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

ICLR REPRODUCIBILITY CHALLENGE
COMP6248 DEEP LEARNING
UNIVERSITY OF SOUTHAMPTON

Dominika Woszczyk
University of Southampton
dcw1g18@soton.ac.uk

Melissa Córdova
University of Southampton
mcm1n18@soton.ac.uk

Ankur Sharma
University of Southampton
as3g15@soton.ac.uk

ABSTRACT

This work aimed to review and replicate the paper "Fast and accurate text classification: Skimming, rereading and early stopping" submitted to the ICLR 2018 Conference (Yu et al. (2018)) where it describes an agent which evaluates whether to classify, skip text, or re-read by using an RNN module and a policy module to form decisions. We re-implement from scratch the above-mentioned model, as well as an simplified version of that model[1] and compare the results on the IMDB movie dataset for sentiment analysis at a word-level. We did not manage to reproduce the paper performance but successfully replicated the simpler model.

1 INTRODUCTION

The paper "Fast and accurate text classification: Skimming, rereading and early stopping" argues that it is not always necessary to read the entire input in order to classify a review. Hence, it implements an approach of fast reading for text classification by building an intelligent recurrent agent which evaluates whether to make a prediction, to skip some text, or to re-read part of the sentence. The agent uses an RNN module that encodes information from past and current tokens and applies a policy module to form decisions. In this report, we attempt to reproduce their first experiment, evaluating the accuracy of the model for the sentiment analysis on the IMDB dataset.

The authors did not make available any code, therefore we re-implemented their approach from scratch. The paper implements the advantage actor-critic to train their policy gradient, which was implemented as well. For additional comparison of the performance, we implemented an earlier version of this paper Yu et al. (2017) with a simpler model and simple LSTM.

1.1 DATASET

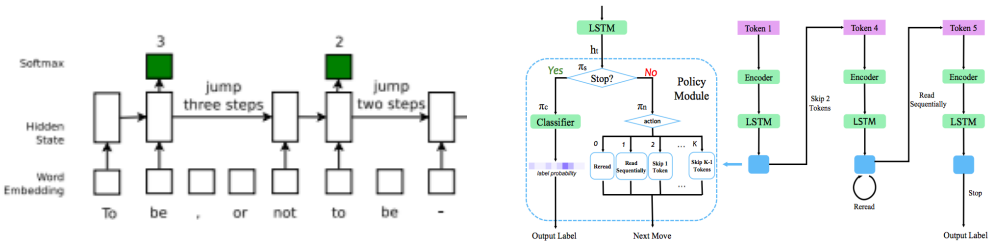
The Large Movie Review Dataset (Maas et al. (2011)) is a 2011 dataset based on IMDB movie reviews for binary sentiment classification. The training set was divided into 20000 training samples and 5000 validation samples.

2 MODEL

The architecture of the model follows an end-to-end approach and given a document outputs the predicted sentiment. It is composed of two main modules: the encoder (RNN) and the Policy, as shown in Figure 1.b. We implemented the networks in Python using the Pytorch library.

The recurrent neural nets (RNN) takes an input x at step size t and outputs a feature h_t containing the information of the previous feature and the current token x. The tokens were word level. The RNN is a Long Short Term Memory (LSTM) network in this case.

[1] https://github.com/COMP6248-Reproducability-Challenge/Differentiables_FastAccurateTextClassification



(a) Skip-model

(b) Fast-model

Figure 1: Models

Feature h_t is fed into the policy module which decides what happens next: rereading, skipping words or stopping and classifying.

Following the model guidelines, we implemented the model structure as follows:

```
RNN_Module:
    embedding = Embedding(input_dim, embedding_dim)
    lstm_cell = LSTMCell(embedding_dim, hidden_dim, bias=True)
```

Taking the feature h_t as input, policy module outputs a probability distribution π_t over the possibles actions: stop signal or step size k (move to the $(t + k)$-th word). The step size k $\in\{0,1...N\}$. For k=0, the token will be re-read, for k=1 the module moves to the next word and for k¿1, it skips k-1 words.

π_S is the Bernoulli distribution used to make the binary decision of whether to stop and classify or continue reading. π_C is a classifier which takes in the latent representation h_t and outputs a conditional multinomial distribution over the classification classes π_N outputs a conditional multinomial distribution over the step sizes k

The policy module was implemented as follows:

```
Policy_Module:
    pi_s_1 = Linear(hidden_dim, 128)
    pi_s_2 = Linear(128, 128)
    pi_s_3 = Linear(128, 2)
    pi_C = Linear(hidden_dim, output_dim)
    pi_N_1 = Linear(hidden_dim, 128)
    pi_N_2 = Linear(128, 128)
    pi_N_3 = Linear(128, 4)
```

The skip-model On Figure 1.a (Yu et al. (2017)) was similar to the fast-model described above but did not have the stop classifier separately, and only had one linear layer only for both classification and jumping decision. Also, it was trained using vanilla reinforce algorithm, not actor-critic algorithm. Both of these models and a simple baseline LSTM were implemented.

2.1 TRAINING

In order to optimise the policy allow for back-propagation, the Reinforce policy gradient (Williams (1992)) using Monte Carlo roll-outs with advantage actor-critic algorithm was implemented. (Konda & Tsitsiklis (2000)).

The authors do not specify any details for their advantage actor-critic implementation.

In the REINFORCE algorithm, the network gives us the policy value and computes the reward with equation 1. After each batch, it computes the loss function with equation 2. Finally, backpropagation is done.

