Template-Filling in Little Space

Max (Donald) Ziff - W266 - 2021-12-04

Abstract

I experiment extensively with the (perhaps) state-of-the-art implementation of Template-Filling: Gu et al and show that substantially the same performance can be achieved using much smaller BERT models (i.e. “BERT-Medium” and “BERT-Small”) that are roughly 40% and 30% the size of the base model. In addition, I show that the text-generation approach reported in the Gu paper interacts poorly with the evaluation framework chosen, leading to uncertainty about the true effectiveness of this approach.

Introduction

This project is motivated by a use case from my work at Tripit. Tripit maintains travel itineraries: users forward emails to plans@tripit.com and TripIt creates an itinerary for every trip, combining, for example, an airline confirmation with a hotel reservation. Details are extracted from emails using an enormous library of regex-based parsers that are fragile and difficult to maintain. Using ML for this phase is tempting, but the sunk cost of the regex parsers makes it difficult to migrate to an ML-based system: it seems easier to do incremental maintenance on the clunky old system than to throw it out in favor of an ML-based system.

Instead of modeling that problem, I consider a complementary approach: can the Tripit phone-based app do the information extraction on-device? A privacy-conscious user might prefer that the Tripit app display locally the travel details it discovers before uploading only that extracted information. For this approach, an ML model would need to be small enough to fit on the device. It would obviously be impossible to port the legacy solution.

This project is intended to be a step in planning a practitioner’s approach, by considering a similar problem in the public research space: Template Filling. The MUC-3 and MUC-4 conferences established a public formulation of Template Filling involving:

- A corpus of news articles
- A small set of template events (“Attack”, “Bombing” and others) intended to be found in those articles
- For each template, a set of roles that might be present, such as “Perpetrator”, “Target”, “Weapon”, etc. There is some overlap in the roles for each template
- For each article, the matching templates and roles filled in by human experts

Note that a single article might match more than one template, or there might be more than one instance of a templated event in a single article.

I aimed to consider a state-of-the-art reference for this problem and to see how minimizing the underlying model affects performance. However, I discovered some things I didn't expect.

Background

For the reference implementation, I chose the work of Gu et al 2021 [1] ("Template Filling with Generative Transformers", Xinya Du, Alexander Rush, Claire Cardie) which uses a single Bert model. I studied the performance of this approach using some of the off-the-shelf minimized Bert models described in Iulia Turc et al [2] ("Well-Read Students Learn Better: On The Importance Of Pre-Training Compact Models", Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova).

One striking aspect of the Gu et al approach is their unusual use of Bert. Although most of my analysis treats the Gu implementation as a black box, it is nonetheless interesting to pry inside. The Gu implementation builds on techniques they developed and published earlier (Gu et al 2020 [3]). The approach uses a single Bert model as both encoder and decoder

For training, the model is fine-tuned on a representation of the text that starts with a [CLS] followed by the names of all the templates, followed then by the text of the news article (tokenized), then by the templates and role fillers to be learnt, with appropriate separator tokens to separate the template names from the role.

For prediction, an attention mask is constructed which initially allows view of the entire test document but no more. Instead of text generation, softmax is used to represent the probability of the next token as a pointer into the tokens of the encoded document. These tokens could be entities or template or role names. At every step, the attention mask is extended so that the entire sequence so far generated is visible. The model is repeatedly invoked in this manner until an end token is predicted or until some maximum. In addition, a hyperparameter controls the generation of separator tokens, to make it more or less likely that the model predicts multiple templates for a single text. Also, the model is constrained so that constituents of a role are in text order.

The authors note that previous work has used two separate models, an encoder and decoder. Because of the authors' approach, not all Bert models work in this framework. In particular, DistilBert (Sanh et al [4]) does not work with this model because of differences in how masks are represented.

Methods

I experimented by slotting in a range of Bert models and observing the performance at many epochs of training. The BERT models considered were the following:

- BERT-Base (the model referenced in the published paper) 110.1 M parameters

- BERT-Medium 41.7 M parameters
- BERT-Small 29.1 M parameters
- BERT-Mini 11.3 M parameters
- BERT-Tiny 4.4 M parameters

I trained these models up to 100 epochs each and evaluated them after each epoch.

