CS263 NLP: Final Report
Earnings Sentiment Analysis: A Comparison of Transfer Learning Methods

Kwang Jun Kim
UCLA Henry Samueli School of Engineering

Jiajin Li
UCLA David Geffen School of Medicine

Athanasse Zafirov
UCLA Anderson School of Business

June 2020

Abstract

The popularity of employing language models has increased for a variety of uses, called downstream NLP (natural language processing) tasks in the context of pre-trained models. While the BERT (Bidirectional Encoder Representation from Transformers) language model constitutes a landmark on this scene, widespread interest has led to a steady pace of research suggesting ever new and improved methods. Given this constantly evolving landscape, we contrast the results and implementations of two newer models, ALBERT and XLNet, to better understand their relative strengths and pinpoint the current state of the art, as well as evaluate their performance gains over BERT. The chosen downstream NLP task for this benchmark study is the sentiment analysis of firms' earnings announcements, classifying them as either negative or positive based on the company's relative sales performance for the period. Our final results showed that ALBERT performed better than the alternatives with fewer parameters, highlighting the relevance of considering multiple language models when aiming for the highest performance.

1 Introduction

We aim to improve on the results of Siano and Wysocki's work in 2019 to classify and predict the sentiment of firms' earnings announcement disclosures by using two of the newest state-of-the-art generalized pretraining methods, XLNet (Yang et al., 2019) and ALBERT (Lan et al., 2019). Using uncased text, ten-fold cross validation and a 2e-5 learning rate, they fine tuned a pre-trained BERT model (Devlin et al., 2018) with the computationally leaner baseline version 12 layers of parameters instead of the expanded 24 layers. The authors achieved testing accuracy of 0.77 in classifying revenue performance into "positive" and "negative" based on the first 10 sentences of their earnings announcement.

Advances in language models are on-going and frequent. We want to understand and quantify the improvements of newer models, such as XLNet's use of a Permutation Language Model and ALBERT's parameter-reduction techniques, on a particular downstream task. Earnings announcements are categorized as "positive" when a company's year-on-year revenue growth is above the sample median for that quarter, and "negative" otherwise. This automatic labelling allows us to focus more time on our state-of-the-art model implementations.

2 Data

We downloaded quarterly earnings announcements from the Factiva database for our sample of January 1990 to December 2019. We confirmed with their tech support that there was no API access or way to download all results of a query in bulk, just manually one page result at a time (with 100 results per page) from the approximately 21,000 query results. We proceeded one year at a time. The query specified Business Wire (U.S.) as a source, the United States as a region, earnings announcement as a category ($ns = C151$) having "Q1 Results" or similar text in the headline[1]. This departs and appears to be more strict from Siano and Wysocki (2019)'s query because the results obtained using their described steps returned many non-relevant entries for us, whereas our method returned only quarterly announcements from what we observed. However like them we excluded articles which had confounding events, announcements and conference calls. Only the first quarter announcements are used in order to isolate the period being discussed in the text as much as possible (later fiscal year entries often describe year-to-date performance). Once the Factiva articles were downloaded, they were individually split using "BWR" marker, then their publication date, ticker, stock exchange, title and text body were separated. Entries which could not have their tickers and exchanges isolated successfully using regular expressions were discarded. As Siano and Wysocki (2019), after removing tables from the texts, we discarded entries with less than 5 sentences and kept only the first 10 sentences as our text feature x_i.

[1] The Factiva query was as follows: ns=C151 and hd="Results for First Quarter" or hd="First Quarter Results" or hd="Q1 Results" or hd="Results for Q1" not ("to report" or "will report" or "to announce" or "will announce" or "Call" or "Webcast")

Sales performance data was acquired for all U.S. firms going back to 1990 from the Compustat database through WRDS. First quarter year-over-year revenue change was categorized as "positive" when it ranked above that year's sample median, and "negative" when below it, this is our outcome y_i variable. This sales data was merged with the Factiva text entries on three data points: stock ticker, calendar date and exchange (namely NASDAQ, NYSE and AMEX). After examining the results we could confirm that success merger were high-quality. This yielded 7,021 observations, more than twice the size of Siano and Wysocki (2019)'s sample of 3,123.

3 ALBERT Model

3.1 Model Overview

ALBERT (Lan et al., 2019) is an upgrade to BERT that advances the state-of-the-art performance on 12 NLP tasks including GLUE, SQuAD v2.0, and the SAT-style reading comprehension RACE benchmark. It is a lite BERT architecture that solves the memory limitation problem in scaling large pre-trained models and achieves optimized performance.

