声明：我已知悉学校对于考试纪律的严肃规定，将秉持诚实守信宗旨，严守考试纪律，不作弊，不剽窃；若有违反学校考试纪律的行为，自愿接受学校严肃处理。

2018-2019 学年第二学期 COMP130137.01 《模式识别与机器学习》课程项目 基于 SkipRNN 的 RNN 速度改进

学号：16307130316 , 姓名：蒋艳琪，贡献: 100%，签名：

Abstract

Recurrent Neural Networks (RNNs) show excellent performance in sequence modeling tasks because it inherently utilizes the whole input sequences and takes their dependencies into consideration. However, it often suffers from slow training and heavy computational burden. The Skip RNN model is proposed which subsamples from the input sequences and learns to skip unnecessary state updates, thus reducing computation cost and unstableness in training process. The model is evaluated on two classic RNN tasks and shows that it can achieve similar, or even improved performance while requiring fewer state updates and hardware resources.

1 Introduction

Recurrent Neural Networks (RNNs), including its gated variants such as Long Short-Term Memory [11] and Gated Recurrent Unit [9], have become the standard model architecture as deep learning approaches to solve sequence modeling tasks.

RNN repeatedly applies a function with trainable parameters to a hidden state. By recurrently updating the hidden states and doing backward propagation, RNN inherently uses information from the whole input sequence; by stacking RNN layers together, its representational power and accuracy can be further increased. Therefore, it has widespread use in natural language processing domain, especially in text classification [14] and character-level language modeling [19] tasks. It also acts as basic building block in many other tasks associated with temporal dimension.

The major weakness of RNN, if any, is that the computation of features in different time steps cannot occur in parallel, which means a lot of challenges a machine learning practitioner may face: slower convergence, memory leakage when dealing with long sequences, etc.

However, not all input tokens are equally important in many language processing tasks. For example, in question answering tasks, only the keywords in the question need special attention. As a result, many sequence shortening techniques have been researched to accelerate RNNs. By learning which samples (elements in the input sequence) need to be used and paid more attention to, RNNs can perform better in tasks involving very long sequences such as document classification.

Skip RNN [8] proposed a novel modification for existing RNN architectures that allows them to skip state updates, decreasing the number of sequential operations performed without any manually defined supervision, thus reducing the total number of floating operations. The network can also be encouraged to perform fewer state updates by adding a penalization term during training, so we can train models under different computation budgets. The modification can be integrated with any RNN(LSTM and GRU), and is orthogonal to many other performance boosting techniques like recurrent batch normalization [10], so they can be easily integrated without any collision.

The main contribution of this project is listed as follows:

1. Implemented the basic and skipped version of RNN cells (LSTM and GRU), which provides the same input and output interfaces as standard RNN does and can be easily used to replace RNNs in existing models.
2. Experimented on two traditional RNN tasks to compare between the basic and skipped version.
3. Customized the loss functions and metrics in FastNLP [2] to suit the network.
4. Visualized the training process and results with fitlog [3].

2 Related work

Fast neural networks As neural networks become increasingly widespread in real-world applications, making neural networks faster has recieved much attention, especially for those targeted at sequential data. Among them, accelerating RNNs is a heated research topic. The efforts are mainly paid to speeding up RNN inference or parallelizing the computation.

As for accelerating inference in RNNs, LSTM-Jump [18] **skips** some input tokens by augmenting the LSTM cell with a classification layer that will decide how many steps to jump between RNN updates, which is quite similar to Skip-RNN. However, the model has many hyperparameters, such as a reasonable reward signal to be used in REINFORCE [17], the number of tokens read between jumps, the maximum jump distance, etc. These hyperparameters are likely to result in a fixed set of subsequences that the model can sample from, instead of a sampling scheme with more freedom and learning ability. Skim-RNN [16] is slightly different in that, instead of totally skipping parts of the input sequence, it can make a fast decision whether to **'full read'** or **'skim'** the input token and switch between a smaller RNN which only updates a fraction of the hidden state and a default RNN cell. Variable Computation in RNN(VCRNN) [12] is also concerned with dynamically controlling the number of units to update at each time step, but it has to make a d-way decision (where d is the hidden state size) rather than a binary decision as Skip-RNN does.

