SQuAD 2.0 Project Report

Konrad Morzkowski
Stanford University
konradm@stanford.edu

Liyang Sun
Stanford University
liyangs@stanford.edu

Phillip Hoovestol
Stanford University
philhoov@stanford.edu

Abstract

The 2.0 version of the large-scale dataset *Stanford Question Answering Dataset* (SQuAD 2.0) allows researchers to design AI models for reading comprehension tasks under challenging constraints. Our goal is to implement, tune and compare various existing popular NLP models on this dataset, and also to understand how changes (regularization, hyperparameters, ...) affect these models' performance. We especially focused on the BERT model, as it is part of almost every state-of-the-art model on this task. Using qualitative analysis and ensemble methods, we successfully managed to improve our model, while understanding what caused each improvement.

1 Introduction

Machine reading comprehension has become one of the most popular centers of interest in natural language understanding. In particular, one of the machine reading comprehension task, the problem of general question answering, has many potential applications. Imagine being able to ask your phone about anything, like what Google Assistant and Siri are trying to do, but with more complex questions. Imagine how much it could help lawyers if they could simply ask a machine specific questions about the law instead of having to sift through tons of material themselves.

While not exactly aiming for this level of complexity, the SQuAD 2.0 challenge is a way to measure how well a machine can answer general questions by extraction, based on paragraphs from Wikipedia articles (see [1]). The major improvement from dataset SQuAD 1.1 to SQuAD 2.0 is the addition of unanswerable questions. Indeed, the models which performed well on the SQuAD challenge (without unanswerable questions) were not robust to distracting or partially false sentences. Furthermore, in SQuAD 2.0, the unanswerable questions are created so that they are relevant to the associated paragraph and so that there can be a plausible, but wrong answer. This method is more challenging than the existing automatic or rule-based methods and enforces models on SQuAD 2.0 to robustly understand when it faces an unanswerable question, while being able to accurately extract right answers.

We will describe in this paper our work and findings, which models we implemented and studied, as well as our experiments with hyperparameters tuning and ensemble methods.

2 Related Work

The task of question answering on the SQuAD 2.0 dataset is very new, thus there are few research papers about it. However, looking at the current leaderboard[1], we can see that BERT ([4]) and ensemble models are ubiquitous. Many models which performed very well are unknown, as the research teams are still working on them, but some of them are already openly published, such as the Bi-Directional Attention Flow (BiDAF, [2]) and the Attention-over-Attention (AoA, [6]) models. We

[1]https://rajpurkar.github.io/SQuAD-explorer/

mainly based our work on these papers, to get a first global overview of how models can handle this task, and to see if different models could be relevantly combined.

Concerning the ensemble methods, there was not many papers, but Dietterich's work on ensemble methods ([8]) gave us a very complete overview on how to ensemble various models.

We also took a look at which models performed well on the SQuAD 1.1 challenge (without the unanswerable questions), but were not performing as well on the second version of the dataset, such as QANet ([9]).

Finally, we explored other less known methods and techniques, such as one described by Sachan et. al. in [7]. Synthetics-self training is used by one of the top models on the leaderboard, and this paper outlines an approach where we augment the data by creating a model that creates its own questions based on the context paragraph. Which questions to feed into the model is decided by multiple heuristics measuring the difficulty of each questions. However, we did not have time to apply it to our models.

3 Our Models

3.1 Baseline Model

The baseline model is a simplified version of the Bi-Directional Attention Flow (BiDAF) model, which is defined in [2]. Contrary to the original model, the baseline model only considers word-level embeddings for the inputs.

In brief, the model is composed of the following layers:

- Embedding layer: inputs are embedded using GloVe pre-trained word vectors, then projected and passed through a two-layer Highway network (described in [10]).
- Encoder layer: output of the embedding layer is passed through a bidirectional LSTM.
- Attention layer: bi-directional attention flow layer, we do not detail it here, as it is already detailed in the project handout.
- Modeling layer: two-layer bidirectional LSTM.
- Output layer: concatenation of outputs of attention layer and modeling layer (resp. modeling layer after being passed through an other bi-directional LSTM), which is then projected and passed through a softmax to get the probability of each word being the start (resp. end) of the answer span.

3.2 Bi-Directional Attention Flow Model with Character-Level Embeddings

We implemented the original BiDAF model, which differs from the baseline model only through its embedding layer, where character-level embeddings are included.

