CS310 Natural Language Processing - Assignment 5: BERT Pretraining
Total points: 60

Tasks

- Pretrain a mini-BERT model using the masked language modeling (MLM) task and next sentence prediction (NSP) task.
- Implement and test your model on a toy dataset in the provided A4_bert_toy.ipynb notebook.
- Train your model on a relatively larger dataset and evaluate its performance in the provided A4_bert_full.ipynb notebook.

Submit

- Two notebooks: A4_bert_toy.ipynb and A4_bert_full.ipynb; any dependent .py files

Requirements

1) Implement `make_batch` function (15 points)

This function is to prepare a piece of toy batch that is ready for the MLM and NSP objectives. The returned `batch` object contains `batch_size` number of the following list of tensors:

`[input_ids, segment_ids, masked_tokens, masked_pos, is_next]`

Here the first tensor `input_ids` contain the integer token IDs of two sentences A and B, with special tokens [CLS], [SEP] and [PAD] properly placed. In the following example:

`[1,3,16,6,21,27,23,7,2,19,8,14,2,0,0,…,0]`

In which the first element "1" is the ID for [CLS], and the second element "3" is the ID for [MASK]. "2" is for [SEP], and "0" is for [PAD]. The entire tensor is padded with 0s to the maximum length specified by the global variable `MAX_LEN`. Its value is 30 for the toy data. This tensor corresponds to text tokens as follows:

`['[CLS]', '[MASK]', 'my', 'baseball', 'team', 'won', 'the', 'competition', '[SEP]', 'oh', 'congratulations', 'juliet', '[SEP]']`

The second tensor `segment_ids` contains integer numbers of 1 and 2s (and 0 for padding as well) to denote the ranges of sentence A and B:

`[1,1,1,1,1,1,1,1,1,2,2,2,2,0,0,…,0]`

The third tensor `masked_tokens` contain the token IDs for all the masked tokens:

`[9,19,0,0,0]`

Together with the fourth tensor `masked_pos`:

`[1,9,0,0,0]`

It tells that the ground truth token at index 1 is "9" (mapped to "great") and the ground truth token at index 9 is "19" (mapped to "oh"). You should note that the lengths of masked_tokens and masked_pos are limited to 5, which is the value of the global variable `MAX_PRED`. That is, in principle 15% of tokens in each sentence are sampled for training, but it should not exceeds the limit. Also, if by chance it is down sampled, then the missing tokens are padded by 0, i.e., the [PAD] token.

Finally, the last tensor `is_next` is a single integer indicating whether sentence B is the actual one following sentence A in the corpus, 1 for true and 0 for false.

There two places you need to implement in `make_batch()`:

First, in the loop for processing the sampled candidates for masks, you need to decide whether to replace a token with [MASK] (80% of chance), or to replace with another randomly sampled word

from the vocabulary (10% of chance), or to leave it unchanged (10% of chance). In order to follow these chances, you can throw a dice by `random.random()`, and use the returned value (between 0 and 1) to decide the action you take: if it is smaller than 0.8, then replace the token with [MASK]; else, you should throw a dice one more time, and this time if smaller than 0.5 (think about why), then you randomly generate a token ID between 0 and `vocab_size-1` using `random.randint()`, and use it for replacement.

Second, towards the end of the while loop, decide whether sentence A and B are the actual adjacent pairs in corpus, and append the correct value for `is_next`.

Notes

- Although we call input_ids, segment_ids etc. as "tensors", but in the implementation, they are merely Python lists. It is okay because we can map them to `torch.LongTensor` in the final training step.
- For the toy data, we are creating only one batch to simulate the training, so in the while loop we use `random.randrange` to create random combinations of sentence A and B. But for the full data, you should use a more "serious" method to make sure that 50% percent of your pairs satisfy `is_next=True` and the other 50% as `is_next=False`.
- We are not using subword tokenization for this assignment. The full data is in Chinese (seven chapters of 《庄子》), and you can treat each character and punctuation as a word.
- About the processing of the full data: 10% of data are held out as the testing set, in which 15% of characters are already sampled and replaced with [MASK]; each line contains a pair of sentences (in string format) interpolated with [CLS] and [SEP] properly, but with no [PAD]; the end of line is appended with a tab "\t" followed by the `is_next` ground truth (0 or 1).

