World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:17, No:2, 2023

On a Conjecture Regarding the Adam Optimizer

Mohamed Akrout, Douglas Tweed

Abstract—The great success of deep learning relies on efficient optimizers, which are the algorithms that decide how to adjust network weights and biases based on gradient information. One of the most effective and widely used optimizers in recent years has been the method of adaptive moments, or Adam, but the mathematical reasons behind its effectiveness are still unclear. Attempts to analyse its behaviour have remained incomplete, in part because they hinge on a conjecture which has never been proven, regarding ratios of powers of the first and second moments of the gradient. Here we show that this conjecture is in fact false, but that a modified version of it is true, and can take its place in analyses of Adam.

Keywords—Adam optimizer, Bock's conjecture, stochastic optimization, average regret.

I. Introduction

IN deep learning, a procedure called backpropagation computes the gradient of a loss or objective function with respect to the network's weights and biases. That gradient information is then passed to an algorithm called an *optimizer*, which decides how best to adjust those weights and biases in order to reduce the loss. For years now, one of the most popular optimizers has been Kingma and Ba's method of *adaptive moments*, or Adam [1]. This method has been favoured owing to its simplicity and its relatively rapid and reliable performance in challenging real-world applications. But the mathematical reasons for Adam's success remain unclear.

When Kingma and Ba first introduced the Adam method in [1], they tried to explain its effectiveness mathematically by proving that it zeroed an error-measure known as *average regret*, in a learning task called *online convex optimization* [2]. Rubio [3] and Bock et al. [4] found mistakes in the proof, and Bock et al. [4] managed to repair most of them, but they could not verify one key statement, called Lemma 10.4 in [1]] and Conjecture 4.2 in [4]. We will show that this conjecture is in fact false, but that a modified version of it does hold. This modified version generalizes an earlier result proven by Reddi and colleagues for their AMSGrad optimizer [5], so our result can replace Bock's Conjecture in analyses of most common variants of Adam.

For tractability, analyses of Adam typically use versions of the algorithm that are slightly different from the one generally employed in deep learning. Here, we will use the version laid out in Algorithm 1, which differs from that of Kingma and Ba [1], and Bock et al. [4] only in that they set $\lambda_m = \lambda_g \in (0, 1)$. We will explain the significance of this difference where it becomes relevant.

In this algorithm, each of the variables g_t, m_t, v_t, $\widehat{m}_t$, $\widehat{v}_t$, and θ_t is a real-valued vector; for instance g_t is the g-vector at time t. But the vector operations in Adam are all element-wise, except possibly in line 2, and therefore we can analyse the parts after that line element-wise, i.e. we can assume throughout this paper that the vectors g_t etc. have one element each (except in Section IV, where calculations of regret depend on non-element-wise operations outside Adam). We will write $g_{1:T}$ for the T-element vector $[g_1, g_2, ..., g_T]$. We also define

$$x_1 \triangleq 1 - \beta_1, \;\; x_2 \triangleq 1 - \beta_2, \tag{1}$$

and

$$s_T \triangleq \sum_{t=1}^{T} \frac{\widehat{m}_t^2}{\sqrt{t\,\widehat{v}_t}}, \tag{2}$$

which is central to Bock's conjecture. We assume $g_1 \neq 0$ because otherwise s_T is undefined.

We can now state:

Bock's conjecture. *In Algorithm* 1*, if* $\lambda_m = \lambda_g \in (0, 1)$ *and* $\gamma = \beta_1^2/\sqrt{\beta_2} < 1$ *then for any* $g_{1:T}$ *we have*

$$s_T \leq \frac{2}{(1-\gamma)} \frac{1}{\sqrt{1-\beta_2}} \left\| g_{1:T} \right\|_2 . \tag{3}$$

We will call the right-hand side of this inequality the *Kingma-Ba* or *K-B bound*.

Algorithm 1 Adam optimizer

Require: $\eta > 0; \beta_1, \beta_2 \in (0, 1)$; $\lambda_m, \lambda_g \in (0, 1]$; duration $T \in \mathbb{Z}^+$; initial parameter (weight and bias) vector θ_0; convex differentiable loss functions $f_t(\theta)$; $m_0, v_0 = 0$.

