Gadam: Combining Adaptivity with Iterate Averaging Gives Greater Generalisation

Diego Granziol\(^*\), Xingchen Wan\(^*\), Stephen Roberts
Machine Learning Research Group
University of Oxford
Oxford, UK
{diego, xwan, sjrob}@robots.ox.ac.uk

Abstract

We introduce Gadam, which combines Adam and iterate averaging (IA) to significantly improve generalisation performance without sacrificing adaptivity. We argue using high dimensional concentration theorems, that the noise reducing properties of IA are particularly appealing for large deep neural networks trained with small batch sizes. We contrast and compare with popular alternatives, such as the exponentially moving average (EMA), batch size increases or learning rate decreases. Furthermore, under mild conditions adaptive methods enjoy improved pre-asymptotic convergence, hence in finite time we expect this combination to be more effective than SGD + IA. We show that the combination of decoupled weight decay and IA allows for a high effective learning rate in networks with batch normalisation, which exerts additional regularisation. For language tasks (PTB) we show that Gadam is superior to finely tuned SGD, SGD with IA and Adam by a significant margin. For various image classification tasks (CIFAR-10/100, ImageNet-32) Gadam is consistently superior to finely tuned SGD and its partially adaptive variant GadamX outperforms SGD with IA.

1 Introduction

Deep learning’s success across a wide variety of tasks, from speech recognition to image classification, has drawn wide-ranging interest in their optimisation, which aims for effective and efficient training, generalisation, which aims to improve its ability to infer on the unseen data. Adaptive optimisers, which invoke a per parameter learning rate, such as Adam (the most prolific) \(^1\), AdaDelta \(^2\) and RMSprop \(^3\) are popular deep learning optimisers, but are known to generalise worse compared to SGD \(^4\). Due to this, many state-of-the-art models, especially for image classification datasets such as CIFAR \(^5\) and ImageNet \(^6\)\(^7\) are still trained using Stochastic Gradient Descent (SGD) with momentum \(^8\). There has been much of the research interest in addressing this issue: For example, \(^9\) suggests dynamic switching between Adam and SGD; \(^10\) suggests Padam, a partially adaptive optimiser uniting Adam and SGD. However, the generalisation difference between SGD and adaptive methods has still not been closed to the best of our knowledge. Furthermore almost all proposed methods aim to strike a middle ground between them: while often improving generalisation, often some the benefits of adaptivity such as fast convergence are partially lost. Furthermore, many of the proposed solutions are heuristically motivated and often lack solid theoretical foundations.

In this work, instead of trading off adaptivity, we propose incorporating iterate averaging (IA) in Adam, to achieve the same objective, which we term Gadam. IA has been recently applied in machine learning with SGD \(^11\)\(^12\)\(^13\), but to the best of our knowledge the combination of IA and adaptive optimisers has not been thoroughly explored apart from a closely related work \(^14\), with

\(^*\) Equal Contribution.
which we compare ourselves empirically. We motivate Gadam from a high dimensional geometry and pre-asymptotic convergence perspective. We then empirically show that our algorithm consistently outperforms test performances of finely-tuned SGD and is competitive against SGD with iterate averaging, without compromising adaptivity or introducing sensitive new hyperparameters, in image classification (CIFAR 10/100, ImageNet 32×32) and natural language processing (Penn Treebank, PTB). Finally, we demonstrate the versatility of our approach: we show that by combining Gadam with partial adaptivity (which we term GadamX), empirical results can be further improved, to even outperform SGD with iterate averaging yet still converge faster.

2 Preliminaries

Adaptive Methods For a data-set of \(N\) training input-output samples \(\{x_i, y_i\}\) and a model parameterised by weights \(w \in \mathbb{R}^P\), the empirical risk \(R_{emp}(w) = \frac{1}{N} \sum_{i=1}^{N} l(h(x_i; w), y_i)\) estimates the latent true risk \(R_{true}(w)\) of the data-generating distribution. Given the inherent unobservability of true risk, the minimisation or \(R_{emp}\) is considered to be the practical optimisation problem of network training \[15\]. Ignoring additional features such as momentum and explicit regularisations, the \(k\)-th iteration of a general iterative optimiser is given by:

\[
w_{k+1} \leftarrow w_k - \alpha B^{-1} \nabla L_k(w)\]

For SGD, the preconditioning matrix \(B = I\) whereas for adaptive methods, \(B\) typically approximates the curvature information whose exact computation is extremely expensive when \(P\) is large. For example, Adam uses the moving uncentered second moment of the per-parameter gradient \[1\].

