On a Conjecture Regarding the Adam Optimizer

Mohamed Akrou, Douglas Tweed

Abstract—Why does the Adam optimizer work so well in deep-learning applications? Adam’s originators, Kingma and Ba, presented a mathematical argument that was meant to help explain its success, but Bock and colleagues have since reported that a key piece is missing from that argument — an unproven lemma which we will call Bock’s conjecture. Here we show that this conjecture is false, but we prove a modified version of it — a generalization of a result of Reddi and colleagues — which can take its place in analyses of Adam.

I. INTRODUCTION

Kingma and Ba [1] tried to prove that their Adam optimizer zeroed the 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. managed to repair most of them, but they could not verify one key statement, called Lemma 10.4 in Kingma and Ba’s paper and Conjecture 4.2 in Bock’s. 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 vectors 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, Ba, and Bock et al. 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 \( g_t, m_t, v_t, \hat{m}_t, \hat{v}_t, \) and \( \theta_t \) 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 4, where calculations of regret depend on non-element-wise operations outside Adam). We will write \( g_{1:T} \) for the \( T \)-element vector \( [g_1, g_2, \ldots, g_T] \). We also define

\[
x_1 = 1 - \beta_1, \quad x_2 = 1 - \beta_2,
\]

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 \( \theta_0 \); convex differentiable loss functions \( f_t(\theta) \); \( m_0, v_0 = 0 \).

Return: updated parameter vector \( \theta_T \).

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

and

\[
s_T = \sum_{i=1}^{T} \frac{\hat{m}_i^2}{\sqrt{\hat{v}_i}}, \quad (2)
\]

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 \( \gamma = \beta_1^t / \sqrt{\beta_2} < 1 \) then for any \( g_{1:T} \) we have

\[
s_T \leq \frac{2}{(1 - \gamma) \sqrt{1 - \beta_2}} \| g_{1:T} \|_2 . \quad (3)
\]

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

II. COUNTEREXAMPLE TO BOCK’S CONJECTURE

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

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

\[
\]
\[
\sum_{t=1}^{T} \frac{g_t}{\sqrt{t}} \leq 2 \|g_{1:T}\|_2.
\]

If we choose \(g_t = 1/\sqrt{t}\) then this inequality becomes
\[
\sum_{t=1}^{T} \frac{1}{t} \leq 2 \sum_{t=1}^{T} \frac{1}{t},
\]
which is false when the left-hand side \(> 4\), as happens when \(T > 30\).

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

![Plot showing example of inequality violation](image)

**Fig. 1.** We ran Algorithm 1 for 200 time steps using \(\beta_1 = \beta_2 = 0.1\), \(\lambda_m = \lambda_g = 1 - 10^{-8}\), and \(g_t = 1/\sqrt{t}\), and we computed the K-B bound and \(s_t\) at each step. \(s_t\) surpassed the bound at \(t = 50\).

### 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 \(\beta_1\) and \(\beta_2\) that are typically used in AI applications of Adam.

**Lemma 1.** In Algorithm 1 if \(\lambda_m = \lambda_g = 1\) and \(\beta_2 \geq 2 \beta_1 - \beta_1^2\) then \(\forall t \in [1, \infty)\)
\[
\frac{m_t^2}{v_t} < K \frac{x_t^2}{x_2}.
\]

**Proof:** By induction:

(i) At \(t = 1\), we have
\[
\frac{m_1^2}{v_1} = \frac{x_1^2 g_1^2}{x_2 g_1} = \frac{x_1^2}{x_2},
\]
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.:
\[
\frac{m_{t+1}^2}{v_{t+1}} - K \frac{x_{t+1}^2}{x_2} < 0,
\]
or equivalently,
\[
\frac{m_{t+1}^2}{v_{t+1}} - K \frac{x_{t+1}^2}{x_2} < 0.
\]

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
\[
(\beta_1 m_t + x_t g_t + 1)^2 - K \frac{x_{t+1}^2}{x_2} (\beta_2 v_t + x_t g_t + 1).
\]

We expand the squared sum, rearrange, and apply (4) to see that (6) is less than
\[
\beta_1^2 m_t^2 + 2 \beta_1 m_t x_t g_t + 1 - K - 1) x_t^2 g_t + 1 - \beta_2 m_t^2.
\]

We break up the first addend into a sum of two terms to get
\[
\frac{K}{K - 1} \beta_1^2 m_t^2 - \frac{1}{K - 1} \beta_1^2 m_t^2 + 2 \beta_1 m_t x_t g_t + 1 - (K - 1) x_t^2 g_t + 1 - \beta_2 m_t^2,
\]
which is
\[
\left(\frac{K}{K - 1} \beta_1^2 - \beta_2\right) m_t^2 - \left(\frac{1}{K - 1} \beta_1 m_t - \sqrt{K - 1} x_t g_t\right)^2.
\]

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

**Lemma 2.** In Algorithm 1 if \(\lambda_m = \lambda_g = 1\) and \(\beta_2 \geq 2 \beta_1 - \beta_1^2\) then \(\forall t \in [1, \infty)\)
\[
\frac{\hat{m}_t^2}{\hat{v}_t} \leq \frac{m_t^2}{v_t}.
\]

**Proof:** From lines 5 and 6 of Algorithm 1 we have
\[
\hat{m}_t^2 = c_t \frac{m_t^2}{v_t}, \quad \text{where} \quad c_t \triangleq \frac{1 - \beta_t^2}{(1 - \beta_t^2)^2}.
\]

