Nadam as a more polyvalent Adam with the increase of capacity in recurrent neural networks

Authors: Lucien Iseli, Marijn van der Meer, Florian Ravasi
Professors: Martin Jaggi, Nicolas Flammarion
Optimization for Machine Learning, EPFL Lausanne, Switzerland
June 12th 2020

***Abstract*—In current state-of-the-art deep learning implementations, neural networks can have up to millions of parameters. In addition, the loss to be optimized is almost always highly non-convex and therefore challenging to optimize. In the early days of deep learning, second-order methods, such as Hessian Free, were widely adopted to tackle these challenging problems. A bit later, in [1], first-order methods such as SGD were investigated with the intent of replacing second-order methods by using momentum and Nesterov accelerated gadient (NAG). Since then, throughout the years, research has focused on first-order methods for their simplicity and low computational cost. Since SGD, new first-order methods have emerged, the most popular being Adam and RMSprop, with promises of greater performance. Adam, which is an extension of RMSprop with momentum is certainly performing well, but seems to struggle on certain problems where RMSprop delivers greater results. Indeed, momentum may impede the performance of Adam in specific cases, for instance when the difference in curvature varies between dimensions. In [1], a similar problem is described with classical momentum (CM). There, the authors propose to use NAG to achieve improved performance. In this paper, we investigate whether Nadam, a version of Adam incorporating NAG, gives more versatile results than Adam. Furthermore, Nadam is compared with RMSprop when Adam does not perform as well as the former. In particular, we examine how Nadam, RMSprop and Adam perform when the number of parameters, and thus the difficulty of the problem, changes.**

I. Introduction

In this paper, four optimisers are analysed and compared to each-other: SGD with NAG, Adam, RMSprop and Nadam.

A. CM and NAG

In specific instances, for example when the difference in curvature between dimensions is significant, Classical Momentum (CM) fails to perform adequately. These limitations are corrected in Nesterov Accelerated Gradient (NAG), which can be rewritten so that it is comparable to CM [1]. In this format, the only difference between NAG and CM lies in the fact that NAG calculates the gradient not at the current position, but at a position where momentum has been added to the present position. In this sense, NAG can be considered as a CM with a peek in the future advantage. In [1], the benefit of adding momentum to SGD is evaluated either with CM or NAG, and its performance is analyzed in a narrow valley with poor initialization. Without momentum, SGD performs poorly once it reaches the flat region of the narrow valley, and does not get very far, compared to CM or NAG. While CM accumulates momentum in the direction of high curvature and eventually bounces back and forth, NAG manages not only to stay steady but also, thanks to momentum, to handle the flat regions fairly well.

B. RMSprop, Adam and Nadam

Compared to SGD, RMSprop [2] is an adaptive learning rate method, *i.e.* it adjusts the learning rate for each dimension and across iterations. Adam [3] is a combination of RMSprop and momentum. It is one of the most commonly used and convenient optimisers for training deep learning models. Nevertheless, as Adam is an extended version of RMSprop that includes momentum, but does not incorporate NAG, it is still sensitive to curvature anomalies. Furthermore, some recent papers [4] have pointed out some specific complex situations where Adam fails to find an optimal solution. Finally, Nadam [5] combines Adam and NAG. In [5], Nadam is presented as having significant performance gains in terms of training speed compared to RMSprop, Adam and SGD.

C. Hypothesis

To compare the optimizers presented in the beginning of I, we pick the task of predicting the next word in a sentence. This is done with the help of recurrent neural networks (RNN) that vary in capacity. Recurrent neural networks are known to be challenging to train. Hence, in what follows, we investigate how the aforementioned optimisers handle the problem at hand once it becomes more complex, *i.e.* when the number of parameters significantly increases. Our hypothesis is that Adam, which uses momentum unlike RMSprop, struggles because of the complex curvatures as it does not implement NAG. Nadam on the other hand should be able to correct it.

