RESEARCH REPORT ON ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Clément Bonnet
École CentraleSupélec
Gif-sur-Yvette, France
clement.bonnet16@gmail.com

Tom Dupuis
École CentraleSupélec
Gif-sur-Yvette, France
tom.dupuis@student.ecp.fr

ABSTRACT

This paper is based upon Kingma & Ba (2017) in which the authors introduced a gradient-based method for stochastic optimization called Adam. This optimizer achieves state-of-the-art performance in non-convex settings. We study the algorithm, the influence of its hyper-parameters and we relate Adam to other optimizers. In a logistic regression problem, we show the importance of tuning each hyper-parameter and address some differences between convex and non-convex settings.

1 INTRODUCTION

Many problems from science and engineering rely on optimizing a scalar objective function with respect to some parameters. For instance, loss function optimization is at the core of machine learning. In the context of a high-dimensional parameter space, higher-order optimization methods are ill-suited. Therefore, as done in Kingma & Ba (2017), we focus on first-order methods for the optimization of stochastic objectives. If the gradient of the objective function can be computed, gradient descent (resp. ascent) is a powerful method for minimizing (resp. maximizing) an objective function. As analyzed in Ruder (2017), there are three variants of gradient descent: batch, stochastic and mini-batch gradient descent. The last one is the most commonly used since it has a lower variance than stochastic gradient descent and its implementations in deep learning libraries can make use of highly optimized matrix operations on graphics processing units (GPU). The machine learning setting most often relies on this stochastic mini-batch gradient descent, generally just called stochastic gradient descent (SGD) whose batch-size parameter is the size of the mini batch.

An optimizer is thus an algorithm based on SGD or a variant to minimize an objective function. The goal is to quickly find a global minimum or a satisfying local one. However, one has to make sure the algorithm does not get stuck in a highly sub-optimal local minimum. An overview of different optimization algorithms for SGD is given in section 2.

In this paper we analyze Adam, an efficient first-order stochastic optimization algorithm that is currently used in most of gradient-based learning algorithms.

2 AN OVERVIEW OF OPTIMIZERS

Before Adam was introduced, the field of optimizers was already filled with variants of SGD that improved its baseline performance both in terms of convergence speed and stability. For notation purposes, we recall below the formulation of batch gradient descent (Vanilla gradient descent):

$$\theta = \theta - \alpha \nabla_\theta J(\theta)$$

Where θ is the parameter being updated at each iteration, α the learning rate and $J(\theta)$ the objective function to minimize.

Mini-batch gradient descent comes as a natural improvement of gradient descent. Then follow multiple adapted versions of SGD that try to increase the convergence rate of the objective function minimization. In all the following update rules, operations on vectors are element-wise.

2.1 ADAPTIVE SPEED AND MOMENTUM

The most notable improved versions of mini-batch gradient descent all try to correct the position or the speed in an online fashion. Correcting the position means implementing automatic re-orientation using the cumulative history of gradients, whereas correcting the speed means defining an automatic learning rate scheduling that prevents the algorithm from exploding or vanishing gradients. It uses the cumulative history of gradient norms. The update rule is the following:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{s_t} + \epsilon} m_t$$

where m_t is the momentum or position term (the gradient in SGD) and s_t is the speed term.

One can note that having the gradient g_t instead of m_t and a constant vector of ones for s_t is equivalent to SGD.

2.2 ADAGRAD

AdaGrad corrects the speed by using the cumulative sum of squared gradients over the iterations as s_t:

$$m_t = g_t$$
$$s_t = s_{t-1} + g_t^2$$

with g_t the stochastic (mini-batch) gradient at iteration t.