As for parallelizing the computation, QRNN [7] introduces a CNN-like layer which applies in parallel across time steps and a minimalist recurrent pooling function that applies in parallel across channels, so it can relax the temporal dependency between consecutive steps. Compared to the models focusing on reducing GPU time (maximizing parellelization), Skip-RNN can also reduce CPU time (minimizing Flops).

Attention models Selecting part of the inputs shares something in common with the hard attention mechanisms in image processing tasks such as object detection [4]. Similar research has been conducted for video analysis tasks, which requires an additional temporal dimension for event recognition [6] and is more relevant to sequence modeling tasks.

Attention mechanism is also natural in human cognition process. Just as human speed reading, it will not affect our overall understanding of the text if we come across some unimportant words and choose to skip them or spend less time processing them.

Skip-RNN can be viewed as generating a hard temporal attention mask given previously seen samples, giving more attention to possibly important input tokens. Moreover, the subsampling techniques it uses is differentiable so it can be trained with simple back propagation, leading to lower computational complexity.

3 Model description

3.1 Model architecture

An RNN takes an input sequence $x = (x_1, ..., x_T)$ and generates a state sequence $s = (s_1, ..., s_T)$ by iteratively applying a parametric state transition model S from $t = 1$ to T:

$$s_t = S(s_{t-1}, x_t)$$

The network can be augmented with a binary state update gate, $u_t \in \{0, 1\}$, which determines whether the state of the RNN will be updated ($u_t = 1$) or copied from the previous time step ($u_t = 0$). The architecture of the model can be described as follow:

$$u_t = f_{binarize}(\widetilde{u}_t)$$
$$s_t = u_t \cdot S(s_{t-1}, x_t) + (1 - u_t) \cdot s_{t-1}$$
$$\Delta\widetilde{u}_t = \sigma(W_p s_t + b_p)$$
$$\widetilde{u}_{t+1} = u_t \cdot \Delta\widetilde{u}_t + (1 - u_t) \cdot (\widetilde{u}_t + min(\Delta\widetilde{u}_t, 1 - \widetilde{u}_t))$$

W_p is a weights vector, b_p is a scalar bias, σ is the sigmoid function and $f_{binarize} : [0, 1] \rightarrow \{0, 1\}$ binarizes the input value.

The model utilizes the observation that the probability of a state update when encountered with a new input will increase if several samples are consecutively skipped. At every time step t, the probability $\widetilde{u}_t \in [0, 1]$ of performing a state update at t is used to compute u_t by a deterministic step function $f_{binarize} = round(\widetilde{u}_t)$. As can be seen, if $u_t = 1$, then s_t will be updated, and the pre-activation of the state update gate for the following time step, $\widetilde{u}_{t+1}$, is flushed and set to $\Delta\widetilde{u}_t$. On the other hand, if $u_t = 0$, then s_t will simply copy the previous state, and $\widetilde{u}_{t+1}$ is incremented by $\Delta\widetilde{u}_t$.

Besides the computational savings of s_t, there is no matrix computation at all whenever $u_t = 0$ because $\Delta\widetilde{u}_t = \sigma(W_p s_t + b_p) = \sigma(W_p s_{t-1} + b_p) = \Delta\widetilde{u}_{t-1}$.

3.2 Back propagation

The whole model is differentiable except for $f_{binarize}$, which outputs binary values. There are several optimizing functions involving discrete variables, including REINFORCE [17]. However, REINFORCE needs additional reward signal, which is hard to define. The straight-through estimator [5] is selected here, which approximates the step function by the identity when computing gradients to be used in the backward pass:

$$\frac{\partial f_{binarize}(x)}{\partial x} = 1$$

By using the straight-through estimator as the backward pass for $f_{binarize}$, all the model parameters can be trained to minimize the target loss function with standard back propagation.

3.3 Computation budget limit

The Skip RNN is able to learn when to update or copy the state without explicit information about which samples are useful. However, there is a trade-off between performance and number of processed samples. For example, one may sacrifice some accuracy in order to run the model faster on a computer with limited computation resources. This model can also be encouraged to perform fewer state updates by adding a penalization term named L_{budget}:

$$L_{budget} = \lambda \cdot \sum_{t=1}^{T} u_t$$

where L_{budget} is the cost associated with a single sequence, λ is the cost per sample and T is the sequence length. It is similar to weight decay regularization, and the network is encouraged to converge toward a state where fewer states updates are required.

