# BERT + Verify

**Kai Marshland**
Department of Computer Science
Stanford University
Stanford, CA 94305
kaimarsh@stanford.edu

## Abstract

BERT provides powerful natural language understanding, and forms the backbone of many state-of-the-art question answering systems. However, as Hu et. al. found in their seminal paper *Read + Verify: Machine Reading Comprehension with Unanswerable Questions*, they could significantly improve performance by adding a distinct answer verification step after the answer generation step. This let them achieve state-of-the-art results on the SQuAD question answering dataset at the time. However, their system used a precursor of BERT, ELMO. We update their methodology to use BERT as a verifier and bidirectional attention flow as an answer generator. We find that although it outperforms bidirectional attention flow without a verifier, it takes an order of magnitude longer to train and the gains are minimal.

## 1 Introduction

Given a piece of text – the context – and a question, we want to be able to answer that question. We develop this using the SQuAD 2.0, which introduced the possibility of a no answer result to a question answering problem. This possibility of a no-answer poses challenges. Unlike in normal question answering, where the answer can always be taken from the text, the possibility of a no-answer is a special case, and intuitively it makes sense to handle it differently.

## 2 Related work

In *Read + Verify: Machine Reading Comprehension with Unanswerable Questions*, Hu et. al. found that adding a verification step to an existing answer generation step let them achieve what were at the time state of the art results (1).

The core issue that the second, verification step seeks to solve is that the confidence in the answer generation step that typically informs a no-answer response might not be accurate. The amount of confidence a network has in an answer is tied to the task of coming up with an answer, but not necessarily the same. A network's best guess might be the correct answer, but if it has too low a confidence, it will respond with a no-answer instead. The verification step decouples these two tasks, leading to better results at least in the case of this paper.

Though Hu et. al. did achieve excellent results at the time, they used techniques that are no longer state of the art. In particular, they depended heavily upon ELMO, which has since been surpassed by the BERT model.

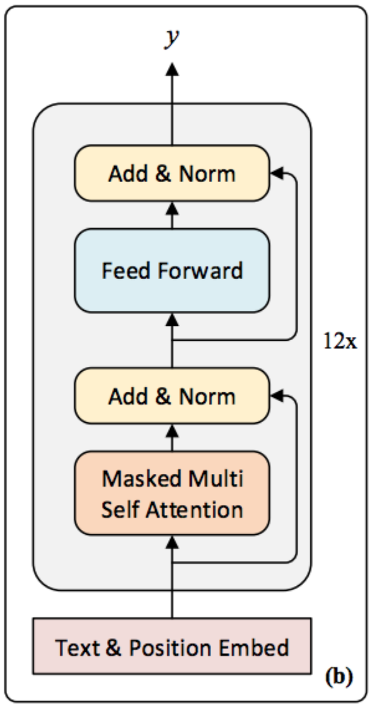
BERT is a powerful language understanding model. Indeed, the majority of the SQuAD leaders use BERT in some way. Two things make BERT so powerful: that it is deeply bidirectional, and that it can use large amounts of plain text data for training (2). The unsupervised aspect is straightforward enough; there's plenty of unannotated data out there, and more data regularly leads to better results in the field of deep learning.

Other researchers have also experimented with verifiers, such as Sun et. al. in *U-Net: Machine Reading Comprehension with Unanswerable Questions* (3). Sun et. al. only split the verifier from the generator in the final step, and achieve fairly good results, though not state-of-the-art.

## 3 Approach

We hypothesized that even better results would be achieved by updating Hu et. al.'s approach to use BERT for answer verification. As a baseline answer generator, we use BIDAF, as introduced in the paper *Bidirectional Attention Flow for Machine Comprehension* (4). For answer verification we use two techniques: the generative pre-trained transformer outlined by Hu et. al., and a verifier that connects a classification head to BERT.

The generative pre-trained transformer answer verifier was the best single verifier found by Hu et. al. It starts by concatenating the text and the question. It feeds that through the transformer to generate an encoding. This transformer was adapted from `https://github.com/huggingface/pytorch-openai-transformer-lm`. It then takes a straightforward linear projection of that output encoding and runs it through a softmax to classify the answer as correct or not.



Generative pre-trained transformer
*Image credit: Hu et. al.*

Architecturally, the verifier is added as an additional layer to the BIDAF question generator. If a question was classified as answerable, it returned the answer directly; if not, it returned vectors representing a no-answer. So that the gradients could back-propagate and so that it was run efficiently, this was structured as a matrix multiplication.

For the BERT-based verifier, we attached a classification head to a pre-trained BERT language model, adapted from `https://github.com/huggingface/pytorch-pretrained-BERT`. This classification head is a simple linear layer. Again, the question and the context are concatenated to provide the inputs to the model. However, they must be re-tokenized to take advantage of the pre-trained BERT weights. After tokenization, the prediction can be made using the pre-trained BERT network. As before, if a question is classified as answerable, it returns the generated answer, and otherwise it returns the vector representing a no-answer.