Iterate Averaging Iterate averaging (IA) \[16\] is a technique that performs a simple average of the iterates \(w_{avg} = \frac{1}{k} \sum_{i=1}^{k} w_i\) throughout the optimisation trajectory. IA has deep roots in optimisation and is required in all stochastic convergence proofs (we explain this in full in App. \[1\]). However, it has been considered not practically useful in deep learning \[17\]. However, a number of recent works have shown strong performance combining SGD and some variants of IA that start at later stages of training only (known as tail averaging) \[12, 13, 18\].

3 Gadam: Adam that Generalises

We propose a simple modification of Adam by incorporating iterate averaging (Algorithm 1 in App. \[A\]). Because pre-averaging Gadam does not differ from Adam, its fast initial convergence is preserved. While two additional hyperparameters are introduced: the starting point for averaging \(T_{avg}\) and the frequency of averaging, we detail below on how to determine the former so that little manual tuning is necessary, and for the latter, we show empirically in App. \[B.2\] that Gadam is little impacted by it (although when analysing theoretically, we assume averaging every iteration). Another key feature is that we require is decoupled weight decay – this as a form of explicit regularisation, as we will show, is critical to the success of Gadam in many contexts and is often the key differentiator from the variant based on plain Adam which is often (but not always) ineffective. We present an explanation for this in Section 4.2.

Gadam requires a starting point for averaging \(T_{avg}\), whose effect will be theoretically discussed in Section 4.1 and empirically justified in App. \[B.3\]. We stress \(T_{avg}\) is not an extra hyperparameter - in normal schedules we need to determine when to decay the learning rate, and \(T_{avg}\) simply replaces that. In general, we find that the performance is not very sensitive to any sensible choice of \(T_{avg}\), as long as we start averaging after validation metrics stagnate. While tuning \(T_{avg}\) actually leads to further performance gains (App. \[B.3\]), in order to make fair comparisons to previous work, we trigger averaging in the 161st epoch out of a 300-epoch budget \[12\]. We also take inspiration from \[18\] for an alternative heuristic to eliminate this free hyperparameter and to start averaging when validation accuracy does not improve for several epochs (the number of which is termed patience); after averaging, we use the same method to determine the early stop point, should a validation accuracy stagnates. We also experiment with a flat learning rate schedule, eliminating the learning rate schedule choice. Our preliminary experiments on this variant, shown in Section \[B.3\] indicate

\[3\] While unlike SGD, adaptive methods are considered more functional without scheduling, unscheduled adaptive methods underperform \[19, 10\] hence scheduling is still almost universally used.
that when automatic schedule determination is used Gadam posts a stronger performance than SGD + IA. This could be particularly helpful for practitioners on limited computational budgets.

4 Theoretical Motivations

We motivate Gadam with 4 key theoretical observations. In this section, we both draw results from Polyak averaging \[16\], which provides the theoretical foundation in many cases, but we also consider the contexts of modern deep learning and adaptive optimisation. Our key claims and findings are:

- IA is particularly relevant as a noise reduction technique when the number of parameters \( P \) is large and the batch size \( B \) is small. For this purpose, the common learning rate decay is not as effective.
- By virtue of the better convergence of Adam, Gadam should outperform SGD-based IA optimisation in term of pre-asymptotic convergence, and we derive this under some mild assumptions.
- High learning rates used in IA implicitly regularises and can be particularly helpful for adaptive methods when properly combined with decoupled weight decay. This effect is further strengthened when batch normalisation (BN) \[20\] is used. Relatedly, we claim IA itself is \textit{not} a form of implicit regularisation contrary to previous beliefs \[12, 21\], but the high learning rate typically used in IA is.
- The “sharpness” of the solution, often believed to be the reason for the better generalisation of IA \[12\] and worse generalisation of adaptive methods \[4\], is not necessarily relevant, at least in the experiments we consider.

