Tackling SQuAD 2.0 with Ensemble Methods and Data Augmentation

Stanford CS224N {Default-PCE} Project

Nikka Mofid
Department of Electrical Engineering
Stanford University
nmofid@stanford.edu

Emanuel Pinilla
Department of Computer Science
Stanford University
epinilla@stanford.edu

Elijah Freeman
Department of Computer Science
Stanford University
elijahf@stanford.edu

Abstract

In recent years, machine reading comprehension has become one of the most central tasks for natural language processing and understanding. Thus, given machine reading comprehension’s interesting challenges, for our final project, we will be tackling the Stanford Question Answering Dataset (SQuAD 2.0), which is one of the largest question answering datasets whose questions and answers emulate real reading comprehension situations [1]. Specifically, we will be utilizing the pre-trained BERT base and ALBERT base along with data augmentation and ensembling in order to develop a powerful model that can perform well on the SQuAD 2.0 dataset. In particular we explore augmenting the dataset using a variation of the “EDA: Easy Data Augmentation” algorithm and building different ensemble models composed of ALBERT, BERT, ALBERT/BERT trained with different hyperparameters, and augmented BERT/ALBERT, in order to analyze how they may improve the performance of our question answering model. We have found that augmenting the no answer questions in the SQuAD dataset was able to successfully improve the performance of BERT and utilizing BERT/ALBERT trained on augmented data in ensembles with weighted voting strategies is able to further boost performance. Currently, we are 11th place on the Dev PCE Leaderboard and 8th place on the TEST PCE Leaderboard with our best ensemble which utilized weighted voting strategies and multiple ALBERTs, including ALBERT with no answer augmentation.

1 Key Information to include

- Project Grading: For Grade (Option 3 on Piazza)

2 Introduction

Over the course of the last decade, Question Answering or “machine reading comprehension” has become one of the most highly research tasks in natural language processing. Though seemingly straightforward, Question Answering is an extremely challenging task as it requires the machine learning agent to comprehend text and the deep relationships between words in context. Our project specifically focuses on building a machine reading comprehension model for the Stanford Question Answering Dataset known as SQuAD 2.0, which is one of the largest question answering datasets whose questions and according answers emulate real reading comprehension situations and include
both answerable and unanswerable question[1]. The aim of this project is specifically to build a model that can efficiently and accurately perform contextual question answering, meaning that given a context paragraph and a question, our model returns the span of the answer in the context.

Though there has been much work upon the Stanford Question Answering Dataset, researchers are still trying to find the best model to outperform the human oracle and handle the no-answer questions in the dataset. Currently, many of the best approaches to tackling the SQuAD 2.0 dataset involve using the pre-trained models such as BERT, ALBERT, and ELMo which not only have great performance and accuracy, given good hyperparameter tuning, but are also significantly more efficient than other contextual models as they do not need to be trained from scratch[2][3]. Thus, for our final project, we will be utilizing pre-trained BERT base and ALBERT base model along with data augmentation for no answer questions and ensembling in order to develop a model that can handle the SQuAD 2.0 dataset and aid in pushing the state of the art. In particular we explore augmenting the no answer questions in the dataset, balancing it, using a variation of the “EDA: Easy Data Augmentation” algorithm and building different ensemble models composed of ALBERT, BERT, BiDaf, and augmented BERT, in order to investigate and analyze how they improve the performance of our question answering model[4]. Doing this, we explore fine tuning of hyperparameters for our models and note the trade-offs between them and the ensembles they make up.

3 Related Work

In the past, there have been many different approaches to tackling the SQuAD 2.0 dataset. Many of the most successful methods have used ensembles of pre-trained models like BERT and ALBERT thus leading us to focus on utilizing pre-trained models for our final project.

BERT was first introduced in the paper “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, in which the authors propose and introduce “BERT”, a fine tuning based pre-trained representation model that achieves state of the art performance on a large suite of sentence level and token level tasks[2]. BERT is essentially a multi-layer bidirectional Transformer encoder which comes in two different sizes BERT base and BERT large with BERT base containing 12 layers and BERT large containing 24. At the time of the paper’s publication, BERT large was able to beat all other submissions on the SQuAD leaderboard and now almost all the top submission to the SQuAD leaderboard utilize BERT in some way. One of the main limitations of BERT, however, is its large model size which makes it difficult to train due to GPU/TPU memory limitations. For this reason, we thought it would be interesting to experiment in our project with ALBERT, a lite BERT.

