Accelerating First-Order Optimization Algorithms

Ange Tato[1,2] and Roger Nkambou[1,2]

[1] Université du Québec à Montréal, Québec, Canada
[2] Centre de Recherche en Intelligence Artificielle (CRIA), Montréal, Canada
nyamen_tato.ange_adrienne@courrier.uqam.ca,nkambou.roger@uqam.ca

Abstract. Several stochastic optimization algorithms are currently available. In most cases, selecting the best optimizer for a given problem is not an easy task as they all provide adequate results. Therefore, instead of looking for yet another absolute best optimizer, accelerating existing ones according to the context might prove more effective. This paper presents a simple and intuitive technique to accelerate first-order optimization algorithms. When applied to first-order optimization algorithms, it converges much more quickly and achieves a better minimum for the loss function when compared to traditional 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. Several tests were conducted with SGD, AdaGrad, Adam and AMSGrad on three public datasets. Results clearly show that the proposed technique, as tested in the field, has the potential to improve the performance of existing optimization algorithms.

Keywords: Stochastic Optimization · First order optimization · Neural Networks · Gradient Descent.

1 Introduction

Several approaches currently exist to improve learning in neural networks : improving the architecture, finding the optimal parameters, finding the best representation of the data, choosing the best optimization algorithm, etc. In this paper, we are interested in the gradient-descent-based method to optimize the learning process in neural network architectures. Optimization algorithms in machine learning (especially in neural networks) aim at minimizing an objective function (generally called loss or cost function). This function represents the difference between the predicted data and the expected values. The minimization consists in finding the set of parameters (weights) of the architecture that gives the best results in targeted tasks such as classification, prediction or clustering. Finding a set of weights to minimize residuals in a feed-forward neural network is not trivial. The process is nonlinear and dynamic in that any change of one weight requires the adjustment of many others [5].

The majority of existing optimization algorithms are first-order methods - using first derivatives of the function to minimize -, based on the gradient descent technique. These techniques - e.g. the back propagation of error -are used to find a matrix of weights that meets the error criterion. Adam (Adaptive Moment estimation) is probably the most popular [6, 9, 25]. However, Adam (while possessing convergence proof issues) has recently been proven unable to converge to the optimal solution for a simple convex optimization setting [19]. A more recent algorithm (AMSGrad) addresses this issue by endowing Adam and other adaptive stochastic methods with a long-term memory. Unfortunately, Adam can still outperform AMSGrad in some cases [3]. This suggests that the optimizers power may be a condition of the data domain it performs upon. Therefore, instead of looking for the absolute best optimizer, finding a way to accelerate the existent would be of greater use. In this perspective, we are investigating a new technique aiming at improving the empirical performance of any first-order optimization algorithm, while preserving their property. We will refer to the solution when applied to an existing algorithm A as AA for (Accelerated Algorithm). For example, for AMSGrad and Adam, 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 tests 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* [4], *CIFAR-10* [5] and *IMDB movie reviews* [17] [6] 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 algorithms. The results suggest that adding the proposed solution to the update rule of a given first-order optimizer makes it performs better.

The rest of the paper is organized as follows: Section 2 summarizes related work; Section 3 describes the proposed solution and provides its theoretical analysis; Section 4 presents the experiments setup and the results; Section 5 discusses performance assessment, the setting of the parameters of the proposed solution and its possible integration in other existing optimizers.

2 Related Work

Learning in neural networks is performed by minimizing an error function. This function therefore measures the difference between the expected and the computed outputs on the entire sample. An error approximating 0 implies that the

[3] https://fdlm.github.io/post/amsgrad/
[4] http://yann.lecun.com/exdb/mnist/
[5] https://www.cs.toronto.edu/ kriz/cifar.html
[6] http://www.cs.cornell.edu/people/pabo/movie-review-data/

network correctly predicts the expected outputs of the data upon which it has learned. Loss minimization involves finding the values of a set of parameters (weights) that reduce the function (this minimum could be local or global). What renders the problem more complex is that the general shape of the loss function is usually poorly understood. Logistic regression (LR) and softmax (multiclass generalization of LR) do not belong to that class as they have well-studied convex objectives [20]. However, that is not the case for neural networks with more than one layer, where the shape of the loss is generally neither convex nor concave, thus able to allow several minima [4]. The loss function is usually minimized using some kind of stochastic gradient descent (SGD) [3], in which the gradient is evaluated using the back propagation procedure [14]. Gradient Descent is one of the most popular algorithms to perform such a task and the much more common way to optimize neural networks [21]. To be used in back propagation, the loss function must satisfy some properties such as the ability to be written as an average, and not be dependent on any activation values of a neural network besides the output values. There exist many loss functions but the quadratic or mean squared error and the cross-entropy are the most commonly used in the domain.