4.1 Noise-reducing Effect of IA in High Dimensions

Mini-batching introduces noise into the gradient observations. Even if we use the entire dataset, if we consider our data to be i.i.d draws from the data generating distribution, the full data gradient will be a noisy estimate of the true gradient. IA has known optimal asymptotic convergence \[16\], with further analysis under less stringent conditions explored in \[22\]. However an explicit analysis of its importance in deep learning, suitable for a machine learning audience, where we have large models and trained with small mini-batches has not been presented before to the best of our knowledge.

To allow for explicit analytical comparisons to the most commonly used noise reduction schemes in the literature, such as exponentially moving average (EMA), batch size increases and learning rate decreases, we consider the optimisation of a quadratic loss function \( f(w), w \in \mathbb{R}^{P \times 1} \) as the proxy of our true loss function. This is not possible under the more general setup such as the twice-differentiable, convex function setting in \[16\]. We optimise using gradient descent with learning rate \( \alpha \) and at each iteration we assume the gradient is perturbed by some i.i.d. zero-mean Gaussian isotropic noise with elementwise variance \( \sigma^2 \). Denoting the weight vector at the end of \( n \)-th iteration as \( w_n \) and the corresponding average of the first \( n \) iterations in the weight space as \( w_{avg} \), formally

\[
\text{Theorem 1. Under the aforementioned assumptions,}
\]

\[
\mathbb{P}\left( \left\| w_n \right\| - \left\{ \sum_{i=1}^{P} w_{0, i} e^{-2\alpha \sigma^2} + \frac{P \alpha \sigma^2}{B} \langle \lambda \rangle \geq t \right\}, \mathbb{P}\left( \left\| w_{avg} \right\| - \left\{ \sum_{i=1}^{P} \frac{w_{0, i}^2}{\lambda_i^2 n^2 \alpha^2} + \frac{P \alpha \sigma^2}{B n} \langle \lambda \rangle \geq t \right\} \right) \leq \nu
\]

where \( \nu = 2 \exp(-ct^2) \) and \( \langle \lambda^k \rangle = \frac{1}{n} \text{Tr} H^k \). \( H = \nabla \nabla L \) is the Hessian of the loss w.r.t weights and \( B \) is the batch size (Proof in App. E).

Theorem\[1\] shows that due the noise-reducing effect of IA, whilst we attain exponential convergence in the mean for \( w_n \), we do not control the gradient noise, whereas for \( w_{avg} \), although the convergence in the mean is worse (linear), the variance vanishes asymptotically – this motivates tail averaging, i.e. starting averaging only when we are somewhat close to minimum. Another key implication of Theorem\[1\] lies in its dependence of \( P \). In high dimensions, typical of large expressive modern neural networks, the low dimensional intuition that majority of the probability mass is concentrated around the mean fails. The relevant quantity is not the probability density at a point, but the integral under the density in the immediate vicinity of that point. In such high-dimensional regimes, the effect of noise may very well dominate the convergence in the mean, provided one starts averaging reasonably close to the minimum. While the iterates will be located in a thin-shell with a high probability, the robustness to the gradient noise will drive the IA closer to the minimum. Unless the per parameter estimation noise scales \( \propto \frac{1}{n} \) which is an odd assumption to make, we expect this effect to be more
and more significant when the number of parameters $P$ gets larger. With $P$ being a rough gauge of the model complexity, this implies that in more complex, over-parameterised models, we expect the benefit of IA to be larger; we validate this result in Section 6.

