Task-specific attention

Stanford CS224N Default Project

Enzo (Chaoqun) Jia
Department of Computer Science
Stanford University
enzojia@stanford.edu
enzo.jia@gmail.com

Abstract

In this paper I explore applications of task-specific attention mechanism, which takes BERT foundation model output and is intentionally overfitted for each task. Experiments proved that this mechanism can help to improve model performance when finetune data is relatively large. Corresponding training cost is not significantly higher compared to traditional finetunes. In this project, task-specific attention layers improve paraphrase detection accuracy from 0.551 to 0.842, with a 10% increase of finetune training time.

1 Key Information to include

- Mentor: N/A.
- External Collaborators (if you have any): N/A.
- Sharing project: N/A.

2 Introduction

2.1 Introduction

BERT (Bidirectional Encoder Representations from Transformers) is designed to be a foundation model which supports different applications, including but not limited to sentiment analysis, paraphrase detection, and semantic textual similarity Devlin et al. (2019). For each document (sentence), BERT produces one document representation which is the self-attention block output corresponding to the first token, and this representation is fed to downstream applications. For example, for sentiment analysis one sentence's representation is fed to a linear layer, and for paraphrase detection we will need representations for two sentences.

In this design, we need the attention block output of only one token to construct a document's representation, because as BERT's name indicates it is a bidirectional encoder model using self-attention as its building blocks, as shown in equation 1 and 2. $X \in \mathbb{R}^{n\times d}$ is the input sentence embedding matrix, $Q \in \mathbb{R}^{D\times d}$ is the query transformation, $K \in \mathbb{R}^{D\times d}$ is the key transformation, $V \in \mathbb{R}^{D\times d}$ is the value transformation, $A \in \mathbb{R}^{d\times d}$ and h is number of heads. As the final output $Y \in \mathbb{R}^{n\times d}$ has the same dimension as the input matrix, the final architecture of the model can stack multiple layers of self-attentions. I will not repeat (too much of) what is illustrated in the original paper Devlin et al. (2019) and CS224N lecture notes.

$$Y_i = \text{softmax}\left(\frac{(XQ_i)(XK_i)^\top}{\sqrt{d/h}}\right)(XV_i) \tag{1}$$

$$Y = [Y_1; \ldots; Y_h]A \tag{2}$$

	BERT return	pretrain / finetune	sentiment dev acc	paraphrase dev acc	STS dev corr	Train time (sec)
Part I	first token output	pretrain	0.389	0.401	0.298	15006
		finetune	0.504	0.595	0.587	37183
Baseline perf	mean sentence output (masks excl'd)	pretrain	0.460	0.386	0.643	15608
		finetune	0.515	0.561	0.854	38161

Table 1: Results on dev datasets, utilizing BERT's outputs on each sentence's first token vs. the whole sentence. One T4 GPU on Google Cloud was used for each task in each experiment.

The purpose of having equations 1 and 2 here is to demonstrate that the first column of final layer output, which corresponds to the first token and is used as the whole document's representation, contains information from all other non-masked tokens in the same document.

2.2 Motivations

In above section 2.1 I demonstrated that in theory, when number of self-attention layers is large enough, final output of the first token's attention block is representative for the whole document. Are 12 and 24 large enough? An experiment answers this question. Table 2.2 gives BERT model's downstream task performances under different settings. Row "first token output" corresponds to the scenario of using BERT's 'vanilla' output which is self-attention output of the sentence's first token, which in the case of this project is a prepended [CLS] token. On the contrast, results in "mean sentence output" row utilizes mean of the whole sentence's all tokens' outputs, after excluding masked tokens. Here we see how the "mean sentence output" helps the STS (semantic textual similarity) task's performance, and this improvement indicates that attention outputs corresponding to tokens other than the first one also have information which is not collected by the first token's output.