Return: updated parameter vector θ_T.

1: **for** $t = 1$ to T **do**
2: $g_t = \nabla_\theta f_t(\theta_{t-1})$
▷ Compute biased moment estimates
3: $m_t = \beta_1 \lambda_m^{t-1} m_{t-1} + (1 - \beta_1 \lambda_g^{t-1}) g_t$
4: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
▷ Bias-correct the moment estimates
5: $\widehat{m}_t = m_t/(1 - \beta_1^t)$
6: $\widehat{v}_t = v_t/(1 - \beta_2^t)$
▷ Update the parameters
7: $\theta_t = \theta_{t-1} - (\eta/\sqrt{t})\ \widehat{m}_t/\sqrt{\widehat{v}_t}$
8: **end**

II. Counterexample to Bock's Conjecture

We consider vectors $g_{1:T}$ where $g_t > 0\ \forall t$. We set β_1 and β_2 equal, i.e. $\beta_1 = \beta_2 = \beta$, and observe that s_T and the K-B bound are right-continuous functions of β at $\beta = 0$. So if we can find a counterexample where $\beta = 0$ then there also exist counterexamples where $\beta \in (0, 1)$.

M. Akrout is with AIP Labs, Budapest, Hungary.
D. Tweed is with the Department of Physiology, University of Toronto, Toronto, Canada (corresponding author, e-mail: douglas.tweed@utoronto.ca).

Letting $\beta \to 0$, we get $\widehat{m}_t = g_t$ and $\widehat{v}_t = g_t^2$ (from lines 3–6 of Algorithm 1) and $\gamma = \beta^{3/2} = 0$. Bock's conjecture then takes the form

$$\sum_{t=1}^{T} \frac{g_t}{\sqrt{t}} \le 2 \left\| g_{1:T} \right\|_2 .$$

If we choose $g_t = 1/\sqrt{t}$ then this inequality becomes

$$\sum_{t=1}^{T} \frac{1}{t} \le 2 \sqrt{\sum_{t=1}^{T} \frac{1}{t}},$$

which is false when the left-hand side > 4, as happens when $T > 30$.

By continuity, (3) is also violated in cases where $\beta \in (0, 1)$. Fig. 1 shows an example.

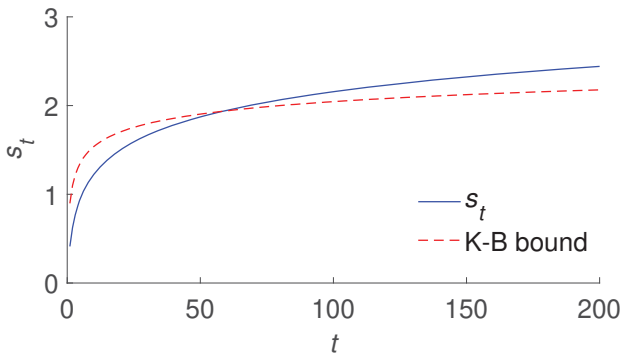

Fig. 1 Evaluation of s_t in Algorithm 1 for 200 time steps using $\beta_1 = \beta_2 = 0.1$, $\lambda_m = \lambda_g = 1 - 10^{-8}$, and $g_t = 1/\sqrt{t}$

III. Modifying Bock's Conjecture

We want to replace the K-B bound on the right-hand side of (3) with a different bound that we can verify, at least for values of β_1 and β_2 that are typically used in AI applications of Adam.

Lemma 1. *In Algorithm* 1*, if* $\lambda_m = \lambda_g = 1$*,* $\rho = \beta_2/\beta_1^2 \in (1, 2)$*, and* $K = \rho/(\rho - 1)$ *then* $\forall t \in [1, \infty)$

$$\frac{m_t^2}{v_t} < K \frac{x_1^2}{x_2}. \tag{4}$$

Proof: By induction:

(i) At $t = 1$, we have

$$\frac{m_1^2}{v_1} = \frac{x_1^2 \, g_1^2}{x_2 \, g_1^2} = \frac{x_1^2}{x_2},$$

and so

$$\frac{m_1^2}{v_1} < K \frac{x_1^2}{x_2}$$

because $K > 1$.