2.3 RMSPROP

RMSProp corrects the speed by using the moving average of squared gradients over the iterations as s_t:

$$m_t = g_t$$
$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) g_t^2$$

2.4 MOMENTUM / HEAVY-BALL

Momentum SGD uses an adaptive value for the gradient to take the past of the trajectory into account. It uses the moving average of previous gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$s_t = [1, ..., 1]$$

2.5 ADAM: THE BEST OF BOTH WORLDS

Adam combines the ideas of RMSProp and Momentum and corrects both speed and momentum terms.

$$m_t = \frac{1}{1 - \beta_1} (\beta_1 m_{t-1} + (1 - \beta_1) g_t)$$
$$s_t = \frac{1}{1 - \beta_2} (\beta_2 s_{t-1} + (1 - \beta_2) g_t^2)$$

We can easily see that Adam is in fact the combination of RMSProp and Momentum, with an additional debiaising factor. In the next section, we focus on this algorithm.

3 ADAM FOR STOCHASTIC OPTIMIZATION

The name Adam is derived from adaptive moment estimation. In Kingma & Ba (2017), the authors state some attractive benefits of using Adam compared to other optimizers. Among others, Adam is straightforward to implement, computationally efficient, it has little memory requirements. It is invariant to diagonal rescaling of the gradients, well suited for problems that are large in terms of data or parameters. It is appropriate for the online learning setting with non-stationary objectives

and for problems with very noisy/or sparse gradients. Furthermore, Adam's hyper-parameters have intuitive interpretation and typically require little tuning.

Before analyzing Adam further, we display the algorithm below in pseudo-code as it is written in Kingma & Ba (2017).

Algorithm 1: Adam algorithm

Require: α: step size
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates
Require: $f(\theta)$: stochastic objective function with parameter θ
Require: θ_0: initial parameter vector
$m_0 \leftarrow 0$ (Initialize first moment vector)
$v_0 \leftarrow 0$ (Initialize second 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)
  $m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1).g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2.v_{t-1} + (1 - \beta_2).g_t^2$ (Update biased second moment estimate)
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)
  $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha.\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ (Update parameters)
end
return θ_t (Resulting parameters)

The goal of the algorithm is to minimize the expected value $\mathbb{E}[f(\theta)]$ of the stochastic scalar function f with respect to its parameters θ. Since the function is stochastic, $f_t(\theta)$ denote its realization at timestep t.

The algorithm keeps updated moving averages of the gradient and the its square as estimates of the first and second moments of the gradient. The hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the decay rate of these moving averages. They require very little tuning as default values $\beta_1 = 0.9, \beta_2 = 0.999$ often lead to satisfying convergence.

To prevent from moment biases toward the initial value of 0, the algorithm correct both moment estimates by rescaling moment i by a factor $1/(1 - \beta_i^t)$. The bias is thus corrected under the assumption that the second moment is stationary in average, i.e. $\mathbb{E}[g_t]$ is constant. This is not true in general, yet the stationarity of the estimate is well approached since the moving average tends to forget past values, making the estimated second moment close to stationary with respect to recent squared gradient values. In the next section we empirically evaluate the convergence of Adam analyze some of its parameters.

4 EXPERIMENTS

The authors of the article studied the performances of Adam in several situations: logistic regression (MNIST dataset), neural networks and CNNs (convolutional neural networks). They concluded that Adam achieves state-of-the-art convergence speed in all 3 scenarios, consistently reaching faster convergence speed than other adaptive methods.

We replicate these results by implementing the Adam optimizer in the context of logistic regression. To do so, we use the "web" database of the LIBCVM package (w8a) from Platt (1999). This dataset is split into training instances (49,749 rows) and test ones (14,951 rows). The dimension of the data is 300, whereas the 301^{th} feature is the label for classification.

To predict the class of our observations we minimize the corresponding logistic loss function to which we add a Ridge regularization term (L^2) as we are in a high-dimensional context, yet not sparse.

$$L(\mathbf{w}) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \mathbf{w}^T x_i)) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

where x_i is the vector of features for the observation i and $y_i \in \{-1, 1\}$ is the class label. In the first subsection, we analyze the different hyper-parameters of the Adam algorithm and their influences on the convergence speed. In the second subsection, we confront Adam to its competitors and predecessors, and we discuss the results.

