ICLR Reproducibility Challenge 2019 : LOOKING FOR ELMOS FRIENDS: SENTENCE-LEVEL PRETRAINING BEYOND LANGUAGE MODELING

Yueh Wu, Chuan-Chun Kuo & Yi-Ming Chen*
Department of Information and Computer Science
*Department of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, CA 92612, USA
{yuehw,chuanchk,yimingc9}@uci.edu

Abstract

As a course project, we try to explore the reproducibility and provide more evidence of *'Looking for ELMo's friends: Sentence-Level Pretraining Beyond Language Modeling'* (Anonymous, 2018). In this report, we provide some analysis on how the authors demonstrated the model designs and experiment settings so as to be recreated by other readers. Furthermore, by closely following the methodologies and applying training details provided in the paper, we present our results on a portion of experiments originally conducted by the authors. We referenced parts of the experiment set-ups from GLUE-baselines and also made our complete implementation available[1]. Finally, from the part we conducted, we show that our own results concord with those claimed in the paper with high reproducibility .

1 Introduction

This report is our submission to the ICLR 2019 reproducibility challenge, where we tried to reproduce the result of the paper, "*Looking for ELMo's friends: Sentence-Level Pretraining Beyond Language Modeling*" (Anonymous, 2018). The goal of this challenge is to determine whether the result and conclusion presented in the original paper are valid, and to analyze the reproducibility of this paper.

In their paper, the authors presented a systematic study to compare different pretraining tasks of language modeling and how it would affect the performance. They mainly applied *ELMo* Matthew et al. (2018), a deep bi-directional language model (biLM), for pretraining tasks. In conclusion, they stated that Language modeling works well as a pretraining task. However, given a small margin between baseline and pretraining tasks metrics, and low correlations between target tasks, such a pretraining method that underlies ELMo and other Language models might not be a promising method for future development.

To reproduce this work, we followed the same experiment at the paper by applying several language understanding tasks in the *GLUE benchmark* (Alex et al. (2018)), and evaluated the pretrained model after each task-specific training. Afterward, we would compare our result with theirs to verify the conclusion and reproducibility of this paper.

The remainder of this paper is organized as follows: section 2 analyzes the reproducibility of this paper, section 3 illustrates the specification, steps and result of the experiment. Finally, the conclusion is proposed in section 4.

[1]https://github.com/brian2213/CS273A/

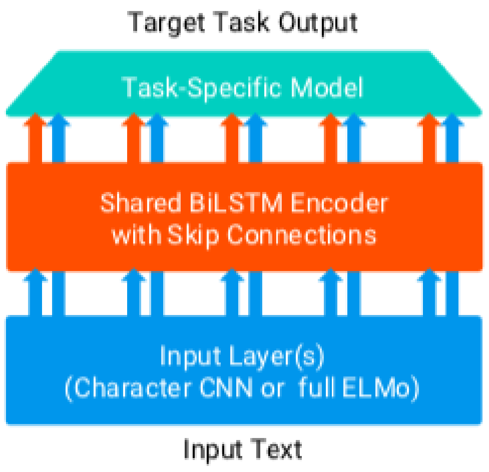
Target Task Output
Task-Specific Model
Shared BiLSTM Encoder with Skip Connections
Input Layer(s) (Character CNN or full ELMo)
Input Text

Figure 1: Common model design is taken from the original paper. The whole architecture contains three major components: Input Layer(s), Shared Encoder, and Task-Specific Model. During pretraining, both the shared encoder and task-specific model are trained. During target task evaluation, the task-specific model has trained anew while the shared encoder is frozen.

2 Reproducibility Analysis

2.1 Clarity of Task Description

Before quickly jumping into the details of the model architectures, the authors spent a whole section introducing tasks which serve for pretraining a shared encoder. These include 1 baseline task, 9 GLUE benchmark tasks, and 5 outside (non-GLUE) tasks. For each task, they give some descriptions as to how it is used and what purpose it serves in this study. In general, we find this useful for selecting a subset of tasks to test reproducibility.

