SQuAD 2.0 Based on ALBERT and Ensemble

Stanford CS224N Default Project

Yanpei Tian
Department of Electrical Engineering
Stanford University
yanpeit@stanford.edu

Abstract

Recent years, Pre-trained Contextual Embedding models bring revolutionary changes to many NLP tasks. Fine-tuning on a Pre-trained model achieves state-of-the-art performance in lots of aspects in NLP. In this project, I want to explore the advantage of pre-training + fine-tuning paradigm to solve the SQuAD 2.0 challenge. Particularly, I've shown fine-tuning on the ALBERT model achieves higher EM and F1 scores, compared to a BiDAF model with character-level embedding. Lastly, I further improves the performance of my model by ensembling a SQuAD solver (a single model for solving SQuAD) and a classifier that gives prediction of answer/no answer.

1 Introduction

Stanford Question Answering Dataset (SQuAD)[1] is a reading comprehension dataset, consisting of questions on a set of Wikipedia articles, where the answer to every question is a segment of text from the corresponding reading passage. A new feature introduced in SQuAD 2.0 is unanswerable questions, where a question cannot be answered given the context. Such a reading comprehension task gives us a gauge in how well a Natural Language Processing (NLP) system can perform reading comprehension and information retrieval.

Currently, there are two approaches to solve a question answering task like SQuAD: Non-Pre-trained Contextual Embedding (Non-PCE) and Pre-trained Contextual Embedding (PCE). The Non-PCE approach builds a neural network architecture from scratch specifically for the question answering task and train the model from scratch. Usually, this approach includes some sort of attention[2] mechanism that let the question attend to the context and vice versa. On the other side, a PCE approach usually involves the pre-training + fine-tuning paradigm. We start from a pre-trained model with predefined architecture and pre-trained parameters and slightly modify its architecture to fit our need. Then, we train the overall model on our own data to produce a final model. Usually, we use a general-purposed "language understanding" model that has been trained on a large text corpus as the pre-trained model. The approach allows us to utilize the training of the pre-trained model for language comprehension. Generally, fine-tuning on a pre-trained model is more efficient and can lead to better results.

In this project, I tried both approaches before try to ensemble a final model. For the Non-PCE part, I build a Bidirectional Attention Flow model with character level embedding based on Seo et al., 2016[3]. For the PCE part, I fine tune the ALBERT[4] on the SQuAD 2.0 dataset. Lastly, I combine a usual SQuAD solver with a classifier that predicts answer/no answer for a question as the final model. A classifier specifically trained for predicting answer/no answer can give better predictions compared to a stand-alone SQuAD 2.0 solver. As a result, the ensemble increase EM by 0.7 and F1 by 0.6 on my best model.

2 Related Work

Chen[5] introduced the simple yet powerful Stanford Attentive Reader. The model first encode both the context and question into two lists of vector representations. Then, utilizing the concept of attention, two separate classifiers were trained to predict the start and ending position for the answer span. BiDAF model also makes use of the concept of attention with "bidirectional" stands for both question attending to context and context attending to question. In the project, I use the BiDAF model with character-level embedding as the baseline.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[6] was introduced in 2018 and it is one of the most influential work in NLP recent years. BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with minimal architectural change and achieve state-of-the-art performance. ALBERT, introduced in 2019, is a variant of the original BERT model. It bring the following two improvements to address the issue of model size and training time.

- Matrix factorization for word embeddings;
- Parameter sharing between layers in the neural network;

In this project, I perform fine-tuning based on ALBERT.

However, the SQuAD 2.0 essentially involves tow parts:

1. Predictions of answerable/unanswerable;
2. Find the correct span in the context;

Regarding the first point, the QNLI task in the GLUE Benhmark[7] provide a good reference, converting SQuAD 2.0 into sentence pair classification by forming a pair between each question and each sentence in the corresponding context. The training goal is to correctly predict whether a question is answerable given a context.

