Machine Learning in Analyzing Linguistic Markdown and Code Prediction

Yifan Zhang [1] **Yitao Long** [1] **Anna Xie** [1]

Abstract

Understanding the relationships between code and comment could lend to improvements across many aspects of AI-assisted development. In this paper, we investigate and compare the performance of different methods to reconstruct the order of markdown cells in a given notebook based on the order of the code cells, demonstrating comprehension of which natural language references which code. The result shows that DeBERTa, retrained under a Masked Language Model (MLM) first and added a BiLSTM layer onto its output of the last transformer layer, obtains the best performance. And the score is 0.8565 under the evaluation metric called Kendall Tau Correlation.

1. Introduction

Machine learning is considered to be one of the biggest innovations nowadays. The need for Machine Learning is high in demand due to the evolving technology and the generation of huge data. And recent progress has greatly improved the performance of many natural language processing tasks. Models like BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019) learn effective contextual information from self-supervised objectives such as Masked Language Model and Next Sentence Prediction, thus improving the state-of-the-art performance of many tasks.

Understanding the relationships between code and comment is also an NLP task, which can assist software developers greatly. For instance, it could help us to do the construction of better data filtering and pre-processing pipelines for model training, as well as automatic assessments of the readability of code. Recent works usually focus on one-to-one pairs of programming language and natural language. In our work, each piece of data is a

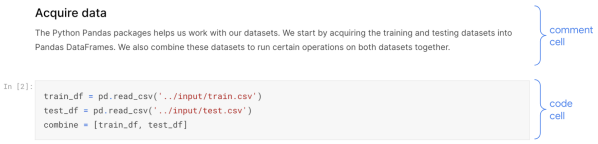

Figure 1. Example of notebook cell

Jupyter notebook, it contains multiple code cells and markdown cells, and the situations that consecutive code cells and markdowns cells can exist, rather than one code cell corresponding to one markdown cell. And we are going to predict the correct ordering of markdown cells based on the order of code cells. The example format of data is shown in Figure 1.

Moreover, the length of the data also becomes a potential problem that we need to consider. Therefore, we propose two ways to deal with long text. The first one is to encode each cell separately. The second way is to use the larger model to encode as many as possible cells at a time. In the first method, we use UniXcoder (Guo et al., 2022) to get the representation of each cell and obtain the position of markdown cells based on similarity. In the second method, we use two models, DeBERTa (He et al., 2021) and CodeBERT (Feng et al., 2020b), to encode multiple cells at a time, and add one BiLSTM layer after the output of BERT variant models to obtain the prediction of the ordering of markdown cells.

2. Related Work

2.1. Multi-Modal Pre-Trained Models

The success of the pre-trained model in NLP drives the development of multi-modal models. Usually, those methods are learned from bimodal data such as CLIP (Radford et al., 2021) learns from pair of vision and text; VideoBERT (Sun et al., 2019) learns from pair of video and text data. Similarly, CodeBERT and UniXcoder are learned by the bimodal data of programming languages and natural language. The differences between CodeBERT and UniXcoder from the above-mentioned models are these two models are not only learned by pairs of different programming languages and

[1]New York University, Courant Institution of Mathematics, New York, United States. Correspondence to: Yifan Zhang <yifan.zhang@nyu.edu>, Yitao Long <yl8251@nyu.edu>, Anna Xie <awx201@nyu.edu>.

their corresponding comments but also learn on unimodal data, i.e. the programming language or comment itself.

2.2. Long Text in NLP

In common tasks, researchers can solve problems by means of BERT models. However, when it comes to long text, in specific, the number of tokens is much larger than 512, which is usually the maximum length of BERT models, we need some other ways to deal with it. Typically, sliding windows is one of the common methods such as Hierarchical Transformers (Nawrot et al., 2021). Using models can support longer text in another way such as Longformer (Beltagy et al., 2020). Long text of data usually includes a large number of unimportant sentences, so researchers propose some extractive methods to extract important contents related to the specific task. This type of method includes fastText (Bojanowski et al., 2016), DPR (Karpukhin et al., 2020) and etc.