I tried other Bert-like models but discovered that the Gu implementation was incompatible with most. So I finally restricted my experiments to models that were usable without code change. I did not vary the “thresh” hyper-parameter, which influences the likelihood that the model predicts multiple templates. Instead I used the value cited in the published paper throughout. For comparison between models, I focused on the “micro-average F1” scores, the same metric used for comparison in the published paper.

I did not study the prediction timings. Based on the results published by the Google researchers, one might expect a speedup roughly between a factor of 2 to 50 for the medium to tiny models.

Results and Discussion

Although I attempted to reproduce the results in the paper by using the same model (“bert-base-uncased”) and training epochs (20), I found that my results came close but differed noticeably from the published results. Table 1 below shows the published results and my results after 20 epochs, along with the best results I obtained using BERT-Base.

Table 1:

Model	Epochs	P	R	F1
GTT (published)	20	61.69	42.36	50.23
BERT-Base	20	55.82	34.48	42.63
BERT-Base	42	61.88	36.82	46.17

Table 2 below shows the best results obtained by each model, along with the published results for comparison.

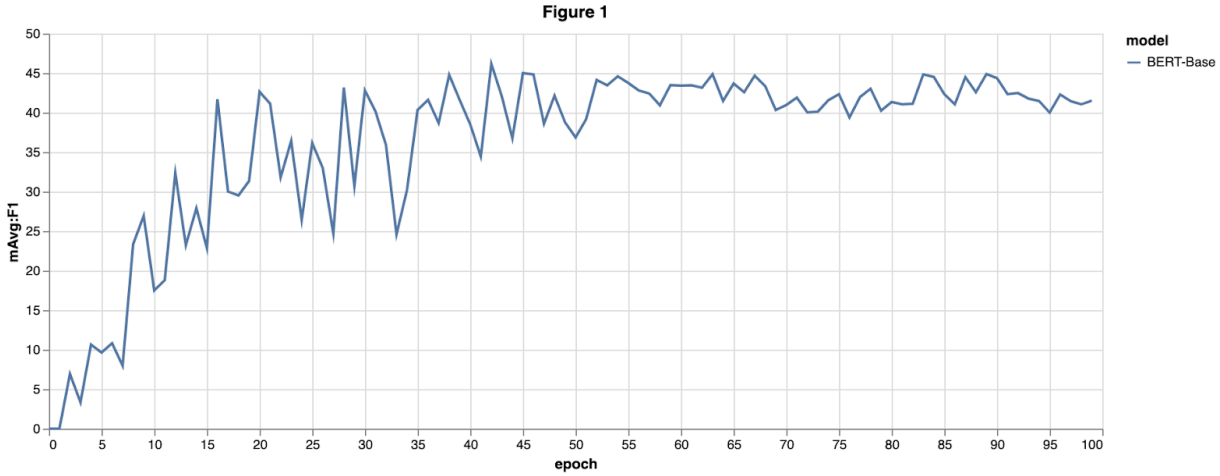
Table 2:

Model	Epochs	P	R	F1
GTT (published)	20	61.69	42.36	50.23
BERT-Base	42	61.88	**36.82**	46.17

BERT-Medium	84	**66.90**	35.47	**46.36**
BERT-Small	85	63.43	33.74	44.05
BERT-Mini	58	57.58	32.27	41.36

We see that the BERT-Medium model out-performs the Base model slightly, after being trained for twice as long. The Small model also performs competitively, while the Mini and Tiny models trail.

However, looking closely at the micro-average F1 score for the baseline model as a function of training epochs yields a surprising insight.



This shows the result of running the model on the reserved test set after each training epoch. We see that the curve is very jagged. Although the best score was after epoch 42, the neighboring epochs dip significantly lower. As training continues, score variability decreases.

I propose that the variability between adjacent epochs is a proxy for how that same model might perform on different test sets. We do not expect that one more or fewer training epochs should have a dramatic effect on performance but we see variation in an almost 12 point range. Table 3 below is an excerpt from the series around epoch 42:

