Ryan QA

Ryan Rice
ryanrice@stanford.edu

Abstract

Open domain question answering is a field of natural language processing in which great strides are currently being made. The release of the SQuAD dataset kicked off a transformation within the field as it provided, for the first time, a large amount of high quality data. More recently, Google has developed and released a pre-trained language model, BERT, that can be extended to a variety of NLP tasks. Open domain question answering models have greatly improved in accuracy upon adopting BERT, and current state-of-the-art models fine-tune and extend this language representation model in their implementations. Therefore, I seek to train an effective question answering system by doing the same. The final proposed model is a weighted ensemble network consisting of individual networks that have been trained using a variety hyperparameters and architectures. This model achieves scores of 74.472 EM and 77.129 F1 on the test dataset.

1 Introduction

In the context of this report, open domain question answering refers to the answering of, or identifying as unanswerable, questions based on a context passage in which the subject of these question-context pairing inputs is not pre-specified. This task is difficult for two reasons. The first is that the questions are open domain. The lack of subject requires models to understand language more generally as a means of parsing out the relationships between questions and their contexts in order to find answers. This portion of the task was the focus of the SQuAD 1.1 dataset, and over time, models achieved very high performance on this with Google's BERT eventually surpassing human performance. This success sparked an update to the dataset in which questions were added that could not be answered based on their given contexts. The is the second difficulty – the potential for questions to be unanswerable. For the SQuAD 2.0 dataset, models must now develop a deeper understanding of how or if the question and context relate.

Extensions of BERT – as opposed to the plain fine-tuning that is state-of-the-art for SQuAD 1.1 – are making progress although the task is far from solved. Models, in general, are biased towards not answering questions due to the construction of the dataset – about half of the questions cannot be answered – and this tendency hurts performance. In order to cope with these shortcomings in single models, an ensemble network in which the individual components are trained with different architectures was used. This allows the overall network to capture the union of answered questions and mitigate the bias towards marking questions unanswerable. The ensemble network outperforms its individual components, and further analysis of accuracy and which examples benefit from this architecture can be found in the Analysis section.

2 Related Work

There have been a variety of approaches to question answering since the initial release of the SQuAD dataset; however, the SQuAD 2.0 dataset has almost exclusively seen models architected as an extension of Google's BERT.

At its core, BERT is a language model that it is architected using bidirectional Transformers. It can be fine-tuned to a variety of NLP tasks by adding and training additional layers. At the time of its release, it achieved state-of-the-art results on eleven NLP tasks, so its capabilities extend far beyond question answering. The model described throughout this paper takes advantage of these great results and fine-tunes BERT in its implementation.

Another approach that very loosely inspired the attention architecture described later in the paper is the Dynamic Coattention Network Xiong at al., 2017 [6]. This network pre-dates BERT and was developed by Salesforce Research. At the time of its release, it achieved state-of-the-art for question answering. An important piece of its architecture is its coattention encoder. The encoder uses attention to compute an affinity matrix for representing the context passage and the question in relation to one another. While not the same implementation, the attention architecture used is inspired by this coattention encoder.

3 Approach

The final network is a weighted ensemble consisting of four separately trained networks. Each of these networks utilizes Google's BERT as the underlying language representation model, and details of this language model's architecture can be found in Devlin et al., 2018 [4]. Additionally, each of the contributing networks is trained using different hyperparameters and, in some cases, architectures. This approach allows for the overall model to better capture different features of language and succeed on different types of questions.

3.1 Architecture

All single models are trained by fine-tuning the base BERT model to the SQuAD 2.0 dataset using Google's Tensorflow implementation [1]. HuggingFace's PyTorch implementation was initially fine-tuned as well for comparison although the results were not as good [2].

Two primary architectures are utilized in the final network. The first architecture is a vanilla fine-tuning of the base BERT model that makes use of a single linear layer, in addition to BERT, for predicting the start and end tokens. The second architecture utilizes multiplicative attention by adding a linear layer to BERT that generates start token predictions and an intermediate end token state. The start token predictions are then multiplied element-wise to the end state to generate the end token predictions based on the start token predictions.