ALBERT was introduced soon after BERT in order to provide an answer to BERT’s size issues by the paper “ALBERT: A Lite Bert for Self Supervised Learning of Language Representations”[3]. ALBERT uses parameter reduction techniques to lower memory consumption and increase the training speed of BERT, making it more amenable to different memory and efficiency constraints, leading to models that have been shown to scale better to the original BERT. Like BERT, ALBERT based models have been shown to high scores on a number of number of natural language tasks, with ALBERT often performing similarly or, with proper hyperparameter tuning, slightly better than BERT despite having less parameters on the SQuAD 2.0 challenge[3].

With the introduction of 50,000 no answer questions to the SQuAD dataset with SQuAD 2.0, the dataset, now suffers from an imbalance between answerable and unanswerable questions[1]. Data augmentation has been shown to be a viable method of balancing a dataset by generating more examples based on the ones provided and also has been shown to be an effective technique to improve the performance of models on many Natural Language Processing tasks[4]. One of the most popular data augmentation techniques for NLP tasks is Easy Data Augmentation, proposed in the paper “EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks”[4]. Easy Data Augmentation is designed to work by augmenting text for classification tasks, taking in one sentence from the training set and performing one of the following operations chosen at random: synonym replacement (SR), random swap (RS), random insertion (RI), and random deletion (RD) to create a new augmented example[4]. EDA has been shown to improve the performance of machine learning classifiers and in our project we adapt this technique to work for Question Answering with the intention of augmenting the no answer questions in the dataset in order to balance it.
4 Approach

4.1 Fine Tuning BERT and ALBERT

We spent quite a large amount of time fine tuning our BERT and ALBERT baselines so they would be able to reach peak performance. We first began with BERT and tried a number of different learning rates ranging from 7e-5 to 3e-5, settling on 3e-5 as the optimal one. We also tried a number of different batch sizes ranging from 5 to 8, and found that BERT performed best with a batch size of 8. For ALBERT, we also experimented with the different learning rates ultimately choosing the learning rate of 5e-5. We also settled on a batch size of 8 for ALBERT. From there, we fine tuned the number of checkpoint files or "savesteps" ("--savesteps") that our model saves so that they would not only fit on the VM, but also contain enough data for both BERT and ALBERT to make proper evaluations. After experimenting with a number of different save steps, we found that "savesteps" 1000 was the best parameter for performance and memory. In order to speed up training, we decided to use mixed precision training and downloaded the apex library from github and set the --fp16 flag in the command to run HuggingFace’s "run_squad.py" to turn the feature on in the HuggingFace API. For both models, we decided to use maximum sequence length of 384 and doc stride of 128.

We chose to train both BERT and ALBERT with the given hyperparameters for 2 and 3 epochs in order to get more BERT and ALBERT models for our ensembles. As shown in the Results section, we will refer to BERT with a learning rate of 3e-5 when trained with 3 epochs as BERT-A and when trained with 2 epochs as BERT-B. In turn, we will refer to ALBERT with a learning rate of 5e-5 when trained with 3 epochs as ALBERT-A and when trained with 2 epochs as ALBERT-B.

4.2 No Answer Question Augmentation with Easy Data Augmentation

Easy Data Augmentation is a state of the art algorithm for data augmentation for binary text classification. EDA makes use of four simple operations to generate augmented sentences given a basis sentence. The algorithm takes in one sentence from the training set and performs one of the following operations chosen at random: synonym replacement (SR), random swap (RS), random insertion (RI), and random deletion (RD) [4]. In text classification long sentences can absorb the noise caused by random swap (RS), random insertion (RI), and random deletion (RD) while maintaining their original class label. Unlike in text classification, however, in Question Answering any deletions, swaps, or insertions may change the underlying meaning and flow of the sentence which impedes comprehension. Thus, in order to augment the SQuAD 2.0 data without impacting the answer span, context, or any underlying meanings, we implemented our own version of EDA focusing on "synonym replacement (SR)", where we would choose 1 word in a question that is not a stop word and replace it with a synonym chosen at random.