2.2 Clarity of Model Description

The common model design in this paper is fairly simple, with slight variations for each target task. Figure1 shows the image taken from the original paper. The model consists of three major components: word-level encoding, sentence-level encoding, and task-specific model. As the authors pointed out:

1. The core of the model design is a shared encoder with skip connections, made up of a two-layer 1024D bidirectional LSTM. Through the skip connection design, the subsequent module sees both the contextual representation taken from the sequence of top-layer hidden states as well as the input word representation.
2. The pretrained character-level CNN word encoder from ELMo(Matthew et al., 2018) is used as a standard input layer in every model. Some of the experiments use the full pretrained ELMo contexualized word representations, which includes training two sets of scalar weights for ELMo's three layers. One of which for input to shared encoder and the other for input to task model via skip connections.
3. As to the task-specific model, a one-hidden-layer MLP is applied after linear projecting the output states of the shared encoder and max-pooling over those projected states for single sentence tasks and smaller sentence-pair tasks. For sentence-pair tasks with larger training datasets, attention mechanism and 512D*2 biLSTM with max-pooling over time as in BiDaF(Seo et al., 2016) is used before applying the MLP layer.

Though rather brief, we found the descriptions of tasks and models being straightforward and comprehensible. However, in case of re-implementing the full model architecture, one has to go through considerable external resources for details that is merely covered in the context.

2.3 Availability of Datasets

For tasks included in the GLUE-benchmark, links for the datasets can be found at section 5 in the paper. For outside tasks, datasets such as WikiText-103, 1 Billion Word Language Model Benchmark, WMT14 English-German, WMT17 English-Russian, and Reddit are used. The links are not directly provided in the paper, so some web search would be needed to acquire them. As to data preprocessing, the authors noted that Moses tokenizer is used for encoder inputs. Also, since ELMo's character-based input layer is used, no input vocabulary is needed.

2.4 Availability of Code

As mentioned by the authors, the models are implemented using the AllenNLP toolkit(Gardner et al., 2018), and the design of the models roughly follows those in GLUE and ELMo. However, a link to the documented code will only be added provided the paper is accepted.

2.5 Computational Requirements

Though the amount of GPUs used isn't indicated in the paper, it's clearly stated that required time for running the whole experiment is more than 100 GPU-days on NVIDIA P100 GPUs. As students working on a course project, our evident lacking of such resources in terms of hardware and time suggests we only select some portion of experiments conducted in the original paper.

2.6 Ease of Re-implementation

The typical hurdle for machine learning project can be attributed to different aspects. First, the easiness of acquiring the datasets necessary for workload and the cost corresponded. Second, the related framework and tools are existed or not. Finally, the accessibility of powerful computing machine.

Mentioned from above sections, this project falls into the category of a comprehensive evaluation in methodology. Which means, the datasets used in this project are open to the public and allows researchers to apply the data in their interest of fields without any cost. Also, this project is built upon existing frameworks and libraries that have good eco-system for NLP. Therefore, the hardness of building fundamental components is also minimized. Then the only concern left here is the computing resources.

As the referenced paper clearly pointed out more than 100 GPU days are consumed, which might make people dodge away from reimplemenating it at their first thought. In our work, we overcome the obstacle through the resources available from Google Colab without any charge though lower spec. Then, Jupyter script and corresponding shell script are coded up to be responsible for loading dependency and executing the tasks automatically. Finally, multiple running machines are executed in parallel by executing the same scripts with different task specified and with different Google account that instantiated multiple virtual machines to approximate the performance of Nvidia P100.

3 Reproduction Experiments

3.1 Computational Resource

Since this project is initiated from a college course, the resources of computation power are typically limited to the student, but undeniable, training the learning model purely on CPU is not a wise choice and would possibly have to take the enormous amount of time in order to produce a single result. As an alternative, we resorted to Google Colab, which provides free remote GPU access with its running environment on Jupyter Notebook, and used its Nvidia K80 GPU to reproduced the results in this report.

However, the hurdle of computation power in this project isn't completely removed since any instance of machine created by Google Colab would be reset for every 12 hours. Which means all running training tasks have to be terminated and the results would just be lost after the timeout. In order to perform the GLUE Benchmark and try to reproduce the result using the same approach as original paper suggested, we organized each of our training tasks would be fitted within 12 hour time slot instead of running all tasks at once. By doing so, the whole GLUE benchmark could be divided into different small tasks and combining all the tasks result together afterward.