Please note in table 2.2, for finetune rows, each downstream task was trained for this task only, without finetuning for other two tasks. This is because finetune optimizes foundation model's parameters for this specific task, and this optimization does not necessarily work for other tasks. The train time in this table is sum of 3 finetuning procedures' running times. To keep train time of pretrain rows comparable to finetune rows, I also ran pretrain procedures 1 time for each task and sum 3 run times.

Instead of taking mean of the whole sentence's attention value, I propose arranging different weights to tokens in the sentence for aggregation – that is what attention does. Detailed discussion on approaches can be found in section 4.

2.3 Baseline performance

Because purpose of this exploration is to test attention layers' capability to collect information distributed in the whole sequence, I use row "mean sentence output (masks excl'd)" in table 2.2 as baseline, which collects information in the whole sentence in an euqally-weighted fashion.

3 Related Work

Related work on BERT and other transformer-architecture large models have been introduced in CS224N lectures, so here I focus on using attention blocks outside of the foundation model. This straightforward idea is very possibly already explored by other researchers, however a 'quick' search did not find scholar papers discussing about it. My guess to explain the lack of related works include:

1. Additional attention layers may not significantly improve models' performance. If true, this may be caused by the fact that foundation models already have multiple attention layers and have exhausted the potential capability of attentions.
2. Attention layers/blocks are resource-consuming to train, so adding them outside of the foundation model may not be meaningful because end-users don't have enough data and computing resources to train the layers.

4 Approach

4.1 Loss functions

The focus of this project is exploring whether task-specific attentions help a large model's performance, so I select loss functions in a straightforward way.

1. Cross entropy for sentiment analysis. Sentiment analysis is an ordinal classification problem which can be simplified to be a categorical classification problem, so cross entropy loss is an intuitive choice. However this setting does not use all information we can obtain from ordinal sentiment labels, so I put "loss function exploration" in the future works list.
2. Binary cross entropy for paraphrase detection, which is a binary classification problem.
3. Negative Pearson correlation for semantic textual similarity analysis, which is an ordinal classification problem.

4.2 Task-specific attention

Second row of table 2.2 takes mean of the whole sentence's attention output, which hurts the paraphrase detection task's performance, and that motivates the author to collect information from each token's output in a smarter way. To be specific, I use one or multiple self-attention layer(s) for sentiment analysis, one or multiple cross-attention layer(s) for paraphrase detection, and another one or multiple cross-attention layer(s) for semantic textual similarity analysis. Having this strategy is because sentiment analysis works on one sentence, and the other two tasks are comparing two sentences. All these attention layers are implemented outside of BERT model and are trained in a task-specific fashion.

4.2.1 Attention outputs

For each sentence I take the last task-specific attention layer's output corresponding to the first token ([CLS] token) as the whole sentence's encode. This is $Y[:, 0]$ extracted from Y in equation 2, which uses a softmax to distribute weights to all tokens' transformed values calculated from previous layer's output. Meanwhile because output corresponding to pad tokens are masked, what I finally get is a weighted mean of the sentence's all tokens' previous attention outputs.

4.2.2 Attention initialization

Without initialization of attention layers' weights, for sentiment analysis and paraphrase detection, task-specific attentions do not help, and on the contrast addition of these layers lowers corresponding dev performance scores. This can be explained by item $\#2$ as discussed in section 3, that attention layers require large amounts of data to train, however my training datasets for these two tasks are small. Considering another fact that in table 2.2 taking mean of a whole sentence's tokens' encode improves model performance, which indicates a special case of uniformly distributed attention (without value transformation), I add another series of experiments which have attention layers initialized to have their weight entries close to each other. This initialization results in softmax output close to a uniform distribution, and meanwhile weight entries are not equal so the optimizer can update them. A uniform distribution $\mathcal{U}(0.9, 1)$ is used for weight initialization.

