Stratified Bagging and Boosting of ALBERT Models for Question Answering

Fu Jiaxiang jfu003@e.ntu.edu.sg

Abstract

In this paper we study the behavior of bagging and boosting of deep learning NLP models (specifically ALBERT V2) when applied onto the SQuAD 2.0 dataset for extractive question answering. We show that, with carefully designed stratified sampling, bagging and boosting can produce better results than standard techniques. Our final model based on bagging results in 1.05-point improvement of F1 score over baseline.

We also explored the effect of regularization and neural architecture variations on model performance. We did not find significant performance improvement with the use of L1 regularization on ALBERT models similar to those found by (Qin et al., 2019) with BERT models. This may in part be explained by the fact that ALBERT by design has reduced redundancy and is less sensitive to regularization due to its smaller size. Furthermore, we showed with experiments that adding a fully connected layer to ALBERT improves performance, and this improvement can be amplified with more fine-tuning.

1 Introduction

Since the discovery of word2vec in 2013 (Mikolov et al., 2013), deep learning has become increasingly popular in applications to natural language processing tasks. In 2018, the advent of Bidirectional Encoder Representations from Transformers, a.k.a BERT (Devlin et al., 2018), asserted the dominance of large scale unsupervised pre-training in the field of NLP to date.

As the algorithms and techniques in deep learning NLP advances forward, the pool of tasks defined by researchers to train and measure these algorithms and techniques are getting added and updated year by year. With the new tasks came curated datasets. These datasets help deep learning researchers develop and train their models and provide widely recognized standards for evaluating and comparing the performance of new models and techniques. The SQuAD 2.0 (Rajpurkar et al., 2016) is one such dataset designed for extractive question answering. Unlike its older version, SQuAD 2.0 includes unanswerable questions in both training and testing examples, which makes it a much more challenging problem. Capable systems that perform well on SQuAD 1.1 often performs much worse on SQuAD 2.0 (Rajpurkar et al., 2016).

This study bases itself on the work of pre-trained contextual embeddings and aims to study the effectiveness of various machine learning techniques such as adding neural architecture, regularization and ensemble learning. In doing so we aim to develop a model that is able to produce better results on SQuAD 2.0 dataset than our baseline model.

Preprint. Under review.

2 Related Work

2.1 BERT & ALBERT

2.1.1 BERT

BERT is a powerful pre-training technique developed by Google for the field of NLP. Upon introduction, it achieved new state-of-the-art performance for eleven NLP tasks (Devlin et al., 2018). Similar to OpenAI GPT (Radford et al., 2018) and ELMo (Peters et al., 2018), it uses unsupervised pre-training to generate contextual embeddings for text tokens. However, unlike the other two, BERT uses Masked Language Modelling task to train its encoders, which allows them to capture information from both left and right contexts.

2.1.2 ALBERT

ALBERT (Lan et al., 2019) is a variant of BERT with two key design differences. The first is the factorization of the embedding parametrization, which allowed the model to reduce the sizes of input-level embeddings without affecting the hidden-layer embeddings. The second is parameter sharing across multiple attention-feedforward block layers. The two techniques combined achieve a 80 - 90% reduction in overall model size, while only slightly affect model performance. However, the size reduction allows the model to scale up further. This is shown by ALBERT-xxlarge, which obtained massive performance improvement over BERT-large in multiple benchmarks including SQuAD 2.0 while maintaining a 30% size reduction.

It is worth pointing out that ALBERT uses SentencePiece (Kudo and Richardson, 2018) for text tokenization, which is different from BERT with its own WordPiece tokenizer. The SentencePiece finds tokens directly from raw sentences hence allows models to be purely end-to-end and language independent.

Like BERT, ALBERT uses a two-step framework for specific downstream tasks such as question answering: *pre-training* and *fine-tuning*. This study only concerns itself with the second step, i.e. *fine-tuning*, and uses pre-trained models from the official sources.

2.2 Bagging & Boosting

Ensemble learning has been an effective machine learning technique since the 1990s. Particularly, it has been shown that Bagging (Breiman, 1996) and boosting (Schapire, 1990) are able to combine multiple *"weak learners"* and transform them into a *strong learner*. Building on Schapire (1990)'s work, Freund (2001) proposed an adaptive version of the boost by majority algorithm called AdaBoost, in which the authors provided specific formulae to calculate adjusted sample weights after each boosting iteration.

However, the ALBERT models we are using to build the ensemble are not *weak learners*. In fact, they are quite the opposite. Therefore, we needed some adjustments in our approaches. We explain this further in Sections 3.4 and 3.5.