The character-level embeddings are obtained following the method of Kim in [3]. In particular, we use convolutional neural networks for the character-level embedding of each word: characters are embedded into vectors (which are randomly initialized), which are passed through the CNN and then max-pooled over the entire width. The character-level embedding is then concatenated with the word-level embedding, which are then projected and passed through the two-layer Highway network.

3.3 BERT Model

We are briefly going to describe the BERT model (defined in [4]) and the changes that are required for question answering. BERT is based on the transformer encoder architecture ([5]) and is currently used as part of most state-of-the-art models in many natural language processing tasks.

The inputs are embedded through three different components. First, they are tokenized as usual and embedded using pre-trained token embeddings. Then, a positional embedding is added in order to give the model positional information, since it is attention based. Lastly, a sentence embedding is added, which differs from sentence to sentence but is the same for every word in a given sentence. There are also two minor additions: we add a <CLS> token at the beginning of the input, and a <SEP> token between the context and the question.

The embeddings are then passed into the transformer encoder, which is a stack of identical encoder blocks:

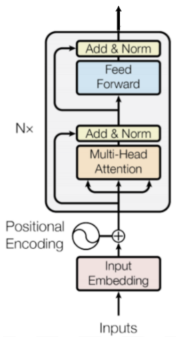

Figure 1: A visual representation of one encoder block, with the input layer

The most important part of one block is the multi-head attention layer. It computes several scaled dot product attention vectors called "heads", then concatenates them into a single attention vector, which is projected using a learnable weight-matrix so that the output is the same shape as the original input.

The output of the multi-head attention layer is then passed through the Add&Norm layer: the output goes through a dropout layer, is added to the original input embeddings (to not forget the former learned information) and finally normalized.

The final component of the encoder block is the feed-forward linear layer. It's a very simple layer consisting of two linear layers with a ReLU activation function in-between.

The major contribution of [4] in the BERT model is its pre-training. BERT is pre-trained on two tasks: Masked Language Modeling, which allows the model to be bidirectional and ensures that the model understands how every single word relates to its context, and next sentence prediction which helps the model learn relative positions between sentences, which is essential for question answering and other natural language inference tasks. The important part here is that we can get the pre-trained weights for our own implementation.

The adapted output layer of BERT for SQuAD 2.0 in particular is to pass all the hidden outputs through two linear and softmax layers so that we get the probabilities of every word being the start of an answer and the end of an answer respectively. Here, the <CLS> token is used as the no-answer token: if it has higher score than all the possible answer spans, then BERT predicts no-answer.

All that is left is fine-tuning the pre-trained BERT model with our own data !

3.4 Attention Over Attention Model

In an attempt to improve Bert's performance, we implemented a modified version of the Attention-over-Attention network described in [6]. In the original version, all of the attention computations are done on the output of a bi-directional GRU. Instead we did the computations on the output of BERT.

The computations are as following:

- Get a similarity matrix M by doing a matrix multiplication between the context matrix and the transpose of the question matrix.
- Two matrices α and β are calculated by taking the column-wise softmax and row-wise softmax respectively.
- Both α and β are also averaged and stored in different vectors $alpha\_sum$, and $beta\_sum$. Computing $alpha\_sum$ is the first difference between our approach and the paper's.
- The document attention is then computed by doing a matrix multiplication of α with $beta\_sum$, resulting in a vector with the same length as the context.
- Because BERT takes the whole sequence (context and question) as input we had to do the same thing for the question by doing a matrix multiplication of β with $alpha\_sum$.

We then concatenate the results to get our final result of the same length as the original sequence. In order not to lose BERT's hidden states, we simply use this result to scale them.

3.5 Ensemble Model

Ensembling models is a method of combining different individual models' outputs to get a "group decision" output. As stated in [8], the goal of this technique is to improve overall predictions by reducing bias and variance on one side, and on the other side to capture and merge the different key strengths of each individual model.

We decided to implement ensemble models based on majority votes between outputs of different models we built. Indeed, this method is one of the more common and reliable methods and is straightforward to implement. Commonly used in classification task, each individual model is trained separately, then for each output the ensemble model picks the output which has the most occurrences among individual models.

On top of the majority vote method, rankings can be made between models to weight each vote. In particular, this allowed us to set deterministic rules for breaking ties in our voting: given a predetermined ranking of all the models by how well they performed individually, any time there is a tie during the voting phase, the output of the highest ranked model wins.