2.1 How Gradient Descent Method Works ?

Let J(x) be a function parameterized by a models parameters $x \in \mathbb{R}^n$, sufficiently differentiable of which one seeks a minimum. The gradient method builds a sequence that should - in principle - work towards finding the minimum. For this, we start from any value x_0 (i.e. a random value) and we construct the recurrent sequence by:

$$x_{n+1} = x_n - \eta \cdot \nabla_{x_n} J(x) \tag{1}$$

where η is the learning rate. For adaptive methods like Adam, the learning rate varies for each parameter. This method is ensured to converge - even if the input sample is not linearly separable - to a minimum of the error function for a well-chosen learning rate. There exist several variants of this method in two basic categories: first-order and second-order methods. While first-order methods use first derivatives of the function to minimize, second-order methods make use of the estimation of the Hessian matrix (second derivative matrix of the loss function with respect to its parameters) [22, 15]. The latter determines the optimal learning rates (or step size) to take for solving quadratic problems. While such approach provides additional information useful for optimization, computing accurate second-order derivatives is too computationally expensive for large models. Furthermore, the computed value usually equals to a bad approximation of the Hessian. In this paper, we will only focus on first-order techniques (the most popular in machine learning domain as well as the most suited for large-scale models).

2.2 Existing First Order Optimization Algorithms

Adam [11] is a first-order gradient-based algorithm of stochastic objective functions, based on adaptive estimates of lower-order moments. The first moment, once normalized by the second moment, gives the direction of the update. Adam updates are directly estimated using a running average of the first and second moment of the gradient. It computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. It has two main components: a momentum component and an adaptive learning rate component. It can be viewed as a combination of RMSprop [7]) and momentum techniques NAG [18]. Adam has a bias-correction feature that helps slightly outperform previous adaptive learning rate methods towards the end of optimization when gradients become sparser [21]. To our knowledge, Adam is one of the latest state-of-the-art optimization algorithms used by many practitioners of machine learning. There exist several techniques to improve Adam such as fixing the weight decay [16], using the sign of the gradient in distributing learning cases [2], or switching from Adam to SGD [10]. However, it has been recently proven that Adam, in addition to encompassing some mistakes in the convergence proof, is unable to converge to the optimal solution for a simple convex optimization setting [19]. A more recent algorithm (AMSGrad) fixes this problem by endowing the techniques used with a long-term memory. The main difference between AMSGrad and ADAM is that AMSGrad maintains the maximum of all v_t (exponentially decaying average of past squared gradients) until the present time step and uses this maximum value for normalizing the running average of the gradient. The original paper claims that AMSGrad either performs similarly, or better than Adam on some commonly used problems in machine learning. The update rule of AMSGrad is as follows :

$$\begin{aligned}
x_{n+1} &= x_n - \frac{\alpha}{\sqrt{\hat{v_n}+\epsilon}}\hat{m_n} \\
m_n &= \beta_1 \cdot m_{n-1} + (1-\beta_1) \cdot \nabla_{x_n} J(x) \\
v_n &= \beta_2 \cdot v_{n-1} + (1-\beta_2) \cdot \nabla_{x_n} J(x)^2 \\
\hat{m_n} &= \frac{m_n}{1-\beta_1^n} \\
\hat{v_n} &= max(\hat{v}_{n-1}, \frac{v_n}{1-\beta_2^n}).
\end{aligned}$$

Recall that the aim of an optimizer is to look for parameters that minimize a function, knowing that we do not have any knowledge on what this function looks like. If we knew the shape of the function to minimize, it would then be easy to take accurate steps that lead to the minimum. While we can't plan for the shape, each time we take a step (using any of the optimizers) we can know if we passed a minimum by computing the product of the current and the past gradients, and check whether it is negative. This informs us on the curvature of the loss surface and is - as far as we know - the only accurate information available in real time. The proposed method exploits this knowledge to improve the convergence of an optimizer. Momentum techniques are one of the many ways of improving each optimizer.

