Improving First-Order Optimization Algorithms (Student Abstract)

Tato Ange,[1,2,3] **Nkambou Roger**[1,2]
[1]Department of Computer Science, Université du Québec À Montréal, Canada
[2]Centre de Recherche en Intelligence Artificielle, Montréal, Canada
[3]nyamen_tato.ange_adrienne@courrier.uqam.ca

Abstract

This paper presents a simple and intuitive technique to accelerate the convergence of first-order optimization algorithms. The proposed solution modifies the update rule, based on the variation of the direction of the gradient and the previous step taken during training. Results after tests show that the technique has the potential to significantly improve the performance of existing first-order optimization algorithms.

Introduction

Optimization algorithms in machine learning (especially in neural networks) aim at minimizing a loss function. This function represents the difference between the predicted data and the expected values. We propose a new technique aiming at improving the empirical performance of any first-order optimization algorithm, while preserving their properties. We will refer to the solution when applied to an existing algorithm A as AA for (Accelerated Algorithm). For example, for AMSGrad (Reddi, Kale, and Kumar 2018) and Adam (Kingma and Ba 2014), the modified versions will be mentioned as AAMSGrad and AAdam. The proposed solution improves the convergence of the original algorithm and finds a better minimum faster. Our solution is based on the variation of the direction of the gradient with respect to the direction of the previous update. We conducted several experiments on problems where the shape of the loss is simple (has a convex form) like Logistic regression, but also with non-trivial neural network architectures such as Multi-layer perceptron, deep Convolutional Neural Networks and Long Short-Term Memory. We used *MNIST* [1], *CIFAR-10* [2] and *IMDB movie reviews* (Maas et al. 2011) datasets to validate our solution. It should be mentioned that although our experiments are limited to SGD, AdaGrad, Adam and AMSGrad, the proposed solution could be applied to other first order algorithms.

[1]http://yann.lecun.com/exdb/mnist/

[2]https://www.cs.toronto.edu/ kriz/cifar.html

Intuition and Pseudo-code

The pseudo-code of our proposed method applied to the generic adaptive method setup proposed by (Reddi, Kale, and Kumar 2018) is illustrated in Algorithm 1, .

Algorithm 1 *Accelerated - Generic Adaptive Method Setup*

Input: $x_1 \in$ F, step size $(\alpha_t > 0)_{t=1}^{T}$, sequence of functions $(\varphi_t, \psi_t)_{t=1}^{T}$
$t \leftarrow 1$
$S \leftarrow$ threshold
repeat
  $g_t = \nabla f_t(x_t)$
  $m_t = \varphi_t(g_1, ..., g_t)$
  $V_t = \psi_t(g_1, ..., g_t)$
  if $g_t \cdot m_t > 0$ and $\mid m_t - g_t \mid > S$ **then**
    $gm_t = g_t + m_t$
    $m_t = \varphi_t(gm_1, ..., gm_t)$
  $\hat{x}_{t+1} = x_t - \alpha_t m_t V_t^{-1/2}$
  $x_{t+1} = \prod_{F, \sqrt{V_t}}(\hat{x}_{t+1})$
  $t \leftarrow t + 1$
until t > T

Let's take the famous example of the ball rolling down a hill. If we consider that our objective is to bring that ball (parameters of our model) to the lowest possible elevation of the hill (cost function), what the method does is to push the ball with a force higher than the one produced by any optimizer, if the ball is still rolling in the same direction. This is done by adding the sum between the current computed gradient (g_t) and the previous gradient (g_{t-1}) or the previous moment (m_{t-1}) to the step currently being computed instead of only g_t. The ball will gain more speed as it continues to go in the same direction, and lose its current speed as soon as it will pass over a minimum. We gain $d = -\eta \cdot g_{t-1}$ (η is the learning rate) for each step at time t where the direction of the gradient did not change. To avoid the oscillation of the ball over a minimum, a constraint was added concerning the difference between the past gradient and the current one. This difference should be more than a threshold ($S <\mid g_{t-1} - g_t \mid$) in order for the acceleration to be effective when updates became smaller.