4 Experiments

4.1 Data

We are working on the SQUAD 2.0 challenge, with given training, development and testing sets. The train/dev/test split is according to the handout and in particular have respectively 129941/5951/5915 examples.

4.2 Evaluation Method

We are evaluating our models with the EM and F1 metrics, while putting a larger emphasis on F1 over EM since that is how the competition will be judged. Additionally, during training we looked at the negative log-likelihood, which can show us how well our models fit the data, and if they start overfitting.

4.2.1 Adjustments in Implementation

We initially successfully ran HuggingFace's PyTorch implementation of BERT[2] for question answering. The only changes we made is to reduce the batch size to 6.

We have included HuggingFace's version into the starter code we received, however we did not adapt it so that we could follow the training process on TensorBoard. We changed the default parameters according to our hardware constraints (5 hours of training per epoch), experimented on wrapping the BERT layers into a *no_grad()* to only train the last linear fine-tuning layer, which didn't work out since the model ended up only producing N/A answers. As for the inputs to the model we concatenated the context and question tensors and created an indicator tensor, in accordance with the documentation. We also did a log-softmax on the output in order to match it with the output from the baseline model.

All the other models and techniques were implemented from scratch by us.

In particular, the ensemble models were implemented by iterating through all the input models' answer csv files then recording for each question how many of each answer were chosen by the models. Then a new set of answers was generated based on majority voting and models' ranking. We tried ensembling different combinations of our models to see which would perform the best (the results of that experiment are in a later section).

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

4.3 Preliminary Results

The baseline and character-level BIDAF were trained with the exact same configuration, batch-size of 64, 30 epochs and a learning rate of 0.5 decaying over time. Since the character-level BIDAF does more processing in its embedding layer, its running time was slightly longer, but we believe that the trade-off between performance and running time is worth it.

Here are the obtained results on the development set:

Model	Dev F1	Dev EM	Training time (in h)
Baseline	61.44	57.92	8.5
Char-BiDAF	62.24	59.049	10.5

We did not spend time tuning the hyperparameters on these models, as our main focus in this project was to qualitatively explore how relevant changes could be made in order to improve models, more than doing "blind" gridsearch for each hyperparameter.

We can however observe that the addition of character-based embeddings improved the performance of the BiDAF model consequently. This was expected as adding the character embeddings allowed for a better representation (with different granularity) of each sentences.

4.4 BERT Results and Analysis

We spent a lot of time experimenting with BERT, as it is currently the most popular question answering model. Unless stated otherwise, we ran BERT with the default parameters: batch size of 6, maximum sequence length of 384, maximum question length of 64, learning rate of 5.10^{-5}, 2 epochs, maximum answer length of 30 and a **null score difference threshold** of 0 (we will detail this last parameter later).

Here are the obtained results on the development set, each epoch taking approximately 5h to train:

	Model	Dev F1	Dev EM
1	BERT (max_ans_len 15)	72.98	69.96
2	BERT (lr 3e-5, thresh 1)	75.067	71.915
3	BERT (batch 8, lr 2e-5, thresh 2, accumulation gradient 2)	74.851	71.603
4	BERT (starting from 2. BERT, lr 1e-5, thresh 0.5)	75.436	72.178
5	BERT (lr 3e-5, thresh -1.0)	76.117	72.985
6	BERT (from 5. BERT, lr 2e-5, thresh -2.0)	75.164	71.915
7	BERT (from 5. BERT, lr 3e-5, thresh -1.0)	74.438	71.306
8	BERT (lr 3e-5, thresh -3.0)	75.852	73.034
9	BERT (lr 5e-5, thresh 3.0)	72.326	68.871

In the table, *thresh* designates the null score difference threshold, and *starting from 2. BERT* means that we loaded the weights from the second BERT model and trained it further.

The first observation we can make is that BERT performs a lot better than the BiDAF models. However, this is a bit unfair, as it is a PCE model, contrary to the other two models which are non-PCE models.

The second observation we made is that the **null score difference threshold** seems to have a substantial role in the performance of BERT models. Given a single example, BERT computes a "null score" (score for no-answer) and a "best non-null score" (best score for an answer). If the difference between the null score and the best non-null score is higher than this threshold, then BERT outputs no-answer.

