REPRODUCTION OF FAST AND ACCURATE TEXT CLASSIFICATION: SKIMMING, REREADING AND EARLY STOPPING

Feng Xie & Wei Deng & Peng Liu
School of Electronics and Computer Science
University of Southampton
Southampton, UK
{fx1n18, wd1u18, pl1u18}@soton.ac.uk

ABSTRACT

In this report, we presented the results from our own reproduction of the chosen paper which applied reinforcement learning to encourage fast and accurate text classification. Comparing our results with that in the chosen paper, we found that the experiments specified in the chosen paper cannot be fully reproduced due to the lack of implementation details. Therefore, based on our best knowledge and understanding on the chosen paper, an analysis and a discussion were made at the end of our report.

1 INTRODUCTION

Our chosen paper proposed a new model based on recurrent neural networks and reinforcement learning, aiming at faster text classification with high accuracy. The model has two modules: a policy module and a recurrent encoder module. With the policy module, the agent is able to reread and skim tokens and gives predictions when it is confident enough. In our implementation and experiments, we found that the paper did not give details about how they applied REINFORCE with baseline and the advantage actor-critic as well as the hyper-parameters in the networks. Considering the fact that the idea of our chosen paper was inspired by (Wei Yu et al. (2017)) and had many similar experiments, we used some same settings to fill our missing details.

We evaluated our models on the IMDB dataset but failed to reproduce the results described in our chosen paper. The results we got from our models using reinforcement learning are even worse than the whole reading model which did not have reinforcement learning module. We concluded that the missing details may be the reason that led us to failure.

2 METHODOLOGY

2.1 OVERVIEW

There are three baseline models proposed in the paper including a whole reading model, a partial reading model and an early stopping model. The authors of our chosen paper applied the models in different datasets covering the topics of sentiment analysis and topic classification, trying to prove the accuracy and efficiency of their proposed model. However, in our experiments, we only applied the models on the IMDB dataset due to the difficulty of training and more importantly, we did not find the models as effective and accurate as described in the paper.

2.2 SPECIFICATIONS

2.2.1 GENERAL PROCESSING

In our chosen paper, each review of the IMDB dataset was padded or truncated to a length of 400. The chunk size was set to 20 and we read one chunk at each time step. In the proposed model, they

did not specify the maximum number of reread/skip times, which may result in the agent keeping reading until the last chunk. In another closely related paper (Wei Yu et al. (2017)), they proposed a model which only applied skimming and they set the maximum number of skimming to five. Therefore, inspired by their approach, we set the maximum number of reread/skim time to five as well. In order to accumulate the history information from the first chunk to the current chunk, we recorded the hidden state of the LSTM from the last chunk and took it as the initial hidden state for the next chunk.

2.2.2 REWARDS FUNCTION

Our chosen paper took the normalised FLOPs (floating point operations) into consideration in the reward function as shown below:

$$r_j = \begin{cases} -\mathcal{L}(\hat{y}, y) - \alpha \mathcal{F}_t & \text{if } j = t \text{ is the final time step} \\ -\alpha \mathcal{F} & \text{otherwise} \end{cases}$$

In our implementation, we wrote a script to compute the FLOPs of most models beforehand with random input. For the LSTM network, the FLOPs was calculated by hand using the specific input size for the IMDB dataset. By adding the LSTM FLOPs with the cost of the convolutional layer, we had the FLOPs of the recurrent encoder module. The normalised FLOPs at each time step was computed in the following way: collect the FLOPs needed at all steps when processing one review and normalise them by subtracting the value list from its mean value and divide them by the standard deviation. The computational costs of models were taken as constants in training.

The alpha parameter serves as a tradeoff between accuracy and efficiency as described in the chosen paper because as the alpha increases, the penalty at each time step increases as well, causing the agent to read fewer chunks in each sample. However this is not the case in our experiments, which will be discussed in 3.3.

The reward at the last time step included a function that measured the loss between the true label and the predicted label. The specific loss function used was not mentioned in the paper. We chose the cross entropy loss in our experiments.