However, none of the existing techniques uses the variation of the direction of the gradient as an information to compute the next update. The proposing solution is not specific to a particular optimizer. It can also be applied on an already proposed accelerated version of the algorithm, which makes it easily adaptable on any optimizer.

3 A method to accelerate first-order optimization algorithms

3.1 Notation

$\|x_i\|_2$ denotes the l_2-norm of i^{th} row of x. The projection operation $\prod_{\mathcal{F},A}(x)$ for A $\in \mathcal{S}_+^d$ (the set of all positive definite $d * d$ matrices) is defined as arg $min_{x\in\mathcal{F}} \left\|A^{1/2}(x-y)\right\|$ for $y \in \mathbb{R}^d$. $\mathcal{F}$ has bounded diameter D_∞ if $\|x-y\|_\infty \leq D_\infty$ forall $x, y \in \mathcal{F}$. All operations applied on vectors or matrices are element-wise.

3.2 Intuition and Pseudo-code

The intuition that led to the creation of the proposed solution is presented in this section. In Algorithm 1, the pseudo-code of our proposed method applied to the generic adaptive method setup proposed by [19] is illustrated. For Adam, $\varphi_t(g_1,...,g_t) = (1-\beta_1)\sum_{i=1}^t \beta_1^{t-i} g_i$ and $\psi_t(g_1,...,g_t) = (1-\beta_2)\sum_{i=1}^t \beta_2^{t-i} g_i^2$. For AMRSgrad, $\psi_t(g_1,...,g_t) = (1-\beta_2)$ diag $\left(\sum_{i=1}^t \beta_2^{t-i} g_i^2\right)$.

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
```

Lets take the famous example of the ball rolling down a hill (see figure 1). 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 $d = -\eta \cdot g_{t-1}$ (see figure 1) 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. The

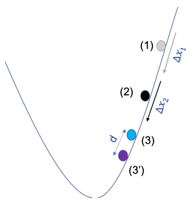

Fig. 1. Intuition of the proposed method for acceleration. (1) initial position of the ball; (2) position of the ball after the first iteration by GD; (3) position of the ball after 2 iterations by GD; (3') position of the ball after 2 iterations by AGD. $d = -\eta \cdot \nabla f(x_1)$ is the gain in the step size.

higher the threshold, the closer we will get to the original algorithm. However, S should not be too small (see section 5.3). Another solution could be to stop accelerating the ball and let the original optimizer take full control of the training once the direction changes for the first time. This solution may also work, but would converge slower. In this paper, we will only 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 Adam and AAmsGrad (see algorithm 1 for details) 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. That change we made to the optimizer will not alter its convergence since the quantity Γ_t [19] - which essentially measures the change in the inverse of learning rate of the adaptive method with respect to time - will keep its property. We provided in the appendix a theoretical proof that the method we are proposing does not change the bound of the regret for AAMSGrad. We only focus on AAMSGrad since the proof of convergence for AGD is straightforward. Note that this technique can be applied to any first-order optimizer, as shown in algorithm 2.

Algorithm 2 *Accelerated - Generic Gradient Descent Method*

