Improving Adam Optimization

Submitted by:

Sparsh Gupta || *Tanmay Singh*

170001049 || 170001051

Computer Science and Engineering

3rd year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Autumn 2019

Index

Introduction

Adam Optimizer is one of the fastest iterative optimization algorithms, used in many areas, especially machine learning. Over the past 50 years, there has been extensive research in this field, and numerous algorithms have been proposed. All the algorithms improve upon the previous algorithms, and Adam has been a result of many of these improvements. In this report, we explore the various algorithms, whose evolution led to development of Adam optimizer and further improving Adam.

Evolution of Iterative Optimization Algorithms

Stochastic Gradient Descent:

It is a first order iterative method for finding the minimum of a function. It is based on the fact that the function value decreases in the opposite direction of the gradient at a particular point, if it is not a local minimizer.

Algorithm 1 Gradient Descent

$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{g}_t$

Challenges and Drawbacks –

- Choosing a proper step size- if too small, slow convergence; if too large, may cause function to fluctuate around minimum or even diverge.
- Same step size for update of all components of Θ.
- Same alpha for all iterations, unable to adapt to dataset characteristics.

Adding Momentum:

Gradient descent has trouble navigating ravines, i.e. areas where surface curves more steeply in one dimension than in the other. In these scenarios, GD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. Adding momentum accelerates

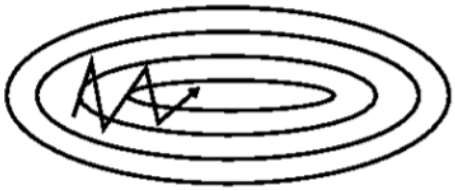
GD in the relevant direction and dampens oscillation. It is done by adding a fraction of the update vector of the past time step to the current update vector.

Algorithm 2 Classical Momentum

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \mathbf{g}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{m}_t$$

The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Gradient Descent without momentum



Gradient Descent with momentum

Challenges and Drawbacks –

- In this method, we first compute the gradient at current location, then take a big jump in the direction of the updated accumulated gradient. So, it does not know when to slow down, and can overshoot the minimum.

Nesterov Accelerated Gradient:

Nesterov momentum has a kind of prescience. It has a notion of where it is going and knows to slow down before the hill slopes up again. First, it makes a big jump in the direction of the previous accumulated gradient. Then it measures the gradient where it ends up, and makes a correction. This corrective update prevents Θ_t from going too fast and overshooting the minimum.

Algorithm 3 Nesterov's accelerated gradient

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta \mu \mathbf{m}_{t-1})$$
$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \mathbf{g}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{m}_t$$

Challenges and Drawbacks –

- Up until now, we have adapted the direction to the slope of the objective function and sped up the descent. However, we would also like to adapt our step size to each individual component Θ_t .

AdaGrad:

It adapts step size rate to the components, performing larger updates for components in whose direction, the gradient is less steep, and smaller updates for those components in whose direction, the gradient is steeper. This prevents unnecessary oscillations in the steeper direction, and moves faster in the less steep direction. It also uses different step size for every component Θ_t.

Algorithm 4 AdaGrad

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{n}_t \leftarrow \mathbf{n}_{t-1} + \mathbf{g}_t^2$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\mathbf{g}_t}{\sqrt{\mathbf{n}_t + \varepsilon}}$$

Challenges and Drawbacks –

- AdaGrad's main weakness is its accumulation of the squared gradients in the denominator. Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the step size to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge, preventing the model from reaching the local minimum.

RMSProp:

Restricts window of accumulated past gradients to some fixed size, by using exponential averaging.

$$R_{t,i} = \gamma R_{t-1,i} + (1 - \gamma) g_{t,i}^{\,2}$$

By the above formula, R approximately becomes the average of last $1/(1-\gamma)$ square gradients.

RMSProp, an alternative to AdaGrad that replaces the sum in n_t with a decaying mean parameterized here by v. This allows the model to continue to learn indefinitely.

Algorithm 5 RMSProp

$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\mathbf{n}_t \leftarrow \nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\mathbf{g}_t}{\sqrt{\mathbf{n}_t + \varepsilon}}$

ADAM

This algorithm also computes adaptive step sizes for each Θ_t. In addition to dividing the step size by the decaying average of past square gradients like RMSProp, Adam also replaces the simple gradient term by an exponentially decaying average of past gradients, thus incorporating momentum.

It only requires first-order gradients with little memory requirements. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; The name Adam is derived from adaptive moment estimation.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Adam works well in practice and compares favorably to other adaptive optimization algorithms.