In a closely related paper (Wei Yu et al. (2017)), they had a different design of the reward function described as below and our chosen paper mentioned that their proposed model had a better performance due to a better reward design. Therefore, we implemented the different reward function to see how the performance differed:

$$R = \begin{cases} 1 & \text{if prediction correct;} \\ -1 & \text{otherwise} \end{cases}$$

If the predicted label is the same as the true label at the final time step, there will be a reward of 1 for that sample. Otherwise the reward is -1. There is no reward at other time steps.

2.2.3 REINFORCE WITH BASELINE

For the update timing in the policy gradient, we followed our choson paper which updated once after each data was processed instead of updating the gradient at each time step like the original policy gradient algorithm. The gradient of the objective in our chosen paper was defined as follows: $\widetilde{\nabla_\theta J}$

$$= \nabla_\theta \left[\log \pi_S \left(1 | h_t\right) + \log \pi_C \left(\hat{y} | h_t, 1\right) + \sum_{j=1}^{t-1} \left(\log \pi_S \left(0 | h_j\right) + \log \pi_N \left(k_j | h_j, 0\right) \right) \right] \sum_{j=1}^{t} \gamma^{j-1} r_j$$

The reward function defined in our chosen paper included a loss function which should be updated and minimised as well. In our experiments, we found that if we combined the gradients of the loss with the gradients needed for the policy gradient algorithm and updated them together just as the above formula shows, the performance was worse than the case when we only used the scalar value of the loss when it was applied in the reward function and optimised the gradients of the loss function separately. Our chosen paper mentioned that they implemented the policy gradient algorithm same as (Wei Yu et al. (2017)) which used a baseline value network and had a different objective function which we followed in our experiments: $J\left(\theta_m, \theta_a, \theta_b\right) = J_1\left(\theta_m\right) - J_2\left(\theta_a\right) +$

$\sum_{s=1}^{S}\sum_{i=1}^{N}(R^s - b_i^s)^2$ In our implementation, θ_m and θ_a referred to the parameters of the cross entropy loss and the policy module respectively. And θ_b referred to the parameters of the baseline value network as mentioned in (Wei Yu et al. (2017)).

Like what we discussed above in 2.2.2, we applied two different rewards settings. So the R in the above function was either 1/-1 or the discounted rewards $\sum_{j=1}^{t}\gamma^{j-1}r_j$.

3 EXPERIMENTS

In general, we implemented the proposed model along with two baseline models(i.e. whole reading and early stopping). Due to different reward definitions, the proposed model and the early stopping model had two variations. One is using $\{-1, 1\}$ as the agent's reward given its predictions. The other is using the sum of discounted rewards that took the FLOPs and loss function into consideration. The script used to compute FLOPs was modified from the code written by Fish (2016). The FLOPs of the models used are shown in Table 1. The FLOPs of CNN and LSTM were measured by a whole sample in the whole reading model whereas they were measured by a chunk of 20 words in other cases.

Table 1: Computation cost measured in FLOPs

Cost	Policy C	Policy S	Policy N	CNN	LSTM
Skim, reread and early stopping & Early stopping	16,770	50,050	50,310	1,024,000	286,720
Whole reading	16,770	-	-	25,344,000	6,881,280

3.1 IMPLEMENTATION DETAILS

PyTorch was used as the main library. Policy s, c, n, the baseline value network and the recurrent encoder were set to have the dropout probability of 0.1. Each hidden linear layer was followed by a rectified linear unit as the activation function. To explore the trade-off between efficiency and accuracy as mentioned in our chosen paper, we tested our models with a range of different $alpha$ from 0.2 to 0.8 with step size at 0.2. The IMDB dataset was split into 20,000 training, 5,000 validation and 25,000 test samples. The numbers of layers in each network were the same as those in the chosen paper.

We used GloVe as the word embeddings and they were updated during training. For each model, the number of training epochs was set to 10. The average FLOPs per data in the testing set was used as the computational metric for each model.