4.1 Adam parameters

After a grid search on the parameters, we chose their default value to be used for all the experiments: $\alpha = 0.01$, $\beta_1 = \beta_2 = 0.9$, $\lambda = 10$, $\epsilon = 10^{-10}$ and a batch size of 25. We then change independently the different parameters, leaving the others unchanged.

For each experiment, we train the model 4 times and plot the average loss over the trials, as well as a confidence interval around it, to remove the randomization introduced by the stochastic nature of the sampling while computing the gradient.

4.1.1 Learning rate

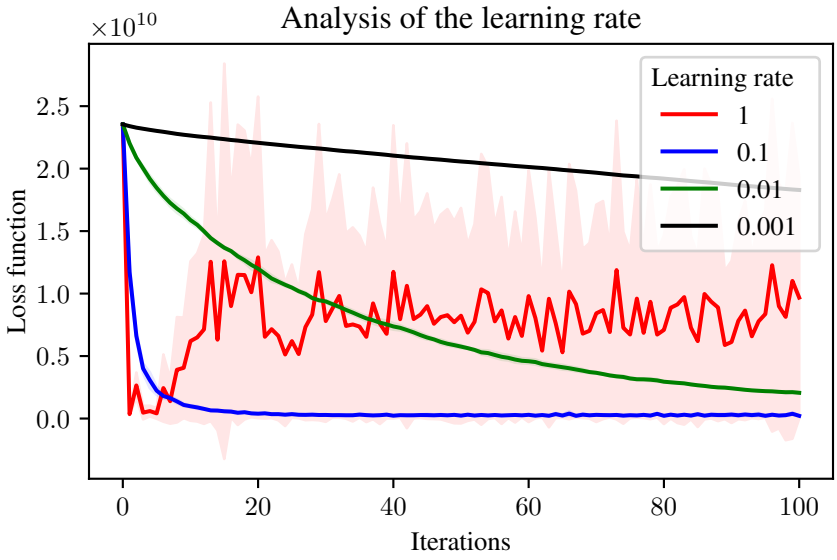

Figure 1: Different learning rates used in Adam.

When training the logistic regression algorithm with Adam as an optimizer, we study the influence of the learning rate α in figure 1. On one hand, for large values of the learning rate, the steps taken by SGD are too large and thus the algorithm fails to converge. On the other hand, if the learning rate is too small, the algorithm is slow to converge, requiring many more iterations.

Therefore, one can typically observe the trade-off between convergence warranty and convergence speed as a larger learning rate could either make convergence faster or make the algorithm diverge. In our setting, a learning rate in between 0.1 and 0.01 seems to be a reasonable choice.

4.1.2 Batch size

As it could be expected, the larger the batch size, the faster the convergence. As a matter of fact, when the batch size increases up to the size of the dataset, mini-batch gradient descent becomes equivalent to batch gradient descent, which is faster to converge than purely stochastic gradient descent or mini-batch. Yet, although the convergence is faster as it requires fewer iterations, the