```
Input: $x_1 \in$ F, step size ($\alpha > 0$)
$t \leftarrow 1$
$S \leftarrow$ threshold
repeat
    $g_t = \nabla f_t(x_t)$
    $new\_g = g_t$
    if $g_t \cdot g_{t-1} > 0$ and $| g_{t-1} - g_t | > S$ then
        $new\_g = g_t + g_{t-1}$
    $x_{t+1} = x_t - \alpha_t \cdot new\_g$
    $t \leftarrow t + 1$
until t > T
```

3.3 Convergence Analysis

We assume that if we are able to prove that modifying one optimizer with the proposed method does not alter its convergence, then the same applies for the other optimizers. Thus, we only analyse the convergence of one deterministic optimization method AGD (Accelerated Gradient Descent) and one non-deterministic method (AAMSGrad).

For deterministic methods such as GD, the convergence analysis consists in finding an upper bound of the difference between the function value at the t-th iteration and the minimal value $(f(x_t) - f(x^*))$, where f is the function to minimize, x^* is the optimal solution and x_t is the solution found by the algorithm at time t.

Theorem 1. *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a L-Lipschitz convex function and $x^* = argmin_x f(x)$. Then, GD with step-size $\alpha \leq 1/L$ satisfies the following :*

$$\begin{gathered}\sum_{t=0}^{T-1}(f(x_{t+1}) - f(x^*)) \leq \frac{L}{2}[\|x_0 - x^*\|^2 - \|x_T - x^*\|^2] \\ f(x_T) - f(x^*) \leq \frac{L}{2T}\|x_0 - x^*\|^2\end{gathered} \tag{2}$$

In particular, $\frac{L}{\epsilon}\|x_0 - x^*\|^2$ iterations suffice to find an ϵ-approximate optimal value x. GD has a sublinear convergence rate of $O(1/T)$.

For AGD, since $\nabla f(x_{T-1}) = k\nabla f(x_T)$ with $k \geq 1$ we have :

Theorem 2. *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a L-Lipschitz convex function and $x^* = argmin_x f(x)$. Then, AGD with step-size $\alpha \leq 1/L$ satisfies the following :*

$$f(x_T) - f(x^*) \leq \frac{L(1+k)}{2T}\|x_0 - x^*\|^2 \tag{3}$$

AGD has a sub-linear convergence rate of $O(1/T)$ which is the same as GD. The acceleration method does not alter the convergence of the algorithm.

For AAMSGrad, the analysis differs slightly from GD (deterministic methods) but the underlying goal remains the same. We used the online learning framework [26]. As such, the convergence result for AAMSGrad is taken with respect to the *regret* since expected values of the gradients of the stochastic objective are difficult to compute. The *regret* is defined as :

$$R(T) = \sum_{t=1}^{T} [f_t(x_t) - f_t(x^*)] \tag{4}$$

Theorem 3. *Assume that $\mathcal{F}$ has bounded diameter D_∞ and $\|\bigtriangledown f_t(x)\|_\infty \leq G_\infty$ for all $t \in [T]$ and $x \in \mathcal{F}$. With $\alpha_t = \frac{\alpha}{\sqrt{T}}$, AAMSGrad achieves the following guarantee, for all $T \geq 1$:*

$$\begin{aligned} R_T \leq{} & \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} (\hat{v}_{T,i}^{-1/2}) + \frac{D_\infty^2}{4(1-\beta_1)} \sum_{t=1}^{T} \sum_{i=1}^{d} \frac{\beta_{1t} \hat{v}_{t,i}^{1/2}}{\alpha_t} \\ & + \frac{2\alpha\sqrt{1+log(T)}}{(1-\beta_1)^2(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \|g_{1:T,i}\|_2 \end{aligned} \tag{5}$$

The proof of this bound is given in the appendix. The following result falls as an immediate corollary of the above result :

Corollary 1. *Let $\beta_{1t} = \beta_1 \lambda^{t-1}, \lambda \in (0,1)$ in Theorem 2, then we have :*

$$\begin{aligned} R_T \leq{} & \frac{D_\infty^2 \sqrt{T}}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} (\hat{v}_{T,i}^{-1/2}) + \frac{\beta_1 D_\infty^2 G_\infty}{4(1-\beta_1)(1-\lambda)^2} \\ & + \frac{2\alpha\sqrt{1+log(T)}}{(1-\beta_1)^2(1-\gamma)\sqrt{(1-\beta_2)}} \sum_{i=1}^{d} \|g_{1:T,i}\|_2 \end{aligned} \tag{6}$$

The *regret* of *AAMSGRAD* can be bounded by $O(2G_\infty\sqrt{T}) \simeq O(G_\infty\sqrt{T})$. The term $\sum_{i=1}^{d} \|g_{1:T,i}\|_2$ can also be bound by $O(G_\infty\sqrt{T})$ since $\sum_{i=1}^{d} \|g_{1:T,i}\|_2 << dG_\infty\sqrt{T}$ [11].