The first architecture is modeled by Figure 1. The question and context passage are concatenated and fed into the pre-trained BERT language model to obtain a final hidden state, $H_{final} \in \mathbb{R}^{h \times l}$ where h is the number of hidden units and l is the maximum sequence length. The output of BERT is then passed through a single linear layer, $W \in \mathbb{R}^{2 \times h}$, such that

$$L = H_{final}W + b = [L_{start}; L_{end}] \in \mathbb{R}^{2 \times l}$$

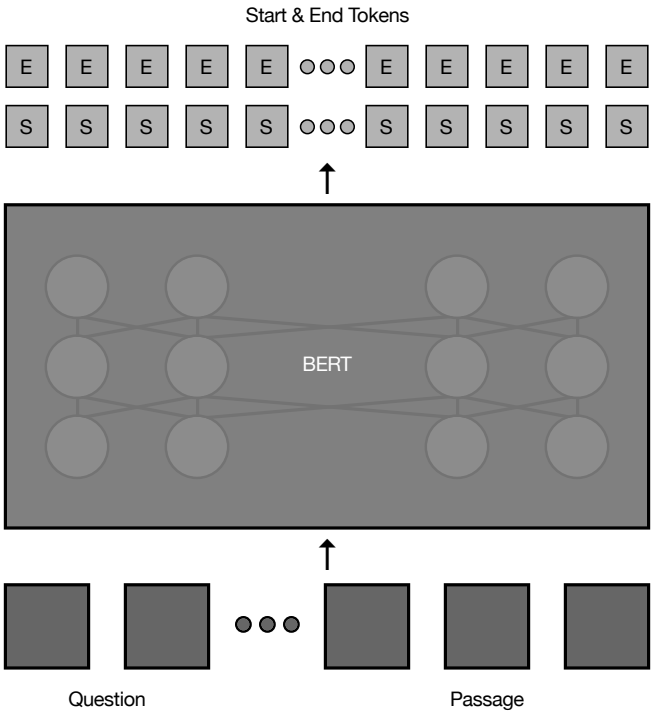

Figure 1: Vanilla Architecture

A softmax is then computed over L to find the most probable start and end tokens. The constraint that the end token must appear after the start token is added to generate logical outputs.

The second architecture is modeled by Figure 2. As before, the question and context passage are concatenated and fed into BERT to obtain the final hidden state, $H_{final} \in \mathbb{R}^{h \times l}$ where h is the number of hidden units and l is the maximum sequence length. This is then also passed through a single linear layer, $W \in \mathbb{R}^{2 \times h}$, such that

$$L = H_{final}W + b = [L_{start}; L_{end}] \in \mathbb{R}^{2 \times l}$$

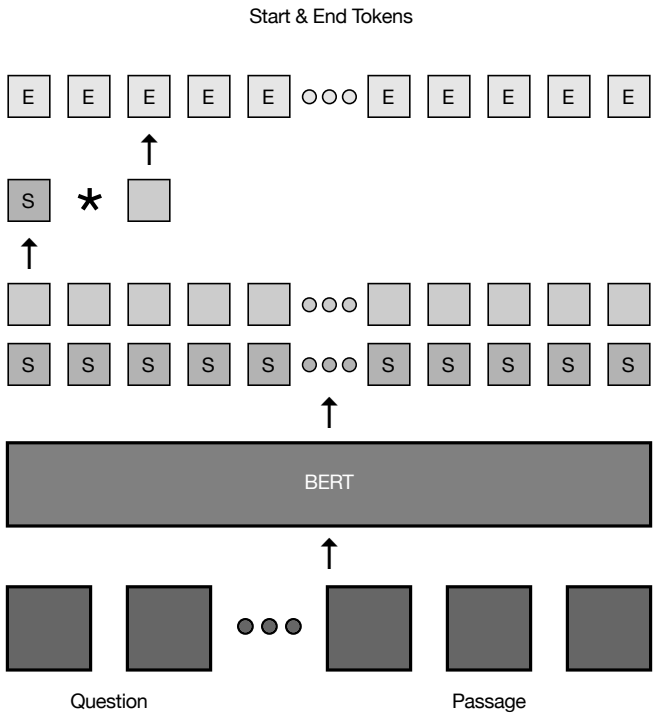

Figure 2: Attention Architecture

Attention is then applied between the start token predictions and the intermediate end token predictions to obtain the final end token predictions.