3.4 Model implementation

This project mainly implements the skipped LSTM and GRU cells, multi-layer RNN also supported. For the purpose of comparison with the standard RNN cells, similar basic LSTM and GRU cells are also implemented so the results will not be affected by some underlying optimization mechanisms of Pytorch-based RNN. The file tree is as below:

- SkipRNNCells.py
  - SkipLSTMCell

  - MultiSkipLSTMCell
  - SkipGRUCell
  - MultiSkipGRUCell
- BasicRNNCells.py
  - BasicLSTMCells
  - BasicGRUCells

BasicRNNNetworks.py and SkipRNNNetwoks.py provides encapsulation for the cells and defines the initialization and forward functions for multi-layer model, so the cells can be directly used to replace the standard RNN part in other existing models.

In order to support computation budget limit, I defined a new loss function called SkipBudgetLoss which inherits the LossBase in FastNLP and combines CrossEntrophy loss with a penalization term on state updates as stated above. I also added a new metric called UsedStepsMetric which inherits the MetricBase and yields the fraction of used samples in an input sequence.

4 Experiments

The influences of adding this state skipping mechanism to two common RNN architectures, LSTM and GRU, are investigated in the following section. In addition to the accuracy metric, the fraction of RNN state updates (the number of 'word' in the input sequence used by the network compared with the total sequence length) and the total training time are also reported for comparison.

Data loading and training are performed with the FastNLP [2] framework. Fitlog [3] is used for auto commit and visualization. Bias b_p is initialized to 1 so that all samples are used at the beginning of training. The initial hidden state s_0 is learned during training, while $\widetilde{u}_0$ is set to 1 in order to force the first state update at t=1.

Experiments are implemented with Pytorch and run on several different NVIDIA GTX1080 GPUs.

4.1 Text classification task

First I tested the performance of Skip-RNN on the text classification task in assignment 4. Skip-RNN is rather easy to use. We only need to replace the LSTM or GRU cells with its skipped version, and the remaining part of the model does not need any modification.

Eight categories from the 20 newsgroups text dataset [1] are selected. The RNN models are single-layer, trained with 128 hidden units, optimized by Adam [13] optimizer with learning rate of 10^{-3} on batches of 32 to minimize the SkipBudgetLoss(CrossEntrophy loss, plus the penalization term if applicable, $\lambda = 10^{-6}$). The results are shown in Figure 1,2,3.

As can be seen from Figure 1, Skip-LSTM achieves higher accuracy on both training and test datasets, performing roughly only half of state updates and costing much less training time. According to the learning curve, Skip-LSTM converges faster and shows lower variance. These phenomenon prove the hypothesis that learning to skip updates enables Skip-RNNs to work on shorter subsequences, capture long term dependencies and simplifies the optimization process.

hyper					metric					
					AccuracyMetric	test		UsedStepsMetric		
model...	task	batch_...	learnin...	time	acc	AccuracyMetric acc	UsedStepsMetric updated_steps_f...	updated_steps_f...	step	epoch
	text_cl									
skip_lstm	text_classification	32	0.001	12599.555839	0.796656	0.796656	0.499303	0.493947	4002	29
basic_lstm	text_classification	32	0.001	15839.375855	0.776186	0.775503	1	1	4002	29