4 Experiments

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 [11] 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 [7] and the experiments

[7] https://keras.io/

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. However, both solutions ($m_{t-1} + g_t$ or $g_{t-1} + g_t$) worked well during our experiments. 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 number of epochs varied from 10 to 30 for all experiments in this paper (there is a good chance for models to over fit with a higher number of epochs). For AMSGrad, we only considered the version with 'max' on v_t, not the diagonal. The full code for this project is available on Github [8].

4.1 Basics problems

We first tested the proposed method on two basics functions: a convex function x^2 (with the minimum fixed at 0) and a non-convex function x^3 (minimum $\to -\infty$). As seen in figure 2, the accelerated versions are able to reach a better minimum compared to the originals in both cases.

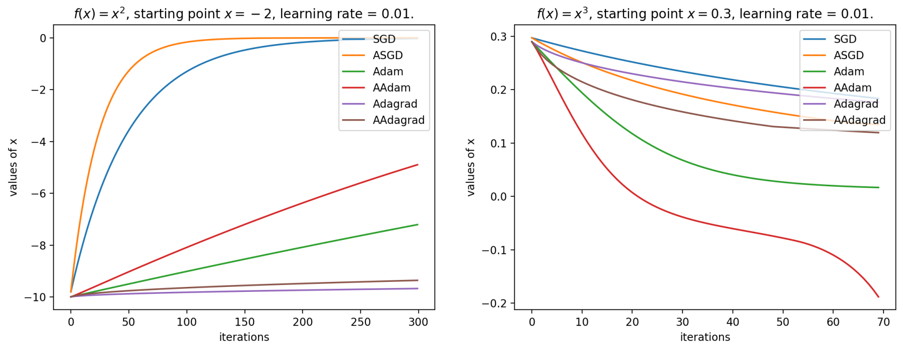

Fig. 2. Minimizing basics functions with starting points x = -2 for x^2 and x = 0.3 for x^3. For AAdam, we used g_{t-1} instead of m_{t-1} in the if-else condition and the computation of the new step. No threshold was set.

4.2 Image Recognition

No pre-processing was applied to the set of MNIST images. All optimizers were trained with a mini-batch size of 128. All weights were initialized from values

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

truncated using normal distribution with a standard deviation of 0.1. We used the 'softmax cross entropy' as the activation function. Again, for AAMSGrad we used g_{t-1} instead of m_{t-1} in the if-else condition and the computation of the new step.

Logistic regression has a well-studied convex objective, making it suitable for comparison of different optimizers without worrying about local minimum issues [11]. We implemented the same model used for comparing Adam with other optimizers in the original paper on Adam. In figure 3 (left side), we can see that the accelerated versions outperform the original algorithms from the beginning to the end of the training. We postulate that our method was able to improve the convergence and find a better minimum in convex problems. Multi-layer perceptron (MLP) is a powerful model with non-convex objective

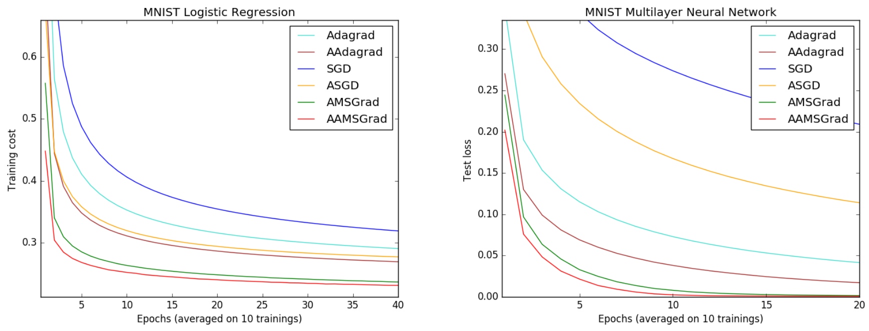

Fig. 3. Logistic regression training negative log likelihood on MNIST images (left). MLP on MNIST images (right). Loss on training step.

functions that consists in an input layer, an output layer, and n hidden layers where n >= 1. In our experiments, we selected models that were consistent with previous publications in the area. A simple neural network model with one fully connected hidden layer with 512 hidden units and ReLU activation was used for this experiment with a mini-batch size of 128. Figure 3 (right side) shows the results of running a MLP on MNIST data. Again, the accelerated versions outperform (reach better minima on training step) the original algorithms.

4.3 CIFAR-10 Dataset

CNNs [13] are neural networks where layers represent convolving filters [12] applied to local features. CNNs are now used in almost every task in machine learning (i.e. text classification [24], speech analysis [1], etc.). We considered the multi-class classification problem on the standard CIFAR-10 dataset, which consists of 60,000 labelled examples of 32 X 32 images. We used a convolutional neural network (CNN) with several layers of convolution, pooling and non-linear units. In particular, the first architecture (which results are shown in figure 4, left side) contained 4 convolutional