Previous studies have shown that a large neural network is crucial for getting state-of-the-art performance. But it is easy to hit the memory limitations of available hardware if there are hundreds of millions or even billions of parameters to train. To address this challenge, ALBERT incorporates two parameter reduction techniques. First, ALBERT employs the factorization of the embedding parametrization. The large vocabulary embedding matrix is decomposed into two small matrices, while the hidden layer embeddings use higher dimensionalities. So the size of the hidden layers is separated from the size of input-level embeddings. Therefore, it is easier to grow the size of the hidden layers without substantially increasing the parameters of the input-level embeddings. Second, ALBERT applies cross-layer parameter sharing. Transformers like BERT depend on independent layers stacked on top of each other. However, it is observed that neural networks usually learn to perform similar operations at different layers using different parameters, leading to redundancy. Cross-layer parameter sharing prevents the parameter size from increasing along with the depth of the neural network. These two techniques can significantly reduce the number of parameters for BERT without seriously lowering performance. These parameter reduction techniques can also act as a type of regularization to avoid overfitting and improve the generalization of the model. In addition, ALBERT introduces a self-supervised loss for sentence-order prediction, which focuses on inter-sentence coherence and improves the effectiveness of the next sentence prediction loss in the original BERT. With these designs, ALBERT can achieve much better performance than BERT with significantly fewer parameters.

3.2 Fine-tuning ALBERT for Earnings Sentiment Analysis

In our project, ALBERT was implemented with PyTorch and transformers libraries in Google Colab and trained with GPU acceleration. We used the pre-trained ALBERT models provided in transformers library and fine-tuned it for the sentiment analysis of financial statements. The data was randomly separated into training set, validation set, and test set, which account for 70%, 20%, and 10% of the entire dataset, respectively. We used accuracy, area under the ROC curve (AUC), and Matthews Correlation Coefficient (MCC) as the metric to evaluate model performance. As there are eight ALBERT architectures in transformers library, ranging from "albert-base-v1" to "albert-xxlarge-v2", we compare their performance using the same hyperparameter settings and datasets. After choosing the ALBERT model with the best performance, we fine-tuned its hyperparameters including learning rate, batch size, and maximum sequence length to further increase its performance in classifying financial statements. Ten-fold cross-validation was also used to reduce biases and find the suitable setting of hyperparameters.

3.3 Results

3.3.1 Model Selection

Because of the limited GPU memory of Google Colab computation environment, we compared four ALBERT architectures with relatively few parameters, i.e., "albert-base-v1", "albert-base-v2", "albert-large-v1", and "albert-large-v2". The base version of ALBERT has 12 repeating layers, 128 embedding, 768-hidden, 12-heads, and 11 million parameters, while ALBERT large model contains 24 repeating layers, 128 embedding, 1024-hidden, 16-heads, and 17 million parameters. We trained each model for 5 epochs on the same training set created as described above, with batch size of 8, maximum sequence length of 512, learning rate of 2e-5. Then we evaluated four fitted models on the same test set. To avoid possible biases introduced by the choice of training set and test set, we applied ten-fold cross-validation and reported the average evaluation metrics. As shown in Table 1, "albert-base-v2" had the best performance on our financial dataset, with the lowest loss, highest accuracy, highest AUC, and highest MCC among the tested models. Therefore, we selected "albert-base-v2" in our downstream analysis.

Architecture	Loss	Accuracy	AUC	MCC
albert-base-v1	0.6476	0.6404	0.6388	0.2785
albert-base-v2	**0.6444**	**0.6475**	**0.6477**	**0.3048**
albert-large-v1	0.6408	0.6463	0.6453	0.2907
albert-large-v2	0.6938	0.5176	0.5183	0.0250

Table 1: Comparison of four ALBERT architectures. It reported the average loss, accuracy, AUC, and MCC of four models in ten-fold cross-validation.

3.3.2 Fine-tuning Hyperparameters

After choosing the suitable architecture for our task, we next needed to fine-tune its hyperparameters, including learning rate, batch size, and maximum sequence length, to further improve its performance. We applied ten-fold cross-validation and performed grid search to determine the hyperparameters to achieve the best performance. In this experiment, we trained the model using the same strategy as mentioned above. In each time, we changed one hyperparameter and fixed others to examine which values could achieve the best performance. To fine-tune learning rate, we trained the "albert-base-v2" model with batch size of 8, maximum sequence length of 512, and learning rate of 2e-4, 2e-5, and 2e-6. To fine-tune batch size, we trained the "albert-base-v2" model with maximum sequence length of 512, learning rate of 2e-5, and batch size of 2, 4, 8, and 16. The batch sizes were bounded by the limited GPU memory in Google Colab. To fine-tune maximum sequence length, we trained the "albert-base-v2" model with learning rate of 2e-5, batch size of 8, maximum sequence length of 64, 128, 256, 512, and 816, because the longest sentence in our dataset has 811 words. To fine-tune learning rate, we trained the "albert-base-v2" model with batch size of 8, maximum sequence length of 512, and learning rate of 2e-4, 2e-5, and 2e-6. As shown in Table 2, we found that ALBERT had the lowest loss, highest accuracy, highest AUC and highest MCC on our dataset with learning rate of 2e-5, batch size of 16, and maximum sequence length of 512.