3 Approach

3.1 Character level embedding for BiDAF

Note: Please refer to the default project handout for the BiDAF model with no character-level embedding.

Following Kim[8], I add an additional character level embedding to the input layer of the BiDAF model.The extra character-level embedding can help to solve the Out-of-Vocabulary (OOV) problem and increase the model's performance.

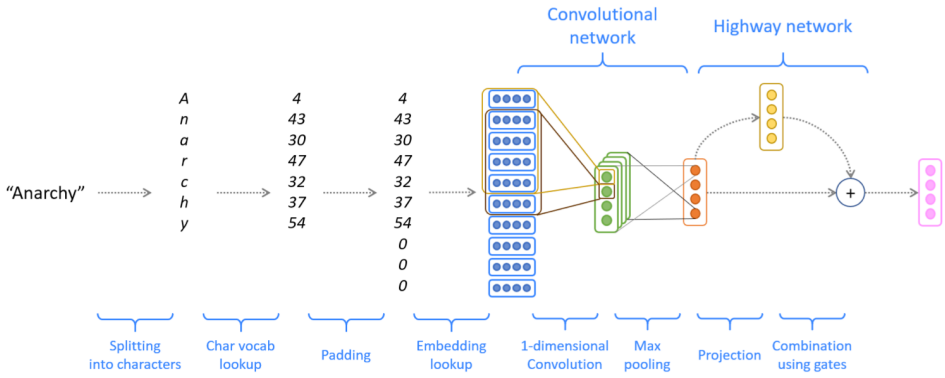


Figure 1: Character-based convolutional encoder

The process of generating a character-level embedding from a word is summarized below:

1. Convert word to character indices: by table look-up, we can represent a word as a vector of integers.
$$x = [c_1, c_2, ..., c_l] \in \mathbb{Z}^l$$
2. Embedding lookup: for each of these characters, we lookup a dense character embedding.
$$x_{emb} = CharEmbedding(x) \in \mathbb{R}^{l \times e_{char}}$$
3. Convolutional network: to combine these character embeddings, a 1-dimensional convolution is used, followed by ReLu activation and MaxPool.
$$x_{conv} = MaxPool(ReLu(Conv1D(x_{emb}^T))) \in \mathbb{R}^{e_{word}}$$
e_{word} is the size of a word embedding.
4. Highway layer and dropout.
$$x_{proj} = ReLu(W_{proj} x_{conv} + b_{proj}) \in \mathbb{R}^{e_{word}}$$
$$x_{gate} = \sigma(W_{gate} x_{conv} + b_{gate}) \in \mathbb{R}^{e_{word}}$$
$$x_{highway} = x_{gate} \circ x_{proj} + (1 - x_{gate}) \circ x_{conv} \in \mathbb{R}^{e_{word}}$$
$$x_{out} = Dropout(x_{highway}) \in \mathbb{R}^{e_{word}}$$

3.2 BiDAF based classifier

I modified the output layer of the BiDAF model to fit a classification job for predicting answer/no answer for a SQuAD example. While the original model's outputs are two probability distributions predicting the answer span, the output of the classifier simply keep the concatenation of attention layer output and modeling layer output corresponding to position 0 (no answer). A logistic regression layer is used to generate the answer/no answer prediction.

