On Optimization Algorithms for Recurrent Networks with Long Short-Term Memory

Hieu Pham[*]
Stanford University
hyhieu@stanford.edu

Zihang Dai[*]
Carnegie Mellon University
dzihang@andrew.cmu.edu

Lei Li
Baidu Research
lilei22@baidu.com

Motivation and contribution Recurrent neural networks (RNN) with long short-term memory (LSTM) are recently proposed to model sequences without prior domain knowledge [3, 6]. In these work, the authors empirically observed that RNN-LSTMs trained with vanilla optimization algorithms, such as stochastic gradient descent (SGD) with a simple learning rate annealing schedule, achieves good performances on a wide range of tasks. While this observation makes it easy to train RNN-LSTMs, it remains a question whether more advanced optimization techniques, such as AdaGrad [1] or momentum and its variations [2], can improve their performances.

We attempt to answer this question. We consider two benchmark problems: (1) language modeling on Penn TreeBank and (2) named entity recognition (NER) on OntoNotes (v5.0) [5]. On these tasks, we compare the performances of RNN-LSTMs trained with various optimization algorithms such as SGD, AdaGrad and momentum. We further propose a novel optimization technique that achieves better performance on both tasks. Our preliminary results show that our method is robust against different scales of data, hence requires minimal tuning effort. For language model in particular, it leads to 7% improvement for single model perplexity.

A new optimization algorithm We propose a novel training algorithm for RNN-LSTMs. Suppose we have a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and we want to find a vector $\theta \in \mathbb{R}^d$ to minimize $f(\theta)$. We randomly initialize $\theta_0 \in \mathbb{R}^d$, then at time step $t \geq 1$ we perform the following update

$$\nu_t \longleftarrow \mu_t \nu_{t-1} + \frac{\eta}{\varepsilon + \sqrt{\sum_{j=0}^{t-1} \left(\nabla f(\theta_j)\right)^2}} \cdot \nabla f(\theta_{t-1}) \tag{1}$$

$$\theta_t \longleftarrow \theta_{t-1} - \nu_t \tag{2}$$

Here, ν_t and μ_t are the accumulated velocity vector and the momentum rate at time step t, respectively; η is the learning rate, which remains unchanged throughout the process. All operations in Eq. (1) are element-wise. Furthermore, $\nu_0 = 0$ and so we add a small amount ε into Eq. (1) to avoid the singularity. Since this method is inspired by combining momentum and AdaGrad, we call it *AdaMomentum*.

Experiments on language modeling To investigate the effect of different optimization algorithms on the performance of RNN-LSTMs, we first consider the task of language modeling over the Penn TreeBank corpus. We use the same settings with [6]: $929K$ training, $73K$ valid, $82K$ testing words and a vocabulary of size $10K$, with the out-of-vocabulary words mapped to the token ⟨unk⟩. Our architecture is the RNN with 2 stacked layers of LSTMs [3]. Each LSTM layer, as well as our word

[*]Work done when the authors were interning at Baidu. Equal contribution.

embeddings, has dimension 256. We train the model using SGD, SGD with momentum, AdaGrad and AdaMomentum, each for 13 epochs. We provide the hyper-parameters of these methods in Table 1. We present the valid and test perplexity recorded every 500 batches Figure 1.

Table 1: Learning and momentum rate η, μ of the algorithms.

Model	η	μ
SGD	1.0	–
Momentum	1.0	0.6
AdaGrad	0.01	–
AdaMomentum	0.02	0.9

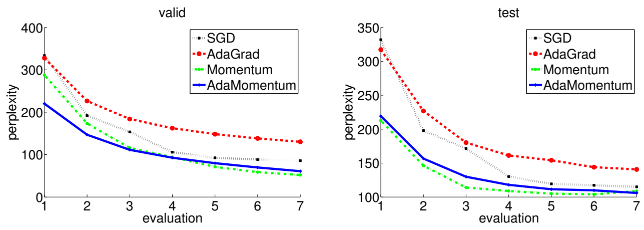

Figure 1: Valid (left) and test (right) perplexities every 500 batches of the algorithms. The final test perplexities of SGD, Momentum, AdaGrad and our proposed AdaMomentum are 115, 108, 140, and 106 respectively. This implies that momentum is very crucial to training language models with RNN-LSTMs, and that combining AdaGrad further improves the performance.

Experiments on named entity recognition We also consider the 4-type Chinese named entity recognition task on the OntoNotes (v5.0) dataset [4]. The task is to recognize and classify the words in a sentence into one of the 4 types: GPE, LOC, ORG, and PERSON. We keep only the sentences of length between 2 and 80, leading to 14.700 training, 1.447 valid, and 1.697 test sentences. We set the embedding size to 256 and use a 2-layer Bi-LSTM with 256 hidden units on each layer. As in the language model experiment, we use different algorithms, including AdaMomentum, to train different models for 30 epochs. We present the test token-level $F1$ score after each epoch in Fig. 2.

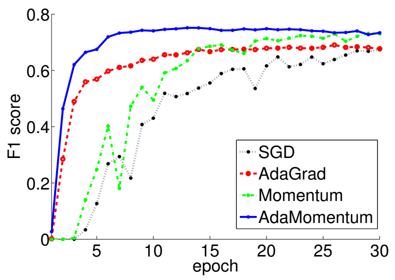

Figure 2: Our proposed AdaMomentum achieves 0.75+ token-level $F1$ score on test data, outperforming other methods. AdaMomentum converges quickly and stably, possessing the merits of both AdaGrad and Momentum. Our performance is lower than as reported [4] because we use a simpler model.

Conclustion and future work With these preliminary results, we strongly believe that although RNN-LSTMs can be trained quite easily with vanilla optimization algorithms, we can improve their performances on many tasks by using more advanced techniques. Our future work will aim at examining those techniques, especially AdaMomentum, on other tasks involving RNN-LSTMs and at various scales of data, such as neural machine translation or question answering.

References

[1] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[2] I. Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[3] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[4] M. Wang, W. Che, and C. D. Manning. Joint word alignment and bilingual named entity recognition using dual decomposition. In *ACL*, 2013.

[5] R. Weischedel, E. Hovy, M. Marcus, M. Palmer, R. Belvin, S. Pradhan, L. Ramshaw, and N. Xue. Ontonotes: A large training corpus for enhanced processing. *Handbook of Natural Language Processing and Machine Translation. Springer*, 2011.

[6] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. In *ICLR*, 2015.