Figure 1: Performance summary of 20 Newsgroups task

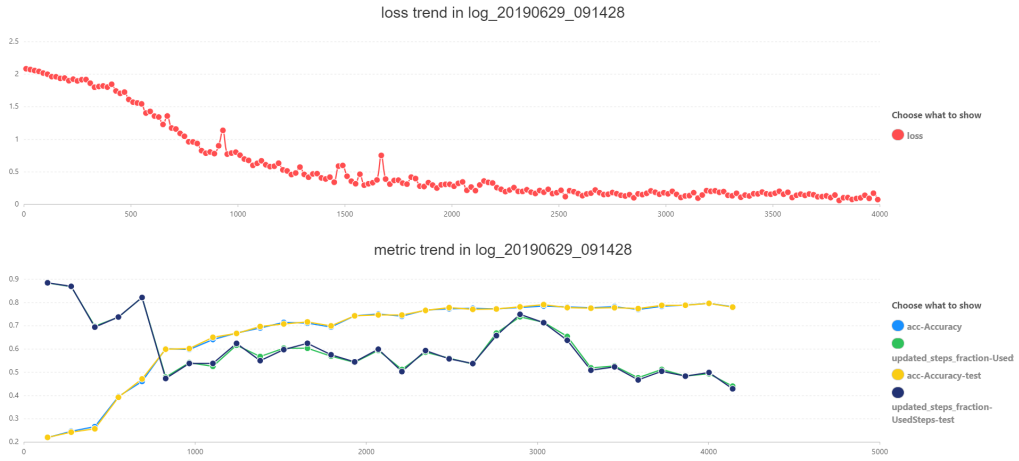

Figure 2: Skip-LSTM model learning curve

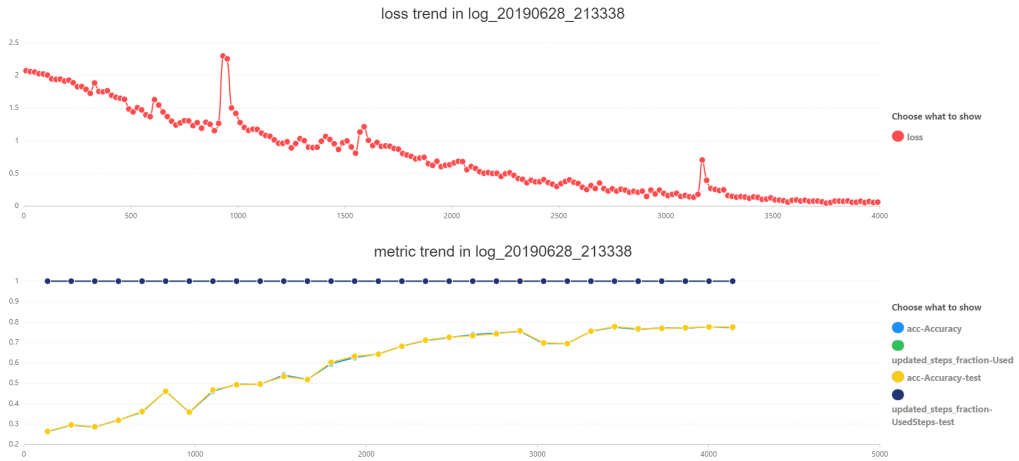

Figure 3: Basic-LSTM model learning curve

4.2 MNIST classification from a sequence of pixels

The MNIST handwritten digits classification benchmark [15] is traditionally considered a Convolutional Nerual Networks(CNNs) task because CNNs can efficiently exploit the spatial dependencies through weight sharing. Recently, it has also become a challenging task for RNN to solve. The 28*28 image can be seen as an input sequence of length 28, each with 28 dimensions. The dataset is fetched from torchvision and splitted into training (55000 samples), validation (5000 samples) and testing (10000 samples) datasets. After processing all pixels with an RNN with 128 hidden units, the average of the hidden states is fed into a linear classifier predicting the digit class.

hyper					metric					
					AccuracyMetric	test		UsedStepsMetric		
model…	task	batch_…	learnin…	time	acc	AccuracyMetric acc	UsedStepsMetric updated_steps_f…	updated_steps_f…	step	epoch
	mnist									
skip_gru	mnist	128	0.0001	3473.623354	0.9482	0.9431	0.938358	0.936628	4300	10
basic_gru	mnist	128	0.0001	2603.71771	0.959	0.955	1	1	4300	10