Currently, "run_squad.py", HuggingFace’s provided SQuAD PCE transformer run-file, turns questions and answers in train.json into "SQuADExample" objects to be used in training using a "SQuAD-Processor" class which parses the input train file [5]. In order to augment the questions, using EDA synonym replacement, we wrote our own SQuADProcessor class "SQuADEDAProcessor" which intercepts the parsed questions from the json file, and passes them into a synonym replacement method we adapted from the EDA paper and its according github using nltk WordNet to produce synonyms [6] [4]. From there, we append the augmented question and the original question with the same answers to the list of examples the PCE model will train on – augmenting the dataset in place.

Initially, we chose to augment a random sample of 5% and 20% of our questions for augmentation but we noticed that our F1 and EM scores were low because there was an imbalance between the No Answer and Has Answer questions that were being augmented. For this reason, we chose to simply augment all the No Answer questions in the dataset, in order to balance it. As detailed in the results section, we saw an improvement in the performance of BERT’s F1 and EM score with No Answer Question augmentation.

4.3 Ensembling and Tie Breaking

One of the key goals of our project is to experiment with ensembling our different models with diverse weighting and voting strategies. Specifically, we decided to experiment with two different popular ensembling methods: Unweighted and Weighted Majority Voting.

In our Unweighted Majority Voting ensembling method, we essentially gave each model one vote as
to what they believe the correct answer for a question is, based on the answers in the predictions file and then from there selected the majority consensus answer to be written the ensemble’s outputted predictions file. We performed tie breaking by choosing the answer of the model with the highest F1 score 70% of the time and for the rest of the time randomly choosing an answer from one of the other models. For the Weighted Majority voting, we would assign a number of "votes" to each model based on their F1 score and then take the highest voted answer as correct, writing it to the ensemble’s outputted predictions files. This way the best model would have the greatest number of "voting shares" and models which were not as accurate would be able to sway the choice of the answer less. In the case of a tie for Weighted Majority Voting, we randomly selected an answer from one of the models.

5 Experiments

5.1 Data

We used the Stanford Question Answering Dataset, SQuAD 2.0. The dataset is split into training, test, and dev data, all of which are publicly available except for test. The training data contains about 129,941 examples, the dev set contains about 6078 examples, and the test set contains about 5915 examples. The dataset contains many (context, question, answer) triples, where context refers to a Wiki excerpt, the answer is a span of text from the context, and the question may or may not be answerable given the context. The data is stored in a series of json files and is preprocessed by running the given default project code in setup.py.

5.2 Evaluation method

For evaluation, we utilized both F1 and exact match (EM) scores. The F1 score, calculated as \[ \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \], considers precision and accuracy in making the evaluation. EM is a boolean metric that determines whether or not the produced output is an exact match to the correct output.

5.3 Experimental details

We used BERT trained with a learning rate of 3e-5 and 2 epochs (BERT-B in Table 1 in Results Section) and ALBERT trained with a learning rate of 5e-5 and 2 epochs (ALBERT-B in Table 1 in Results Section) as our baseline models. For all of our BERT and ALBERT models, we used the PCE versions from the Huggingface library [5]. Please refer to the appendix for the exact command we used to fine tune BERT and ALBERT, but to summarize, we used a learning rate of 3e-5 for all BERTs and 5e-5 for all ALBERTs and for all BERT/ALBERT models, a batch size of 8, max sequence length of 384, and docstride of 128 along with mixed precision (fp16) training through the apex library. We adjusted the number of epochs according to the experiment we were running. The key hyperparameters for each model are listed in the tables below. It took approximately 8 hours to train BERT and 7 hour to train ALBERT. For BERT with No Answer Augmentation, we simply ran BERT-B and ALBERT-B but used our "SQUADEDAProcessor" class instead of "run_squad.py"s' original "SQUADProcessor" which parses the train json file in order to intercept the parsed questions and run those marked as "No Answer" or specifically "is_impossible==True" through question augmentation with Easy Data Augmentation based synonym replacement as detailed in Section 4.2. As there are about 43,498 no answer questions, our no Answer Augmentation doubled this number to 86,996 no answer questions and thus increased the size of our train dataset from 129,941 to 173,439 question and answer pairs. We trained both BERT and ALBERT using No Answer Augmentation and the results can be found in Table 1 in the Results section below. It took about 10 hours to train BERT with No Answer Augmentation and 9 hours to train ALBERT with No Answer Augmentation. For our ensembles, we simply took different combinations of our models predictions files and passed them to our ensembling script for either weighted or unweighted majority voting as detailed in Section 4.3.