4.3 Implementation details

1. For each of the 3 downstream tasks, the upstream output tensor is further transformed by a linear layer. For sentiment analysis the final output is a 5-element softmax probability tensor, and for each of the other 2 tasks the final output is a scalar produced by a dot multiplication between two input sentences' transformed upstream output encodes.
2. In cross-attention implementation, I align the two sentences by padding the shorter one to the length of the longer one. This is for implementation convenience and doesn't impact prediction performance.

5 Experiments

Because the purpose of this exploration is to test whether extra attention layers on top of BERT bring performance improvements, I control Data and most experimental settings as discussed in subsection 5.2.

5.1 Evaluation method

To better understand two possible reasons for the lack of related work on task-specific attention mechanism, as discussed in section 3, this exploration looks into two metrics:

1. For recognition performance I will use the same metrics as demonstrated in table 2.2(accuracy for sentiment analysis and paraphrase detection, and Pearson correlation for semantic textual similarity). These metrics will be collected on test datasets.
2. For computational resource requirement I record running time on one Nvidia T4 GPU to finish corresponding tasks. I don't expect a significant change in this metric, as both baseline and new experiments have the same training time complexity of $O(n^2)$ for each sentence in each epoch, where n is number of tokens in one sentence. The required large amounts of resources discussed in section 3 come from processing large amounts of data. Because the three downstream tasks are trained separately, I record running time as the sum of three training times.

5.2 Experimental details

For this project I control experiment settings, except task-specific attention settings, to be the same across all 3 experiments: baseline, experiment #1, and experiment #2, corresponding to 1st row, 2nd row, and 3rd row in result table 5.3, respectively. Controlled settings include:

- Loss function: cross entropy for sentiment analysis, binary cross entropy for paraphrase detection, and negative Pearson correlation for semantic textual similarity analysis.
- Data: Stanford Sentiment Treebank data for sentiment analysis, Quora dataset for paraphrase detection, and SemEval STS Benchmark dataset for semantic textual similarity analysis.
- Optimizer: all experiments for all tasks use AdamW optimizer.
- Learning rate: $1e-3$ for pretrain and $1e-5$ for finetune.
- Number of epochs: 10 epochs for pretrain, and 10 epochs for finetune.

Differences between baseline vs. experiment #1 vs. experiment #2 are in task-specific attention settings, and can be found in table A, that baseline has no task-specific attention layer, experiment #1 has uninitialized task-specific attention layers, and experiment #2 has task-specific attention layers with their weights initialized under uniform distribution $\mathcal{U}(0.9, 1)$. Number of layers of task-specific attention varies for three downstream tasks.

5.3 Results

Experiment results are collected and organized in table 5.3. **My best test leaderboard submission was an assembly including experiment #2 output for sentiment analysis, experiment #1 output for paraphrase detection, and baseline output for semantic textual similarity analysis**. This submission was scored at $(0.524 + 0.842 + 1.826)/3 = 0.759$, and ranked #30 out of 80 submissions on March 15th, 2024.

In table 5.3, 'Train time (sec)' column records sum of three downstream tasks' training times for each row, in seconds. Also 'Train time vs. baseline' is the ratio between the train time in that row over corresponding train time in the baseline row.

Table 5.3 is further discussed in section 6, to answer questions including why paraphrase detection sees a significant score improvement while the other two tasks don't.

	Processing on BERT return	pretrain / finetune / test	sentiment accuracy	paraphrase accuracy	STS correlation	Train time (sec)	Train time vs. baseline
Baseline	mean sentence output (masks excl'd)	pretrain	0.460 (dev)	0.386 (dev)	0.643 (dev)	15608	1
		finetune	0.515 (dev)	0.561 (dev)	0.854 (dev)	38161	1
		test	**0.517**	**0.551**	**0.826**		
Experiment #1	Uninitialized attention	pretrain	0.253 (dev)	0.469 (dev)	0.251 (dev)	18086	1.159
		finetune	0.509 (dev)	0.843 (dev)	0.822 (dev)	41827	1.096
		test	**0.533**	**0.842**	**0.791**		
Experiment #2	Initialized attention	pretrain	0.262 (dev)	0.429 (dev)	0.148 (dev)	17648	1.131
		finetune	0.539 (dev)	0.838 (dev)	0.386 (dev)	41036	1.075
		test	**0.524**	**0.839**	**0.374**		