(ii) Next we show that if (4) holds at any time t then it still holds at $t + 1$, i.e.:

$$\frac{m_{t+1}^2}{v_{t+1}} - K \frac{x_1^2}{x_2} < 0, \tag{5}$$

or equivalently,

$$m_{t+1}^2 - K \frac{x_1^2}{x_2} v_{t+1} \; < 0.$$

If we substitute the formulas for m_{t+1} and v_{t+1} from lines 3 and 4 of Algorithm 1, and use the definitions in (1), the left-hand side becomes

$$(\beta_1 \, m_t + x_1 g_{t+1})^2 - K \frac{x_1^2}{x_2} (\beta_2 \, v_t + x_2 g_{t+1}^2). \tag{6}$$

We expand the squared sum, rearrange, and apply (4) to see that (6) is less than

$$\beta_1^2 \, m_t^2 + 2 \, \beta_1 \, m_t \, x_1 \, g_{t+1} - (K - 1) \, x_1^2 \, g_{t+1}^2 - \beta_2 \, m_t^2.$$

We break up the first addend into a sum of two terms to get

$$\frac{K}{K-1} \beta_1^2 \, m_t^2 - \frac{1}{K-1} \beta_1^2 \, m_t^2 + 2 \, \beta_1 \, m_t \, x_1 \, g_{t+1} \\ -(K - 1) \, x_1^2 \, g_{t+1}^2 - \beta_2 \, m_t^2,$$

which is

$$\left(\frac{K}{K-1} \beta_1^2 - \beta_2 \right) m_t^2 \\ - \left(\frac{1}{\sqrt{K-1}} \beta_1 \, m_t - \sqrt{K-1} \, x_1 \, g_{t+1} \right)^2 .$$

By the definition of K, the top line here equals 0, and the quantity as a whole ≤ 0, proving (5). ■

Lemma 2. *In Algorithm* 1*, if* $\lambda_m = \lambda_g = 1$ *and* $\beta_2 \ge 2 \, \beta_1 - \beta_1^2$ *then* $\forall t \in [1, \infty)$

$$\frac{\widehat{m}_t^2}{\widehat{v}_t} \le \frac{m_t^2}{v_t}.$$

Proof: From lines 5 and 6 of Algorithm 1 we have

$$\frac{\widehat{m}_t^2}{\widehat{v}_t} = c_t \frac{m_t^2}{v_t}, \quad \text{where} \quad c_t \triangleq \frac{1 - \beta_2^t}{(1 - \beta_1^t)^2}.$$

Note that $c_1 = x_2/x_1^2$, which is ≤ 1 when $\beta_2 \ge 2 \, \beta_1 - \beta_1^2$. To prove that $c_t \le 1 \; \forall t \in [1, \infty)$, we define this function of continuous time:

$$h(t) \triangleq (1 - \beta_2^t) - (1 - \beta_1^t)^2. \tag{7}$$

We will show that $\forall t \in [1, \infty)$

$$\frac{dh}{dt} \le 0 \text{ whenever } h(t) = 0,$$

because that means $h(t)$, starting at $h(1) = x_2 - x_1^2 \le 0$, can never cross over to any positive value, and therefore c_t stays ≤ 1.

We have

$$\frac{dh}{dt} = - \log \beta_2 + (1 - \beta_2^t) \log \beta_2 + 2 \, \beta_1^t \, (1 - \beta_1^t) \log \beta_1,$$

and if $h(t) = 0$,

$$\frac{dh}{dt} - \log\beta_2 + (1-\beta_1^t)^2 \log\beta_2 + 2\,\beta_1^t\,(1-\beta_1^t)\log\beta_1,$$

because then $(1-\beta_2^t) = (1-\beta_1^t)^2$ by the definition in (7).