5.4 Results

On the test PCE leaderboard, our highest F1 score is 81.466 and our highest EM score is 77.887. On the dev PCE leaderboard, our highest F1 is 82.257 and our EM is 79.516. Overall, our results are what we expected for our models both augmented and non-augmented.
Looking at Table 1, we found that ALBERT-B was the best non augmented model, and that No Answer augmentation was able to improve the performance of BERT-B by 0.02 points for F1 and 0.068 for EM, but was unable to improve the performance of ALBERT-B or ALBERT-A. We hypothesize this is the case because ALBERT is a lighter version of BERT which uses factorized embedding parameterization and lighter feature extraction making it less sensitive to the changes and noise which No Answer augmentation brings to the dataset to improve the model. Regardless, the improvement of BERT-B with No Answer Augmentation (BERT-B-NoAnsAug) shows that Question Augmentation is a viable choice to strengthen the PCE BERT model. In Table 2, we showcase the different unweighted ensemble models we built finding that an ensemble of ALBERT-B, ALBERT-A, and ALBERT-B-NoAnsAug performing the best. In turn in Table 3, we showcase how weighting the models in the ensembles based on their performance can improve the F1 and EM scores. In Table 4, we show that by weighting the best models higher, giving them more votes in the majority voting process, we can hit new high scores for all our ensembles with our best ensemble achieving an F1 of 82.257 and EM of 79.516. Overall, these results indicate that our approach makes sense and is taking steps in the right direction.

Table 1: BERT NonAugmented and Augmented Models with Tuning (Dev Set Results)

<table>
<thead>
<tr>
<th>BERT Model</th>
<th>LR</th>
<th>Epochs</th>
<th>F1</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-A</td>
<td>3e-5</td>
<td>3</td>
<td>76.498</td>
<td>73.053</td>
</tr>
<tr>
<td>BERT-B</td>
<td>3e-5</td>
<td>2</td>
<td>76.762</td>
<td>73.659</td>
</tr>
<tr>
<td>BERT-B-NoAnsAug</td>
<td>3e-5</td>
<td>2</td>
<td>76.782</td>
<td>73.727</td>
</tr>
</tbody>
</table>

Table 2: ALBERT NonAugmented and Augmented Models with Tuning (Dev Set Results)

<table>
<thead>
<tr>
<th>ALBERT Model</th>
<th>LR</th>
<th>Epochs</th>
<th>F1</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALBERT-A</td>
<td>5e-5</td>
<td>3</td>
<td>81.219</td>
<td>77.854</td>
</tr>
<tr>
<td>ALBERT-B</td>
<td>5e-5</td>
<td>2</td>
<td>81.912</td>
<td>78.973</td>
</tr>
<tr>
<td>ALBERT-B-NoAnsAug</td>
<td>5e-5</td>
<td>2</td>
<td>80.069</td>
<td>77.393</td>
</tr>
</tbody>
</table>

Table 3: Best Unweighted Majority Voting Ensembles (Dev Set Results)

<table>
<thead>
<tr>
<th>Ensemble Model</th>
<th>F1</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALBERT-B + BERT-B-NoAnsAug + BERT-B</td>
<td>79.380</td>
<td>76.407</td>
</tr>
<tr>
<td>ALBERT-B + BERT-B-NoAnsAug + BERT-B + BERT-A</td>
<td>81.025</td>
<td>78.019</td>
</tr>
<tr>
<td>ALBERT-B + ALBERT-A + BERT-B-NoAnsAug + BERT-B</td>
<td>81.526</td>
<td>78.310</td>
</tr>
<tr>
<td>ALBERT-B + ALBERT-A + BERT-B-NoAnsAug + BERT-B + BERT-A</td>
<td>81.105</td>
<td>78.052</td>
</tr>
<tr>
<td>ALBERT-B + ALBERT-A + ALBERT-B-NoAnsAug</td>
<td>81.550</td>
<td>78.578</td>
</tr>
</tbody>
</table>