One thing worth to mention is that Google also provides another commercial mean of access to its computing machine which is Google Cloud. A virtual machine for this project can be instantiated in minutes by choosing the option of image file from Google Cloud Platform. The option of Deep Learning VM image is optimized for machine learning framework and GPU workload by providing software dependencies as it comes such as Python library and Nvidia GPU library(CUDA, CuDNN, NCCL). The virtual machine instance set up on Google Cloud is able to obtain more powerful GPU such as Nvidia P100 and Nvidia V100 which is roughly 3 to 10 times quicker than the K80 of Google Colab in practice. And indeed, some results in this report are populated by using Google Cloud with Nvidia P100 to speed up.

3.2 Architecture and Dependency

Our project follows the same architecture as proposed in the referenced paper. Two major components give critical functionality for Natural Language Processing. One is PyTorch, an open source Python library for machine learning, and another one is AllenNLP, a NLP library built on PyTorch.

Maintained by Facebook AI Reaseach Group, PyTorch is built on scientific computing framework, Torch, to process matrix and tensor. It provides similar operation like Numpy but with more advanced features such as GPU acceleration and automatic differentiation system for deep neural network. Especially in our project, the GPU acceleration plays an important role to perform the benchmark. The same task could be executed for weeks by CPU or hours by GPU.

AllenNLP is maintained by Allen Institute and built upon PyTorch. It provides models and tools for NLP problem such as tokenizer, indexer, tagger, embedder and encoder. These handy features give us the advantage to build LSTM model and ELMo machine necessary for GLUE benchmark and allow us to put more attention on the training model structure and verification of benchmark results instead of low-level implementation of each small components.

In regards to the GLUE benchmark, the GLUE-baselines[2], managed by NYU, demonstrates the implementation of test and the access to datasets in decent details. For example, the paradigms of COLA/SST/MRPC benchmark, the hook of pretrained model layer, the scripts of acquiring data and the checkpoint of training task. We fabricated our own version of codes by referencing this repository and performed our own analysis with modification to provide results in this report.

3.3 Tasks Selection

Based on our computational resources, we chose the following three GLUE tasks: CoLA, SST and MRPC, to perform the experiment. By applying these tasks, not only could we finish each pretraining/ task-specific task within 12 hours to avoid resetting at Google Colab, but we could also retrieve analytical result to verify the reproducibility of the original paper. Moreover, both of these tasks are categorized as Medium-Data at the original paper, which denotes that we could use similar hyper-parameters to perform the experiment. (Detailed hyper-parameters we applied are shown in Appendix)

3.4 Training and Evaluation

As the training procedure of the original paper, we specified the tasks to pretrain the share-encoder, and then freeze the first two layers to train task-specific model respectively by three GLUE tasks. Three single GLUE tasks, multitask and N/A(random initialized encoder) were applied to pretrain, and each with or without the pretrained ELMo BiLSTM layers.

[2]https://github.com/nyu-mll/GLUE-baselines

Pretr.	**Avg**	**CoLA**	**SST**	**MRPC**
Baselines				
N/A	52.8	13.2/70.1	75.2	69.9/75.9
Single-Task	53.8	14.8/70.4	76.8	69.9/75.9
N/A(E)	59.3	24.3/64.2	82.7	71.1/76.7
Single-Task(E)	58.7	21.5/70.9	83.0	71.6/77.2
GLUE Tasks as Pretraining Tasks				
CoLA	54.9	14.8/70.4	76.5	73.3/77.9
SST	55.1	15.1/69.8	76.8	73.3/78.3
MRPC	48.6	0.0/69.12	76.0	69.9/75.9
CoLA(E)	59.3	21.5/70.9	82.7	73.8/78.6
SST(E)	60.4	24.2/61.2	83.0	74.0/78.9
MRPC(E)	57.8	18.9/67.2	83.0	71.6/77.2
Multitask Pretraining				
All	55.3	14.4/69.6	78.8	72.8/78.1
All(E)	60.9	23.0/71.7	83.4	76.5/80.5

Table 1: GLUE results. ”(E)” denotes that full ELMo used as the input layer

To evaluate the result, we applied the same accuracy metric at the original paper, except that we added categorical accuracy of CoLA to measure the performance of binary classification.