This lead us to analyze the errors made by each model by putting them into three categories:

- Type 1: predicting no answer when there is an answer
- Type 2: predicting an answer when there is no answer
- Type 3: predicting the wrong answer

The errors for some of our models can be seen in the table below.

Model	Type 1	Type 2	Type 3	Total
Char-BiDAF	371	1023	512	1906 (less examples than BERT)
BERT 1 (thresh 0)	428	984	494	1906
BERT 2 (thresh 1)	329	960	498	1787
BERT 3 (thresh 2)	270	1025	513	1808
BERT 9 (thresh 3)	240	1128	538	1906
BERT 4 (thresh 0.5)	337	912	529	1778
BERT 5 (thresh -1)	451	782	477	1710
BERT 8 (thresh -3)	647	637	416	1700
Ensamble*	423	778	446	1647

*Ensemble with BERT 3, 4, 5, 6, 8, 9 + BiDAF + Baseline + null (BERT 9), described below.

Initially splitting up the errors in this way made it easier for us to think about changes we could make in order to decrease one of the categories and guided us throughout the project. Indeed, we can see that the null score difference threshold allows us to control the trade-off between errors of type 1 vs 2 and 3 !

By increasing the threshold we decreased the amount of errors in the Type 1 category while increasing errors in the two other categories. Decreasing the threshold has the opposite effect. Additionally, when changing the threshold from the initial value of 0 the overall number of errors decreased. This is ideal for ensembling, since we will have many decent models that are good at avoiding different types of errors. The ensamble, as expected, decreases errors of every type as compared to the first version.

We also tried doing a fully qualitative approach and analyze every question by itself in order to figure out why BERT models made each error, but that turned out to be less fruitful since it is harder to classify the errors and way more time-consuming to get a substantial enough amount of data to make a decision, while the process above is straightforward.

4.4.1 AoA Results

We implemented the Attention-over-Attention mechanism on top of the BERT model. However, it performed very poorly (worse than the baseline).

The reason we believe AoA didn't perform well is that the scaling of BERT's inputs is unnecessary since it already has calculated its own attention in a more sophisticated way than the AoA mechanism. Thus, by rescaling it, we are essentially bringing it down to the level of AoA and not augmenting it.

However, some of the top 10 models on the leaderboard use AoA, meaning that there might be something wrong with our implementation. What we could have done instead is implemented the AoA model according to the paper, instead of trying to augment BERT, and then used it as part of our ensemble model. The reason it worked in the original paper is that no form of attention was used before doing the AoA computations, so the AoA output would also be the final output.

(Although the paper didn't mention any testing on the SQuAD 2.0 dataset, we found a powerpoint made by the authors where they mentioned that they sucessfully adapted the AoA code for SQuAD).

4.5 Ensemble Results

To ensemble the models, many alternatives were possible:

- Naive majority: majority vote without tie-breaking rules
- Majority with ranking: majority vote with models' ranking as tie-breaker
- Different combinations of models

Based on [8], where Dietterich claims that ensemble model performs especially well if its individual models are good and diverse (in the sense that they make different errors), we added another custom trick to our ensemble model: using the qualitative analysis on the null score difference threshold in BERT models, we included one BERT model with quite high threshold, and another one with low threshold, even if they had lower performances. Indeed, the former model makes less type 1 errors and the latter makes less type 2 errors.

Furthermore, for the model with highest threshold, it is less likely to predict no-answer on a given question and therefore when it did predict no-answer, we know it was done with a high degree of confidence. We then adjusted our implementation further, so that whenever no-answer was predicted by this model, our ensemble immediately outputs no-answer for that question.

Below is a table with various ensemble experiments on the development set. When not specified:

- All of our individual models are included, except BERT 9, as we included it last.
- We use majority, with ranking tie-breaker.

Model	Dev F1	Dev EM
Ensemble (naive majority, no ranking)	76.473	73.774
Ensemble	76.610	73.873
Ensemble + null (BERT 3)	76.725	73.988
Ensemble without baseline + null (BERT 3)	76.449	73.643
Ensemble without worse 2 BERT + null (BERT 3)	76.764	73.939
Ensemble with BERT 3, 4, 5, 6, 8 + null (BERT 3)	76.827	73.791
Ensemble with BERT 3, 4, 5, 6, 8 + BiDAF + Baseline + null (BERT 3)	76.829	73.972
Ensemble with BERT 3, 4, 5, 6, 8, 9 + BiDAF + Baseline + null (BERT 9)	77.065	74.186

In the table, *null(BERT 3)* refers to our custom trick: if BERT 3 predicts no-answer, then the ensemble model predicts no-answer.