2.3. Ordering Task

The problem of sentence order can be formulated as finding an order with maximum coherence. Typically, there have two types of models. One is a pairwise model. The pairwise model first predicts the order of pairwise sentences, then rank the final order. For example, (Chen et al., 2016) ranks the order by summing up scores in pairwise sentences. The second type is the generation model. For instance, (Chowdhury et al., 2021) uses BART to reconstruct the order in the decoder part.

3. Methods

3.1. UniXcoder

UniXcoder, which is known as a unified cross-modal pre-trained model for programming language. The model uses mask attention matrices with prefix adapters to control the behavior of the model and leverages cross-modal contents and code comments to enhance code representation. It utilizes a similar architecture as RoBERTa in the encoder.

Here, we firstly retrained UniXcoer under MLM task. Then, obtain the representation of cell from last layer hidden-state of the first token of the sequence. And we can rank cells based on cosine similarity.

3.2. DeBERTa

DeBERTa, which is known as the decoding-enhanced BERT with Disentangled Attention. It is a transformer-based neural language model for improving the BERT and RoBERTa models with disentangled attention mechanism and enhanced mask decoder. Disentangled attention mechanism means DeBERTa disentangle the word embedding and position embedding. Also, positional embedding here is relative position. So, it lacks of absolute information of sequence, and that is why they incorporate absolute position embedding after last transformer layer. The comparison of BERT and RoBERTa at decoding layers is shown in figure 2.

Here, we also retrained the DeBERTa under MLM task. Then in the downstream ordering task, we add one more BiLSTM onto ouput of final layer of DeBERTa, and then forward to a fully connected layer to predict the value for each markdown cell.

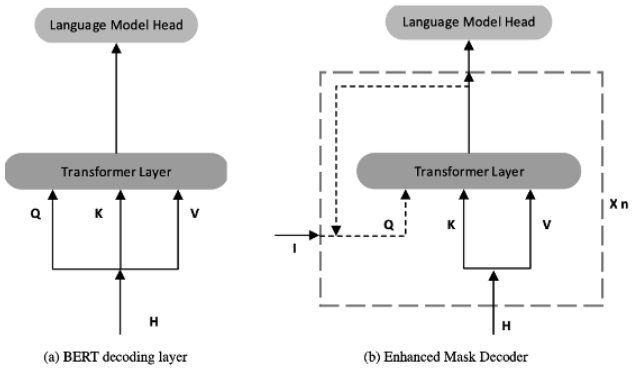

Figure 2. Comparison of the decoding layer

3.3. CodeBERT

CodeBERT, is also a bimodal pre-trained model first introduced in 2020 (Feng et al., 2020a). The interesting thing in CodeBERT is that it uses a concept of "Pair" in processing dataset. CodeBERT is based on Transformer neural architecture, and are trained to detect tractable sampled from generators. It can process programming language such as Python, Java and JavaScript and natural language pairs which close under our topic that finding the relationship between users' code and notations in Juypter Notebooks.

4. Experiments

In this section, we did experiments between UniXcoder, DeBERTA and CodeBERT model with following dataset. The hardware we used to train is 2 V100 GPU.

4.1. Dataset

The data set consists of 139,256 Jupyter Notebook Code in .json files, published by the Jupyter Community. It is downloaded from Kaggle's competition "Google AI4Code – Understand Code in Python Notebooks". The evaluation metric is called Kendall-Tau Correlation. It is a coefficient that

Table 1. Description of Dataset

CELL TYPE	MIN	MAX	MEAN
CELL	2	1005	45.75
CODE CELL	1	809	22.87
MARKDOWN CELL	1	537	22.88

Table 2. Average Tokens

CELL TYPE	MEAN
CODE CELL	41.79
MARKDOWN CELL	33.10

represents the degree of concordance between 2 columns of ranked data. The greater the number of inversions, the smaller the coefficient. Table 1 describes some statistics of this dataset. For example, the minimum number of cells in a sample is 2, while the maximum is 1005. And Table 2 demonstrates the average tokens in different types of cell.