Figure 4: Performance summary of MNIST task

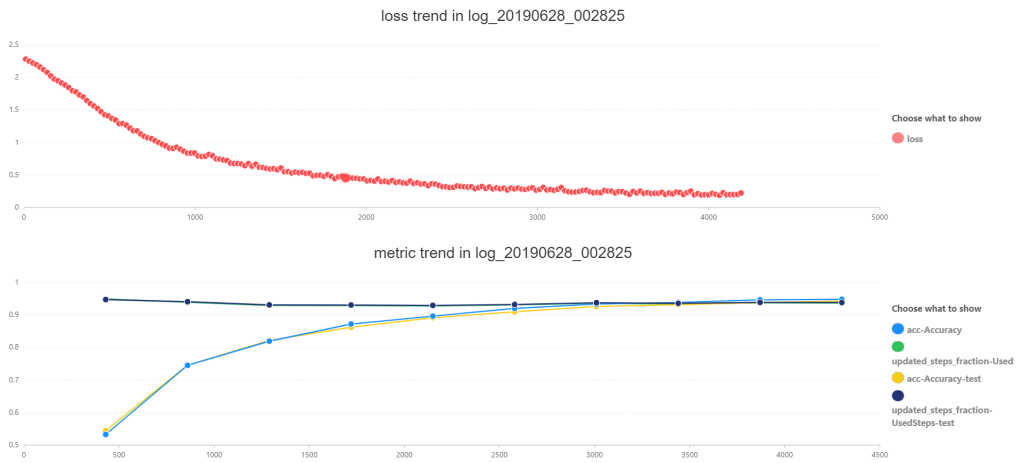

Figure 5: Skip-GRU model learning curve

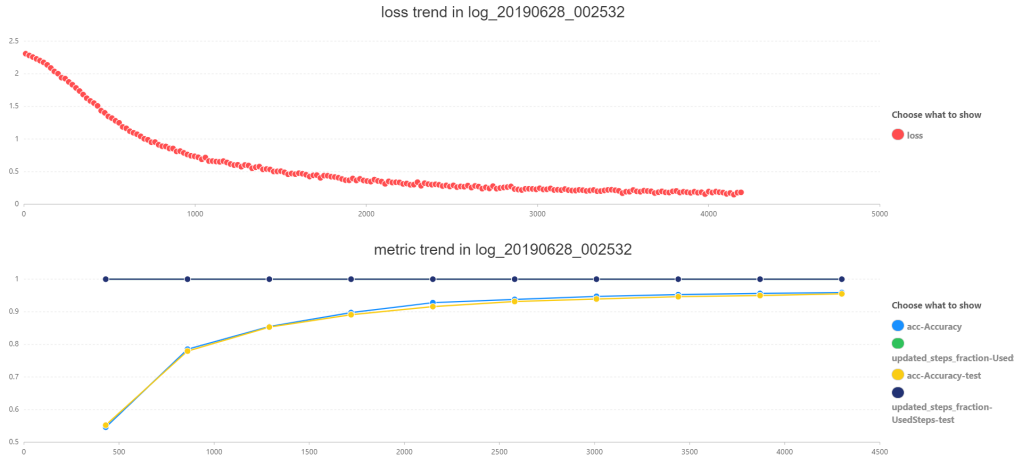

Figure 6: Basic-GRU model learning curve

Figure 4,5,6 summarize classification results after 20 epochs of training. As is shown, the average state updates performed by Skip-GRU are fewer than the basic GRU model, yet they achieve similar accuracy. It is notable that the training time of Skip-GRU is longer, which may be explained by the fact that the sequence length (28) is not as long as the former task, thus the advantage of skipping state updates is not that trivial; also, fewer Flops do not necessarily mean faster training.

5 Conclusion

Skip RNN can act as an extension to existing recurrent architectures. By adaptively determining whether the state needs to be updated, Skip RNN can be trained faster and more stably owing to its shorter back propagation sequences for long inputs or complex models. All the parameters can be trained with gradient descent, resulting in a simple optimization mechanism. Experiments have shown that its performance is promising on a series of sequence modeling tasks. Moreover, the proposed computation budget control by adding a penalization term is especially valuable under those circumstances where hardware capability is limited.

Acknowledgements

I acknowledge the support of GPUs, and the FastNLP framework.

References

[1] http://qwone.com/ jason/20Newsgroups/, 2008. [Online; accessed 29-June-2019].

[2] https://github.com/fastnlp/fastNLP, 2019. [Online; accessed 29-June-2019].

[3] https://github.com/fastnlp/fitlog, 2019. [Online; accessed 29-June-2019].

[4] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[5] Yoshua Bengio. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pages 1–37. Springer, 2013.

[6] Subhabrata Bhattacharya, Felix X Yu, and Shih-Fu Chang. Minimally needed evidence for complex event recognition in unconstrained videos. In *Proceedings of International Conference on Multimedia Retrieval*, page 105. ACM, 2014.

[7] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

[8] Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[10] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[12] Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural speed reading via skim-rnn. *arXiv preprint arXiv:1711.02085*, 2017.

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

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

[19] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.