**Learning Rate Scheduling and Adaptive Methods** For tractable analysis in Theorem 1 we compare $w_{\text{avg}}$ with $w_n$ of the same learning rate, which is not what is used in practice. Optimisers utilising IA decay the learning rate much modestly or does not decay it at all [12, 18] and this is theoretically justified [22, 16]. It is therefore of great interest on whether the analysis holds for comparisons with realistic learning rate scheduling, as reducing the learning rate $\alpha$ or increasing the batch size $B$ also reduces noise effect in Theorem 1. On this, we first note that there is a limit to how much one may reduce $\alpha$: for very small $\alpha$, $w_n$ converges at a rate $O(\frac{1}{n})$ whereas $w_{\text{avg}}$ converges at $O\left(\frac{(1}{n})^\frac{\alpha}{2}\right)$. In fact, formally [16, 22] show that for the expected loss of the final iterate to be smaller than that of the average of the iterates, the learning rate must decay faster than $O(\frac{1}{n})$, and IA has optimal convergence rates for learning rate schedules which decay less aggressively than that. This implies that for IA to be sub-optimal we need the learning rate to decay faster than the theoretically proposed $O(\frac{1}{\sqrt{n}})$; this schedule is already almost never used in practice, as it decays the learning rate too fast and strongly harms the convergence in mean. Note that the batch size can also only be increased up to a maximal value of $N$ the dataset size, it also incurs a large memory cost which is often impractical for modern networks. Up to this point, we have used results derived using SGD and now we show the results may be generalised to adaptive methods. [23] derive asymptotic general bounds on the regrets of $w_{\text{avg}}$ and $w_n$ using optimisers with both identity (first-order) and non-identity preconditioning matrices $B$ and we summarise the results in Table 1. In the asymptotic limit of $n \to \infty$, regret of the average of the iterates tends to 0 regardless of the preconditioning matrix, thereby establishing the validity of previous claims on Adam. Furthermore, under some mild assumption on the properties of the conditioning matrix of Adam, we show that IA in Adam could lead to even better expected loss bound compared to SGD – intuitively this is not surprising, given one of the major practical advantages of Adam is its ability to make more progress on a per-iteration basis; we derive this claim in App. E.3.

**Note on Assumptions** To make the analysis in this section applicable to generalisation performance, we make the similar assumption as [24] that the batch gradients are draws from the true latent gradient distribution (not the empirical one) and all gradients are i.i.d. By the Central Limit Theorem, the analysis of in this section holds. Nonetheless, it is natural to argue that for a finite dataset, bias, in addition to noise, can be present in the batch gradient estimate of the true gradient. We argue that even with bias present IA still leads to a better solution: if we assume each gradient $\nabla \mathcal{L}(w_n)$ is strictly smaller than the $\nabla \mathcal{L}(w_{\text{avg}})$, the learning rate must decay faster than $O(\frac{1}{\sqrt{n}})$, and IA has optimal convergence rates for learning rate schedules which decay less aggressively than that. This implies that for IA to be sub-optimal we need the learning rate to decay faster than the theoretically proposed $O(\frac{1}{\sqrt{n}})$; this schedule is already almost never used in practice, as it decays the learning rate too fast and strongly harms the convergence in mean. Note that the batch size can also only be increased up to a maximal value of $N$ the dataset size, it also incurs a large memory cost which is often impractical for modern networks. Up to this point, we have used results derived using SGD and now we show the results may be generalised to adaptive methods. [23] derive asymptotic general bounds on the regrets of $w_{\text{avg}}$ and $w_n$ using optimisers with both identity (first-order) and non-identity preconditioning matrices $B$ and we summarise the results in Table 1. In the asymptotic limit of $n \to \infty$, regret of the average of the iterates tends to 0 regardless of the preconditioning matrix, thereby establishing the validity of previous claims on Adam. Furthermore, under some mild assumption on the properties of the conditioning matrix of Adam, we show that IA in Adam could lead to even better expected loss bound compared to SGD – intuitively this is not surprising, given one of the major practical advantages of Adam is its ability to make more progress on a per-iteration basis; we derive this claim in App. E.3.