Prior to connecting the BERT-based answer verifier with the BiDAF-based answer generator, we trained it in isolation as a classifier on the training data, saving the result and inputting it to the subsequent generation + verification process. This additional bit of pre-training was motivated by the complex nature of the task; we wanted to have a reasonable verifier before trying to do answer generation.

## 4 Experiments

### 4.1 Data

We used the SQuAD 2.0 dataset, as established for the default project.

### 4.2 Evaluation Method

We evaluated the results using the F1 and EM score.

### 4.3 Evaluation Details

For the generative pre-trained transformer verifier, we ran it with the default hyperparameters, most notably a batch size of 64. It was run for 30 epochs, which took approximately twelve hours.

For the BERT verifier, we ran it with a batch size of 8, which was all that could fit into the GPU memory of an Azure NV6 machine. To begin, we ran the verifier on its own to train it. The verifier was trained for 4 epochs. This again lasted twelve hours, by design. In this period, average loss shrunk from 0.71 to 0.41, showing improvement in the classifier's ability to distinguish answerable and non-answerable questions.

After training the classifier on its own, it was attached to the BiDAF model. The joint model was then trained for a further twelve hours, in which it managed to train for three epochs. For speed reasons, this joint model did not train the BERT-based verifier further.

### 4.4 Results

| **Model** | **F1 Score** | **EM Score** |
|---|---|---|
| Baseline (30 epochs) | 60.49 | 57.26 |
| GPT verifier (30 epochs) | 52.19 | 52.19 |

Table 1: Results after 30 epochs

A further interesting piece of qualitative data is NLL. NLL was dramatically lower with the GPT verifier, 0.1438 relative to the baseline's 2.280 after 30 epochs. After the 3 epochs, 2.798 with the BERT-based verifier compared to 3.547 with the baseline.

| Model | F1 Score | EM Score |
|---|---|---|
| Baseline (3 epochs) | 51.89 | 49.04 |
| BERT verifier (3 epochs) | 49.94 | 49.68 |

Table 2: Results after 3 epochs

There are several potential explanations for the transformer's performance. The first is that adding the verifier added too many new parameters and made both the verifier and the answer generator hard to train. This is somewhat corroborated by a relatively slow decrease in NLL, a much slower decrease than was observed in the baseline. The other potential explanation is that the core hypothesis behind verification is wrong: that confidence in the answer generation step that typically informs a no-answer response actually is accurate. Given the increased power of the natural language understanding models that generate answers, this might well be the case.

The BERT-based verifier, on the other hand, was able to beat the baseline, at least on EM score. This indicates that the BERT-based verifier really is more powerful than the GPT-based verifier, which squares with current state-of-the-art research. BERT seems to have learned the relationship between answerability, question, and context better that the GPT did. However, the gain in EM score seems to have come at the cost of a decrease in F1 score. This would correspond to prediction no-answer too often. It would be an match when there really is no answer – a frequent enough occurrence to increase the EM score – but an overabundance of no-answer predictions will decrease the F1 score, since a no-answer would not match any part of the real answer. This bias toward no-answers is explained by the structure of the algorithm. It can check if a question is answerable, and if it's not, it will return no-answer. However, if it is marked as answerable, it will return the answer predicted by the BiDAF – which may well be a no-answer itself. While this oversight could be corrected given more time, applying this verifier in the current manner will only ever increase the frequency of no-answer responses. To that end, it seems better to evaluate the verifiers on the basis of EM score.

It is promising that BERT could beat the baseline EM score. However, the margin by which it did beat the baseline was minimal. It seems likely that the verifier simply wasn't good enough. The likeliest explanation is that the core hypothesis behind verification is wrong, as suggested for the GPT verifier. The confidence in the answer generation step that typically informs a no-answer response is good enough. It may not be entirely accurate. The gains from using a verifier do indicate there is something wrong with the process. However, the verifier isn't an oracle. Where a verifier might be able to do a slightly better job at figuring out if a question is answerable or not, it will not do a dramatically better one.

## 5 Analysis

The BERT-based verifier and GPT-based verifier regularly caused predictions of no-answer when in fact there was an answer. This fits with the explanation above; given an oversight, the verifiers will only increase the number of no-answer predictions made.

Notably, the baseline only occasionally predicts an answer when none exists. While it does happen, it is a comparatively rare problem, occurring in approximately a tenth of the questions. Solving this would give gains – as the BERT-based verifier showed – but even in the best-case scenario, it would not give radical improvement. Instead, as we in fact found, the potential upside is small, given the power of current answer generation models.

One issue with the verifier system as currently designed is that its representation of data is completely incompatible with the answer generator. This manifested in the form of the additional re-tokenization step, converting GloVe vectors to strings and then back to embeddings to be compatible with BERT's pre-trained weights. To a large extent, this is specific to the implementation, but further research into verifiers might run into it again. It isn't unreasonable to believe that the optimal answer generator could use a different system than the optimal verifier, and these systems would almost certainly rely on pre-trained weights. Any sort of conversion, despite being necessary in practice, adds wasted computation.