Hyperparameter	Value	Loss	Acc	AUC	MCC
Learning rate	2e-4	0.71	0.49	0.50	0
	2e-5	**0.61**	**0.67**	**0.67**	**0.36**
	2e-6	0.63	0.66	0.65	0.33
Batch size	2	1.07	0.66	0.66	0.31
	4	0.67	0.64	0.64	0.28
	8	0.65	0.64	0.64	0.29
	16	**0.64**	**0.66**	**0.66**	**0.34**
Maximum sequence length	64	0.70	0.57	0.57	0.15
	128	0.68	0.62	0.62	0.25
	256	0.64	0.68	0.68	0.36
	512	**0.64**	**0.69**	**0.69**	**0.37**
	816	0.64	0.67	0.67	0.34

Table 2: Performance of ALBERT with different selection of learning rate, batch size, and maximum sequence length. The average loss, accuracy (Acc for short in the table), AUC, and MCC in ten-fold cross-validation were shown.

3.3.3 Comparison Between ALBERT and Baseline BERT Model

As our goal is to improve the results of Siano and Wysocki (2019), which proposed to fine-tune a pre-trained BERT base uncased model for earnings sentiment analysis. However, we used a dataset different from the dataset of Siano and Wysocki (2019), so we trained a BERT base uncased model with the hyperparameters described in their paper using our data, and compared it with the fine-tuned ALBERT model. To avoid the biases introduced by the random choice of training set and test set, we applied ten-fold cross validation and calculate the average evaluation metrics. The results showed that the fine-tuned ALBERT had better performance than BERT used in Siano and Wysocki (2019), as demonstrated by lower loss, higher accuracy, higher AUC, and higher MCC. Therefore, we successfully improved the prediction of earnings sentiment analysis by fine-tuning a pre-trained ALBERT model.

Model	Loss	Accuracy	AUC	MCC
ALBERT	**0.6066**	**0.7083**	**0.7052**	**0.4135**
Baseline BERT	0.6109	0.6863	0.6898	0.3810

Table 3: Comparison of ALBERT and the baseline BERT model. It showed the average loss, accuracy, AUC, and MCC of four models in ten-fold cross-validation.

4 XLNet Model

4.1 Model Overview

XLNet (Yang et al, 2019) is auto-regressive pretraining method that is combined with permutation language modeling, and it achieved state-of-the-art results on various types of NLP benchmark tests such as RACE, SQuAD, MNLI, and SST-2 when published in 2019. Before the advent of XLNet, BERT, a previous state-of-the-art pre-trained model, has a couple of limitations. Firstly, it is unable to reckon phrasal words. Secondly, its technique of masked token does not exist in real-world documents and masked tokens disappear in downstream which makes token prediction tricky. XLNet overcomes such BERT's downsides through permutation language modeling, two-stream self-attention, and Transformer-XL integration.

Permutation language modeling is a derivative of auto-regressive modeling where the order of token sequence is shuffled. Such randomized ordering shows a similar effect of bidirectional auto-encoding so that it can predict tokens ahead with a sequence of previous tokens. Two-stream self-attention is a technique that separates query stream and content stream in order to predict the information of a token. Lastly, XLNet fetches a couple of ideas from Transformer-XL which are relative positional encoding scheme and segment recurrence mechanism. Relative positional encoding scheme enables positional encodings when using old hidden states, and segment recurrence is a mechanism to propagate states of previous segments. Embracing these ideas, XLNet becomes better in capturing deep contextual information.

4.2 Feasibility Check

For this project, our strategy with XLNet started with checking feasibility with CoLA dataset which is a binary classification of grammar acceptability. We chose PyTorch and transformers as our machine learning libraries for implementation and fine-tuned XLNet for classification with CoLA, and trained it in Google's Colab environment with GPU acceleration. The pre-trained XLNet model, "xlnet-base-cased", was used because its single linear classification layer shows good performance for binary classification. As for hyperparameters, we set batch size as 32 and epoch as 4 based on our finding that training loss and validation loss no longer decreased after the fourth epoch with batch size 32. With XLNet fine-tuned for CoLA as a preliminary step, we confirmed that overall Matthews Correlation Coefficient (MCC) was 0.51.