3.5 Result

Table 1 shows the results of all our pretrained encoders. We took N/A (random initialization) and Single-Task (one-on-One pretraining/task-specific training) as our baselines, and compared with single GLUE tasks and multitask pretraining tasks at each row.

Obviously, pretraining with ELMo BiLSTM layers does improve scores of tasks. It is easy to tell by comparing different pretraining tasks with and without ELMo in different rows. Another observation is that multitasking pretraining also improved the performance. As we compare its average with any of GLUE tasks, N/A or Single-Task, the score of multitask pretraining is the highest among all.

However, the small margin between result reveals the limitation of language modeling and multitask pretraining. For example, the score of MRPC with ELMo pretraining is merely different from the one without ELMo, which denotes that some GLUE tasks can’t benefit from ELMo pretraining. Moreover, even though the average score of multitask pretraining exceeds all the other corresponding tasks, it can’t beat some of their best individual scores. For instance, score of CoLA task pretrained on multitask(E) is lower than the one pretrained on SST(E).

4 Conclusion

The following conclusion of the overall reproducibility regarding to our selected paper comes in two perspectives.

According to the original paper (Anonymous, 2018), even though pretraining tasks works better with ELMo language modeling and multitask pretraining comparing to those tasks without them, the authors address that pretraining and freezing the sentence encoders might not be an ideal platform for future works. Given that margins between some pretraining tasks are small, and performance of multitask pretraining tasks fail to exceed their best individual component.

Our result supports the above statement. After carefully examining our result and comparing with their conclusion, we could observe that pretraining tasks with ELMo and multitask task do improve the performance generally. Nevertheless, we could still list a small portion of outliers or small

	Hyper-parameters
Classifier dropout rate	0.2
Classifier hidden dim	256
Max vocabulary size	20000
Early stopping	20 (Epoch)
Min learning rate	1e-6

Table 2: Other hyper-parameters we used to reproduce the experiment

margin of difference if we compare individual cases separately. It limits the possibility to extend this pretraining method for future development.

On the other hand, in terms of acquiring necessary datasets, labraries and code platform to build upon and to re-run the experiments, we haven't met too much obstacles. In such a systematic study comparing the effectiveness of various pretraining tasks, we agree that the authors had successfully introduced their intuition, explained the procedures, and provided especially extensive analysis of their findings. One way to add clarity that we could think of is giving more details to the implementation of task-specific models, since in the paper, descriptions of models and training procedures are intertwined. Probably due to strict limitations of computation resources, which happens to be our biggest constraint, we could not have the opportunity to explore every aspect of their experiment. Nevertheless, the fact that our own results share consistency with theirs gives rise to our overall positive attitude towards reproducibility.

5 Appendix: Hyperparameters Details

We tried to follow every the hyper-parameters setting of paper to reproduce the result. Below are two setup that worth to be mentioned. As for other hyper-parameters, they are shown in Table 2.

Task Sampling We applied proportional scaling to determine the probability of sampling the task. The probability is $P_j = N_j / \sum_j f(N_j)$, where N_j is the number of training data at task j. Comparing with uniform sampling, it could avoid imbalance sampling of small-data task and overfitting.

Loss Scaling We also utilized proportional scaling to calculate loss of task. The weight of loss is $w_j = N_j / \sum_j f(N_j)$.

References

Wang Alex, Singh Amanpreet, Michael Julian, Hill Felix, Levy Omer, and R. Bowman Samuel. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv*, pp. preprint 1804.07461, 2018.

Anonymous. Looking for elmos friends: Sentence-level pretraining beyond language modeling. *Seventh International Conference on Learning Representations (2019 ICLR)*, 2018.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*, 2018.

Peters Matthew, Neumann Mark, Iyyer Mohit, Gardner Matt, Clark Christopher, Lee Kenton, and Zettlemoyer Luke. Deep contextualized word representations. *2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1(Long Papers):2227–2237, 2018.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.