We define $\alpha \triangleq 1-\beta_1^t$ to get, $\forall \alpha \in [x_1, 1)$,

$$\begin{aligned}\frac{dh}{dt} &= -\log\beta_2 + \log\beta_2\,\alpha^2 + 2\,\log\beta_1\,\alpha\,(1-\alpha)\\ &= \log\beta_2\,\underbrace{\Big((1-r)\,\alpha^2 + r\,\alpha - 1\Big)}_{P(\alpha)},\end{aligned} \tag{8}$$

where $r \triangleq 2\,\log\beta_1/\log\beta_2$, which > 1 when $\beta_2 \geq 2\,\beta_1 - \beta_1^2$. Because $r > 1$, the polynomial $P(\alpha)$ is concave down. It follows that $P(\alpha) \geq 0$ on $[x_1, 1)$, because $P(1) = 0$ and $P(x_1) \geq 0$ by the conditions on β_1 and β_2 (see the Appendix). Therefore by (8), $dh/dt \leq 0$ at any $t \in [1, \infty)$ where $h(t) = 0$, which means h can never cross 0 and c_t can never exceed 1. ■

Result 1. *In Algorithm* 1*, if* $\lambda_g = 1$*,* $\beta_2 < 2\beta_1^2$*,* $\beta_2 \geq 2\,\beta_1 - \beta_1^2$*,* $K = \beta_2/(\beta_2 - \beta_1^2)$ *as in Lemma 1, and* $\tau = \lfloor -log(2)/log(\beta_1) \rfloor$ *then* $\forall T \in [1, \infty)$

$$s_T < (2+\sqrt{\tau})\sqrt{1 + K\,\frac{x_1^2}{x_2}\,\log T}\quad \|g_{1:T}\|_2\,. \tag{9}$$

Proof: We may assume that $\|g_{1:T}\|_2 = 1$, as s_T is a homogeneous function of degree 1 of $g_{1:T}$; that is, if we multiply every element of $g_{1:T}$ by a constant, ζ, then the effect on s_T is to multiply it by ζ as well. We can also say that $g_t \geq 0\ \forall t \in [1, \infty)$, as we are seeking an *upper* bound for s_T, and given any $g_{1:T}$ with negative elements, we could always increase s_T by flipping the signs of those negative g_t. And we can assume that $\lambda_m = 1$, because any $\lambda_m \in (0, 1)$ would only shrink s_T, as is clear from (2), lines 3 and 5 of Algorithm 1, and the non-negativity of all the g_t.

The definition of s_T in (2) shows that it is the dot product of two vectors:

$$s_T = \widehat{m}_{1:T} \cdot \mu_{1:T}, \tag{10}$$

where $\mu_{1:T}$ is the vector with elements $\mu_t = \widehat{m}_t/\sqrt{t\widehat{v}_t}$.

The first vector in this dot product has a bounded 2-norm. First of all,

$$\|m_{1:T}\|_2 \leq \|g_{1:T}\|_2 = 1$$

because $m_{1:T}$ is an exponential moving average of $g_{1:T}$, and the 2-norm of such an average cannot exceed the 2-norm of its input. Then we get each $\widehat{m}_t$ by multiplying m_t by the factor $1/(1-\beta_1^t)$. For all $t > \tau$, those factors are < 2, so $\|\widehat{m}_{\tau+1:T}\|_2 < 2\,\|m_{\tau+1:T}\|_2 < 2$. And $\|\widehat{m}_{1:\tau}\|_2 \leq \sqrt{\tau}$ because $\widehat{m}_t \leq 1\ \forall t \in [1, \infty)$. Therefore

$$\|\widehat{m}_{1:T}\|_2 < 2 + \sqrt{\tau}. \tag{11}$$

The second vector in the dot product (10) also has a bounded 2-norm. Using the definition of the norm and then Lemmas 1 and 2, we get

$$\begin{aligned}\|\mu_{1:T}\|_2^2 = \sum_{t=1}^{T}\frac{\widehat{m}_t^2}{t\,\widehat{v}_t} &\leq 1 + \sum_{t=2}^{T}\frac{1}{t}\left(K\,\frac{x_1^2}{x_2}\right)\\ &\leq 1 + \left(K\,\frac{x_1^2}{x_2}\right)\int_{t=1}^{T}\frac{1}{t}\,dt\\ &= 1 + K\,\frac{x_1^2}{x_2}\,\log T,\end{aligned}$$