II. Experiments structure

The performance of SGD, RMSprop, Adam and Nadam are analyzed during the training of three different recurrent neural networks with LSTM cells whose task is to predict the next word in a sentence. The dataset, WikiText-2, contains texts from articles taken from Wikipedia with a total of $2'051'910$ words and a vocabulary of $33'277$ words. That vocabulary is divided into sentences of 40 words each. For each predicted word, the output of the network consists therefore of $33'277$ logits corresponding to each word in the vocabulary. On this, a cross-entropy loss is used. There are

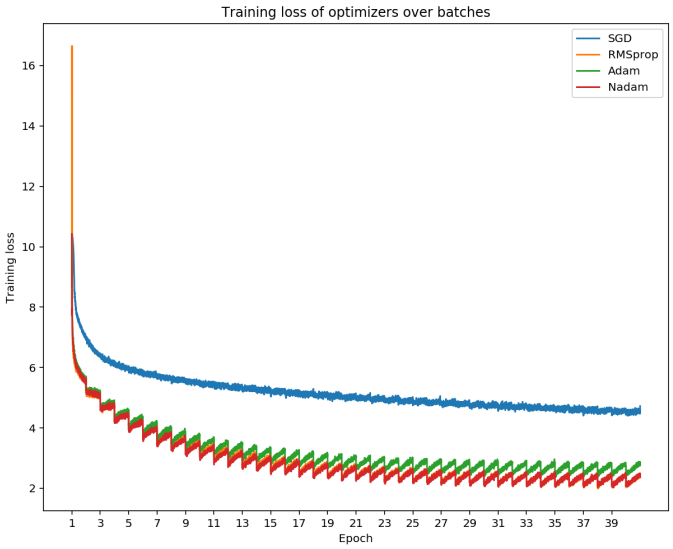


Fig. 1: Losses comparison on small network (RNN1)

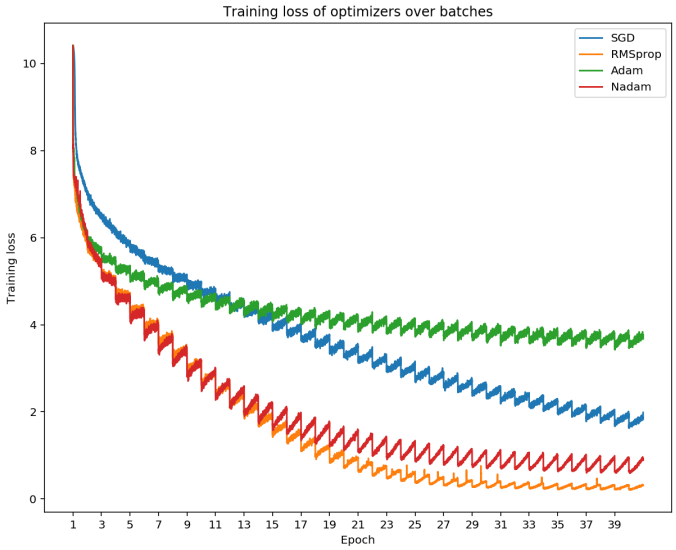


Fig. 2: Losses comparison on big network (RNN3)

four types of parameters that we could adjust in order to increase the capacity of our model: the number of features of the hidden state in the LSTM, the embedding size for each word, the number of recurrent layers (*i.e.* how many cells are stacked) and the capacity of the fully connected layer that predicts the logits.

The first network (RNN1) has a rather limited number of features concerning the hidden state (300) and embedding (300). In addition, it does not stack any LSTMs. In the second one (RNN2), only the number of hidden (1000) and embedding (1000) features is increased. Finally, in the third one (RNN3), 3 stacked recurrent layers are used rather than one. Note that the capacity of the fully connected layer only changes because of the embedding, as the output size remains unchanged over the three networks and that the number of words (40) in each sentence does not change.

For each network, the optimal learning rate is identified using grid search and the RNNs are initialized with exactly the same weights. More details about the learning rates used in the experiments can be found in the Appendix. The momentum used for SGD with NAG is again identified using grid search and the optimal value for all three experiments is 0.99. Additionally, 0.9 and 0.999 are used for β_1 and β_2 concerning Adam as well as Nadam as advised in [3]. Even if 0.975 is used in the paper introducing Nadam [5], we decide to stick with 0.9 and 0.999 so that it is comparable to Adam. Finally, at the beginning of each epoch, the sentences that form the batches are shuffled. However, the order in which they occur is the same between the different trainings.