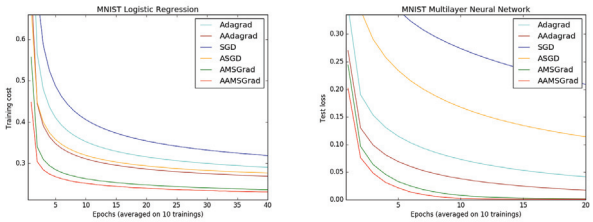

Figure 1: Logistic regression training negative log likelihood on MNIST images (left). MLP on MNIST images (right). Loss on training step.

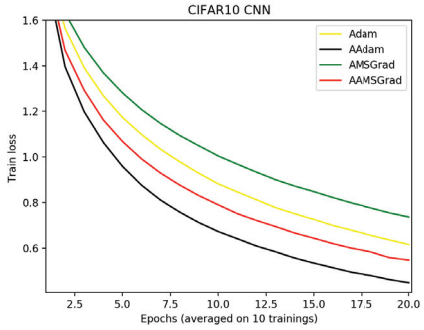

Figure 2: CNN on cifar10. Training cost over 20 epochs. CIFAR-10 with c32-c64-c128-c128 architecture.

We will focus on the case where the gradient g_t is replaced by $g_t + m_{t-1}$ or $g_t + g_{t-1}$ when computing m_t in AAdam and AAmsGrad or g_t in other optimizers (SGD and AdaGrad for example) if the direction does not change. Note that g_t could also be replaced by $g_t + v_{t-1}$ when computing V_t.

Experiments and results

In order to evaluate the proposed solution empirically, we investigated different models: Logistic Regression, Multilayer perceptron (MLP), Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM). We used the same parameter initialization for all of the above models. β_1 was set to 0.9 and β_2 was set to 0.999 as recommended for Adam and the learning rate was set to 0.001 for Adam, AAdam, AMSGrad, AAMSGrad and 0.01 for the remaining optimizers. The algorithms were implemented using Keras [3] and the experiments were performed using the built-in Keras models, making only small edits to the default settings. For each model, all the optimizers started with the same initial values of weights. For the accelerated versions AAdam and AMSGrad, we will only show the variant where the value of m_t was changed according to algorithm 1. The value of v_t stayed unchanged. The experiments were repeated 10 times and we calculated the mean of the estimated mean model skill. Thus, each point in the following figures represents the average result after 10 trainings on one epoch (one pass over the entire dataset). The choice of the parameter S is very important to ensure the effectiveness of the approach. When the value is too small, the accelerated versions converge faster at the beginning but fail to reach a better minimum. The full code for this project is available on the Github [4] of the first author. As shown on figures 1,2 and 3 the accelerated versions largely outperformed the original algorithms. During all our experiments, we noted that Adam outperformed AMSGrad. In fact, the performance of an optimizer depends on the data, the starting points etc. Hence the solution could be applied to improve any optimizer, as the best optimizer will always depend on the task at hand.

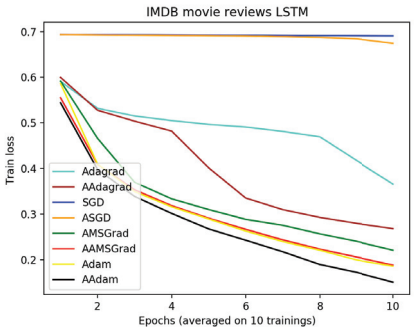

Figure 3: LSTM training negative log likelihood on IMDB movie reviews. Loss on training step. No threshold was set when training the LSTM model.

Conclusion

The proposed solution has the potential to speed up the convergence of any first-order optimizer based on the variation of the direction of the gradient or the first moment (m_t). The only drawback of this contribution is that it requires slightly more computation than the standard approach, and implies to specify a new parameter S. However, we found that the extra time taken by the technique was compensated by the better minimum reached. Future work will aim at analysing the technique on sparse data, determine best empirical value of S for each context, or optimize the technique by making sure that the test stops once the threshold has been reached).

References

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, 142–150. Association for Computational Linguistics.

Reddi, S. J.; Kale, S.; and Kumar, S. 2018. On the convergence of adam and beyond.

[3] https://keras.io/

[4] https://github.com/angetato/Custom-Optimizer-on-Keras