3.3 ALBERT fine-tune and classification

3.3.1 Pre-train

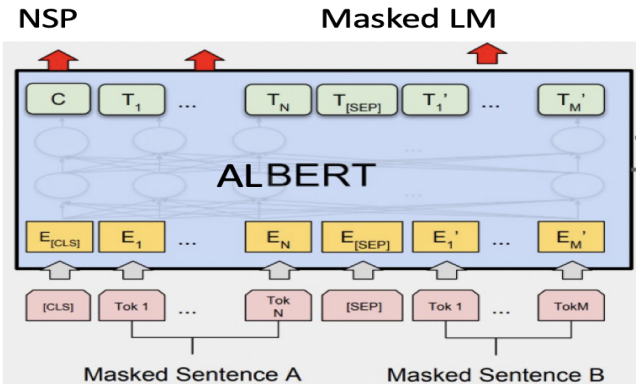


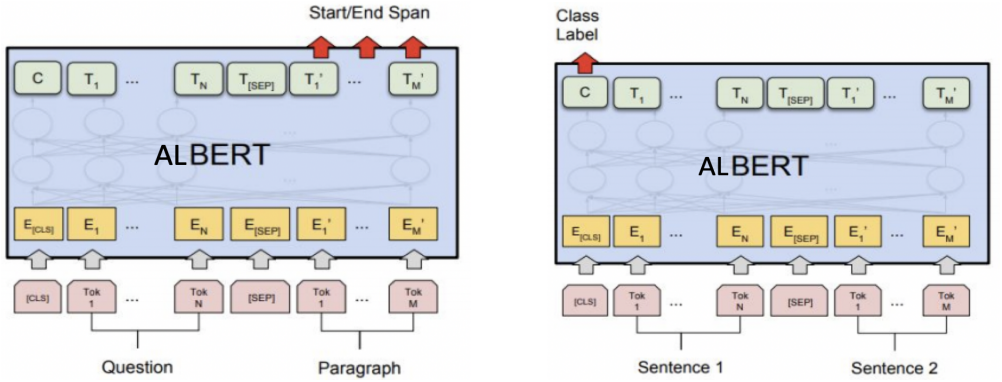
Figure 2: ALBERT pre-training

The pre-training of ALBERT involves two tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

3.3.2 Fine-tune and classification

The fine-tuning is performed on a ALBERT Model with a span classification head on top for extractive question-answering task. The output layer is a linear mapping on top of the hidden-states to compute start logits and end logits.

`https://huggingface.co/transformers/model_doc/albert.html#albertforquestionanswering`[9]



(a) Fine-tune (b) Classification

Figure 3: Fine-tuning and Classifier based on ALBERT

The classifier is built from ALBERT Model with a sequence classification head on top. The output layer perform linear mapping and logistic regression after pooling the hidden-states.

`https://huggingface.co/transformers/model_doc/albert.html#albertforsequenceclassification`

3.4 Ensemble

I performed three ensembles: BiDAF, ALBERT-base, and ALBERT-large. In all of the ensembles, I use stacking to combine the span prediction from the SQuAD solver and answer/no answer classification from the classifier to generate the final output.

Algorithm 1: Ensemble algorithm

Result: Final output for SQuAD 2.0
1. Generate SQuAD spans for every (context, question) pair as if every question has an answer;
2. Generate answer/no answer predictions use the classifier;
3. Combine the outputs from step 1 and step 2 to generate the final output;

4 Experiments

4.1 Data

I use the provided SQuAD 2.0 dataset for the default project customized from the official dataset of SQuAD 2.0. A summary of the train, dev, test set is as follows:

- **train** (129,941 examples): All taken from the official SQuAD 2.0 training set.
- **dev** (6078 examples): Roughly half of the official dev set, randomly selected.
- **test** (5915 examples): The remaining examples from the official dev set, plus hand-labeled examples.

The dataset contains many (context, question, answer) triples. Each context is taken from Wikipedia and the question is to be answered based on the context. The answer is a span from the context (for unanswerable question, the span is empty).

4.2 Evaluation method

Performance is measured via two metrics: **Exact Match (EM)** and **F1**.

- **EM** is a binary measure of whether the model's output matches exactly with the truth. This metric gives no "partial credit".

- **F1** is the harmonic mean of precision and recall. Precision is the percentage of correct predictions among all predictions. Recall is the recovery rate of the truth in the prediction. This metric is less strict than **EM**.