4.2. UniXcoder

We encode each cell separately in a file and obtain their representations. Then we can simply rank the markdown cells based on cosine similarity.

First of all, calculate the mean of every two consecutive representation of code cells. If the code cell rank first in a file, then the representation is the code representation itself. Then for each markdown cell, compare the similarity of the markdown cell to all meaned code cells and the position of a markdown cell is the highest similarity between this markdown cell to a meaned code cell. For instance, if a markdown cell $markdownA$ has the highest similarity to mean of $codeB$ and $codeC$. So the position of this markdown cell is among this two code cells.

In this dataset, there may have several markdown cells between two code cells. At this time, we would compare the similarity of markdown cells and the back code cell. For instance, $markdownA$ and $markdownB$ were predicted between $codeC$ and $codeD$. Now we compare the similarity of $markdownA$ to $codeD$ and $markdownB$ to $codeD$. The higher similarity, the closer to the $codeD$. The reason is that we usually write some comments after a code cell and then describe what we are going to do before we start writing next code cell. In inference, the score we get at this method is 0.7303.

4.3. DeBERTa

Using DeBERTa, we normalized the rank of the markdown cells to the range [0, 1]. At this time, we encoded as many units of cells as possible and added a special token between two cells. Then after the output of final fully connected layer, we used MAE loss to evaluate the differences between the output and normalized rank value from the corresponding output of special token.

Table 3. Ablation Study of DeBERTa

MODEL	SCORE
DEBERTA	0.8255
DEBERTA WITH MLM	0.8348
DEBERTA WITH MLM, BILSTM HEAD	0.8565

Since the model is large, we used 2 GPUs to train, the batch size is 2 and learning rate is 1e-6, the training loss is shown in Figure 3. The score we finally got was 0.8565, and we also evaluate the score by using another and smaller model called CodeBERT in the same way, which is decribed in Section 4.4.

In order to evaluate the effect of MLM task and BiLSTM head, we did an ablation study, the result is shown on Table 3. From Table 3, we can know that retrained under MLM helps in performance. And BiLSTM layer could make model extract more bidirectional information from data.

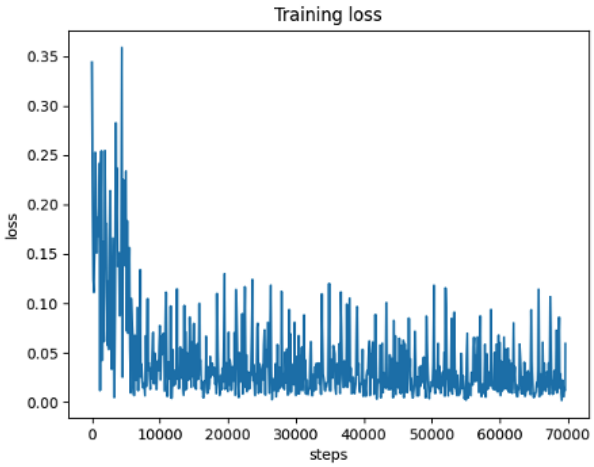

Figure 3. Training loss of ordering task

4.4. CodeBERT

On the third approach, we used CodeBERT, a bimodal Pre-trained model for programming languages and natural language. It produces general-purpose representations that can broadly support NL-PL understanding tasks

and generation tasks. It uses a multi-layer bidirectional Transformer as the model architecture to pair Natural Language (Markdown) and Programming Language(Code) with MLM.

Shown in Figure 4, both natural language and programming languages are masked into the generator and archive masked sample to the discriminator. NL-Code discriminator is used for producing general-purpose representations in the fine-tuning step.