Table 2: Final results on dev and test datasets. One T4 GPU on Google Cloud was used for each task in each experiment.

	sentiment	paraphrase	STS
Baseline	3840	0.59M	2.36M
Experiment #1 & #2	4.72M	3.54M	11.2M

Table 3: Number of task-specific parameters to train for each task in each experiment.

6 Analysis and discussions

6.1 Number of task-specific parameters

In baseline models, each task has one linear projection layer, which is 768 by 5 for sentiment analysis, 768 by 768 for paraphrase detection, and 768 by 3072 for semantic textual similarity analysis. These linear projection layers also exist in experiment #1 and #2. In each self-attention block, there are four linear projection layers, three of which are for transformation of query, key, and value matrices, of size 768 by 768, and the fourth one is another 768 by 768 layer for transformation of final output. Each cross-attention block is similar to a self-attention block in parameter settings, but with one more 768 by 768 linear layer for intermediate transformation. In each of experiments #1 and #2 there are two self attention layers for sentiment analysis, one cross attention layer for paraphrase detection, and three cross attention layers for semantic textual similarity analysis. With all the above information we get the number of task-specific parameters in table 6.1. So for experiments with task-specific attentions there are 4.72M/3.54M/11.2M parameters to train, for each task respectively, and this echos concern item #2 as discussed in section 3 that training task-specific attentions require large amounts of data and corresponding computing resources.

There is one note for paraphrase detection task. Training data for this task has 17688 labeled pairs of sentences, and this is a relatively large data set compared to other two tasks, as sentiment analysis has 1068 labeled sentences and semantic textual similarity analysis has 755 labeled pairs of sentences. With a larger amount of training data, and longer training times which indicate larger amounts of computational resources, paraphrase detection task is benefited the most from addition of task-specific attentions and has its test accuracy increased from 0.551 to 0.842.

6.2 Performance: experiment #1 vs. baseline

For sentiment analysis and semantic textual similarity tasks, experiment #1's test scores increase by 0.013 and -0.035, respectively, both of which are relatively small. On the contrast, for paraphrase detection task, experiment #1's uninitialized task-specific cross-attention layer helps to increase detection accuracy from 0.551 to 0.842, which is a significant improvement. In subsection 6.1 I discussed about three tasks' different amounts of performance improvements by examining numbers of parameters and training data sizes, and in this subsection the discussion is from a different perspective.

Improvement in paraphrase detection task's score proves that the task-specific cross-attention layer has a better capability than a simple mean in collecting information from all token's encodes. However why this better capability doesn't work as well for the other two tasks? I looked into the test leaderboard for best scores for the 3 tasks, that by March 15th 2024, the best scores were: 0.557 for sentiment analysis, 0.9 for paraphrase detection, and 0.891 for semantic textual similarity analysis. Comparing these best scores to my baseline test scores, we see that for sentiment analysis and semantic textual similarity analysis, my baseline scores are $> 90\%$ of best scores. This can be interpreted as that for these two tasks, a simple mean of all tokens' BERT encodes is already capable of extracting most information out, and there is not much potential left for a weighted mean to improve each task's performance. However for paraphrase detection a simple mean only catches $0.551/0.9 = 61\%$ of the best submission score, this leaves a large amount of improvement potential for a weighted mean, which can be implemented by task-specific attention layers.