Another issue with the original AdaBoost algorithm is that it only deals with binary classification. In the case of multi-class classification, AdaBoost in its original form requires each of the *weak learners* to correctly predict class label at least 50% of the time. This is a much stricter requirement than the original intention of AdaBoost, which only requires the *weaker learners* to be slightly better than random guess (e.g. 34% accuracy for a 3-class classification problem). Zhu et al. (2009) solved this problem in their SAMME boosting algorithm with some modifications to how AdaBoost adjusts the sample weights after each boosting iteration.

The problem this study is trying to tackle is close in nature to the multi-class classification problems that Zhu et al. (2009) solved. As we break up sentences into tokens, and our models try to predict the start and end tokens, we are essentially doing multi-class classification. Therefore, for boosting, we derived our approach from the SAMME algorithm. We will discuss how we apply the SAMME algorithm to solve our particular problem in details in Section 3.5.

3 Approach

3.1 Baseline

Since all the models and techniques explored in this paper are based on the ALBERT-base V2 model pre-trained by Google, it is natural we use it as the baseline. Google published their result using this model on SQuAD 2.0, and it managed to score 82.1(%) for F1 and 79.3(%) for exact match.

While used as a baseline in this study, ALBERT-base V2 is by no means an easy-to-beat benchmark. The model itself is extremely powerful. Furthre, Google has optimized the fine-tuning for the task of SQuAD 2.0. Therefore, there is little room for improvement by simply tweaking the training parameters.

3.2 Regularization

Neural networks are known to be prone to overfitting. To harness the power of neural networks while avoiding overfitting, regularization techniques are commonly used in modern deep learning systems. Qin et al. (2019) showed that L1 regularization improves BERT models significantly for the task of question answering on SQuAD 2.0. In this study, we explored the use of L1 regularization for ALBERT models.

The L1 regularization adds an additional term to the training objective function, as illustrated below.

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}_i, y_i) + \frac{\lambda}{n}||\theta||_1,$$

where

- J is the objective function to minimize
- θ represents all model parameters
- n is the number of training instances
- L is the loss function
- $\hat{y}_i$ is the model predictions
- y_i is the ground truth

3.3 Adding Fully Connected Layer

In general, more hidden layers in a neural network can potentially lead to better performance, provided the model does not overfit. In the baseline ALBERT model, most of the complex neural architecture are generic and only a single linear layer was used in the end for making problem specific predictions. In this study, we experiment adding a fully connected layer to the model, and fine-tune the model slightly longer.

3.4 Bagging

In the original bagging approach (Breiman, 1996), each contributing model is a *weak learner*. In order to avoid overfitting, we used an iterative approach in our application of bagging. We start with only 2 models in the ensemble, and gradually increase the number of models until more models no longer leads to better validation performance.

Furthermore, we experiment different mixes of training samples. Apart from randomly bootstrapped subsets, we also experiment the inclusion and exclusion of their complementary subsets. As we are sampling 50% of training examples every time, these complementary sets are of the same sizes and can be conveniently added to our bootstrapped sample pools.

3.5 Boosting

We use the SAMME algorithm adapted to our problem. Similar to bagging, we sample 50% of the training examples each time for boosting. At each iteration, we draw a sample from the training set

with a predefined probability weights, and train a new classifier with the drawn subset. The weights are then adjusted based on the results of trained classifier: those classified incorrectly would get a higher chance of being sampled in the next iteration.

The precise algorithm is depicted below. Note that we simplified the mathematical formulae presented by Zhu et al. (2009). While remaining mathematically equivalent, our version below is simpler in form and in our opinion better for practical implementation.

SAMME Algorithm

1. Initialize the observation weights $w_i = 1/n, i = 1, 2, ..., n$.
2. For $m = 1$ to M:
   (a) Draw a subset sample from all training data using weights w_i.
   (b) Train a classifier $T^{(m)}(x)$ with the sample
   (c) Compute error rate
   $$err^{(m)} = 1 - \sum_{i=1}^{n} w_i \delta(\hat{y}_i, y_i)$$
   (d) Compute adjustment factor
   $$adj^{(m)} = \frac{1 - err^{(m)}}{err^{(m)}} \cdot (K - 1)$$
   (e) Set
   $$w_i \leftarrow \begin{cases} w_i, & \text{if } \hat{y}_i = y_i, \\ w_i \cdot adj^{(m)}, & \text{if } \hat{y}_i \neq y_i. \end{cases}$$
   (f) Normalize w_i
3. Output
   $$logits = \sum_{m=1}^{M} log(adj^{(m)}) \cdot T^{(m)}(x)$$

where

- n is the number of training examples
- M is the total number of boosting iterations
- $\hat{y}_i$ is the model predictions
- y_i is the ground truth
- $\delta(a, b)$ is the Kronecker delta function which equals 1 if $a = b$ and 0 if $a \neq b$.
- K is the number of classes, which equal to the max sequence length