2) Implement the model (20 points)

The majority of BERT model are already implemented, and there are two places that require your implementation: the `EncoderLayer` and `BERT` classes.

The `EncoderLayer` class defines how input go through the multihead self-attention layer and the feedforward layer properly. The input to the former includes the encoded sequence and an attention mask. Note that the attention mask here is not the one for masking out the future tokens in training a transformer decoder, but rather it is for masking out all the [PAD] tokens.

The BERT class defines the data flow of the entire model. Your implementation focuses on the forward function, in which you need to produce the outputs for the MLM and NSP tasks:

- For NSP, you goal is to use the last layer representation of [CLS] to produce the logits for a binary classifier. You should use the provided fully-connected layer, tanh activation, and classifier components accordingly.
- For MLM, you goal is to gather the last layer representations of all the masked tokens, whose indices are denoted by the input tensor maked_pos. The gathered mask representations are then passed through a fully-connected layer (with a special GELU activation) and a layer norm, and finally a $V \times d$ decoder. The decoder's job is to map the representations to logits of vocabulary size, and it shares the same weight as the input token embedding layer.
- The `forward` function of `BERT` takes as input three tensors: `input_ids`, `segment_ids`, and `masked_pos`. You should use `masked_pos` to gather the mask representations. The function returns two logits, `logits_lm` and `logits_clsf`, which you will use later in the training loop to compute the loss.

3) Implement the training loop (10 points)

In the training loop, you conduct forward pass on the model, take the returned logits to compute the cross-entropy losses, `loss_lm` and `loss_clsf`, respectively. You should sum them up as the final loss. Note that `masked_tokens` serves as the ground truth for the MLM task, and `is_next` is the ground truth for the NSP task. By default, we use the same loss function instance (`criterion`) for the two tasks, but you can feel free to try separate losses.

Grading rubrics for Requirement 1 through 3

For Requirement 1 and 2, you get full credits if your outputs match the expected ones; otherwise, you get 0 points for the corresponding component (make_batch = 15, EncoderLayer = 5, BERT = 15 points).

For Requirement 3, if your loss decreases to around 3 or below, you get full credits; otherwise, you receive 0 points

****

4) Training on full data (15 points)

You are provided with three data files, `train.txt`, `test.raw.txt`, and `test.pairs.txt`. `train.txt` contains 90% of the text from the first seven chapters of 《庄子》(内七篇). Each line is a sentence, which is obtained from a simple segmentation by the regular expression pattern `[，。！？；]`. Chapters are separated by a blank line. `test.raw.txt` contains10% of the text, and is not separated by chapter. You should build your vocabulary on the union of `train.txt` and `test.raw.txt`.

Test.pairs.txt is the test set to evaluate the trained model. It is already in the format for MLM and NSP tasks: 15% of tokens are replaced with [MASK]; the first 29 lines are positive next sentence pairs, and the rest 28 lines are negative pairs. The format of each line is:

`[CLS] sentence_A [SEP] sentence_B [SEP] \t is_next \t mask_ground_truth`

For example:

`[CLS] [MASK] 者 庄 周 梦 为 胡 蝶 ， 栩 栩 然 胡 蝶 也 。 [SEP] 不 [MASK] 周 之 梦 为 胡 蝶 与 ？ [SEP] 0 昔 知`

In which the `is_next` label is 0, and “昔” is the ground truth word for the first [MASK], and “知” is for the second [MASK]. Both sentences are already tokenized by space.

Your goal is to train a BERT model on `train.txt`, and evaluate its performance on `test.pairs.txt`, in terms of accuracy of MLM (percentage of correctly predicted [MASK] words) and NSP (percentage of correctly predicted `is_next` labels)

Grading rubrics for Requirement 4

We are still testing the potential performance of BERT on the data, so the grading rubrics in terms of MLM and NSP accuracies will be released shortly.

****