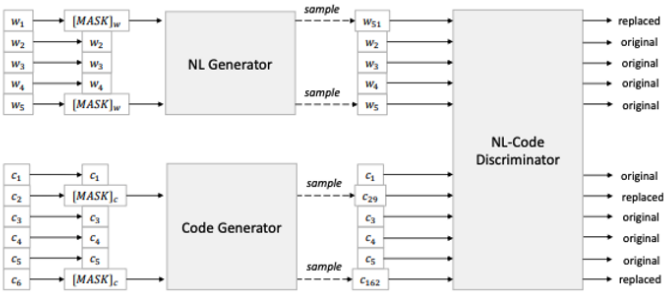

Figure 4. Illustration about the replaced token detection objective

Using an encoder and decoder in the fine-tuning step, we initialized the encoder of a generative model with CodeBERT. We calculated the loss function using the equation:

$$L_{MLM}(\theta) = \sum_{i \in m^w \ \& \ m^c} -logp^{D_1}(x_1 | w^{masked}, c^{masked}) \tag{1}$$

The final score we got was 0.812, with a training loss decent shown in Figure 5.

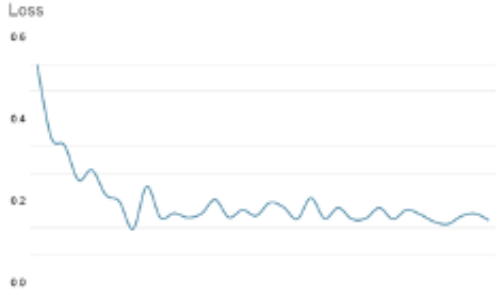

Figure 5. Training loss of ordering task

5. Discussion and Conclusion

In this study, we attempted to understand the relationship between code and comment cells through python notebooks in order to explore new ways that machine learning can assist software developers.

We experimented with three different approaches: using models UniXcoder, DeBERTa, and CodeBERT. Looking at the result score, DeBERTa obtains the best performance with a score of 0.8565, compared to the score of UniXcoder 0.7303, and the score of CodeBERT is 0.812.

We did the ablation studies for DeBERTa and it shows that retraining DeBERTa under MLM task first and adding a BiLSTM layer into the output of DeBERTa can improve the performance of the model.

The possible reason that UniXcoder performs the worst is only using a UniXcoder is not enough to identify the differences between the cells. Later, the difference in performance between DeBERTa and CodeBERT shows that training with the larger model can improve performance.

We believe all the performance can still be improved. We only run one epoch in both MLM and ordering task since training takes much longer time than our expected because of the volume of data and model size even though we take advantage of NYU HPC and data parallelism in our model.

References

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching word vectors with subword information, 2016. URL https://arxiv.org/abs/1607.04606.

Chen, X., Qiu, X., and Huang, X. Neural sentence ordering, 2016. URL https://arxiv.org/abs/1607.06952.

Chowdhury, S. B. R., Brahman, F., and Chaturvedi, S. Is everything in order? a simple way to order sentences, 2021. URL https://arxiv.org/abs/2104.07064.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., and Zhou, M. Codebert: A pre-trained model for programming and natural languages. *CoRR*, abs/2002.08155, 2020a. URL https://arxiv.org/abs/2002.08155.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., and Zhou, M. Codebert: A pre-trained model for programming and natural languages, 2020b. URL https://arxiv.org/abs/2002.08155.

Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., and Yin, J. Unixcoder: Unified cross-modal pre-training for code representation, 2022. URL https://arxiv.org/abs/2203.03850.

He, P., Gao, J., and Chen, W. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2021. URL https://arxiv.org/abs/2111.09543.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering, 2020. URL https://arxiv.org/abs/2004.04906.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

Nawrot, P., Tworkowski, S., Tyrolski, M., Kaiser, , Wu, Y., Szegedy, C., and Michalewski, H. Hierarchical transformers are more efficient language models, 2021. URL https://arxiv.org/abs/2110.13711.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Sun, C., Myers, A., Vondrick, C., Murphy, K., and Schmid, C. Videobert: A joint model for video and language representation learning, 2019. URL https://arxiv.org/abs/1904.01766.