DS-GA 1008 Homework 5

Zian Jiang (zj444)

November 29, 2020

Problem 1

a

- B:Bidirectional
- E: Encoder
- R: Representations
- T: Transformers

b

- BooksCorpus: 800 million words
- English Wikipedia: $2,500$ million words

c

A multi-layer bidirectional Transformer encoder with

- Number of layers: 16
- Number of self-attention heads: 16
- Hidden size: 1024

d

Unlike in RNN-based language modeling, in the Transformer encoder we do not have the notion of word order. All words of input sequence are fed to the network with no special order or position so model has no idea how the words are ordered. Thus, position embeddings are added to each word embedding to help the model incorporate the order of words.

e

- Masked LM: given data, choose 15% of tokens and mask them, keep them unchanged, or replace those tokens with other tokens based on a probability. Then try to predict those masked tokens.
- Next Sentence Prediction (NSP): Given two sentences A and B, try to predict if B is the actual next sentence that follows A.

f

$$\mathcal{L}_{\text{MLM}} = - \sum_{\underbrace{i : m_i = 1}_{\text{MASKED POSITIONS}}} \log p_\theta(\underbrace{y_i}_{\text{TRUE TOKEN}} \mid \underbrace{\tilde{y}_1, \ldots, \tilde{y}_N}_{\text{MASKED SEQUENCE}})$$

Thus in our case it is

$$\mathcal{L}_{\text{MLM}} = -\log p_\theta(y_{t_1}|\tilde{y}_1, \ldots, \tilde{y}_N) - \log p_\theta(y_{t_2}|\tilde{y}_1, \ldots, \tilde{y}_N)$$

g

According to Table 5, MLM performs better than left-to-right language modeling on all 5 tasks: MNLI-m, QNLI, MRPC, SST-2, and SQuAD. According to Figure 5, even though MLM only predicts 15% of tokens in each batch, it outperforms left-to-right language modeling almost immediately in terms of absolute accuracy on the MNLI task.

h

16 Cloud TPUs (64 TPU chips total). Took 4 days to complete.

i

The latter approach: all parameters are fine-tuned. Compared to the other approach, fine-tuning all parameters end-to-end should lead to better performance because more parameters are being optimized, while still being relatively inexpensive compared to pre-training.

j

It is the classification token. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. Thus, we put this token at the beginning of every input, then feed it through BERT, and then we will feed the final hidden vector of this token to the linear classification layer to predict and train.

k

Because we still trained on the NSP task, the separation token [SEP] is still trained on. For MNLI, we will ignore [CLS], and feed the premise and hypothesis, with [SEP] separating them, to our trained model. We will use the mean of all last hidden vectors as the final aggregate sequence representation. Lastly, we feed it the final aggregate sequence representation to the linear classification layer for fine-tuning and predicting.

l

According to the paper, for the first $10,000$ steps learning rate warmup is applied, which means in the beginning of training they start with a much smaller learning rate and then increase it over $10,000$ steps until it reaches that "initial" learning rate. Then, learning rate decays linearly every few steps.

Problem 2

This answer is inspired by an ICRL 2020 paper titled Incorporating BERT into Neural Machine Translation. Instead of replacing the Transformer encoder stack directly with English BERT, we keep the current Transformer-to-Transformer architecture and add another set of attention layers: from the encoder to BERT and from the decoder to BERT. Thus, the model does hot have to rely on representations from BERT but can cherry-pick a useful piece of information from BERT when necessary. In other words, in both Transformer encoder and decoder stack, instead of feeding forward just self-attention outputs at each layer, we will feed forward both BERT-side attention and self-attention.

Problem 3

a

Catastrophic forgetting in sequential fine-tuning on tasks results in forgetting information learned in earlier stages of transfer learning. For example, while fine-tuning on several tasks sequentially, the model we get after having fine-tuned on the last task may perform terribly on the first task, even though the model had been fine-tuned on the first task as well earlier.

b

- As different tasks have different noise patterns, a model that learns two tasks simultaneously is able to learn a more general representation. Learning just task A bears the risk of overfitting to task A, while learning A and B jointly enables the model to obtain a better representation through averaging the noise patterns.

- Some features may be difficult to learn from task A but easy to learn from task B. Multi-task learning allows the model to learn features more easily through different tasks.

c

- Sharing all parameters between tasks results in deterioration of performance for a subset of tasks.
- Requires access to all the tasks at the same time making it difficult to add more tasks on the fly.
- As the different tasks have varying sizes as well as loss functions, effectively combining them during training is quite a challenge.

Problem 4

a

RTE is a binary classification problem with "entailment" vs. "not entailment", while MNLI breaks them down further into "entailment", "contradiction" and "neutral". Also, the RTE dataset is significantly smaller than the MNLI dataset.

b

Validation accuracies:

- 0.534296 (seed 24)
- 0.545126 (seed 24)
- 0.541516 (seed 8)

mean: 0.5403
standard deviation: 0.0055

c

Validation accuracies:

- 0.685921 (seed 42)
- 0.675090 (seed 24)
- 0.678700 (seed 8)

mean: 0.6799
standard deviation: 0.0055

d

$109,483,778$

e

Without using any pre-trained weights, RTE is a small dataset and simply the RTE task alone does not have enough data to tune both the classifier and BERT weights to a level of performance that pre-trained BERT is on, which can be regarded as a language model whose weights are pre-trained on a whole larger corpus. This limiting factor is overcome by swapping untrained BERT with a pre-trained BERT, and fine-tuning starts at the loaded pre-trained BERT weights instead of randomly initialized weights.

Problem 5

b

Besides RTE, the other 3 adapters I used are SICK, SciTail, and MNLI. The reason is that like RTE, these 3 tasks are also natural language inference tasks, so I believe that this similarity may help train the RTE task.
Validation accuracies:

- 0.7328519855595668 (seed 42)
- 0.7653429602888087 (seed 24)
- 0.7364620938628159 (seed 8)

mean: 0.7449
standard deviation: 0.0178