layers with 32 channels for the first two layers and 64 channels for the other two. The kernel of each of the 4 convolutional layers had a size of 3 x 3 followed by one fully connected layer of size 212. The network used 2 x 2 max pooling. It also contained three Dropout layers with keep probabilities of 0.25, 0.25 and 0.5, respectively applied 1) in between the two first convolutional layers and the two other convolutional layers; 2) between the two last convolutional layers and the flatten layer; and 3) between the fully connected layers. The mini-batch size was set to 128 like in previous experiments. The code for this architecture [9] was taken from the Keras library. Results for this problem are reported in figure 4. As we can see, accelerated algorithms performed considerably better than originals on the two different architectures.

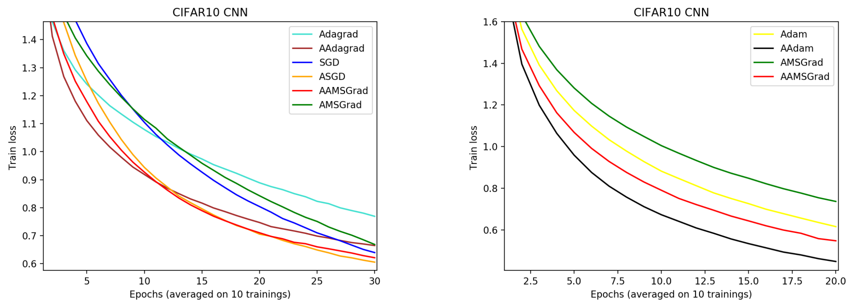

Fig. 4. Convolutional neural networks on cifar10. Training cost over 30 (left) and 20 epochs (right). CIFAR-10 with c32-c32-c64-64 (left) and c32-c64-c128-c128 (right) architectures.

4.4 IMDB Movie Reviews Dataset

Finally, we experimentally evaluated our method on imdb movie reviews data [10]. This dataset consists of 25,000 movies reviews from IMDB, labelled by sentiment (positive/negative). Reviews have been pre-processed and encoded as a sequence of word indexes. We then ran a Logistic Regression and LSTM models. LSTM has become a core component of neural Natural Language Processing (NLP) [23]. The LSTM architecture [8] can learn long-term dependencies using a memory cell that is able to preserve states over long periods of time. We used the built-in LSTM architecture of Keras with 32 units, a recurrent dropout of 0.2 and a dropout of 0.2. An embedding layer preceded the LSTM. The dimensionality of character embedding was set to 32 and the batch size was set to 32. We took the first 5,000 most frequent words. Results are presented in figure 5. As expected,

[9] https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py
[10] https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification

accelerated versions consistently outperformed original algorithms. Nevertheless, for Adam and AMSGrad, the results were not very clear from logistic regression (left), which may be due to the sparse nature of the data. This clears the way for more analyses on sparse problems with logistic regression.

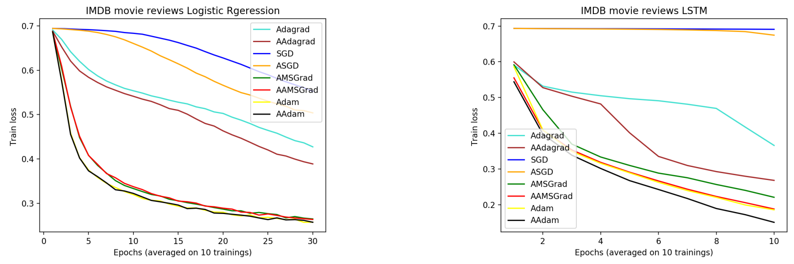

Fig. 5. Logistic Regression training negative log likelihood on IMDB movie reviews (left). LSTM training negative log likelihood on IMDB movie reviews (right). Loss on training step. No threshold was set when training the LSTM model.

5 Discussion

5.1 Analysis of the Execution time

The proposed method add to the original algorithms, a test (if-else) which might slow the execution time. What if we gave the same execution time to all algorithms, investigating whether the proposed method allows the algorithm to reach a better minimum? In figure 6, we performed two experiments where we evaluated the minimum reach by each optimizer given a fixed time. For function x^2, we set the starting point at $x = 2$ and the time at $t = 0.0005s$. As we can see (Figure 6 - left side), the accelerated versions did less iteration but reached a better minimum when compared to the original algorithms under the same execution time. Moreover, we ran the logistic regression on the MINST data with a fixed time of $t = 10s$. Again, as we can see (Figure 6 - right side) accelerated versions largely outperformed the original algorithms. Thus, even if the method takes time compared to original algorithms, it is able to reach a better minimum when it is given the same time of execution.

5.2 Adam vs AAdam

