector generator produced contains identity information. The structure of the network can be seen in Fig (CITE).

The loss function of the generator is given by Eq 1.

Strategy 2

Although GAN has a strong ability of learning and manipulating data distribution, it is hard to be trained, and can easily suffer from gradient vanishing (CITE). Therefore, we propose an MLP-based network to disentangle identity and semantic information. The overlook of the network can be seen in Fig.

The network takes embedding vectors extracted by CodeBERT as the input. Then, the input vector will be put into MLPs and can obtain two high-dimensional vectors: and We hope that possesses identity information and possesses semantic information. Therefore, we use V1 for classification and V2 for code search, in order to extract identity and semantic information, respectively. Then, we maximize KL divergence, which is illustrated in Eq, between V1 and V2. Finally, V2 can be viewed as the sentence embedding of an input code snippet without identity information and can be further used for code search.

According to the above, learning the disentangle network involves finding parameters that minimize the hybrid loss on the given dataset, and the objective is given as Eq 4. Besides, Eq2 represents the code search loss, intends to keep search performance, where sim is the similarity score and is the ith code and is the ith query. is the classification loss, the purpose is to extract identity information. leverages KL divergence to maximize the distribution of the embedding vectors output by the disentangle network.

hyperparameters

Experiment

1. Experiment Setting

Dataset. The dataset we used is the Code Search Net dataset preprocessed by GraphCodeBERT. The parameter of the dataset is given below. We remove the data of Ruby and Javascript, as they are much less than other data.

Baseline model. We choose CodeBERT as the baseline model. It plays an important role in extracting code and query features. It is the encoder that first embed a code snippet or a natural query into a high-dimensional vector.

Disentangle network. We use multiple linear perceptron as the basic layers of the network. We use ReLU as the activation.

Training setting. We use an RTX 3090 GPU for training. The batch size of the training procedure is 128. The total training epoch number is 3. We use package of Transformers 4.24.0 and Pytorch 1.12.0, based on Python 3.8.13.

Analysis of training process.

Disentangle Network 1.

We illustrated the change of discriminator loss during training. It is obviously that the loss of the discriminator is able to converge, while there is still slightly increment in some period, e.g., step 50 and step 100. Fig 6 shows the loss change of the generator during training. There might be overfitting, as the loss first decreases and then rises with increasing iterations.

Disentangle Network 2.

We list hybrid loss change of network based on strategy 2 in Fig 7. It is apparent the loss is decreasing and able to converge. Fig 8 represents the evaluation index change in the training period, using MRR, which is shown in Eq, as the index and with the evaluation step of 2, 000.

Analysing embedding vector after disentangling.

Intuitively, identity and semantic information can be divided easily after applying the disentangle strategy we proposed. Thus, we first use t-SNE, which is a dimensionality reduction method, to

reduce high-dimensional vectors to 2 dimensions. Then we visualize the 2-dimensional vectors. We sample 5, 000 code snippets from the whole dataset. And then draw the embedding graph. For the strategy 1 network, we visualize the vectors output by the origin CodeBERT and the generator. For the strategy 1, we illustrate embedding vectors output by CodeBERT and disentangle network. Compared to the CodeBERT vectors, i.e., blue points, the generator output, i.e., yellow points, is more concentrated in the same area. For the strategy 2, we visualize v1 and v2 vector, which is considered have identity and semantic information, respectively. From the graph we can see that, two types of the vectors can basically be separated.

Analysing code search result.

In this section, we try to figure out the code search performance of two disentangle networks. We use MRR as the evaluation indicator, which is the reciprocal of the correct result among all returned results and it is given as Eq~\ref{MRR}, where Q is the total test number and $rank\_i$ represents the correct rank among all results.

Table 1 shows the code search MRR results. The left first column indicates the language of data. The top row indicates the strategy we proposed.

From the results we can see that there is a small performance increment in Python, compared to the original results in~\ref{section:1}, while a slightly decrease occurs in other languages.

It is clear that strategy 1 and strategy 2 can reach a similar result, while Java and Php data have a better MRR in strategy 1, and Python and Go data are better in strategy 2.

Discussion and Conclusion

In this paper, we propose two disentangle strategies for tackling performance decline when fine-tuning CodeBERT for the code search task with multiple language data. In strategy 1, we leverage a GAN based model, to eliminate identity information. In strategy 2, we use MLPs to divide embeddings by maximize their KL divergence. Experiments show our work achieve some results.

However, we only consider identity information as the signal that distinguish program languages from each other, e.g., a classification task. Whether the identity information is equal to syntax information, needs more experiments to proof.