The first comment we can make, is that as expected, adding a ranking tie-breaker improves the model, and likewise for the custom trick.

Furthermore, the ensemble model does slightly better when it is composed of diverse individual models. For instance, most of the time, adding the baseline and BiDAF models, even if they have worse performance, does not hurt, and can even improve the model slightly. A more substantial example is the addition of the BERT 9 model with higher threshold (3.0), which improved the model a lot more, supporting our analysis and strategies.

Another interesting observation is that some individual models hinder the learning of the ensemble model. This is very likely due to the majority vote system: if bad models all predict the same wrong answer, then it can win the vote... We did not find an efficient way to rule out these cases, apart from trying various combinations of individual models.

5 Test Results

An important thing we have kept in mind is that all of our analysis are based on the results on the development set. This is non negligible as it is possible that some of our findings could be biased towards its content and examples. However, as our findings seem coherent and not too biased, we assumed they were generalizable.

Under this assumption, we picked our best ensemble model and our best individual model, and we got the following results on the test set:

Model	Test F1	Test EM
Ensemble with BERT 3, 4, 5, 6, 8, 9 + BiDAF + Baseline + null (BERT 9)	77.398	74.489
BERT 5	76.688	73.609

These results are coherent with all of our previous analysis and techniques. Our models do not seem to overfit on the development set, which supports our previous assumption.

6 Conclusion

This project on machine reading comprehension using the SQuAD 2.0 dataset was very interesting and helped us get a good understanding on current state-of-the-art question answering models (especially BERT), on natural language programming tasks and models, but most importantly, on how to build, train, evaluate and adapt neural networks in a more general point of view.

We attempted to implement a wide variety of models, and while many succeeded, there were others where optimal performance was not reached or that we could not implement fully in the time we had. However, these trials and errors were a key part of the process for finding which methods worked best with our problem and coding framework, an integral skill in developing these models in any setting. Being able to look into errors and make modeling decisions based on what we saw was a rewarding process, since it feels like it gave us more control over the performance of our model and the direction we wanted to go in.

Developing our ensemble method and noting the improvement it gave our models provided significant takeaways in particular. The process gave us a chance to go through the steps of researching a new method all the way to implementing it successfully without much prior knowledge of it. The ensemble's success in improving overall performance makes it a huge tool to potentially use in future projects. This is also the case because it is both intuitive and relatively simple to implement from scratch for any type of machine learning model of similar task, as it can be handled in many cases solely from the output of the models being ensembled.

As for future work, there is definitely some more performance to be squeezed out of our current models by simply training longer. A good combination with that would be the synthetic self-training we mentioned earlier but didn't have time to implement since we wanted to focus on improving our models by changes of parameters or architecture instead of simply training for longer.

References

[1] Rajpurkar P., Jia R., & Liang P. (2018) Know What You Don't Know: Unanswerable Questions for SQuAD. *arXiv preprint arXiv:1806.03822*

[2] Seo M., Kembhavi A., Farhadi A., & Hajishirzi H. (2017) Bi-Directional Attention Flow for Machine Comprehension. *arXiv preprint arXiv:1611.01603*

[3] Kim Y. (2014) Convolutional Neural Networks for Sentence Classification. *Conference on EMNLP, 2014*

[4] Devlin J., Chang M.-W., Lee K., & Toutanova K. (2018) Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*

[5] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., & Polosukhin I. (2017) Attention Is All You Need. *Advances in Neural Information Processing Systems, 5998–6008*

[6] Cui Y., Chen Z., Wei S., Wang S., Liu T. & Hu G. (2017) Attention-over-Attention Neural Networks for Reading Comprehension. *arXiv preprint arXiv:1607.04423v4*

[7] Sachan M., Xing E. (2018) Self-Training for Jointly Learning to Ask and Answer Questions.

[8] Dietterich T. G. (2000) Ensemble Methods in Machine Learning.

[9] Wei Yu A., Dohan D., Luong M.-T., Zhao R., Chen K., Norouzi M., & Le Q. V. (2018) QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. *arXiv preprint arXiv:1804.09541*

[10] Srivastava R.K., Greff K., & Schmidhuber J. (2015) Highway Networks. *arXiv preprint arXiv:1505.00387*