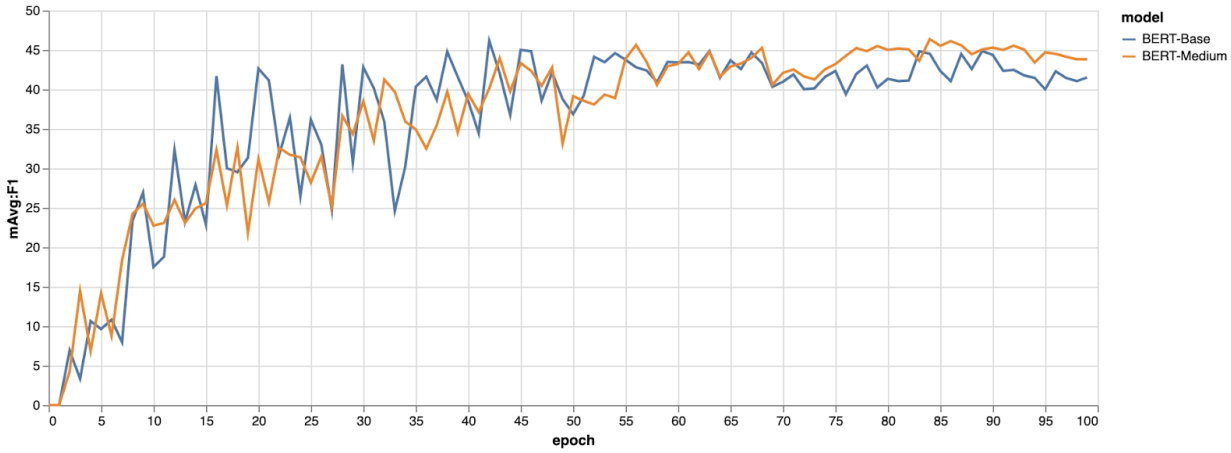
Model	Epochs	P	R	F1
BERT-Base	41	59.34	24.26	34.44
BERT-Base	42	61.88	36.82	46.17
BERT-Base	43	62.08	31.77	42.03

This variation between training epochs is much larger than the delta improvement shown over previous results in the published paper, as well as the difference between the published result

and my best attempt to reproduce. This raises the possibility that the published result was fortuitous.

Looking at the results curve for both the base and the overall next-best performing model, BERT-Medium, shows similar jaggedness:

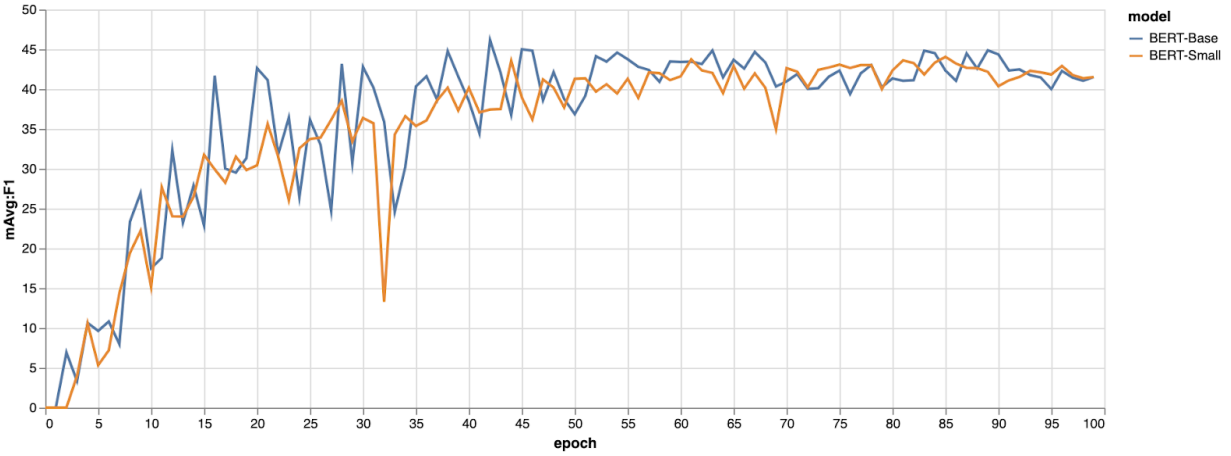
Figure 2



It is striking here that the variation between epochs seems to lessen after multiple epochs, and that the smaller model (BERT-Medium) seems to out-perform the Base model as training continues.

Similar effects are visible when comparing the base model against the next smallest model, BERT-Small.