Depending on the starting point of the algorithm, Adam can reach a minimum that is not optimal or oscillate around the minimum (as shown in Figure 7 - left side). This is principally due to the fact that Γ_{t+1} can potentially be indefinite

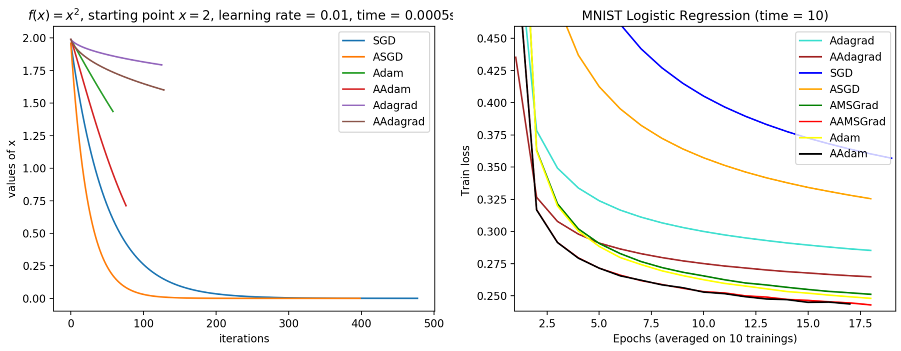

Fig. 6. Convergence analysis with a fixed time.

for t ∈ [T] [19] with:

$$\Gamma_{t+1} = \left(\frac{\sqrt{V_{t+1}}}{\alpha_{t+1}} - \frac{\sqrt{V_t}}{\alpha_t} \right) \tag{7}$$

However, this quantity remains ≤ 0 for non-exponential moving average algorithms such as SGD or AdaGrad (to be observed that AMSGrad resolves this specific problem). Since the technique amplified the step taken by the original algorithm, we could not expect more than for AAdam to behave like Adam (as shown in Figure 7). Thus, depending on the starting point, AAdam may reach a better minimum faster, or only behave like Adam.

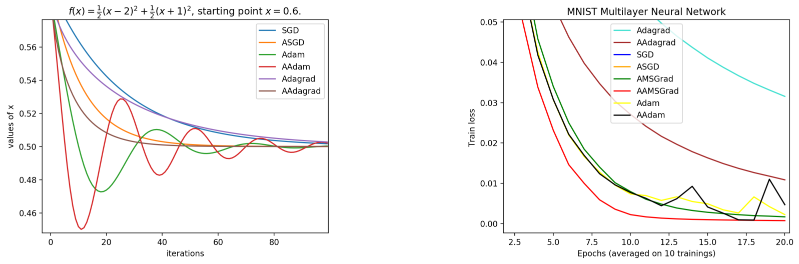

Fig. 7. Convergence analysis with fixed time.

5.3 How to choose the threshold ?

The choice of the parameter S is very important to ensure the effectiveness of the approach. In Figure 8, we tested different values of S with LR on MNIST. As we can see, when the value is high, the accelerated versions seem to behave like the original optimizers. When the value is too small, the accelerated versions converge faster at the beginning but fail to reach a better minimum. Note that, for AAMSGrad, S should be smaller than the one set for AAdam since the v_t used to compute the step size in AMSGrad is higher (the max value) than in Adam. We would like to mention that during all our experiments with SGD and AdaGrad we did not set a value for S. S was only set for AAdam and AAMSGrad. Future work will focus on a more detailed analysis of the impact of the parameter S in order to be able to recommend which value is of use (or not) for a specific model and optimizer.

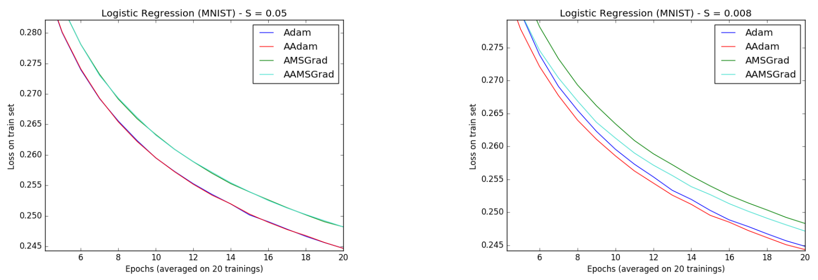

Fig. 8. Experiments on the variation of the value of S

Overall, the proposed technique can be easily applied to other kinds of optimizers. For example, for NAG (Nesterov Accelerated Gradient) we will have (all operations are element-wise):

$$\begin{aligned} \mathrm{x}_{n+1} &= \mathrm{y}_n - \eta \cdot (\nabla J(y_n) + \nabla J(y_{n-1})) \\ &\text{if} \qquad \nabla J(y_n) * \nabla J(y_{n-1}) > 0 \\ \mathrm{y}_{n+1} &= \mathrm{x}_{n+1} + \beta(x_{n+1} - x_n) \end{aligned} \tag{8}$$

During all our experiments, we noted that Adam outperformed AMSGrad. Thus, the performance of an optimizer depends on the data (shape of the loss), the starting points (initial value of parameters) etc. This conclusion is interesting in the sense that our method could be applied to improve any optimizer, as the best optimizer will always depend on the task at hand.

6 Conclusion

In this paper, we have presented a simple and intuitive method that modifies any optimizers update rule to improve its convergence. There can be no optimizer working best for all problems. Each optimizer has its advantages and disadvantages. No one can tell in advance which will be the best choice for a specific case. The solution we have presented 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 (*mt* for moving average methods). Instead of using the gradient as is for computing the next step, we used the sum of the current gradient and the past gradient to compute the current step when both values have the same sign (the direction did not change), and keep the current gradient otherwise. We conducted successful experiments with four well-known optimizers (SGD, AdaGrad, Adam and AMSGrad) on different models using three public datasets (MNIST, CIFRA-10 and IMDB movies reviews). In all our experiments, the accelerated versions outperformed the originals. In worst cases, both had the same convergence, which suggests that