Table 4: Best Weighted Majority Voting Ensembles (Dev Set Results)

<table>
<thead>
<tr>
<th>Ensemble Model</th>
<th>Vote Distribution</th>
<th>F1</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALBERT-B + BERT-B-NoAnsAug + BERT-B</td>
<td>3, 2, 1</td>
<td>82.139</td>
<td>79.237</td>
</tr>
<tr>
<td>ALBERT-B + BERT-B-NoAnsAug + BERT-B + BiDaf</td>
<td>5, 4, 3, 1</td>
<td>79.09</td>
<td>76.900</td>
</tr>
<tr>
<td>ALBERT-B + BERT-B-NoAnsAug + BERT-B + BERT-A</td>
<td>12, 5, 4, 3</td>
<td>81.929</td>
<td>78.990</td>
</tr>
<tr>
<td>ALBERT-B + ALBERT-A + BERT-B-NoAnsAug + BERT-B</td>
<td>11, 5, 4, 3</td>
<td>82.204</td>
<td>79.352</td>
</tr>
<tr>
<td>ALBERT-B + ALBERT-A + BERT-B-NoAnsAug + BERT-B + BERT-A</td>
<td>5, 4, 3, 2, 1</td>
<td>82.231</td>
<td>79.467</td>
</tr>
<tr>
<td>ALBERT-B + ALBERT-A + ALBERT-B-NoAnsAug</td>
<td>4, 4, 4, 2</td>
<td>82.257</td>
<td>79.516</td>
</tr>
</tbody>
</table>

6 Analysis

In this section, we will go through the four main tables in the results section and provide insight into the reported outputs. First, we look at Table 1 which compares the results of different BERT non-augmented and augmented models with tuning for the dev set. Looking at the BERT models, we
see that BERT-B which is BERT trained with a learning rate of 3e-5 using 2 train epochs outperforms BERT-A which is BERT trained with the same learning rate for 3 epochs. This could be because increasing the number of epochs causes the model to fit more to the train set which leads it to perform slightly worse on the dev set. In turn, from the table, we see that we can improve upon BERT-B by using No Answer Question Data Augmentation, and this is the best ranked model we tested for BERT. This makes sense as augmenting the no answer questions balances the dataset and adds more diversity to the questions which improves BERT’s understanding of the context. It also allows BERT to see more frequently words that it may have not seen as often do to the synonym replacement logic.

Table 2 compare the results of different ALBERT non-augmented and augmented models with tuning for the dev set. Similar to BERT, ALBERT-B which is ALBERT trained for 2 epochs with a learning rate of 5e-5 performs better than ALBERT-A which is trained with 3 epochs with a higher F1 and EM score. As with BERT, we suspect this is because the 3 epoch ALBERT slightly overfitted to the training set leading it to do worse on the dev set than 2 epoch. We also ran ALBERT-B, the best performing ALBERT model, with No Answer Augmentation as shown with ALBERT-B-NoAnsAug. We noticed that although we augmented the no answer questions the dataset the performance of ALBERT did not improve. We hypothesize this is because although augmentation added more questions with some slight diversity due to synonym replacement ALBERT was not able as sensitive to the changes due to its factorized embedding parameterization and lighter feature extraction which is used in order to make the model smaller. Thus, because ALBERT was not able to detect the changes in the questions from synonym replacement due to its lighter feature extraction, ALBERT-B-NoAnsAug overfitted to the train set, interpreting the augmented and non-augmented questions as the same, and performing worse in the end on the dev set.