and

$$\|\mu_{1:T}\|_2 \leq \sqrt{1 + K\,\frac{x_1^2}{x_2}\,\log T}. \tag{12}$$

Therefore by (10)-(12), and the Cauchy-Schwarz inequality, we have (9). ■

The range of β values that is permissible, given the conditions in Result 1, is shown in green in Fig. 2. For instance if $\beta_1 = 0.9$ then we must have $\beta_2 \in [0.99, 1)$. This range includes the β values most commonly used in deep learning.

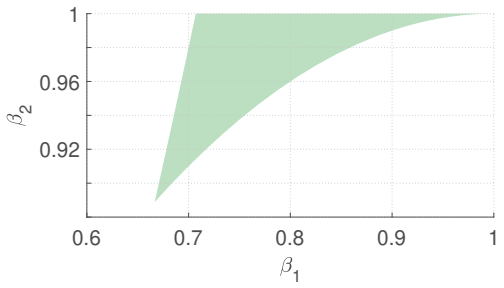

Fig. 2 The range of β values that guarantees the upper bound on s_T in (9)

IV. Analysing Adam

Our Result 1 generalizes an earlier result proven by Reddi and colleagues, namely Lemma 2 in [5]. Our methods of proof are quite different, but both approaches lead to bounds involving the quantity $\log T$ under a square-root sign. Reddi et al. [5] proved their result for their AMSGrad optimizer, whereas ours holds for AMSGrad and for most or all common varieties of Adam itself, with or without bias correction (lines 5 and 6 of Algorithm 1) and with or without λ variables (line 3 of Algorithm 1).

We also cover in Result 1 a wider range of optimizers than does Reddi and colleagues' Lemma 2 [5], but both their result and ours are very generally applicable in other ways. First, they both hold in a setting of online optimization that places no conditions except boundedness on the sequence of gradients, $g_{1:T}$. Second, they do not assume that the functions to be learned are convex, which is important if we aim to derive any conclusions about an optimizer's performance in deep learning, where the loss landscape is usually far from convex, even in a small local neighbourhood of the network's parameter vector. Third, both results concern processes that precede the parameter adjustments in the network, and consequently they

make no assumptions about those adjustments. In particular, they do not require a mechanism that shrinks the learning rate factor as a function of time, as in line 7 of Algorithm 1. This point matters because even though most current analyses of Adam do require that the learning rate factor shrink with time, nonetheless in real deep-learning applications, performance is better without shrinkage.

All of this generality may be valuable, as we are unlikely to understand Adam or other Adam-type optimizers until we analyse them in settings other than convex optimization. Recent analyses in that convex setting have revealed a great many interesting properties of Adam-type optimizers, but nothing so far that explains these optimizers' outstanding performance in deep learning. For example, one of the strongest results yet achieved is Reddi and colleagues' proof [5] that AMSGrad zeroes average regret in the setting of online convex optimization. But its proven rate of convergence is not as good as that of simple gradient descent [2], so this finding does not yet explain why AMSGrad works so much better than gradient descent in deep learning. For Adam, the case is even worse, as Reddi et al. [5] and Bock and Weiss [6] have shown examples where Adam fails to zero the average regret. Strictly, Reddi and colleagues' [5] example was of a failure when network parameters are optimized not by Adam alone but by Adam together with projection into a feasible set, but by adding weight decay it is straightforward to create a version of their example where pure Adam, without projection, also fails to zero the average regret.

Another recent positive result is the proof by Bock and Weiss [7] that Adam converges locally, meaning roughly that if it ever gets inside a convex neighbourhood of an optimum in parameter space then it will converge to that optimum. But again, the same is true of simple gradient descent, so this result shows only that Adam is as good as gradient descent in this respect, not that it is better.