4.3 Experimental details

I trained the BiDAF model with character-level embedding and BiDAF classifier with the given configuration in the starter code. Each training process takes about 12 hours to complete on Azure NV6.

For ALBERT based training, I did four experiments shown below. The training configuration deviates from the default Huggingface script only in learning rate. To expedite the training process, I ran all the ALBERT related experiments on a Tesla V100 GPU.

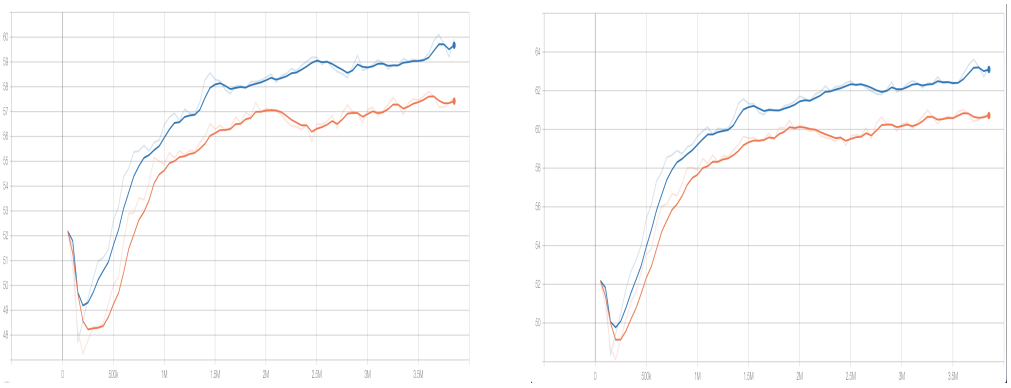
Model Configurations	learning rate	epochs	training time (hours, on Tesla V100)
ALBERT-base-SQuAD	3e-5	2	2
ALBERT-large-SQuAD	3e-5	2	6
ALBERT-base-Classifier	1e-5	3	3
ALBERT-large-Classifier	5e-6	3	9

Table 1: Model Configurations for ALBERT Fine-tuning

4.4 Results

Final submission on the PCE test leaderboard: EM=79.746, F1=82.533.

4.4.1 Effectiveness of character-level embedding on BiDAF



(a) EM (b) F1

Figure 4: Performance measurements for BiDAF model with character-level embedding (blue line) and without character-level embedding (orange line)

According to the plot, the additional character-level embedding boosts performance of the BiDAF by 2, measured by EM and 2, measured by F1.

4.4.2 Single model performance on dev set

- **BiDAF** EM=60.108, F1=63.626;
- **ALBERT-base** EM=77.443, F1=79.604;
- **ALBERT-large** EM=**78.578**, F1=**81.359**;

Compared to PCE-based methods, the BiDAF baseline with character-level embedding performs considerably worse. This is expected as the BiDAF's model size is smaller than ALBERT. And it is trained on much less data as ALBERT is trained on billions of words during pre-training.

4.4.3 Classifier performance (measured in accuracy) on dev set

- **BiDAF** Acc=61;
- **ALBERT-base** Acc=82;
- **ALBERT-large** Acc=**85**;

These numbers agree with my expectation for the following reasons:

1. Compared to the SQuAD solver, the classifier trained on the same architecture (BiDAF, ALBERT-base, ALBERT-large) all get slightly better results as classification is a easier task compared to finding the correct answer span;
2. The performance on different models increases as the model complexity increases;

4.4.4 Ensemble on dev set

- **No classifier, predicting answer for all questions** EM=39.6, F1=43.0;
- **BiDAF classifier** EM=53.5, F1=66.5;
- **ALBERT-base classifier** EM=78.5, F1=80.6;
- **ALBERT-large classifier** EM=**79.3**, F1=**82.0**;

First of all, we can notice that simply predicting an answer for every question leads to a huge drop in performance as about half of the (context, question) pairs are unanswerable. Secondly, ensembling generally leads to a slight performance increase, compared to single-models. The only case where ensemble leads to a performance drop is the EM metric for BiDAF.