We made a few adaptations of this algorithm to fit our use case. Firstly, we do not pre-determine the number of iteration M. Instead, we evaluate the ensemble after each iteration and applies early stop when we detect overfitting or when performance increment is too insignificant. Secondly, for each example the model outputs both the start and end tokens. Here we only consider a prediction correct if both the start and end pointers are correct.

3.6 Stratified Sampling

SQuAD 2.0 includes about half unanswerable questions in the training set. Our model uses a threshold to determine if a question is answerable. Au such, it is imperative that we avoid biased sampling. In all of our bagging and boosting sub-samples, we select the same percentage from the answerable and unanswerable examples separately. Doing so makes sure all sub-samples have the same distribution as the original training data.

4 Experiments & Analysis

4.1 Data

4.1.1 SQuAD2.0

All experiments in this study are conducted on The Stanford Question Answering Dataset Version 2.0 (Rajpurkar et al., 2016). As of now, they can be accessed in the official SQuAD website `https://rajpurkar.github.io/SQuAD-explorer/`.

4.1.2 Data Preprocessing

Tokenization

As per required by ALBERT models, each question and context are split into tokens before they can be encoded into vector embeddings. For tokenization we used the *AlbertTokenizer* implementation in the *transformers* library from the *Hugging Face* organization, which in turn uses the *sentencepiece* library developed by *Google*.

Segmentation & Padding

The tokens of corresponding questions and contexts are concatenated to form a single sequence. While doing so we assign each token a *segment id* or *token_type_id* in technical terms, which equals 0 for tokens generated from the questions and 1 for those of contexts. These *ids* are recorded in accompanying vectors and are subsequently fed into the ALBERT models.

As ALBERT models accept vectors of fixed length, we pad short sequences to a fixed length. In this case this fixed length is 384 tokens. We construct another vector called *attention_mask* to indicate padded tokens.

Doc Stride

For those sequences that are too long, we separate them into multiple sequences using a sliding window approach. In each of these *synthetic sequences*, the question tokens are repeated, with the remaining covering a part of the context tokens, and a certain number of tokens away from each other. This number is called *DOC_STRIDE* and is set to 128 in our case.

4.2 Evaluation Metrics

The main evaluation metric we use in this study is the F1 score, as defined below.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

As a secondary metric we also use the exact match (EM) to help us understand our model performance more holistically.

$$EM = \begin{cases} 1, & \text{if } prediction = answer, \\ 0, & \text{if } prediction \neq answer. \end{cases}$$

As suggested by SQuAD 2.0 team, during testing phase we compare our model predictions with all the ground truths provided for the same testing instance and take the max as our final score.

4.3 Results & Analysis

4.3.1 Baseline

We built our baseline by fine-tuning the pre-trained ALBERT-base V2 model using the training parameters published by Google. As shown below, the performance of our implementation is inline with that published by Google.

Table 1: Baseline Model (Google's Result vs. Our Result)

Model	F1	EM
Google's ALBERT-base V2	82.1	79.3
Our ALBERT-base V2	82.09	79.30

4.3.2 L1 Regularization

We evaluated the addition of L1 regularization. See below for the results.

Table 2: Model Performance with L1 Regularization

λ	F1	EM
0	82.09	79.30
1e-03	70.33	66.37
1e-04	81.38	78.35
1e-05	81.84	78.89
1e-06	81.63	78.46
1e-07	82.32	79.15
5e-08	82.38	79.27
1e-08	81.90	78.81

As can be seen above, the optimal value for λ is about 5e-08. However, even at the optimal point, regularization hardly improved the F1 score over baseline ($\lambda = 0$), and resulted in a worse EM score.

Qin et al. (2019) showed that L1 Regularization significantly helped the performance of BERT models. However, this effect is minimal for the case of ALBERT model, as seen above. This can be partially explained by the fact that ALBERT employed the cross-layer parameter sharing technique, which is a form of regularization by itself as it effectively binds parameters of different layers together, reducing the degree of freedom by a large extent. Further, from the model size point of view, the ALBERT-base V2 model we used is much smaller in size than the BERT-base model used in their study, with only 11% the number of parameters. Given all else the same, ALBERT-base V2 should be less prone to overfitting thus less sensitive to L1 regularization.