Note that \(c_t = x_t^2 / x_1^2\), which is \(\leq 1\) when \(\beta_2 \geq 2 \beta_1 - \beta_1^2\). To prove that \(c_t \leq 1\) \(\forall t \in [1, \infty)\), we define this function of continuous time:
\[
h(t) \triangleq (1 - \beta_t^2) - (1 - \beta_1^2)^2.
\]

We will show that \(\forall t \in [1, \infty)\)
\[
\frac{dh}{dt} \leq 0 \text{ whenever } h(t) = 0,
\]
because that means \(h(t)\), starting at \(h(1) = x_2 - x_1^2 \leq 0\), can never cross over to any positive value, and therefore \(c_t\) stays \(\leq 1\).

We have
\[
\frac{dh}{dt} = - \beta_2 + (1 - \beta_2^2) \log \beta_2 + 2 \beta_1 (1 - \beta_1^2) \log \beta_1,
\]
and if \( h(t) = 0 \),

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

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

We define \( \alpha \triangleq 1 - \beta_1^2 \) to get, \( \forall \alpha \in [x_1, 1) \),

\[
\frac{dh}{dt} = -\log \beta_2 + \log \beta_2 \alpha^2 + 2 \log \beta_1 (1 - \alpha)
\]

\[
= \log \beta_2 \left((1-r) \alpha^2 + r \alpha - 1\right),
\]

where \( r \triangleq 2 \log \beta_1 \log \beta_2 \), which means \( \beta > 1 \) 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 \( \beta_1 \) and \( \beta_2 \) (see the Appendix). Therefore by (8), \( \frac{dh}{dt} \leq 0 \) 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 3.1, and \( \tau = [-\log(2)/\log(\beta_1)] \) then \( \forall T \in [1, \infty) \)

\[
s_T < (2 + \sqrt{\tau}) \sqrt{1 + K \frac{x_2^2}{x_2} \log T} \| g_{1:T} \|_2. \]

**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, \( \zeta \), then the effect on \( s_T \) is to multiply it by \( \zeta \) 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 it 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 it, 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 (9) shows that it is the dot product of two vectors:

\[
s_T = \hat{m}_{1:T} \cdot \mu_{1:T}, \tag{10}
\]

where \( \mu_{1:T} \) is the vector with elements \( \mu_t = \hat{m}_t/\sqrt{v_t} \).

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 \( \hat{m}_t \) by multiplying \( m_t \) by the factor \( 1/(1 - \beta_1^2) \). For all \( t > \tau \), those factors are \( < 2 \), so \( \| \hat{m}_{\tau+1:T} \|_2 \leq 2 \| m_{\tau+1:T} \|_2 < 2 \). And \( \| \hat{m}_{1:T} \|_2 \leq \sqrt{T} \) because \( \hat{m}_t \leq 1 \ \forall t \in [1, \infty) \).

Therefore

\[
\| \hat{m}_{1:T} \|_2 < 2 + \sqrt{\tau}. \tag{11}
\]

The second vector in the dot product (10) also has a bounded 2-norm. Using the definition of the norm and then Lemmas 3.1 and 3.2, we get

\[
\| \mu_{1:T} \|_2^2 = \sum_{t=1}^{T} \frac{\hat{m}_t^2}{v_t^2} \leq 1 + \sum_{t=2}^{T} \frac{1}{t} \left( \frac{K x_2^2}{x_2} \right)
\]

\[
\leq 1 + \left( K \frac{x_2^2}{x_2} \right) \int_{t=1}^{T} \frac{1}{t} \ dt
\]

\[
= 1 + K \frac{x_2^2}{x_2} \log T,
\]

and

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

Therefore by (10), (11), (12), and the Cauchy-Schwarz inequality, we have (9).

The range of \( \beta \) values that is permissible, given the conditions in Result 3.3, is shown in green in the next picture. For instance if \( \beta_1 = 0.9 \) then we must have \( \beta_2 \in [0.99, 1) \). This range includes the \( \beta \) values most commonly used in deep learning.

![Fig. 2. Green shows the range of \( \beta \) values that guarantees the upper bound on \( s_T \) in (9).](image)

**IV. ANALYSING ADAM**

Our Result 3.3 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. 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 \( \lambda \) variables (line 3 of Algorithm 1).

So Result 3.3 covers a wider range of optimizers than does Reddi and colleagues’ Lemma 2, 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 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’ 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 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’ 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 3.2, we said that $P(x_1) \geq 0$, where $P$ was the polynomial in (8), i.e.

$$ (1-r)x_1^2 + r x_1 - 1 \geq 0, $$

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

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

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

$$ \beta_2 \geq \beta_1^{1-2 \log \beta_1 / \log \beta_2}. $$

Now given that $\beta_2 \geq 2 \beta_1 - \beta_1^2$ in Lemma 3.2, it will suffice to show that

$$ 2 \beta_1 - \beta_1^2 \geq \beta_1^{1-2 \log \beta_1 / \log \beta_2}, $$

i.e.

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

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

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

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

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

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

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

So if $y$ were $\leq 0$ at any $\beta_1^2 \in (0, 1)$ then it would stay $\leq 0$ on $(\beta_1^2, 1)$, contradicting the fact that $y(1)$ is a strict local maximum. Therefore $y$ must remain $\leq 0$ 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.