TrimZero: A Torch Recurrent Module for Efficient Natural Language Processing

Jin-Hwa Kim[1], Jeonghee Kim[2], Jung-Woo Ha[2], and Byoung-Tak Zhang[13]
[1]Interdisciplinary Program in Cognitive Science, Seoul National University
[2]NAVER LABS, NAVER Corporation
[3]School of Computer Science and Engineering, Seoul National University

Abstract

Deep learning framework supported by CUDA parallel computing platform boosts advances of studies on machine learning. The advantage of parallel processing largely comes from an efficiency of matrix-matrix multiplication using many CUDA-enabled graphics processing units (GPU). Therefore, for recurrent neural networks (RNNs), the usage of a zero-filled matrix representing variable lengths of sentences for a learning batch is forced for that reason, however, it is still true that these zeros are wasting computational resources. We propose an efficient algorithm which is trimming off zeros in the batch for RNNs providing the same result. The benchmark results validate our method with approximately 25% faster learning. Empirically, a natural language task confirms our results.

Keywords : torch, rnn, trimzero, deep learning

1. Introduction

Recurrent neural networks (RNNs)[1, 2] have been playing an important role in natural language processing, with an advance in parallel computing technology. In the center of this advance, CUDA GPU programming platform invented by NVIDIA facilitates a very fast computation of linear algebra, e.g. matrix-matrix multiplication.

To utilize this advantage, one is inevitably forced to use a zero-filled matrix to represent variable length sentences as an input to the language model. The computational cost involving zeros is not diminished, requiring the same effort.

Many researchers and developers struggle to avoid zero computations in natural language processing using RNNs. However, they implemented it in an *ad hoc* way, so, it is hard to generalize and apply to other projects. In Section 2, we summarize the other attempts permiting a bit of performance loss or computational cost.

2. Related Works

Xu and his collegues (2015) introduce the *homogeneous data* technique[3] to reduce the zero computation problem. In training phase, they made a mini-batch of the sampled length of sentences after randomly sampling a sentence length, instead of a mini-batch of the maximum length of sentences. Though it gives a little distortion to the random sampling, they reported that it greatly shortens the convergence phase. This method is also used in training *Skip-Thought Vectors*[4].

Lu et al. (2015) samples in a purely randomized way for a mini-batch. Instead, it sorts the data ordering by length, then, splits the mini-batch to have the same length for each split of the mini-batch. Next, it forwards the splited data chunks multiple times, i.e. the number of unique lengths, for the mini-batch[5]. Therefore, the additional computational cost comes from sorting, spliting and multiple forwarding for a single forward step. Notice that this technique suffers the worst speed degradation when the length variance of sentences are maximized in a mini-batch.

This work was supported by NAVER Corp. and partly by the Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (R0126-15-1072-SW.StarLab, 10044009-HRI.MESSI).

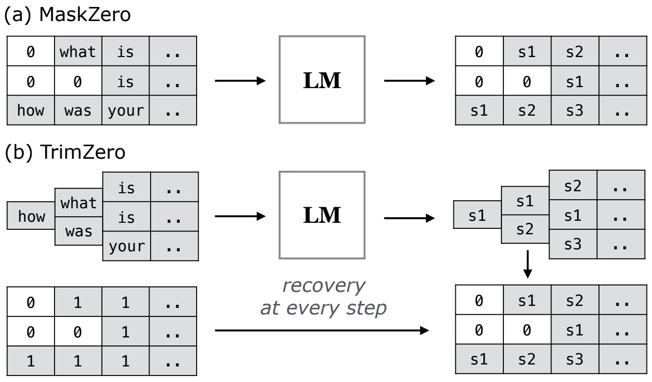

Figure 1: A schematic comparison of *TrimZero* with a naive approach, *MaskZero*. For details, see the text in Section 4.

3. Modularization

Torch *nn*[1] for neural networks provides a modular way of building a model and training it. For a general functionality, *Module* class provides three fundamental methods to override, used by *forward* and *backward*.

3.1 Forward

updateOutput(input) gets the output for a given input, based on the current parameters of the module instance. If the module is sequentially connected to the other module, the output is used for the input to the other module.

3.2 Backward

updateGradInput(input, gradOutput) returns the gradient of the given input, based on the current parameters of the module instance and the given gradient of the output. It takes an advantage of the *chain rule* for calculating a derivative of the composition of multiple functions.

1) `https://github.com/torch/nn`

accGradParameters(input, gradOutput, scale) has a similar functionality to *updateGradInput(input, gradOutput)* except that it computes the gradient of its own parameters. The parameter *scale* is used for the learning rate. The module accumulates the parameter gradients, not replacing the gradients.