$$L_{final} = [L_{start}; L_{start} * L_{end}] \in \mathbb{R}^{2 \times l}$$

Finally a softmax is then computed over L_{final} to find the most probable start and end tokens. Again, the constraint that the end token must appear after the start token is added to generate logical outputs.

Single models based on these architectures are then combined to form an ensemble network. The five best predictions from each model, along with their probabilities, are used. Each single model is given a weight to scale its probabilities relative to the other networks. The final output is the prediction with the highest cumulative probability. More concretely, four single models – three vanilla and one attention model – are used with two of the vanilla models having a weight of 2 while the third vanilla and the attention model have weights of 1. Details about the hyperparameters used to train each of these single models as well as how the weights were reached can be found in the Experiments section.

3.2 Baseline

The baseline model used is a BiDAF whose implementation is provided by Chris Chute [3]. Details of this model's architecture can be found in Seo et al., 2016 [5]. The model was trained for 11 epochs and achieved an EM score of 55.806 and an F1 score of 58.797.

4 Experiments

Experimentation consisted of training single models with various hyperparameters and architectures and comparing these single models for performance. Additionally, a variety of ensemble networks, consisting of different combinations of these single models, were also compared. In the ensemble networks, the weights for each single model were adjusted as a final hyperparameter.

4.1 Data

The SQuAD 2.0 dataset was used to train the model. It consists of a context and question that are concatenated and converted to word embeddings before being fed into the model. The labels are answer spans from the context. The data split is as follows: 129,941 training examples, 6078 dev examples, and 5915 test examples.

4.2 Evaluation Methods

The experimental models were primarily evaluated using EM and F1 scores, and these metrics are fairly standard to question answering. In addition to these automated scores, manual error analysis was key in understanding the shortcoming of the model. Specifically, the weights for the ensemble network were derived by analyzing the comparative failures of the single models.

4.3 Details

Experimentation included the testing of implementation, number of epochs, batch size, learning rate, and max sequence length. The first stage of experiments saw comparisons between implementations and the results are described in Table 1 with the Tensorflow implementation performing the best.

Based on these results, all further models were trained in Tensorflow. The next phase of experimentation included tuning the number of training epochs and analyzing the performance of the attention mechanism. The results of these experiments are outlined in Table 2, and neither variable was found to have a significant impact on the performance of the model.

Given that the performance was approximately the same with respect to both number of epochs and architecture, the vanilla model with 2 training epochs was used in the final experiments as it was the fastest to train. The final, and most important, hyperparameters that were tuned were maximum sequence length and ensemble weights. The results of each are described in Table 3 and Table 4, respectively.

4.4 Results

Below are the results for different fine-tuning different implementations compared with the performance of the baseline. The Tensorflow implementation well exceeds the baseline's performance while the HuggingFace implementation falls short.

Table 1

Implementation	EM	F1
Tensorflow	**72.409**	**75.053**
Baseline BiDAF	55.806	58.797
HuggingFace	46.693	47.066

Results for different numbers of training epochs and architecture types are below. Performance is approximately the same across both variables although the vanilla model at 2 epochs trains the fastest.

Table 2

Architecture	Epochs	EM	F1
Vanilla	2	**72.409**	75.053
Vanilla	3	72.392	**75.250**
Attention	2	71.520	74.465
Baseline	-	55.806	58.797

The hyperparameter with the biggest impact on performance was maximum sequence length. Longer sequences can perform better as the additional context prevents the model from falsely predicting no answer as often. This improvement, however, is not a direct extension of the performance of the smaller sequence length models which is why the weighted ensemble network performs much better. Further analysis of this phenomenon can be found in the Analysis section.