**Table 1: Upper bounds of expected regrets when $n \to \infty$.**

<table>
<thead>
<tr>
<th>Preconditioner</th>
<th>$B = I$ (e.g. SGD)</th>
<th>$B \neq I$ (e.g. Adam, second-order optimisers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>$\frac{\sqrt{n}}{\alpha \lambda} \text{Tr}(H^{-1} \Sigma_g(w^*))$</td>
<td>$\min \left(\frac{1}{n} \text{Tr}(B^{-1} \Sigma_g(w^<em>)), \frac{1}{2} \text{Tr}(B^{-1} \Sigma_g(w^</em>))\right)$</td>
</tr>
<tr>
<td>$n$-th Iterate</td>
<td>$\frac{1}{\sqrt{n}} \text{Tr}(1 - \frac{1}{n} H)^{-1} \Sigma_g(w^*)$</td>
<td>$\frac{1}{2} \text{Tr}(B^{-1} \Sigma_g(w^*))$</td>
</tr>
</tbody>
</table>

Note: $\Sigma_g(w^*)$ is the covariance of gradients evaluated at optimum $w^*$.

**4.2 Regularising Effect of IA via Effective Learning Rates**

We argue IA maintains a high effective learning rate, which is shown to be a strong regulariser that leads to improved generalisation of Adam and SGD [25]. This is obviously made possible by the high (nominal) learning rate schedules of IA, but on networks with BN, weight reduction is also relevant. This can be straightforwardly shown by the Cauchy-Schwartz inequality:

**Theorem 2.** If iterates $\{w_i\}$ are drawn from a uniform distribution in the weight space, then the $L_2$ norm of the expected IA point is smaller or equal to the expectation of the $L_2$ weight norm of the iterates: $||E(w_i)||_2 \leq E(||w_i||_2)$ (Proof in App. E).
Weight reduction is not peculiar to IA; simply decaying the learning rate can already lead to a comparable, if not larger, amount of weight reduction. However, the modern CNNs are almost universally equipped with normalisation schemes such as BN. Weight reduction achieved through $L^2$ regularisation, which is staple of DNN training, works not in the classical way in limiting the expressiveness but rather by encouraging high effective learning rate that scales $\alpha_{\text{eff}} \propto \frac{\alpha}{||w||^2}$ [25].

This implies 1) we expect Gadam (and indeed all optimisers with IA, but we focus on Gadam here) to be more effective with BN since it both keeps $\alpha$ large and $||w||^2$ small. Given that much criticism about adaptive methods is their propensity to “over-adapt” and the “small learning rate dilemma” (i.e., learning rate decaying combined with adaptive momentum leading to no training progress made) [10, 3], this could provide much-needed regularisation and escape from the dilemma; 2) explicitly reducing $\alpha$ has little regularisation benefit, as the aggressive decay easily eclipses any weight reduction. We experimentally validate both are true using a classical VGG-16 network on CIFAR-100 with and without BN (Fig. 1(a)(b)): under the identical setup (detailed in App. C.1), the margin of improvement of Gadam is much larger with BN. While Gadam keeps $\alpha_{\text{eff}}$ high, in scheduled AdamW it quickly vanishes once we start learning rate decay. It is then curious to ask whether using AdamW with no/more moderate schedule can counteract this. However, we find either to underperform the scheduled AdamW by more than $5\%$ in accuracy rendering their discussions irrelevant (not shown in graph).

We believe the same regularisation argument also explains why combining adaptive methods with IA has not been popularised despite the popularity of the former and the theoretical desiderata of the latter. Indeed we find naively combining Adam with IA is ineffective on typical image problems, but we argue that the real culprit is the ineffective yet commonly used $L^2$ regularisation, which counter-intuitively penalises parameters with more fluctuating gradients less than otherwise [19].

4.3 Relevance of Local Geometry Arguments

One argument as to why IA improves generalisation [12] is about the local geometry of the solution found: [12] discuss the better generalisation of SWA to the “flatter” minimum it finds. The same argument is used to explain the apparent worse generalisation of adaptive method: [26] showed empirically that adaptive methods are not drawn to flat minima unlike SGD. From both Bayesian and minimum description length arguments [27], flatter minima generalise better, as they capture more probability mass. [21] formalise the intuition under the assumption of a shift between the training and testing loss surface and investigate the presence of “flat valleys” in loss landscape. They argue that averaging leads to a biased solution to the “flatter” valley, which has worse training but better generalisation performance due to the shift. This suggests IA has an inherent regularising effect, which contrasts with our previous claim that IA should improve both.

