CS475 Spring 2021 Homework 3

Team 1: HyeonJeong Ha, Jio Oh, Seola Choi, Junyoung Choi

May 8, 2021

In this homework, the task is implementing and comparing different techniques for pooling token-level representation in BERT[1]. Pooler gives a "representation" of each sequence in the batch by taking the hidden states. We usually use the output representation of the special classification token `[CLS]`[4] for sentence level tasks. `[CLS]` token represents the meaning of sentences after passing all BERT layers. To improve the performance of pooling methods, we suggest two different poolers, `MeanMax pooler` and `SWAP`.

1 MeanMax Pooling

Currently, the hidden state that is forwarded to the `MeanMax pooler` has the dimension of (Batch Size, L, H). The output of MMT is depicted as

$$C_{\text{MMT}} = \left[\sum_{i=0}^{L-1} T_i \;||\; \max_i T_i\right], \quad C_{\text{MMT}} \in \mathbb{R}^{2H}, \quad T_i \in \mathbb{R}^{H}, \tag{1}$$

where || implies the vector concatenation operator. We utilized the $torch.sum()$ method, by summing up through dimension 1 to implement the first part of C_{MMT}. This part is named as *tmp* in the code. To implement the second part of the vector, we made use of the L_2 Norm to find the maximum size column vector for each batch($torch.norm()$). Then, these maximum sized vectors were concatenated into a vector of size (B,H), named *tmp*2 in the code. Finally, *tmp* and *tmp*2 were concatenated so that we get the desired C_{MMT} with size of (B,2H).

The final target output is depicted as $C = \tanh(C_{\text{MMT}} W_{\text{MMT}}^{\top})$, which was done by utilizing the simple linear neural network ($nn.Linear()$) and the tanh activation function ($nn.tanh()$).

2 M-SWAP (Modified Softmax-Weighted Average Pooling)

$$average\_pool(X, weights = softmax\_per\_window(X)) \tag{2}$$

For `MyBertPooler`, we use `M-SWAP` which applies average pooling but re-weights the inputs by using $softmax$ for each window. A major disadvantage of max pooling is that it only computes the gradient with respect to the maximum value in a window. In other words, max pooling generates sparse gradients, leading to loss of information. In contrast, `M-SWAP` takes advantage of average pooling where all activations are passed to a gradient backwards. Then, it goes through a $softmax$ function to normalize the values into a probability distribution. By using $softmax$, it gives an effect that is similar to max pooling which favors large values[2]. To sum up, `M-SWAP` has nearly identical values to max pooling for forward pass while all elements in the window receive gradient updates, rather than just the maximum one for the backward pass. Actually, original `SWAP` deals with images, which have 2 dimensional data, while our data is 1 dimensional, so we only take the `CLS` token and apply a softmax-weighted operator with *relu* activation.

3 Stochastic Pooling

`Stochastic pooling`[7][3] is a method designed to solve problems with max pooling and average pooling. Max pooling has the disadvantage of overfitting on the learning data. The average pooling tends to reduce

strong stimuli. `Stochastic pooling` is obtained by dividing a particular activation by the sum of the total activations. We could regard this step as a normalization step. The strength of `stochastic pooling` is that it prevents overfitting. So it is more effective when the task requires higher epochs for training. However for the `GLUE` tasks, three epochs are adequate to verify performances. Also, on one-dimensional texts, `stochastic pooling` played the same role as dropout, so we got the conclusion that `stochastic pooling` might not be favorable for this task.

4 Result

We tried to compare the performances across various tasks (`SST-2, CoLA, MRPC`) in `GLUE` [6] so that we can fairly test the implemented poolers. First of all, we chose the `SST-2` dataset, because we believe that the positive/negative sentiment analysis/classification is the most fundamental task in natural language processing. Moreover, we tried to test if our pooler helped boost the model's functionalities in terms of similarity analysis. Hence, we selected the `MRPC` for testing. Lastly, we wanted to test not only the semantic aspects, but also the syntactic parts. Thus we made use of `CoLA` tests, used to see if the model can distinguish grammatically incorrect phrases.

		Batch Size 16			
	metric	**CLS**	**Mean Max**	**SWAP**	**Stochastic**
SST-2	accuracy	.933	.931	.936	.930
CoLA	correlation	.589	.580	.599	.568
MRPC	accuracy / f1	.867 / .903	.828 / .886	.870 / .909	.855 / .898
		Batch Size 32			
	metric	**CLS**	**Mean Max**	**SWAP**	**Stochastic**
SST-2	accuracy	.928	.931	.931	.934
CoLA	correlation	.565	.567	.570	.580
MRPC	accuracy / f1	.828 / .878	.808 / .861	.846 / .889	.840 / .889
		Batch Size 64			
	metric	**CLS**	**Mean Max**	**SWAP**	**Stochastic**
SST-2	accuracy	.923	.930	.928	.916
CoLA	correlation	.562	.549	.563	.560
MRPC	accuracy / f1	.821 / .877	.768 / .849	.806 / .869	.820 / .877

Table 1: Results on GLUE Dataset, `SST-2, CoLA, MRPC`. The metric is an accuracy for `SST-2`, correlation for `CoLA`, and accuracy & f1 score for `MRPC`. Batch sizes of evaluation are 16, 32, and 64.

5 Discussion

Since controlling of hyperparameter affects to the training time or performance, we expected that we can compare them by controlling batch size. For `SST-2, CoLA, MRPC` dataset, the performance decreases with increasing batch size. Generally, as batch size becomes large, we make use of more data to compute the gradient descent, thus optimization becomes handy. However, if the problem space that we ought to optimize is flat, the approximated absolute value of the gradient becomes small, leading to slow convergence. Hence, our results might just dwell in the local minima without any decrease in loss. In a similar vein, when losses are averaged over a large batch, the model fails to travel far enough to reach better solutions when the optimal solutions are far away from the initial point[5]. In conclusion, we reckon that larger batch sizes might result in poor generalization and the problem space for the GLUE dataset is relatively flat.

`M-SWAP` has better performance than `CLS`, even though `M-SWAP` takes `CLS` token-based input. We believe that the following two characteristics of `M-SWAP` results in better performance than `CLS`. First, `M-SWAP` generates a probability distribution of `CLS` tokens in sentences for each hidden state. Second, it gives information about the original `CLS` token to the intermediate representation(probability distribution) which is similar to skip connection. `M-SWAP` has better performance than `Mean_Max`, since all activation are passed to a gradient backward, rather than just max which is sparse.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[2] Shawn Jain. SWAP: Softmax-weighted average pooling. 2020.

[3] Takumi Kobayashi. Gaussian-based pooling for convolutional neural networks. 2019.

[4] Chris McCormick. Bert word embeddings tutorial. 2019.

[5] Kevin Shen. Effect of batch size on training dynamics. 2018.

[6] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[7] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.