5 Analysis

All these analysis is performed with respect to the ALBERT-large configuration.

5.1 SQuAD solver analysis

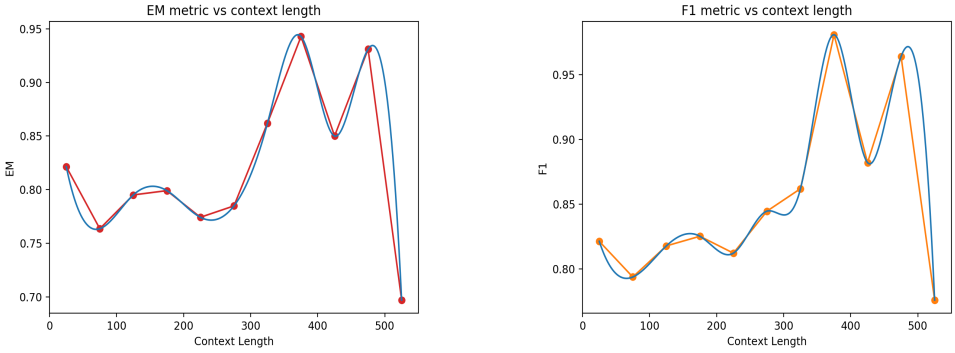


Figure 5: Performance metrics for the SQuAD solver vs length of the context. The subplot on the left is Emact Match (EM) vs context length and the subplot on the right is F1 score vs context length.

To measure the SQuAD solver's ability to correctly answer a question according to contexts of different lengths, I plot the two figures shown above. Surprisingly, the SQuAD solver performs the best for contexts of length around 350-450, instead of the shortest contests. This is in contrast to a RNN-based architecture, where one should expect best performance for shortest lengths. Nevertheless, we can see a performance drop for context lengths greater than 500. This performance drop may come from the maximum allowed input sequence preset in ALBERT model being 512. These curve demonstrate the superior ability to build long-distance dependency in reading comprehension of transformer-based architectures.

5.2 Classifier analysis

Classification Report	precision	recall	F1	support
0	80	90	0.85	3065
1	90	80	0.84	3333
accuracy			0.85	6398
macro avg	0.85	0.85	0.85	6398
weighted avg	0.85	0.85	0.85	6398

Table 2: Classification report of classifier

According to the classification report, the classes of answer/no answer are approximately well-balanced (roughly 10:11). The classifier has a tendency of predicting a (context, question) pair as having an answer. This is reflected by the lower precision for class 0 (has answer) and higher precision for class 1 (no answer). Overall, the classifier achieves 85 accuracy and 85 F1 score on the classification task.

6 Conclusion

In this project, I tried both Non-PCE method and PCE method on the SAuAD 2.0 challenge. With the help of larger model size and more training data during pre-training, PCE-based methods perform noticeably better than the BiDAF (Non-PCE) baseline. Then, I stacked a SQuAD solver and a classifier to ensemble a final model. The ensemble process improves model performance by 0.7, measured by EM and 0.6, measured by F1 in case of the ALBERT-large model. My final model achieves **EM=79.746, F1=82.533** on the test set.

During the process of my experiments, I find the importance of having the correct training configuration. For example, if I use 1e-5 (instead of 5e-6) as the learning rate when training the ALBERT-large classifier, it simply won't work, with prediction accuracy oscillating around 50. Due to the time constraint imposed by this project, I cannot perform very extensive hyper-parameter search on different training configurations. This points out an avenue for future work: how to efficiently find the optimal hyper-parameter sets instead of performing iterative search.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016.

[4] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[5] Danqi Chen. *Neural Reading Comprehension and Beyond*. PhD thesis, Stanford University, 2018.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[7] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2018.