Figure 1: Val. error and $\alpha/||w||^2$ of CIFAR-100 on VGG-16 with and without BN. In (b) only results with BN where the quantity is relevant are shown.

Figure 2: Comparing IA using Adam and AdamW on PRN-110 on CIFAR-100.
An alternative analysis of EMA arriving at similar result was done in [33], but their emphasis of where for simplicity we follow[10] to fix 
As discussed, most related works improve generalisation of adaptive methods by combining them with 
work has shown under certain assumptions that the true risk surface is 
Table 2: Performance and Hessian-based sharpness metrics on CIFAR-100 using VGG-16. The 
numerical results for iterates are in brackets.

<table>
<thead>
<tr>
<th>Optimiser</th>
<th>Terminal LR</th>
<th>Train acc.</th>
<th>Test acc.</th>
<th>Spectral Norm</th>
<th>Frobenius Norm</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdamW</td>
<td>3E–6</td>
<td>99.93</td>
<td>69.43</td>
<td>62</td>
<td>9.3E–4</td>
<td>4.7E–5</td>
</tr>
<tr>
<td>Gadam</td>
<td>3E–5</td>
<td>99.77 (94.12)</td>
<td>69.67 (67.16)</td>
<td>120 (2500)</td>
<td>1.4E–6 (0.86)</td>
<td>6.4E–5 (5.2E–3)</td>
</tr>
<tr>
<td>Gadam</td>
<td>3E–4</td>
<td>98.62 (89.34)</td>
<td>71.55 (64.68)</td>
<td>43 (280)</td>
<td>1.1E–3 (0.023)</td>
<td>1.1E–4 (5.3E–4)</td>
</tr>
<tr>
<td>SGD</td>
<td>3E–4</td>
<td>99.75</td>
<td>71.64</td>
<td>4.40</td>
<td>1.2E–5</td>
<td>4.7E–6</td>
</tr>
<tr>
<td>SWA</td>
<td>3E–3</td>
<td>99.98 (98.87)</td>
<td>71.32 (69.88)</td>
<td>1.85 (14.6)</td>
<td>4.6E–6 (1.3E–4)</td>
<td>1.1E–6 (8.6E–5)</td>
</tr>
<tr>
<td>SWA</td>
<td>3E–2</td>
<td>91.58 (77.29)</td>
<td>73.40 (63.42)</td>
<td>1.35 (12.0)</td>
<td>8.4E–6 (7.0E–5)</td>
<td>1.8E–5 (9.8E–5)</td>
</tr>
</tbody>
</table>

However, one issue in the aforementioned analysis, is that they train their SGD baseline and averaged schemes on different learning rate schedules. While this is practically justified, and even desirable, exactly because IA performs better with high learning rate as argued, for theoretical analysis on the relevance of the landscape geometry to solution quality, it introduces interfering factors. It is known that the learning rate schedule can have a significant impact on both performance and curvature [28]. We address this by considering IA and the iterates, for the same learning rate to specifically alleviate this issue. We use the VGG-16 without BN [4] using both AdamW/Gadam and SGD/SWA. In addition to the test and training statistics, we also examine the spectral norm, Frobenius norm and trace which serve as different measures on the “sharpness” of the solutions using the spectral tool by [30]; we show the results in Table 2. We find a rather mixed result with respect to the local geometry argument. While averaging indeed leads to solutions with lower curvature, we find no clear correlation between flatness and generalisation. One example is that compared to SGD, the best performing Gadam run has 14× larger spectral norm, 92× larger Frobenius norm and 23× larger Hessian trace, yet the test accuracy is only 0.09% worse. Either our metrics do not sufficiently represent sharpness, which is unlikely since we included multiple metrics commonly used, or that it is not the most relevant explanation for the generalisation gain. We hypothesise the reason here is that the critical assumption, upon which the geometry argument builds, that there exist only shifts between test and train surfaces is unsound despite a sound analysis given that. For example, recent work has shown under certain assumptions that the true risk surface is everywhere flatter than the empirical counterpart [31]. Furthermore, for any arbitrary learning rate, as predicted IA helps both optimisation and generalisation compared to iterates of the same learning rate; any trade-offs between optimisation and generalisation seem to stem from the choice of learning rates only.