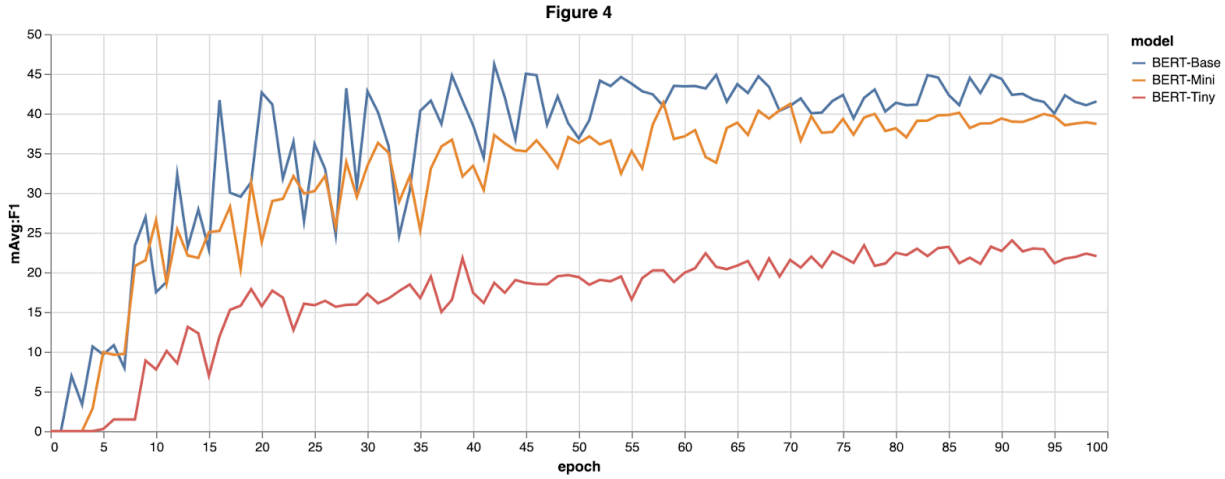
Figure 3



Here we again see the pattern of reduced jaggedness as training continues. We also see that the small model performs competitively.

For completeness, here is a similar graph showing the other models studied: BERT-Mini and BERT-Tiny. The same general patterns are observable: striking jaggedness in the first part of

training (roughly through epoch 50) followed by more stability as the training progresses. However the performance of the smaller models is markedly less.



Another striking insight uncovered by this analysis is that this method does not seem to over-fit. I propose that this apparent effect is due to the difference between the training loss function and the actual prediction methodology. During training, the model is pushed toward learning the examples as texts with full expert-given labels supplied. But the model is not evaluated during learning on the algorithm eventually used for prediction. So although the model is probably over-fitted to its training objective, subsequent training does not negatively affect its performance on the testing objective, but instead seems to lead toward stability epoch over epoch.

Conclusion

I have examined the performance of the Template-Filling implementation of Gu et al in much more detail than was done in the original paper. Towards my goal of evaluating how much the Bert model size affects performance, we see that, in the paper's own evaluation framework, the Medium model performs about as well as the full model, and that performance drops off with smaller models.

However, looking under the hood of this implementation, we have also learned some interesting things. Within the training bounds in the published paper, we see such differences between adjacent training epochs that we wonder whether the published results may have been fortuitously chosen.

We have also observed that this implementation style, though it exhibits interesting and novel ideas, may lead to results that are even more uncertain than usual, due to the discrepancy between training and testing modes.

Returning to the context of a practitioner in the Tripit business case, there is reason to doubt that this implementation style will yield better results than perhaps non-Bert methods. The language of airline confirmation emails is much more stylized than that of news articles.

Implementation Notes

My implementation consists of test frameworks and visualizations that supplement the reference implementation. Thus I have chosen to archive this work as a fork of the original. The forked repo is here: https://github.com/donaldziff/gtt. To see this work as diffs, look here: https://github.com/xinyadu/gtt/compare/master...donaldziff:master

References

[1] “Template Filling with Generative Transformers”, Xinya Du, Alexander Rush, Claire Cardie. (2021) https://aclanthology.org/2021.naacl-main.70/

[2] “Well-Read Students Learn Better: On The Importance Of Pre-Training Compact Models”, Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. (2019) https://arxiv.org/abs/1908.08962

[3] “GRIT: Generative Role-filler Transformers for Document-level Event Entity Extraction”, Xinya Du, Alexander M. Rush, Claire Cardie (2020) https://arxiv.org/abs/2008.09249

[4] “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. (2019) https://arxiv.org/abs/1910.01108