the accelerated versions were at least as good as the originals. This work took those optimizers one-step further and improved their convergence without noticeably increasing complexity. The only drawback of the proposed solution is that it requires slightly more computation than the standard approach (as it has to compute the if-else instruction), 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. The new update rule depends only on the variation of the direction of the gradient, which means that it can be used in any other first-order optimizer for the same goal. The technique is intuitive and straightforward to implement. Future work will aim at analysing the technique on sparse data, defined best value of S for each context and finally optimize the technique (e.g. stop the test once the threshold has been reached).

Acknowledgments

We would like to thank the NSERC (National Science and Engineering Council) of Canada and the UQAM's AI Research Center (CRIA) for their financial support of this research.

References

1. Abdel-Hamid, O., Mohamed, A.r., Jiang, H., Deng, L., Penn, G., Yu, D.: Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing **22**(10), 1533–1545 (2014)
2. Bernstein, J., Wang, Y.X., Azizzadenesheli, K., Anandkumar, A.: signsgd: compressed optimisation for non-convex problems. arXiv preprint arXiv:1802.04434 (2018)
3. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010, pp. 177–186. Springer (2010)
4. Choromanska, A., Henaff, M., Mathieu, M., Arous, G.B., LeCun, Y.: The loss surfaces of multilayer networks. In: Artificial Intelligence and Statistics. pp. 192–204 (2015)
5. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on. pp. 39–43. IEEE (1995)

6. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
7. Hinton, G., Srivastava, N., Swersky, K.: Lecture 6a overview of mini-batch gradient descent. Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture (2012)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
9. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. arXiv preprint (2017)
10. Keskar, N.S., Socher, R.: Improving generalization performance by switching from adam to sgd. arXiv preprint arXiv:1712.07628 (2017)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
13. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10), 1995 (1995)
14. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation **1**(4), 541–551 (1989)
15. LeCun, Y., Simard, P.Y., Pearlmutter, B.: Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In: Advances in neural information processing systems. pp. 156–163 (1993)
16. Loshchilov, I., Hutter, F.: Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101 (2017)
17. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1. pp. 142–150. Association for Computational Linguistics (2011)
18. Nesterov, Y.: A method of solving a convex programming problem with convergence rate o (1/k2). In: Soviet Mathematics Doklady. vol. 27, pp. 372–376 (1983)
19. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond (2018)
20. Rennie, J.D.: Regularized logistic regression is strictly convex. Unpublished manuscript. URL people. csail. mit. edu/jrennie/writing/convexLR. pdf **745** (2005)
21. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
22. Schaul, T., Zhang, S., LeCun, Y.: No more pesky learning rates. In: International Conference on Machine Learning. pp. 343–351 (2013)
23. Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075 (2015)
24. Tato, A., Nkambou, R., Dufresne, A., Beauchamp, M.H.: Convolutional neural network for automatic detection of sociomoral reasoning level pp. 284–289 (2017)
25. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. In: International Conference on Machine Learning. pp. 2048–2057 (2015)
26. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 928–936 (2003)