The reward for the last output action is computed as follows, where L is a loss function that measures the accuracy between the predicted label and the true label.

$$r_j = \begin{cases} -L(\hat{y}, y) - \alpha F_t, & \text{if } j = t \text{ is the final time step} \\ -\alpha F, & \text{otherwise} \end{cases} \tag{1}$$

where α is a trade-off parameter between accuracy and efficiency.

$$\widehat{\nabla_\Theta J} = \hat{\nabla}[\log \Pi_S(1 \mid h_t) + \log \Pi_C(\hat{y} \mid h_t, 1) + \sum_{j=1}^{t-1} (\log \Pi_S(0 \mid h_j) + \log \Pi_N(k_j \mid h_j, 0))] \sum_{j=1}^{t} \gamma^{j-1} r_j \tag{2}$$

where $\gamma \in (0,1)$ is a discount factor.

2.2 Experiment settings

The paper did not specify any of the hyperparameters, hence, in the attempt to reproduce the results we had to set our values as follows in Table 1.

Table 1: Parameters for reproducibility

Parameter	Value
Optimizer	Adam with lr= 0.01
Gamma	0.99
alpha	1e-7
Max skip size k	3
FLOPS constant	10000
Batch size	50

The only parameter specified in the paper was the maximum skip size k, and a learning rate of 0.001 for the Adam optimiser, which had to be changed to 0.01 for the model to train.

As $L(\hat{y}, y)$ was not specified in the paper, we decided to set $L(\hat{y}, y) = 1$ when $\hat{y} = y$ and -1 otherwise.

For fairness of results, all models were trained on the same machine, using CPU only and by training one datapoint at a time instead of training in batches.

3 Results

We compared the accuracy after certain epochs to the paper's accuracies, as the FLOPS count is proportional to the compute time taken given the code is executed on the same machine.

In order to compare performances, we re-implemented an earlier version of the selected paper where the authors describe a simpler model (no rereading) for text skimming. This paper had a lot of implementation details, unlike, the Yu et al. (2018). We will call this model Skip-model and the model from the paper Fast-model. Additionally, we compare the results to a single-layer LSTM that we refer to as simple-lstm-model. This was used as a baseline model.

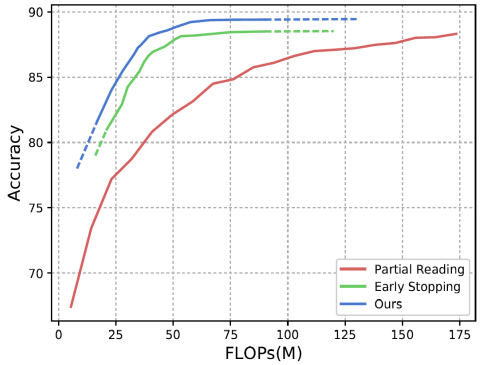
On Figure 2, one can see that our implementation of fast-model did not achieve similar results due to little information given about the hyperparameters. However, the skip model achieved an accuracy of 80% in 500 minutes , for which all the implementation details were described accurately in this paper.

4 Conclusion and Further Directions

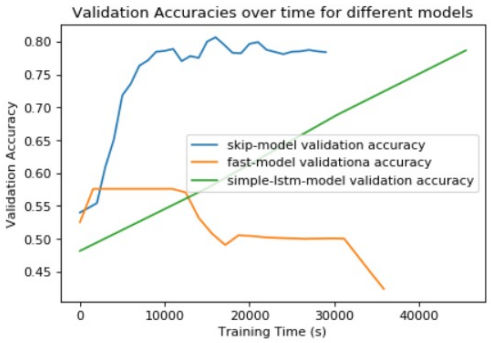
Due to the sparse implementation details, we did not manage to reproduce the results stated in the paper. However, the similar skim-model detailed in Yu et al. (2017) did perform better than a simple LSTM as it reached an accuracy of 80% in less time, confirming the ability of faster classification. It is possible that with further investigation of the hyperparameters space we might be

able to reproduce the results. However we cannot guarantee as we did not manage to achieve similar results. Therefore, we conclude that the results in Yu et al. (2018) are not reproducible easily but those in Yu et al. (2017) are reproducible.

Regarding further directions, for the purpose of text classification, most RNN models have been now replaced by Transformer and Attention models. Further work would include exploring skimming methods applied to those type of models.



(a) Paper results



(b) Validation accuracies over time for different models

Figure 2: Results

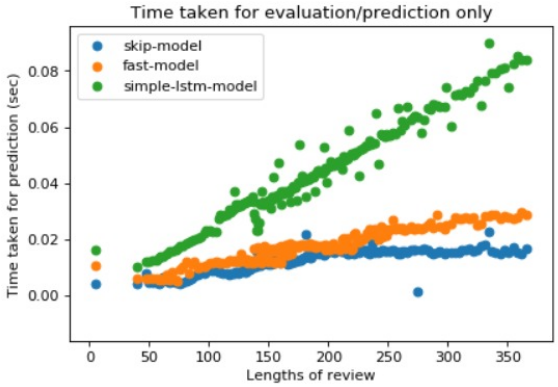


Figure 3: Time taken for evaluation/prediction only

REFERENCES

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150. Association for Computational Linguistics, 2011.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Adams Wei Yu, Hongrae Lee, and Quoc V Le. Learning to skim text. *arXiv preprint arXiv:1704.06877*, 2017.

Keyi Yu, Yang Liu, Alexander G Schwing, and Jian Peng. Fast and accurate text classification: Skimming, rereading and early stopping. 2018.