[8] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[9] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

A Appendix

Training curves for the final model (ALBERT-large SQuAD Solver and Classifier)

A.1 SQuAD solver

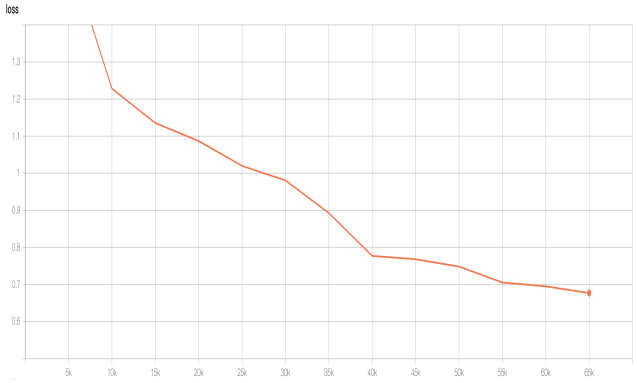
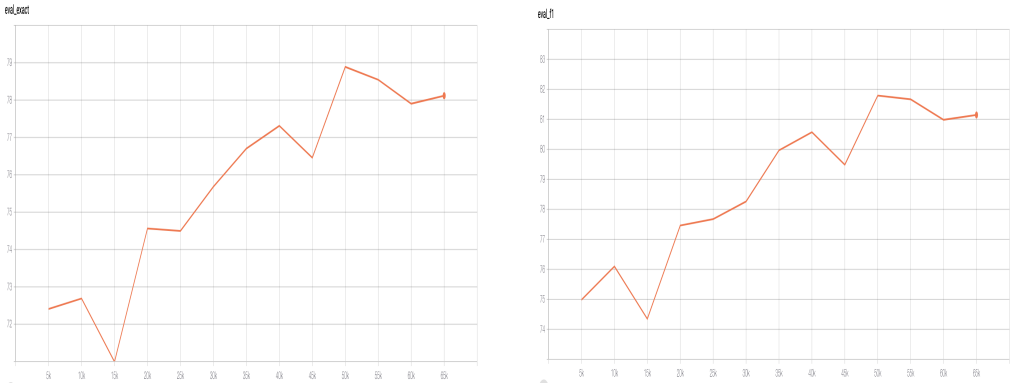


Figure 6: Training loss, logged every 5000 iterations

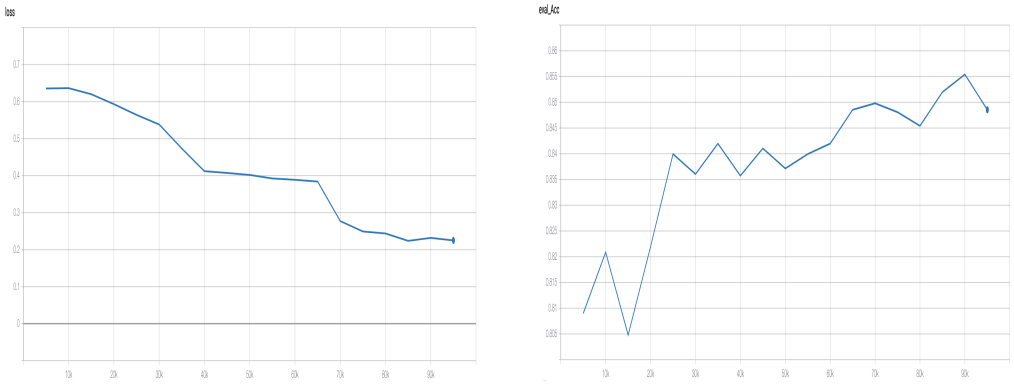


(a) EM on dev set

(b) F1 on dev set

Figure 7: Evaluation metrics on dev set during fine-tuning

A.2 Classifier



(a) Training loss (b) Prediction accuracy

Figure 8: Trainig loss and evaluation accuracy of classifier during training, logged every 5000 iteration