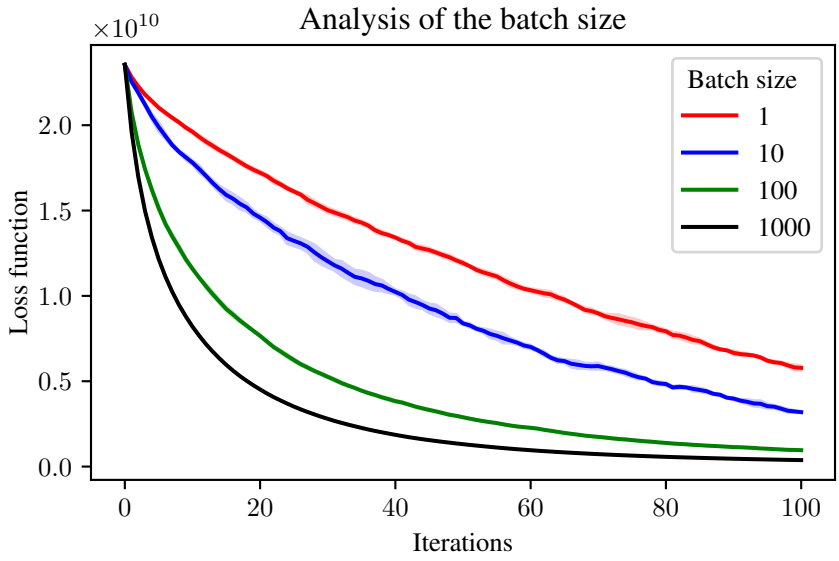

Figure 2: Different batch sizes used in Adam.

number of computer operations increases. Indeed, one can see that in figure 2, the convergence does not scale up with the bath size. 100 iterations with a batch size of 100 achieves better performance than 10 iterations with a batch size of 1000. This is explained by the stochastic convergence properties of SGD and why mini-batch stochastic gradient descent is so powerful.

4.1.3 Momentum decay rate: β_1

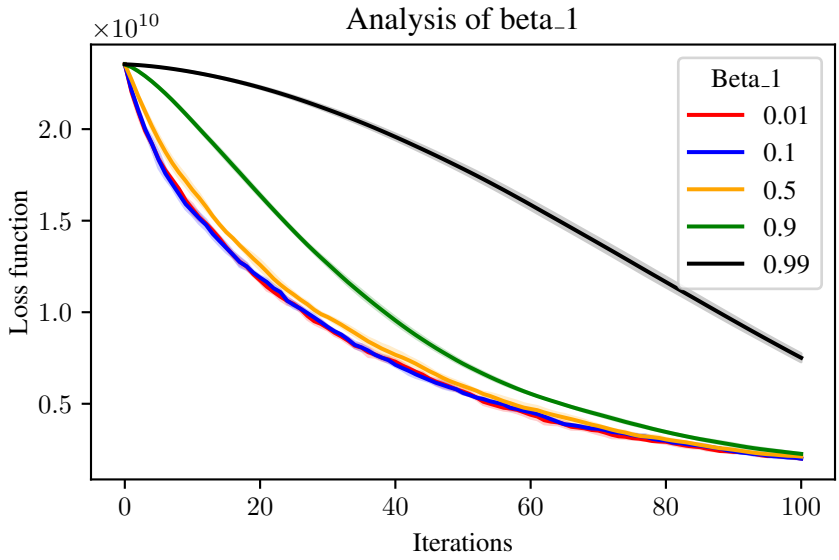

Figure 3: Different values of β_1 used in Adam.

The momentum decay rate β_1 controls the behavior of the moving average of the gradients. The higher the decay rate, the more the first moment estimate takes into account past values of the gradient. If the decay rate is small, the algorithm resembles Vanilla SGD as the momentum converges to the gradient itself and does not take past values of the gradient into account.

In figure 3, one can see that the best convergence is obtained for small values of the momentum decay rate β_1. This can be explained by the convex setting of the logistic regression problem. Since here, the objective function is smooth and convex, there is no need for a momentum and simply using the gradient at each time step is actually more efficient. Thus, a value of 0 for β_1 would actually be an acceptable choice here. However, in non-convex settings such as neural networks training, the momentum makes convergence faster and a value close to 1 for β_1 achieves better performance.

