6th place solution of Google AI4Code – Understand Code in Python Notebooks

Shujun He

1 Background of our team

I participated in the recent competition "Google AI4Code – Understand Code in Python Notebooks", which was hosted during the third quarter of 2022. My team name is Shujun. My final score is 0.9100.

Our team consistes of me, who is a PhD student in chemical engineering at Texas A&M University in College Station, TX.

2 Solution details

2.1 Relative to absolute position prediction

My solution predicts the relative markdown cell positions with respect to every one of the code cells and then calculates the absolute position by averaging all absolute position predictions for each markdown cell:

$$\hat{p}_i = 1/N_{code}(\sum_{j=1}^{N_{code}} \alpha_j + \hat{\delta}_{ij}), \tag{1}$$

where $\hat{p}_i$ is the predicted absolute position for the ith markdown cell (markdown cells are initially randomly shuffled, so i carries no real information), N_{code} is the number of code cells, α_j is the absolute position for code j, and $\hat{\delta}_{ij}$ is the predicted relative position for markdown cell i with respect to code cell j. Absolute positions of code cells are simply assumed to be linearly spaced out between 0 and 1; because the code positions relative to each other is known, the absolute positions can be calculated.

2.2 Network architecture

My network architecture consists mainly of a deberta encoder and a GRU layer; an overview is shown in Figure 1. First, markdown blocks and code blocks are tokenized and concatenated. During training, 10 markdown blocks with 20 code blocks with a max length of 1100 are used first, then 10 markdown blocks with 40 code blocks with a max length of 1600 are used to train another 5 epochs, and then the weights are retrained with 20 markdown blocks with 80 code blocks with a max length of 3160. Following that, a deberta large encoder is used to encode the text, and then each markdown and code block is pooled. Next, features are added to pooled feature vectors (discussed in a later section). Each markdown block feature vector is then concatenated to the sequence of code block feature vectors, and then an GRU network is used to predict the relative position of that markdown block to each code blocks. Lastly, the absolute position of the markdown cell is computed according to Equation 1. MAE loss is used during training.

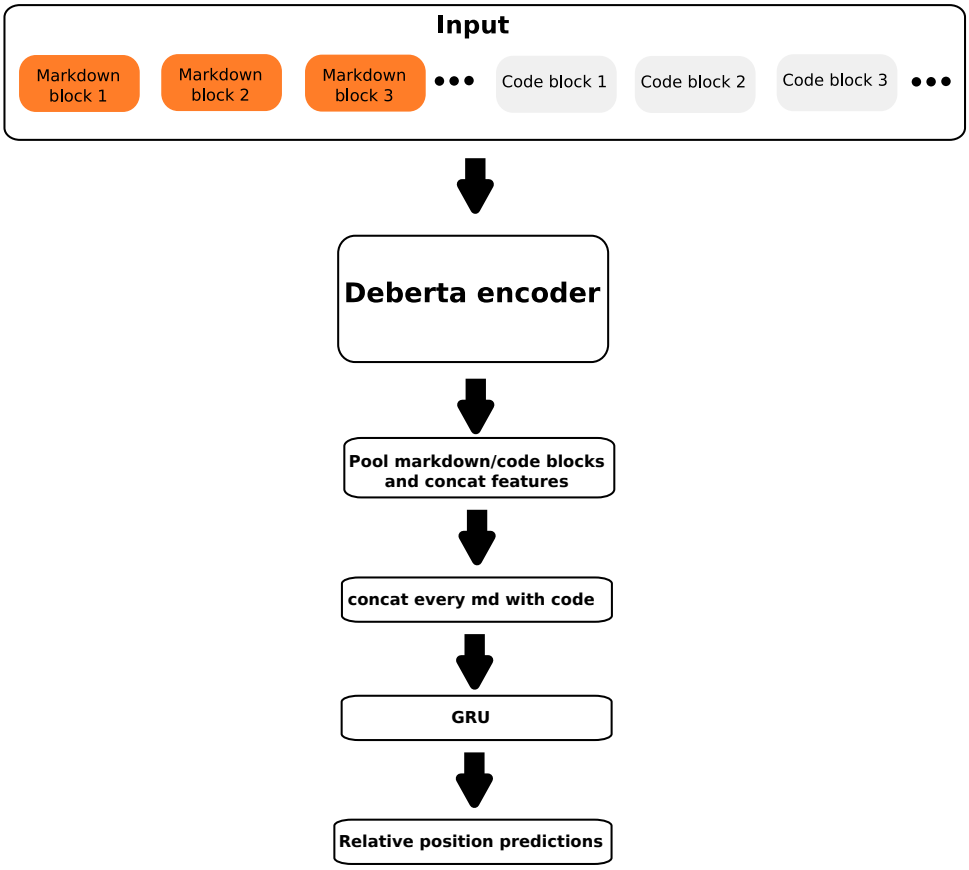

Figure 1: Network architecture overview

2.3 Features

The following features are used:

1. number of markdown cells in given notebook (normalized by 128)
2. number of code cells in given notebook (normalized by 128)
3. ratio of markdown cells to total number of cells

These features are concatenated with pooled feature vectors of markdown cells. Pooled code cells feature vectors are also concatenated with their absolute positions.

2.4 MLM

Masked langauge modeling pretraining is used, where the input text is arranged in the same manner as in finetuning. MLM probability was set to 0.15 and the model is trained for 10 epochs. A max sequence length of 768 is used with at most 5 markdown cells and 10 code cells.

2.5 Inference and TTA

During inference, I tried to predict positions of as many markdown cells as possible. I was able to use a max length of 4250 with 100 code blocks and 30 markdown blocks. Code blocks have at most 26 tokens and markdown blocks 64. Notebooks are sorted based on number of markdown/cell blocks to reduce inference time. Because for longer sequences where there are more than 30 markdown blocks, I cannot make predictions for all of them at the same time, and multiple forward passes are needed to make predictions for all markdown blocks. As a

result, the model never sees all the markdown blocks at the same time. To counteract that, I do test time augmentation by randomly shuffling the markdown cells each time a new inference cycle is started and therefore the model will see different combinations of markdown cells in long sequences. For shorter sequences, this simply changes the order of markdown cells during inference. For my best inference notebook, a total of 3 inference cycles are used (3xTTA). 3xTTA improves from score from 0.9067 to 0.9100.

3 What didn't work

1. Trying to translate markdown cells that are not in English did not result in any improvements for me. I tried facebook's m2m100_418M.
2. Separating code and markdown cells and doing two forward passes through deberta for concatenated code and markdown cells.
3. Cubert did not work because I couldn't fix the tokenizer.
4. Maxpooling instead of meanpooling

4 References

I used the solution from Khoi Nguyen as a baseline: `https://www.kaggle.com/competitions/AI4Code/discussion/326970`.