5 Related Works

As discussed, most related works improve generalisation of adaptive methods by combining them with SGD in some form. As an example representing the recent works claiming promising performances, [10] introduce an additional hyperparameter p, to control the extent of adaptivity: for \( p = \frac{1}{2}, 0 \), we have fully adaptive AdamW or pure first-order SGD respectively and usually a p falling between the extremes is taken. In addition to empirical comparisons, since our approach is orthogonal to these approaches, as a singular example, we propose GadamX that combines Gadam with Padam, where for simplicity we follow [10] to fix \( p = \frac{1}{2} \) for the current work. We note that \( p < 1 \) is regularly considered a heuristic to be used for an inaccurate curvature matrix [23], although the specific choice of \( p = 1/2 \) has a principled derivation in terms of a regret bound [32]. Previous works also use EMA in weight space to achieve optimisation and/or generalisation improvements: [13] entertain EMA in SWA, although they conclude simple averaging is more competitive. Recently, [14] proposes Lookahead (LH), a plug-in optimiser that uses EMA on the slow weights to improve convergence and generalisation. Nonetheless, having argued the dominance of noise in the high-dimensional deep learning regime, we argue that simple averaging is more theoretically desirable for generalisation. Following the identical analysis to Section 4.1, we consider the 1D case w.l.o.g and denote \( \rho \in [0,1] \) as the coefficient of decay, asymptotically the EMA point \( w_{ema} \) is governed by:

\[
N\left( \left( 1 - \rho \right) w_0 + \frac{1 - \alpha \lambda}{1 - \alpha \lambda - \rho} - \frac{1 - \rho \alpha \sigma^2 \kappa}{1 + \rho - \lambda} \right), \text{where } \kappa = (1 - (1 - \alpha \lambda)^{-2}) \tag{5.1}
\]

An alternative analysis of EMA arriving at similar result was done in [33], but their emphasis of comparison is between the EMA and iterates instead of EMA and the IA point in our case. From

\(^3\)It is argued that BN impacts the validity of conventional measures of sharpness [29] hence we deliberately remove BN here, nor do we tune optimisers rigorously, since the point here is for theoretical exposition instead of empirical performance. See detailed setup in App. C.2.
while the convergence in mean is less strongly affected, the noise is reduced by a factor of \(\frac{1}{1+\rho}\). So whilst we reduce the noise possibly by a very large factor, it does not vanish asymptotically. Hence viewing EMA or IA as noise reduction schemes, we consider IA to be far more aggressive. Secondly, EMA implicitly assumes that more recent iterates are better, or otherwise more important, than the previous iterates. While justified initially (partially explaining LH’s efficacy in accelerating optimisation), it is less so in the late stage of training. We nonetheless believe LH could be of great combinable value, and include a preliminary discussion in App. B.1.

### 6 Experiments

We test our methods in multiple vision and language processing tasks, running each experiment 3 times with mean and standard deviation reported. In this section, all non-IA baselines are tuned rigorously with proper schedules for fair comparisons\(^4\), and we also include the results reported in the previous works in Table 6 of App. C, where we also include all the implementation details.

![Figure 3: Test error (a-c) and test improvement of best IA over its base optimiser against number of parameters (d) on CIFAR-100. In (d), the results lifted from [12, 34] are marked with asterisks.](image)

![Figure 4: Top-5 Test Error on ImageNet 32x32 (a)(b) and validation perplexity of 3-layer LSTM on PTB word-level modelling](image)