4. TrimZero

An implementation of *TrimZero* uses the decorator pattern to encapsulate a module, one of RNNs, which is implemented in *rnn* package[6] extending *nn*. Note that the *LSTM* and *GRU* modules inherit *AbstractRecurrent*, which is a grandchild of *Module*.

Before calling *forward* on the inner module, it trims zero vectors while keeping the mask, which indicates the indexes of non-zero vectors in a mini-batch. After the forwarding, it recovers the original shape inserting zero-vectors back using the mask. The same operations are done for a backward step. This technique is based on the assumption that the computational cost of operations on zero-vectors are greater than indexing and memory referencing. To minimize the operational cost, we intensely constrain manipulating operations. Figure 1 shows a schematic representation of our method comparing with a naive approach, *MaskZero*.

The usage of *TrimZero* is the same with *MaskZero* as follows:

```
tz = nn.TrimZero(module, nInputDim)
```

where `module` is a module to be decorated by *TrimZero*, and `nInputDim` is the number of non-batch dimensions.

The implementation of *TrimZero* is released under BSD 3-Clause License[2].

5. Experiments

To validate our method, whether the *TrimZero* gets the same results and faster, we tested the method using the artificial sentences, whose lengths are uniformly sampled, i.e. $l \sim U(L - x + 1, L)$, where $x \in \{1, 2, 4, ..., L\}$ and $L = 26$ is the maximum length of sentences, and x is an experimental variable, called a zero factor.

As Figure 2 represents, *TrimZero* effectively accelerates the learning speed, providing the same results (not shown here) on the input. The acceleration effect is clearer when the zero factor is larger, which means the range of fluctuation of sentence length is wider, and, more importantly, the mean of lengths is smaller. However, if there is no zero-vector, when $x = 1$, the computational cost does not change or gets worsen.

We do not include the empirical results on the human generated natural language data, generally, whose median of sentence lengths is smaller than a half of the maximum length, have a similar effect of reducing the computational time.

2) `https://github.com/Element-Research/rnn`

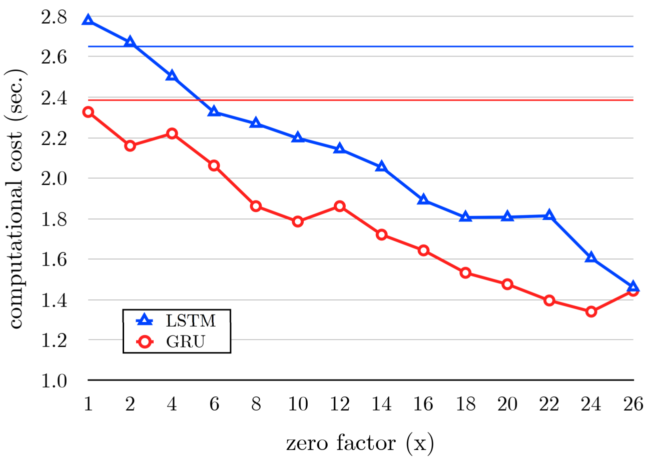

Figure 2: Experimental results varying the zero factor x using *TrimZero* as described in Section 5. The blue horizontal solid line (upper) indicates the computaional cost of *LSTM* using *MaskZero*. The red horizontal solid line (lower) indicates the computaional cost of *GRU* using *MaskZero*.

6. Conclusions

We propose a novel method to reduce the computational cost of forward and backward steps for learning RNNs, which can be generalized in a modular way for Torch *rnn* package. Most cases confirm the superiority of the proposed method, *TrimZero*, over a naive method, *MaskZero*. Therefore, we highly recommend to use *TrimZero* as a method to handle variable-length inputs except that for inputs of the same length.

References

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[3] K. Xu, A. Courville, R. S. Zemel, and Y. Bengio, "Show, Attend and Tell : Neural Image Caption Generation with Visual Attention," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015.

[4] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-Thought Vectors," in *Advances in Neural Information Processing Systems 28*, pp. 3294–3302, 2015.

[5] J. Lu, X. Lin, D. Batra, and D. Parikh, "Deeper LSTM and Normalized CNN Visual Question Answering Model." `https://github.com/VT-vision-lab/VQA_LSTM_CNN`, 2015.

[6] N. Léonard, S. Waghmare, Y. Wang, and J.-H. Kim, "rnn : Recurrent Library for Torch," *arXiv preprint arXiv:1511.07889*, 2015.