III. Results

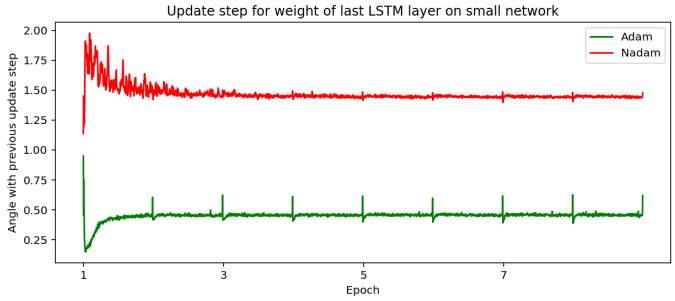
Fig. 1 and 2 illustrate the evolution of the different training losses for the smallest (RNN1) and biggest networks (RNN3) accross epochs. The results for RNN2 can be found in VII in Fig. 4.

As can be seen on Fig. 1, RMSprop, Adam and Nadam behave in a similar manner while SGD with NAG, which serves as baseline, lags behind. However, when strikingly increasing the number of parameters (Fig. 2), Adam falls short to RMSprop and Nadam, ultimately even getting outperformed by SGD. In section I-C, our hypothesis suggests that Adam's momentum may suffer from the same issues as CM and bounces back and forth when the curvatures are more complex. Adam's momentum may hamper the optimiser's performance as it may be a little too naive, whereas NAG tries to be more foresighted. Our results seem to support this hypothesis but this will deeper evaluated in section IV.

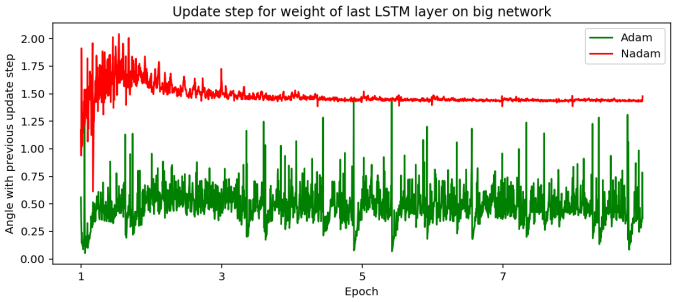
Another thing to notice on Fig. 1, 4 and 2 is that we have a "stairs" effect at each epoch. A suggestion to explain the cause of this effect is that at each epoch, when feeding previously seen batches, the network is better at classifying them as it has seen them one more time since the sentences remain the same, thus leading to a gap when starting the epoch. However, since this effect is consistent throughout all networks and optimizers we decide to ignore this effect.

IV. Discussion

Since NAG seems to prevent variance in the gradient direction, the direction of the update step concerning a subset of parameters, *i.e.* the hidden state of the LSTM from the last recurrent layer, is explored. The last recurrent layer is the one who benefits the most from the gradient flow since it is directly linked to the fully connected layer that predicts the logits. With this aim in mind, the values used in the update step for Adam and Nadam, *i.e.* the one that is multiplied by the learning rate and then subtracted to the parameters, were stored during training. Then, the angle between the value at step t and $t+1$ is evaluated and its evolution plotted (Fig.



(a) Small network (RNN1)



(b) Big network (RNN3)

Fig. 3: Angles update step comparison between Adam and Nadam for the small and big networks

3a, 3b).

Fig. 3a illustrates that Adam and Nadam both have a low variance and thus seem quite stable during training on network RNN1, *i.e.* the one with the fewest number of parameters. This is verified in Fig. 1 where the performance of Adam is similar to Nadam's. However for RNN3, *i.e.* the one with the biggest capacity, Fig 3b demonstrates that Adam has more variance and is not sure what direction to opt for whereas Nadam stabilizes almost as quickly as for RNN1. However, since these evaluations only concern a subset of the loss function parameters and the dimensions used were 300 and 1000, the interpretation of the value of the angle computed here is not entirely unambiguous. Therefore, one should rather focus on the variance of this value, which is strinkinlgy bigger for Adam than Nadam.

Finally, it is noticeable that the curves in Fig. 3a and 3b, are quite similar between networks. Indeed, Nadam shows a small bump at the beginning of training in both cases and the only notable difference seems to reside in the rise in variance once capacity is increased. Moreover, the signals seem to oscillate around the same mean values in both figures. Hence, even though the meaning of the angle is not completely straightforward, we see that it remains consistent.