In Table 3, we showcase our different ensembles which we created using Unweighted Majority Voting, essentially giving each model in the ensemble 1 vote as to what the correct answer was for a given question and then taking the majority consensus answer to be written to the ensembles output file. Looking at the ensembles, we were able to achieve our best results using an ensemble of just ALBERT's, specifically ALBERT-B, ALBERT-A, and ALBERT-B-NoAnsAug with an F1 score of 81.550 and EM of 78.587. This makes sense as our ALBERT models received the highest scores out of all our models and with good ensembling through majority vote we should have been able to further tune their results to improve predictions. The excellent performance of this ensemble shows that ensembling with your highest performing models may be better than ensembling all your models for an unweighted vote. This makes sense as including other lower performing models without any weighting in the vote may lead to an incorrect majority vote for the answer of a question. This can be partially mitigated by introducing weighted voting as we see in the next table, Table 4, but an overarching theme we found in our results was that when using voting based ensembling quality of model predictions is better than quantity of models in an ensemble.

After experimenting with unweighted voting, we decided to add weighting in order to see if they were able improve the performance of the ensembles. We found that this was generally true with the weighted majority voting versions of all our models outperforming the Unweighted Majority Voting versions of our ensembles as shown by Table 4. This makes sense as we were able to weigh or give more votes to our better performing models (e.g. ALBERT based) thus preventing the consensus from being skewed toward a majority vote for an incorrect answer as might have happened if we performed unweighted voting with our lower performance models. Similarly, we achieved our best performance with a weighted voting ensemble of ALBERT-B, ALBERT-A, and ALBERT-B-NoAnsAug in which we gave ALBERT-A and ALBERT-B 4 votes each since they were our best performing models of the three and ALBERT-NoAnsAug 2 votes since it performed slightly worse. We were also able to achieve a significant improvement with our ensemble of ALBERT-B, ALBERT-A, BERT-B-NoAnsAug, BERT-B, and BERT-A with weighted voting over unweighted voting. With unweighted majority voting each model received one vote and this ensemble achieved an F1 score of 81.550 and EM of 78.578 on the dev set, upon adding weighting, giving our best model ALBERT-B 5 votes, second best model ALBERT-a 4 votes, BERT-NoAnsAug 3 votes, BERT-B 2 votes, and BERT-A 1 vote, we show a huge jump in performance to an F1 score of 82.231 and EM of 79.467. This performance shows that giving your best models more weight in majority vote ensemble is a way to improve the accuracy of your ensembles.
7 Conclusion

Overall, our project showed us that augmenting the no answer questions in the SQuAD dataset was able to successfully improve the performance of BERT and utilizing BERT/ALBERT trained on augmented data in ensembles with weighted voting strategies is able to further boost performance. Currently, we are 11th place on the Dev PCE Leaderboard with our best ensemble and 8th on the TEST PCE Leaderboard. Some limitations of our work were memory and computational limits which prevented us from training multiple augmented models with different hyperparameters in order to see if we could further improve performance of augmented model with the correct tuning and add more tuned models to our ensembles. For next steps for our project, we would be interested in training more ALBERTs and BERTs with different hyperparameters and ensembling them to see if we can further improve the performance of our models. Overall, our project really helped open our eyes to the power of augmentation and ensembling in improving the performance of our models on SQuAD 2.0 dataset.

References


A Appendix

Our command to fine-tune BERT was:

```
python run_squad.py
    --model_type bert
    --model_name_or_path bert-base-uncased
    --do_train
    --do_eval
    --train_file {path/to/}train-v2.0.json
    --predict_file {path/to/}dev-v2.0.json
    --per_gpu_train_batch_size 8
    --do_lower_case
    --learning_rate 3e-5
    --num_train_epochs 2.0
    --max_seq_length 384
    --doc_stride 128
    --fp16 8
    --version_2_with_negative
```
Our command to fine-tune ALBERT was:

```bash
python run_squad.py
   --model_type albert
   --model_name_or_path albert-base-v2
   --do_train
   --do_eval
   --train_file {path/to/}train-v2.0.json
   --predict_file {path/to/}dev-v2.0.json
   --per_gpu_train_batch_size 8
   --do_lower_case
   --learning_rate 3e-5
   --num_train_epochs 2.0
   --max_seq_length 384
   --fp16 8
   --doc_stride 128
   --version_2_with_negative
   --save_steps 1000
   --logging_steps 1000
   --output_dir {path/to/output_dir}
```