Accelerated-Adam (AADAM)

Based on Research Paper on "***Improving Adam Optimizer***" by **Ange Tato & Roger Nkambour.**

The main idea behind AAdam is to speed up the progress along dimensions in which gradient consistently point in the same direction. In addition to storing an exponentially decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t like Adam, AAdam also keeps an exponentially decaying average of past updates. Thus, the current update not only depends on the previous gradients, it also depends on the previous values of the update $\Delta\theta$. We keep track of past parameters updates with an exponential decay where β_1 (approximatively 0.9, the same β_1 in Adam) is the constant controlling the decay. It adds a small value 'd' to the current update of Adam. That value (d) is multiplied by the sign of the current gradient. The new update rule is summarized as follows:

$$
\begin{aligned}
\theta_{n+1} &= \theta_n - (\eta \frac{\beta_1}{\sqrt{\hat{v_n} + \epsilon}} \hat{m}_n + d) \\
d &= \Delta\theta_{n-1} * sign(\nabla_{\theta_n} J(\theta)) * (1 - \beta_1) \\
\hat{m}_n, \hat{v}_n &= \text{Adam Parameters} \\
\Delta\theta_{n-1} &= \text{Last update step.}
\end{aligned}
$$

Intuition behind AAdam: If we consider that our objective is to bring a ball (parameters of our model) to a lowest elevation of a road (cost function), what we do is to adapt the speed of the ball by trying to sending it more in the direction of the gradient. That also implies decreasing the step size taken by the ball on the opposite direction. This is done by adding a small portion of past updates to the current updates of Adam. The update is a vector that has the direction of the gradient. In case the gradient changes direction, the size of the step taken by AAdam will be less large than the one taken by Adam step. This new update accelerates the move of the ball towards the minimum (local or global depending on where we started). Since the step added to the step proposed by Adam is not very big, one can hope that if Adam finds a better minimum, AAdam

will find it too but more quickly. It should not be forgotten that finding a better minimum does not imply a better ability to generalize, on the contrary, finding a better minimum could lead to overfitting.

Switching from Adam to SGD(SWAT)

Based on Research Paper on "***Improving Generalization Performance by Switching from Adam to SGD***" by **Nitish Shirish Keskar** and **Richard Socher.**

"*Despite superior training outcomes, adaptive optimization methods such as Adam, AdaGrad or RMSProp have been found to generalize poorly compared to Stochastic gradient descent (SGD). These methods tend to perform well in the initial portion of training but are outperformed by SGD at later stages of training. We investigate a hybrid strategy that begins training with an adaptive method and switches to SGD when appropriate. Concretely, we propose SWATS, a simple strategy which Switches from Adam to SGD when a triggering condition is satisfied. The condition we propose relates to the projection of Adam steps on the gradient subspace.*"

Algorithm 1 SWATS

Inputs: Objective function f, initial point w_0, learning rate $\alpha = 10^{-3}$, accumulator coefficients $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-9}$, phase=Adam.

Initialize $k \leftarrow 0$, $m_k \leftarrow 0$, $a_k \leftarrow 0$, $\lambda_k \leftarrow 0$
while stopping criterion not met **do**
  $k = k + 1$
  Compute stochastic gradient $g_k = \hat{\nabla} f(w_{k-1})$
  if phase = SGD **then**
    $v_k = \beta_1 v_{k-1} + g_k$
    $w_k = w_{k-1} - (1 - \beta_1)\Lambda v_k$
    continue
  end if
  $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
  $a_k = \beta_2 a_{k-1} + (1 - \beta_2) g_k^2$
  $p_k = -\alpha_k \frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \frac{m_k}{\sqrt{a_k}+\epsilon}$
  $w_k = w_k + p_k$
  if $p_k^T g_k \neq 0$ **then**
    $\gamma_k = \frac{p_k^T p_k}{-p_k^T g_k}$
    $\lambda_k = \beta_2 \lambda_{k-1} + (1 - \beta_2)\gamma_k$
    if $k > 1$ and $|\frac{\lambda_k}{(1-\beta_2^k)} - \gamma_k| < \epsilon$ **then**
      phase = SGD
      $v_k = 0$
      $\Lambda = \lambda_k / (1 - \beta_2^k)$
    end if
  else
    $\lambda_k = \lambda_{k-1}$
  end if
end while
return w_k

Switchover point:

$$\left| \frac{\lambda_k}{1 - \beta_2^k} - \gamma_k \right| < \epsilon$$

The implementation of fore-mentioned strategy is being improvised. So, the complete analysis is unavailable.