The fatal flaw of the BERT-based verifier, however, is its long training time. It took an order of magnitude longer to run each iteration with the verifier than it did without it. This is unacceptable, both from a time perspective and from a financial perspective, given the high cost of training on powerful GPUs. From a programmer's perspective, it makes it difficult to iterate quickly and try out novel approaches when an inordinately large amount of time goes toward training even a small sample. For reference, it took approximately three seconds to run a single iteration when training the BERT-based verifier (without the answer generator attached).

This high training time is to some extent dependent on the verifier chosen. The GPT-based verifier was dramatically faster than the BERT-based verifier. However, some degree of slowdown is inevitable. While not every verifier will have over one hundred million parameters like in BERT, anything relatively performant will contain a high number. Models keep on getting bigger and bigger; any verifier that is able to keep up with the answer generator (and thus provide gains relative to the generator alone) is likely to have about the same order of magnitude of parameters. Thus, any verifier will cause the same issues.

The parameters for the verifier cause other issues beyond speed. It eats up a half gigabyte of disk space for a single BERT-based verifier's parameters. It fills up GPU memory, leaving less space for an answer generator to do something complex. Batch sizes have to be reduced, even if the answer generator can accept a larger batch.

## 6 Conclusion

Verifiers are not worth the additional complexity, and they are certainly not worth the additional training time. Though they were once useful, state-of-the art answer generation has long surpassed them and they have few gains to give. Time spent developing a verifier could be better spent developing a superior answer generator. Time spent training a verifier could likewise go directly to training a better answer generator.

The core assumption that a verifier depends upon seems wrong. Though it might make intuitive sense to handle the problems of answer generation and answerability separately, they both depend on similar types of understanding. It is unecessary to run two distinct models when almost all of the computation is shared, particularly in the lower layers. Given the degree of similarity, it seems unlikely that a verifier would be able to specialize to the degree necessary to justify the complexity and cost.

While Hu et. al. acheived good results thanks to a verifier, this was using a now obsolete answer generation model. Similarly, when Sun. et. al. subsequently achieved good results with an architecture that contained a verifier, they had simultaneously introduced a new answer generation architecture. It's unclear that the verifier was actually responsible for the gains. Certainly, a verifier may be useful for some tasks, such as when there is some real world cost that only applies when a wrong answer is given. One such task might be question answering in search; if the model is unsure, it's better to return no highlighted answer than the wrong one. Even for this though, a distinct verifier is unnecessary; the confidence produced by the generator would be sufficient. Similarly, a better verifier could be developed than that outlined in this paper (perhaps even just by training longer), but it's unlikely that it would be able to overcome the inherent disadvantages of having a verifier. Verifiers might be able to squeeze a small amount more performance out of an answer generator, but that performance comes at a cost. Verifiers should not be used.

### 6.1 Future Work

The author would not recommend the pursuit of further work with verifiers. However, if further research is done, a few things seem clear, beyond just experimentation with different verifier architectures:

- The answer verifier should be connected to a state-of-the-art answer generator, not a baseline. While it's somewhat useful to see to what extent a verifier can improve a simple baseline, it would be more valuable to see how it could improve a more complex model. The advantages of a verifier are deeply tied to the weaknesses of the answer generator. One might reasonably expect large variations in the value of a verifier based on what generator it is hooked up to; a state-of-the-art system would thus have more value than the BiDAF baseline.
- Dedicate more training time – on the order of days instead of hours – to training the verifier. Given the number of parameters that a verifier adds, it is reasonable to expect it to require significantly more training time.
- Experiment with pre-training the sub-components more individually as well as experimenting with having them combined. This end-to-end system has an enormous number of parameters that are trained simultaneously, and the layers interact in a complex way. The verifier is not well-suited to be trained with a negative log likelihood loss; a cross entropy loss would be more effective for training it. On the other hand, training the whole model exclusively end-to-end might give better results as well, since the verifier and the generator could rely more on each other.
- One could follow Sun et. al.'s footsteps and work with combining the verifier's lower layers with the lower layers of the answer generator. This would make the system faster and reduce some of the disadvantages of having a verifier in the first place.
- A future researcher should seek to avoid the theoretically unnecessary re-tokenization steps that were required to make different architectures compatible with each other. These waste computation time, and should be eliminated if no dependence on pre-trained weights forbids it.
- One could also try letting the verifier choose whether or not to answer prior to generating an answer in the first place. Given the prevalence of no-answers, this could reduce computation time significantly.
- Eliminate the generator's ability to predict a no-answer at all, and depend solely on the verifier to produce no-answers. It is unecessary for them to be doing the same job, and eliminating this ability from the generator might make it able to specialize into answer generation more easily.

These approaches might lead to better results with verifiers. On the whole, however, experimenting with superior generators seems much more likely to achieve state-of-the-art results on question answering problems.

## References

[1] M. Hu, F. Wei, Y. Peng, Z. Huang, N. Yang, and D. Li, "Read + verify: Machine reading comprehension with unanswerable questions," 2018.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[3] F. Sun, L. Li, X. Qiu, and Y. Liu, "U-net: Machine reading comprehension with unanswerable questions," 2018.

[4] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, “Bidirectional attention flow for machine comprehension,” 2016.