Table 3

Architecture	Max Sequence Length	EM	F1
Vanilla	128	72.409	75.053
Vanilla	256	**72.919**	**76.300**
Vanilla	384	71.405	74.634
Baseline	-	55.806	58.797

Finally, the different single models were combined in various configurations to maximize performance. Results from some of these configurations are below with the full four model ensemble performing the best. This ensemble weights the single models such that the contributions from each max sequence length are equal.

Table 4

Architecture (Max Length + Attention/Vanilla)	Weights	EM	F1
128A + 128V	1, 1	73.050	75.853
128A + 128V + 256V	1, 1, 2	74.679	77.304
256V + 384V	1, 1	74.794	77.603
128A + 128V + 256V + 384V	1, 1, 1, 1	75.617	78.114
128A + 128V + 256V + 384V	1, 1, 2, 2	**75.716**	**78.338**
Baseline	-	55.806	58.797

5 Analysis

The error in the single models that comprise the ensemble network largely lies in a failure to respond to answerable questions. The networks are biased towards marking questions as unanswerable, but this is not surprising given that approximately half of the questions are unanswerable. What is surprising is that the models can fail to answer different questions. This phenomenon allows for the weighted ensemble to capture the different successes of each of the individual models and ultimately maximize performance. An example of this is the following question and an excerpt from its context along with the individual models' predictions.

Who did Rollo sign the treaty of Saint-Clair-sur-Epte with?

...was established by the treaty of Saint-Clair-sur-Epte between King Charles III of West Francia and the famed Viking ruler Rollo...

Table 4

Architecture (Max Length + Attention/Vanilla)	Prediction	Probability
128A	King Charles III of West Francia	0.462
128V	King Charles III of West Francia	0.740
256V	King Charles III of West Francia	0.584
384V	No Answer	0.932

The ground-truth answer is King Charles III. Three of the four models are close but quite not achieving exact match accuracy while the 384V model is the only one that fails to answer this question. It is in this manner that the ensemble network benefits from the variety of single models.

That being said, there still exist answerable questions that none of the single models answer, and this error perpetuates into the final ensemble which limits performance. An example of this is the following question and an excerpt from its context along with the individual models' predictions.

What was the Norman religion?

...descendants of Rollo's Vikings and their Frankish wives would replace the Norse religion and Old Norse language with Catholicism (Christianity) and the Gallo-Romance language of the local people...

Table 4

Architecture (Max Length + Attention/Vanilla)	Prediction	Probability
128A	No Answer	0.746
128V	No Answer	0.517
256V	No Answer	0.724
384V	No Answer	0.830

The ground truth answer is Catholicism but all of the individual models predict no answer such that the ensemble, incorrectly, also predicts no answer.

6 Conclusion

Ultimately, the most successful model was the ensemble including weights normalizing the contributions of the single models with respect to max sequence length. It was surprising to find that not only did sequence length have the largest, if not the only, impact on performance, but that larger sequence lengths are not exclusively better. The merits of the smaller maximum sequences should not be overlooked, and the ensemble allows both to contribute to an overall system that outperforms each by itself.

The primary limitation for this work was memory. For the longer sequence lengths, batch size had to be reduced significantly, perhaps to some performance cost. In all cases, a batch size of 32 – suggested by Google's findings for maximizing performance on this task – was not able to be used. Furthermore, for all experiments the base BERT model was used. Given GPUs with more memory, the large model could have been experimented with in a timely manner which likely would have improved performance.

References

[1] URL: https://github.com/google-research/bert.

[2] URL: https://github.com/huggingface/pytorch-pretrained-BERT.

[3] URL: https://github.com/chrischute/squad.

[4] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018.

[5] Minjoon Seo et al. *Bidirectional attention flow for machine comprehension*. 2016.

[6] Caiming Xiong, Victor Zhong, and Richard Socher. *Dynamic Coattention Networks for Question Answering*. 2017.