4.3 Fine-tuning with Financial Dataset

We fine-tuned XLNet with our earnings statement data for sentiment classification. Similar with feasibility check, we implemented XLNet with PyTorch and transformers libraries on Google Colab with Tesla P100 GPU which has 16 Gb memory. Since our task was also binary classification, we tested "xlnet-base-cased", a pre-trained XLNet model. To align with the experiment of ALBERT, we divided our financial dataset proportionally so that training set, validation set, and test set accounted for 70%, 20%, and 10% of the dataset, respectively.

Regarding hyperparameters, there are three big dimensions, which are learning rate, batch size, and maximum sequence length. For learning rate, we tried with 2e-4, 2e-5, and 2e-6. For the dimension of learning rate, we fixed batch size to be 8 and maximum sequence length to be 512. For batch size, there were three possible values, 2, 4, and 8. Trying with batch size larger than 8 led to GPU memory overflow so we decided to try up to 8. For this dimension, we fixed learning rate to be 2e-5 and maximum sequence length to be 512. For maximum sequence length, we tried with 64, 128, 256, and 512. Maximum sequence length over 512 also induced GPU memory overflow due to expanded arrays with padding so we tried up to 512. For this dimension, we fixed learning rate to be 2e-5 and batch size to be 8. Because over 90% of the sentences had length less than 512, maximum sequence length of 512 covered most sentences in the financial dataset. Epoch was fixed as 4 to avoid overfitting.

4.4 Results

Hyperparameter	Value	Loss	Acc	AUC	MCC
Learning rate	2e-4	0.71	0.53	0.50	0
	2e-5	**0.52**	**0.66**	**0.66**	**0.32**
	2e-6	0.57	0.64	0.65	0.29
Batch size	2	0.71	0.53	0.50	0
	4	0.49	0.61	0.62	0.24
	8	**0.48**	**0.68**	**0.66**	**0.33**
Max sequence length	64	0.70	0.47	0.50	0
	128	0.51	0.57	0.57	0.14
	256	0.52	0.59	0.59	0.18
	512	**0.48**	**0.66**	**0.66**	**0.31**

Table 4: Performance of XLNet with different selection of learning rate, batch size, and maximum sequence length. The average loss, accuracy (Acc for short in the table), AUC, and MCC in ten-fold cross-validation are reported.

We measured accuracy, area under the ROC curve (AUC), and Matthews Correlation Coefficient (MCC) to evaluate model performance for comparison with ALBERT. According to performance result (Table 4), learning rate, batch size, and max sequence length were optimal at 2e-5, 8, and 512 respectively. Note that batch size and maximum sequence length had upper bounds due to potential GPU memory overflow. If we could use GPU with larger memory, it was more likely to get better performance with increased values of batch size and maximum sequence length. In our study, with the best hyperparameter setting, XLNet had worse performance than ALBERT, because it had lower accuracy, AUC, and MCC, despite its lower loss.

5 Conclusion

For earnings sentiment analysis, ALBERT showed better performance than XLNet when hyperparameters for each model were set optimally. Performance of ALBERT was optimized when learning rate was 2e-5, batch size was 16, and maximum sequence length was 512. Meanwhile, XLNet achieved its best performance when learning rate was 2e-5, batch size was 8, and maximum sequence length was 512. In addition, with ALBERT, we successfully improved on the results of earnings sentiment analysis conducted by Siano and Wysocki (2019), because ALBERT had better performance on our dataset than baseline BERT.

6 Discussion

Results for our financial dataset with ALBERT and XLNet are less promising than results with other datasets such as CoLA. One of speculated reasons is the small size of our financial dataset which has total 7021 sentences, and the size of our financial dataset might not be big enough to train models thoroughly. We should get more data to further improve our results. Another reason for relatively low performance might be the limited hardware resources. Both ALBERT and XLNet show performance improvement when hyperparameters such as batch size and maximum sequence length increase. If we can secure high-end GPU for experiments, we would come up with better results.

We can see that hyperparameter settings for both models are similar to each other when they show their best results. However, there is a difference between the models. the performance of XLNet is more sensitive to its hyperparameters, compared to ALBERT. In that sense, XLNet is likely to benefits more than ALBERT when there are more GPU memory for larger batch size and maximum sequence length.

In our project, ALBERT had better performance than BERT and XLNet, although ALBERT has significantly fewer parameters than BERT and XLNet. And ALBERT base model with fewer parameters performed better than ALBERT large model with more parameters. It indicates that NLP tasks are complicated and the model performance for a specific task does not solely depend on the number of parameters. Different models have different properties, and thus learn differently from the data. Therefore, in practice, it is reasonable to try multiple models for a NLP task.

7 References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805.*

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv preprint arXiv:1909.11942.*

Federico Siano and Peter D. Wysocki. 2019. Transfer Learning and Textual Analysis of Accounting Disclosures: Applying Big Data Methods to Small(er) Data Sets. Available at *SSRN: https://ssrn.com/abstract=3560355*

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R. Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237*