Overall, then, the message seems to be that Adam is inferior to gradient descent in the setting of online convex optimization (OCO). The choice to analyse Adam in that setting goes back to Kingma and Ba [1], and it was reasonable because OCO shares with deep learning the crucial feature that the gradients change unpredictably from moment to moment. But there are disanalogies, because in deep learning the gradients vary for two distinct reasons. First, they fluctuate from minibatch to minibatch. This effect, known as gradient noise, is probably well modelled by the random gradients of OCO. But second, the gradients of deep learning also drift as the network moves into new regions of parameter space where the local geometry of the loss function is different. So even without gradient noise — even with whole-batch as opposed to minibatch learning — the gradients would still vary unpredictably. This feature is not reflected in the OCO setting, and its absence may be preventing Adam-type optimizers from displaying their true worth. For instance, Reddi et al. [5] have shown that the reason Adam can fail in OCO is that its memory for past gradients is, in a certain sense, too short. But in deep learning, a short memory may let Adam discard gradient information that is obsolete because it belongs to regions of parameter space from which the network has already moved away.

V. Conclusion

Our upper bound on s_T in (9) can replace the Kingma-Ba bound in analyses of the Adam optimizer.

Appendix

In our proof of Lemma 2, we said that $P(x_1) \geq 0$, where P was the polynomial in (8), i.e.

$$(1-r)\,x_1^2 + r\,x_1 - 1 \geq 0, \tag{13}$$

where $r \triangleq 2\log\beta_1/\log\beta_2$. To verify (13), we observe that it is equivalent to

$$r \geq \frac{1-x_1^2}{x_1-x_1^2},$$

which, by the definition of r, is in turn equivalent to

$$\beta_2 \geq \beta_1^{\left(1-\frac{\beta_1}{2-\beta_1}\right)}.$$

Now given that $\beta_2 \geq 2\,\beta_1 - \beta_1^2$ in Lemma 2, it will suffice to show that

$$2\,\beta_1 - \beta_1^2 \geq \beta_1^{\left(1-\frac{\beta_1}{2-\beta_1}\right)},$$

i.e.

$$y(\beta_1) \triangleq \beta_1^{\frac{\beta_1}{\beta_1-2}} + \beta_1 - 2 \leq 0, \tag{14}$$

for $\beta_1 \in (0,1)$.

Straightforward calculations show that

$$y(1) = 0, \quad \frac{dy}{d\beta_1}(1) = 0, \quad \frac{d^2y}{d\beta_1^2}(1) = -2,$$

i.e. $y(1)$ is a strict local maximum.

To see that $y \leq 0$ in $(0,1)$, we compute

$$\frac{dy}{d\beta_1} = \beta_1^{\frac{\beta_1}{\beta_1-2}}\left(\frac{-2\log\beta_1 + \beta_1 - 2}{(2-\beta_1)^2}\right) + 1 \tag{15}$$

and observe that if y were 0 at any $\beta_1 \in (0,1)$, then by (14) the term

$$\beta_1^{\frac{\beta_1}{\beta_1-2}}$$

would $= 2-\beta_1$, and (15) would become

$$\frac{dy}{d\beta_1} = \frac{-2\log\beta_1}{2-\beta_1} > 0.$$

So if y were ≥ 0 at any $\beta_1' \in (0,1)$ then it would stay ≥ 0 on $(\beta_1', 1)$, contradicting the fact that $y(1)$ is a strict local maximum. Therefore y must remain ≤ 0 in $(0,1)$, confirming (14) and (13).

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[2] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.

[3] David Martínez Rubio. Convergence analysis of an adaptive method of gradient descent. *University of Oxford, Oxford, M. Sc. thesis*, 2017.

[4] Sebastian Bock, Josef Goppold, and Martin Weiß. An improvement of the convergence proof of the adam-optimizer. *arXiv preprint arXiv:1804.10587*, 2018.

[5] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[6] Sebastian Bock and Martin Weiß. Non-convergence and limit cycles in the adam optimizer. In *International Conference on Artificial Neural Networks*, pages 232–243. Springer, 2019.

[7] Sebastian Bock and Martin Georg Weiß. Local convergence of adaptive gradient descent optimizers. *arXiv preprint arXiv:2102.09804*, 2021.