4.3.3 Adding Dense Layer

We added a fully connected linear layer to ALBERT model before the final output layer. The size of this hidden layer is 384, equal to the max sequence length thus equal to the final output size.

As the additional layer increases the network complexity, we decided to experiment fine-tuning it longer. The results are shown below.

Table 3: Model Performance with Added Dense Layer

Model	epochs	F1	EM
baseline (optimized)	2	82.09	79.30
baseline (more training)	3	81.64	78.25
baseline + dense layer	2	82.15	79.45
baseline + dense layer	3	82.70	79.57

As shown above, adding a fully connected layer improves the model performance, and training for one more epoch increased performance further. By contrast, training the baseline model for one more epoch led to a drop in performance due to overfitting.

All subsequent experiments are done with the added dense layer.

4.3.4 Bagging

Due to limitations on computing resources and diminishing marginal utility, we stopped at training 6 models on 6 sub-samples. Out of the 6 sub-samples, 4 are created by randomly sample 50% of

training examples from answerable and unanswerable respectively - stratified sampling. These 4 sub-samples and the models trained on them are labelled A, B, C and D. Two other sub-samples, A' and B' are complementary set for set A and B. To illustrate, A and A' are mutually exclusive and together they form the full training set. Due to how they are sampled, set A' and B' also have the same distribution of answerable vs. unanswerable questions as set A, B, C and D.

We evaluated different combinations of trained models. See below for the results.

Table 4: Model Performance with Bagging

Experiment Seq No.	Models	F1	EM
1	A & B	82.08	79.17
2	A & A'	82.25	79.32
3	B & B'	82.82	80.03
4	A, B, C & D	82.80	80.09
5	A, A', C & D	82.92	80.27
6	A, A', B & B'	83.14	80.49
7	A, A', B, B', C & D	83.05	80.40

Three conclusions can be draw from the results. Firstly, ensembling 4 models generally leads to better performance than 2 models. This can be seen by comparing Experiments 4 - 6 with Experiments 1 - 3. Secondly, having more than 4 models leads to overfitting and deteriorates performance. This can be seen by comparing Experiments 7 with Experiment 6. Thirdly, given the same number of models, having complementary sets produces better results over purely random sub-samples. This can be seen by comparing Experiments 2 - 3 with Experiment 1, and by comparing Experiments 5 - 6 with Experiment 4.

Among all available combinations, (A, A', B & B') are trained on two pairs of complementary sets and when ensembled they give the best results - 1.05 point improvement in F1 over baseline.

4.3.5 Boosting

For boosting, we start with only 1 model. We then go through the boosting iteration described in Section 3.5, adding one model at a time. The results can be seen below.

Table 5: Model Performance with Boosting

No. of Boosted Models	F1	EM
1	80.77	77.77
2	82.12	79.14
3	82.25	79.27
4	82.62	79.66

The result shows that adding more models leads to better performance up to 4 models. However, due to performance shortfall compared to bagging and diminishing marginal utility, we stopped at 4 models.

5 Conclusion & Discussion

In this paper we have shown that adding fully connected layer to ALBERT improves its performance. This performance can be further amplified with proper tuning on training parameters. We also showed that with stratified sampling and the inclusion of complementary sets for training, bagging and boosting improves performance of ALBERT-base V2 model despite it being a *strong learner*.

However, there are a few questions remain to be answered. Firstly, does the performance improvement introduced by bagging and boosting applicable to large sizes of ALBERT models such as ALBERT-xxlarge, given they are even stronger learners? Secondly, what happens if we keep adding more models in boosting? In the course of this study we argued it's unlikely to surpass the performance of our best bagging model, but the exact answer remains to be ascertained. These 2 questions can be 2 areas of potential future work.

References

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24:123–140.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Freund, Y. (2001). An adaptive version of the boost by majority algorithm. *Machine Learning*, 43:293–318.

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, abs/1802.05365.

Qin, Z., Mao, W., and Zhu, Z. (2019). Diverse ensembling with bert and its variations for question answering on squad 2.0.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:197–227.

Zhu, J., Zou, H., Rosset, S., and Hastie, T. (2009). Multi-class adaboost. *Statistics and Its Interface*, 2(3):349–360.