4.1.4 Speed decay rate: β_2

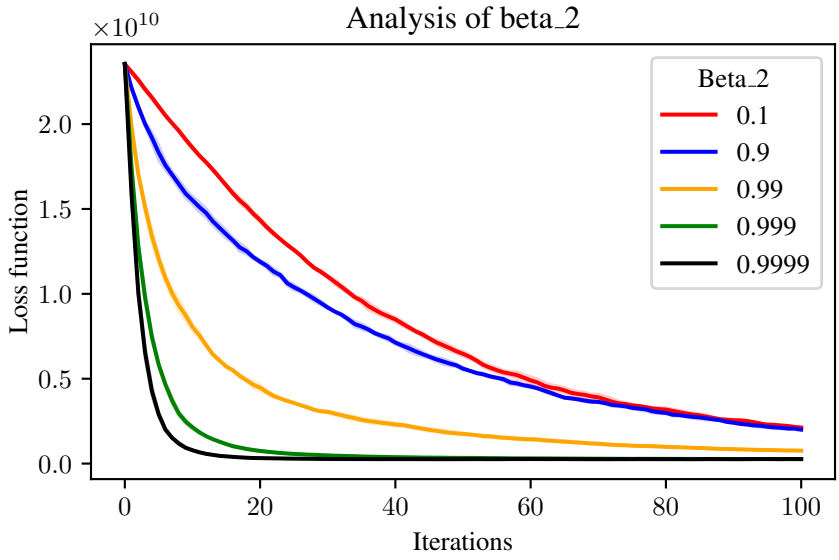

Figure 4: Study of β_2.

The second moment decay rate β_2 controls the behavior of the moving average of the squared gradients. As one can see in figure , the speed of convergence increases with β_2. Although high values are often used in general, here it seems that the closer to 1, the better. This can be explained by the convex shape of the objective function. In general, a high value of β_2 is recommended. However, for highly non-convex settings, if β_2 is too large, the learning direction does not adapt to current values of the squared gradient, which can cause a decrease in the convergence speed.

4.1.5 Effect of debiasing

We study the influence of debiasing terms for both first and second moment estimates of the gradient. The theoretical explanation for both debiasing terms is given in section 3. As shown in figure 6, keeping a bias toward the initial values of the gradient and its square actually makes convergence faster here. Once again, this may arise from the convex setting of the problem, where a caution for debiasing moment estimates is not needed since convergence in a convex setting happens naturally. It was showed by the authors in the original paper that this bias-correction is important for non-convex problems.

4.2 Other optimizers as special cases of Adam

We finally compare Adam with other famous optimizers: Vanilla mini-batch gradient descent, AdaGrad, RMSProp and Momentum. We use the same baseline hyper-parameters as before and plot the evolution of the loss for the 5 optimizers in order to compare their convergence speed. We also add to the comparison the Adam version without the debiasing terms.

The results in figure 6 are rather surprising. Even if Adam is known in the literature to be the best optimizer in most situations, we clearly see that it is not the case in our problem. There are three important points to note here. First, both Momentum and Vanilla mini-batch have the same

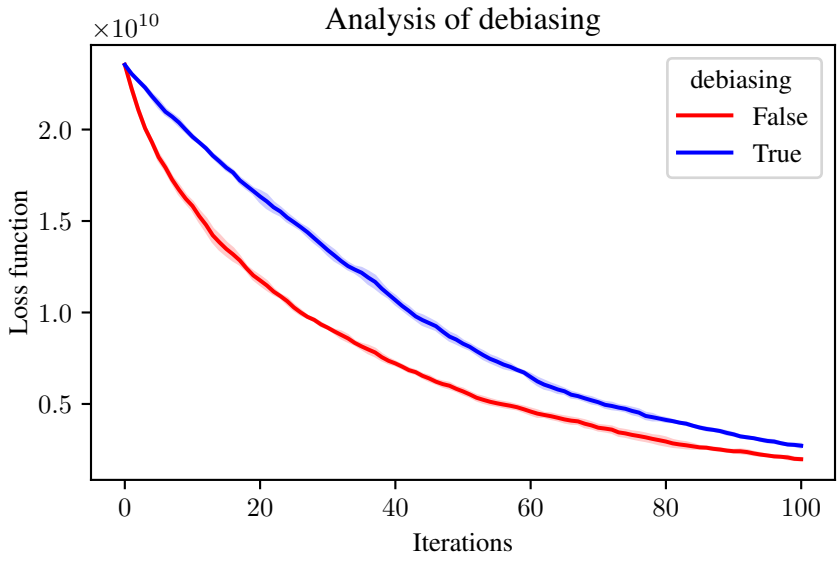

Figure 5: Study of debiasing.