3.2 RESULTS

All the accuracies and average FLOPs with different values of α are shown in Table 2. The model names that have FLOPs in them are the ones that considered FLOPs in the reward definition. Others are the ones used the reward definition from Wei Yu et al. (2017), except that the whole reading model did not use reinforcement learning algorithm.

As the result shows, all the models that considered FLOPs in the reward achieved lower accuracy than the ones simply using $\{-1, 1\}$ as the reward for each data, which was opposite to what was claimed in the chosen paper. And also both early stopping model with FLOPs and skim, rearead and early stopping with FLOPs failed to show the trade off between efficiency and accuracy. In our experiments, the whole reading model achieved the highest accuracy on the IMDB dataset.

3.3 DISCUSSIONS AND ANALYSIS

In Table 2 we can see that although the whole reading model achieved the highest accuracy of 0.85, it had the biggest computational cost because each time it read a whole sample of 400 words instead of a chunk. In comparison, the model of skimming, rereading and early stopping with the reward of $\{-1, 1\}$ achieved the accuracy of 0.70 with far less computational cost.

Table 2: Results form different models

Model	α	Average FLOPs	Accuracy
Skim, reread and early stopping with FLOPs	0.2	7,072,170	0.6578
	0.4	7,072,170	0.6636
	0.6	7,072,170	0.6122
	0.8	7,072,167	0.6632
Skim, reread and early stopping	$None$	2,549,349	0.7021
Early stopping with FLOPs	0.2	27,232,170	0.5114
	0.4	27,232,170	0.5101
	0.6	27,232,170	0.5106
	0.8	27,232,170	0.5119
Early stopping	$None$	1,377,540	0.6572
Whole reading	$None$	32,242,049	0.8566

In terms of the early stopping model with FLOPs, the performance is not better than a random guess. Although we tuned the α, the accuracy stayed at around 0.51. The performance of skimming, rereading, and early stopping with FLOPs is better, but it is still much lower than the expected accuracy as mentioned in the chosen paper.

It is worth noting that we didn't observe the trade-off between efficiency and accuracy by tuning α when we considered computational cost in the reward function. The expected result should be that when we increase the value of α, the agent will be punished more, thus resulting in it reading fewer chunks and the agent will explore more when α is reduced. In our experiments, an interesting observation was that the agent kept rereading the first chunk in the model of skim, reread and early stopping with FLOPs and went as far as it could in the model of early stopping with FLOPs, given different values of α. A potential explanation is that since that the agent's goal was maximising the accumulative rewards, it realised that it could received the highest rewards when it kept rereading the first chunk or reading as more as it could. Therefore we think that the reward function was not defined properly enough or there are some missing implementation details in the paper.

Apart from the low accuracies and absent trade-off when tuning α, with all respect for Yu et al. (2018), we doubted whether the actor-critic algorithm was implemented or not because there was no details in the paper. And in the replies of the ICLR review we found that the authors gave ambiguous responses to the actor-critic implementation and explanation when it was questioned by one official reviewer. Another reason which also led to our failure could be that we did not use batch operations to deal with multiple data in the reinforcement learning process. Instead, we processed one data at each time and used batch update to update policies.

4 Conclusion

Given limited information in the paper, we failed to achieve the high accuracy and the trade-off between accuracy and efficiency, but we tried other approaches in a closely related paper. And the performance was claimed to be better with advantage actor-critic but little information was found.

Code is implemented by ourselves, otherwise, cited, and is available at: COMP6248-Reproducability-Challenge/Fast-And-Accurate-Text-Classification-Reproduction.

References

Big Fish. Pytorch tools, 2016. URL `https://zhuanlan.zhihu.com/p/33992733`.

Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. pp. 1880–1890, 01 2017. doi: 10.18653/v1/P17-1172.

Keyi Yu, Yang Liu, Alexander G. Schwing, and Jian Peng. Fast and accurate text classification: Skimming, rereading and early stopping, 2018. URL `https://openreview.net/forum?id=ryZ8sz-Ab`.