V. Further improvements

To go further into analyzing the performance of Nadam and evaluating if Nadam indeed combines the best of both worlds between Adam and RMSprop, a similar experiment on different training tasks could be conducted; for example, these optimizers could be compared on auto-encoders or reinforcement learning tasks that use deep neural networks. It would bring a more complete view of their performance on various architectures. Indeed DeepMind, a well-renown company in reinforcement learning, usually uses RMSprop but Adam and Nadam seem to outperform RMSprop on auto-encoders [5] [6]. That way, we would know a wider range of tasks that tend to be best trained with either RMSprop or Adam to study Nadam's performance.

Furthermore, analyzing the results of the optimizers over different architectures might bring a lot of insight into their loss landscape [7]. Doing so, we may acquire a better understanding of what kind of loss landscapes bring optimisation challenges for RMSprop and Adam. Then, one could verify if Nadam is indeed more versatile than the aforementioned optimizers on these landscapes.

VI. Conclusion

For this particular task, RMSprop seems to be the most efficient optimizer, particularly when the network capacity increases significantly and the problem to be solved therefore becomes more difficult to optimize. Besides, in this case, Nadam proves to be more stable than Adam. As mentioned in section V, the choice between Adam and RMSprop for a particular optimization task in deep learning is not straightforward. Hence, we can hope that Nadam benefits from the momentum Adam adds to RMSprop without having an oscillating effect. Nevertheless, there seems to be a trade-off between versatility and simplicity as Nadam's implementation becomes cumbersome and therefore tends to be less easily interpreted. As a matter of fact, note that Nadam combines momentum, RMSprop as well as NAG and the effects of this set of implementations used together might not be really clear.

References

[1] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: http://proceedings.mlr.press/v28/sutskever13.html

[2] A. Karpathy, "A peek at trends in machine learning," 2017. [Online]. Available: https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106

[3] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[4] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to SGD," *CoRR*, vol. abs/1712.07628, 2017. [Online]. Available: http://arxiv.org/abs/1712.07628

[5] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[6] A. Basu, S. De, A. Mukherjee, and E. Ullah, "Convergence guarantees for rmsprop and ADAM in non-convex optimization and their comparison to nesterov acceleration on autoencoders," *CoRR*, vol. abs/1807.06766, 2018. [Online]. Available: http://arxiv.org/abs/1807.06766

[7] H. Li, Z. Xu, G. Taylor, and T. Goldstein, "Visualizing the loss landscape of neural nets," *CoRR*, vol. abs/1712.09913, 2017. [Online]. Available: http://arxiv.org/abs/1712.09913

VII. Appendix

	RNN1	RNN2	RNN3
SGD	0.05	0.05	0.05
RMSprop	0.006	0.003	0.001
Adam	0.008	0.006	0.01
Nadam	0.006	0.003	0.003

TABLE I: Learning rates for different optimizers and models

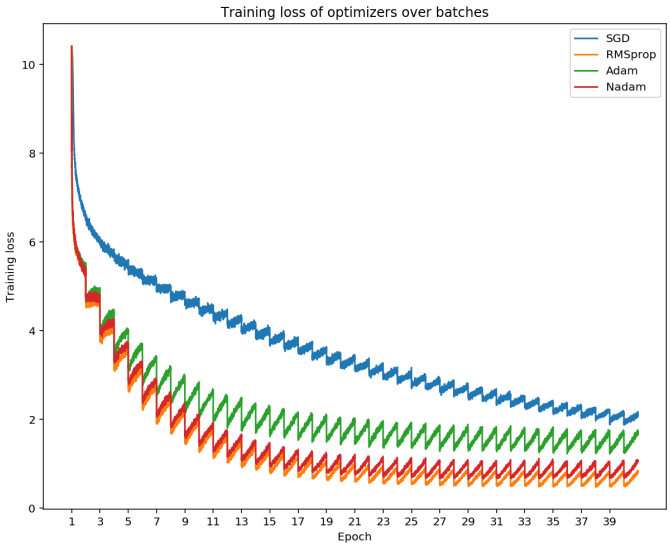


Fig. 4: Losses comparison on medium network (RNN2)