6.3 Performance: experiment #1 vs. experiment #2

For all three downstream tasks, experiment #2's test scores are lower than experiment #1's, indicating that initialization of task-specific attention weights does not help to improve application performances. There are two different scenarios among the three tasks, and one possibile scenario not observed in this exploration:

- For sentiment analysis and paraphrase detection tasks, experiments with and without weight initialization produce close score values. This indicates that parameter searches for both experiments may arrive at the same local optima. Under this scenario initialization does not help to improve model performance.
- For semantic textual similarity analysis, initialization of attention weights reduces test score (label vs. prediction Pearson correlation) from 0.791 to 0.374. This can be explained that initialization of weights leads the optimizer to arrive at a different local optima, where the model performance is not as good.
- There is another possible scenario that, initialization may lead to a better-performing local optima.

Conclusion of this subsection: when resources allow, trying different initialization settings has the potential to help improve the model's performance on tasks, however if there is not redundant resource, I recommend using uninitialized task-specific attention.

6.4 Training time

Compared to baseline, both experiments #1 and #2's training times increase between 7% and 16%. This quantifies the discussion in section 5.1, that addition of task-specific attentions increases computational resource requirement for training, but not by a significant margin because of the same $O(n^2)$ training time complexity for both baseline and experiment #1/#2 settings.

Experiment #1 costs more time than experiment #2, for both pretrain and finetune training scenarios. This indicates that the initialization method which uses $\mathcal{U}(0.9, 1)$ makes starting point of the attention layers to be closer to a local optima, though this closer local optima may not be the same one the optimizer would arrive at without an initialization precedures.

6.5 Performance: loss function for STS

Two loss functions were evaluated on training and dev data for semantic textual similarity analysis. Using an MSE loss results in dev Pearson correlations between 0.3 and 0.4, under different other settings. A customized negative Pearson correlation loss improves result dev Pearson correlation to be above 0.8, as shown in table 5.3.

6.6 Discussion: task-specific attention vs. more attention layers in the foundation model

With multiple attention layers added, a question is raised that, why don't we directly use a foundation model with more attention layers, e.g. BERT with 24 attention blocks? The answer is yes we can use a bigger pretrained foundation model, and meanwhile this does not stop us from trying task-specific attention mechanism. The reason is that task-specific attention is trained more precisely for one type of tasks, so it's intentionally overfitted and can not be generalized to other types of tasks. This is why this mechanism is different from attention blocks inside foundation models. When the finetune dataset is large enough we always want to further explore the remaining potential in a foundation model's outputs.

7 Conclusion

Task-specific attention helps to collect information from the whole document's tokens' encodes in an efficient way, so this is a method worth being tried when building an application of a foundation model. This mechanism has a higher probability to succeed when finetune data size is large. However the author wants to emphasize that task-specific attention is a downstream mechanism on top of results by finetuning the foundation model, so the scientist or engineer who is designing the application shall first focus on upstream tasks, including foundation model selection, task loss function design, etc., and then use task-specific attention mechanism as an easy add-on option to check whether the upstream outputs have remaining information which can be captured by this machanism.

8 Future works

There is a list of items not included in my experiment plan because of limited time for this exploration, including:

- Different loss functions, especially for sentiment analysis. Cross entropy loss sees model outputs as categorical classfication results, however for sentiment analysis the label is ordinal data, with more information unused.
- More training data, including CFIMDB data for sentiment analysis.
- Different foundation models.
- Different experiment settings, including different optimizers, learning rates, etc.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

	Processing on BERT return	sentiment	paraphrase	STS
Baseline	mean sentence output (masks excl'd)	no attn	no attn	no attn
Experiment #1	Uninitialized attention	2 self-attn layers no initialization	1 cross-attn layer no initialization	3 cross-attn layers no initialization
Experiment #2	Initialized attention	2 self-attn layers init: $\mathcal{U}(0.9, 1)$	1 cross-attn layer init: $\mathcal{U}(0.9, 1)$	3 cross-attn layers init: $\mathcal{U}(0.9, 1)$

Table 4: Experiment settings. Attention here refers to task-specific attention layers, not layers in BERT.

A Appendix (optional)