Test Functions

1. Sphere Function –

$$f(x_1,x_2)=x_1^2+x_2^2$$

2. Rosenbrock Function –

$$f(x_1,x_2)=100(x_2-x_1^2)^2+(x_1-1)^2$$

3. Beale Function –

$$f(x_1,x_2)=(1.5-x_1-x_1x_2)^2+(2.25-x_1+x_1x_2^2)^2+(2.625-x_1+x_1x_2^3)^2$$

4. Matyas Function –

$$f(x_1,x_2)=0.26(x_1^2+x_2^2)-0.48x_1x^2$$

5. Booth Function –

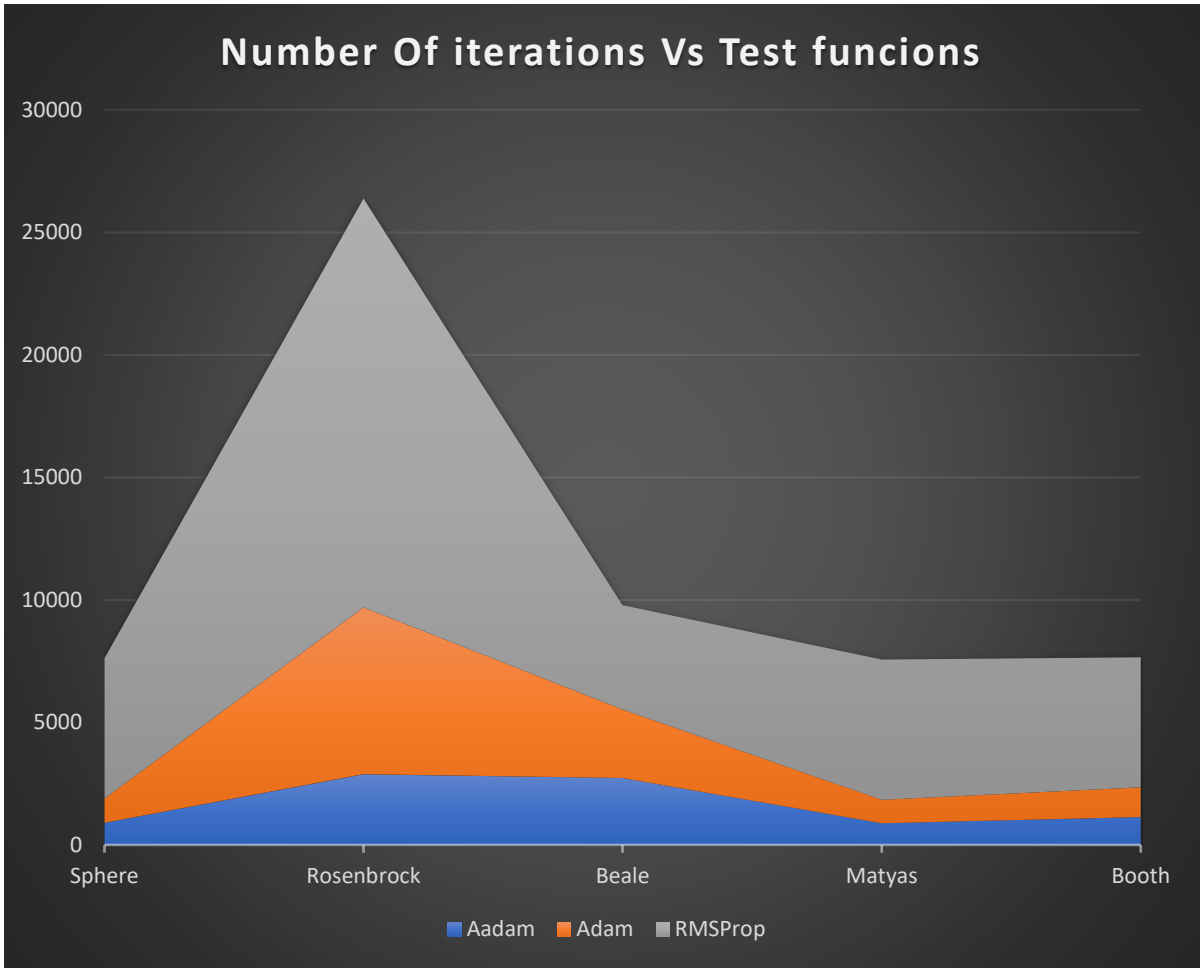
$$f(x_1,x_2)=(x_1 + 2*x_2 -7)^2+(2*x_1 + x_2 - 5)^2$$

Results

	Number of Iterations			
Functions	**AdaGrad**	**RMSProp**	**Adam**	**AAdam**
Sphere	7219	5729	983	913
Rosenbrock	VSC*	16694	6810	2886
Beale	VSC*	4259	2786	2741
Matyas	5577	5722	956	888
Booth	25835	5304	1210	1148

*VSC: Very slow to converge.

Graphical Analysis



Conclusion

From graphical analysis, it is observed that the improved Adam or AAdam is fastest among the four, thereby supporting the hypothesis.

On plotting the relative iterations required to find the minimizer, we observe that –

$$\text{AdaGrad} < \text{RMSProp} < \text{Adam} < \text{AAdam}$$

In summary, RMSprop deals with the radically diminishing learning rates of AdaGrad. Adam then incorporates momentum in addition to dividing by decaying average of past gradients strategy of RMSProp.

Then AAdam speed up the progress along dimensions in which gradient consistently point in the same direction. In addition to storing an exponentially decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t like Adam, AAdam also keeps an exponentially decaying average of past updates. Thus, the current update not only depends on the previous gradients, it also depends on the previous values of the update $\Delta\theta$.

References

- https://www.researchgate.net/publication/334398886_Workshop_track_-ICLR_2018_IMPROVING_ADAM_OPTIMIZER
- https://www.researchgate.net/publication/329039554_An_Optimization_Strategy_Based_on_Hybrid_Algorithm_of_Adam_and_SGD
- https://arxiv.org/pdf/1412.6980.pdf
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- http://cs229.stanford.edu/proj2015/054_report.pdf