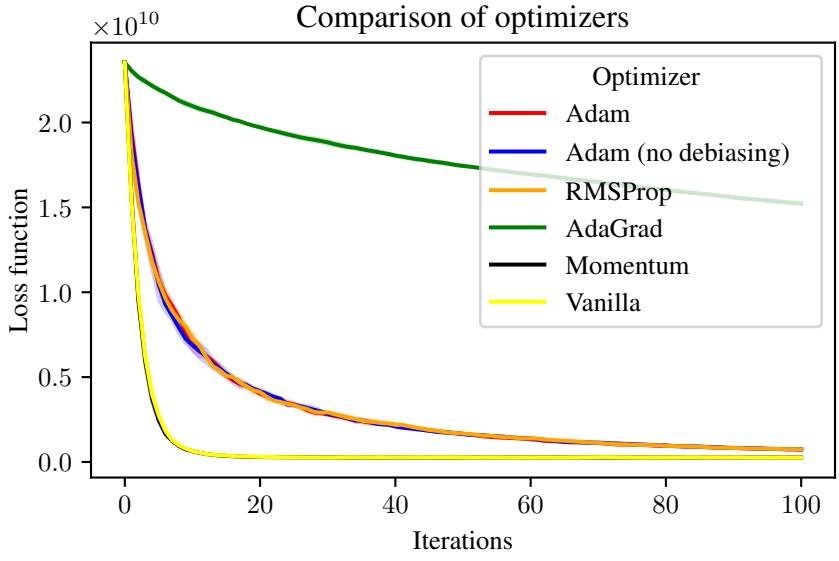

Figure 6: Study of debiasing.

convergence speed and were substantially better than Adam and the rest of the optimizers. Secondly, Adam and RMSProp have the same performance. Whereas AdaGrad performs poorly compared to all other optimizers.

These three observations can actually be explained by the strong convexity of the logistic regression problem. The loss function being convex, there is only one local minimum which is thus global. The use of an adaptive learning rate in order to "jump over" local minima or to prevent from overshooting the minimum is useless in this case. Therefore RMSProp and Adam, which both use speed term correction are actually slowed down by it, whereas it is in fact not necessary.

Vanilla mini-batch and Momentum have both a constant learning rate and perform better here. AdaGrad suffers from a poorly conditioned dataset, making it converge very slowly. Indeed, if A is the

dataset, it is known that if $\kappa(A^TA) = \frac{|\lambda_{max}(A^TA)|}{|\lambda_{min}(A^TA)|}$ is high, AdaGrad has slow convergence speed. In the case studied here, $\kappa = 1.7 \times 10^{17}$, considered as very high and explaining the poor performance of AdaGrad.

5 CONCLUSION

We have presented Adam, an optimizer for stochastic optimization that is vastly used in machine learning and deep learning. We have shown that this algorithm is an improved version of previous methods by combining RMSProp and Momentum update rules and adding a bias-correction term. It is important to note that Adam has inspired more recent optimizers that achieve state-of-the-art convergence speed, such as AdaMax which is generalization of Adam with the infinity norm of the gradient or Nadam, a combination of Adam and Nesterov SGD, not presented in this report. We have eventually shown that when comparing optimizers, one must carefully choose the setting in which to make the comparison since the performance of optimizers may depend on the level of convexity of the problem.

ACKNOWLEDGMENTS

We would like to thank our professors Laurent Le Brusquet[1], Paul-Henry Cournède[2] and Arnaud Gloaguen[1] for their support and their teaching. We also thank the ICLR team for providing us with their 2021 submission template upon which this article is written, although there is no aim of publishing. The code used for the experiments is available on GitHub[3].

REFERENCES

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

John Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods: Support Vector Learning*, pp. 185–208, 02 1999.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[1]L2S, CentraleSupélec, France

[2]MICS, CentraleSupélec, France

[3]https://github.com/clement-bonnet/lab-works/blob/main/optimization/project/code.ipynb