**Image Classification on CIFAR Data-sets** Here we consider VGG-16, Preactivated ResNet (PRN) and ResNeXt \([35, 36, 37]\) on CIFAR datasets \([38]\), with CIFAR-100 results in Fig. 3 and Table 3 and CIFAR-10 results in App. D.1. As AdamW always outperforms Adam in our experiments, the curves for the latter are omitted in the main text; we detail these results in App. D. The results show that optimisers with IA (SWA, Gadam and GadamX) invariably improve over their counterparts without, and GadamX always delivers the strongest performance (except for PRN-110, where the difference between SWA and GadamX is not significant, although GadamX has a smaller standard deviation and converges faster). Without compromising convergence speed, in all cases Gadam outperforms tuned SGD and Padam - suggesting that solutions found by adaptive optimisers does not necessarily generalise worse. Indeed, any generalisation gap seems to be closed by the using IA and an appropriately implemented weight decay. We emphasise that results here are achieved without tuning \(T_{avg}\); if we allow crude tuning of \(T_{avg}\), on CIFAR-100 GadamX achieves 77.22% (VGG-16) and 79.41% (PRN-110) test accuracy respectively, which to our knowledge are the best reported performance on these architectures (See Table 4 and Fig. 7 in App. B.3). Finally, in line

\(^4\)In image classification, we use the linear schedule, which both performs better than usual step schedule (See App. D.3) and is consistent with [12].
with our results from Theorem 1 we observe a roughly linear relation between \( P \) and benefit of IA measured in terms of test improvement compared to the SGD baseline, shown in Fig. 4 where we also incorporate SWA results from the previous works [12]. Whilst we note that the experimental setups of different architectures and implementations can vary significantly and more experiments are needed to rigorously establish any relation, it is at least encouraging to see some agreement with our simplified model in section 4.1 in real networks.

### Image Classification on ImageNet 32 × 32

Again, our methods perform competitively. We also see the implication of Theorem 1 here: In PRN-110, IA only leads to marginal improvements in test performance – in ImageNet, this architecture has the almost no over-parameterisation, since if we use number of parameters \( P \) as a crude estimate of the model complexity we have \( P \approx N \) (this might also explain the out-performance of Adam-based optimisers, as overfitting and the resultant need for regularisation might be less important). However, in WideResNet (WRN) [40] experiments, the gain is much larger: while Gadam does not outperform our very strongly performing SGD perhaps due the default learning rate in AdamW/Gadam we use, it nevertheless improves upon AdamW greatly and posts a performance stronger than baseline in literature with identical [39] and improved [41] setups. Finally, GadamX performs strongly, outperforming more than 3% compared to the baseline [39] in Top-5 accuracy.

### Word-level Language Modelling on PTB

We run word-level language modelling using a 3-layer Long-short Term Memory (LSTM) model [42] on PTB dataset [43] and the results are in Table 4. Remarkably, Gadam achieves a test perplexity of 58.77 (58.61 if we tune \( \tau_{avg} \)). See Table 4 in App. B.3, better than the baseline NT-ASGD in [18] that runs an additional 300 epochs on an identical network. Note that since, by default, the ASGD uses a constant learning rate, we do not schedule the learning rate except Padam which requires scheduling to converge. Also, for consistency, we use a manual trigger to start averaging at the 100th epoch for ASGD (which actually outperforms the NT-ASGD variant). We additionally conduct experiments with scheduling and NT-ASGD (App. 4) and Gadam still outperforms. It is worth mentioning that for state of the art results in language modelling [44, 45, 46], Adam is the typical optimiser of choice. Hence these results are both encouraging and significant for wider use in the community.

### Conclusion

We propose Gadam, a variant of Adam that incorporates IA. We analyse Gadam and IA from a high dimensional geometric perspective, highlighting its benefits in modern over-parameterised networks where estimation noise is large. We also analysed its regularisation effects: We find that previous arguments in terms of solution sharpness are less relevant and that trade-off between generalisation and optimisation is not related to IA, but rather to the learning rate schedule employed. We confirm that Gadam, and the partially adaptive variant GadamX, perform well across different tasks. This
suggests that when properly regularised and implemented with IA, adaptive methods can, and do, generalise well. In the future, we plan to test our methods in models representing the most recent advances in deep learning [47,48] and also hope to explore the prospect of applications in low-bit optimisation [49], another important direction for scalable and portable deep learning.

References


A Algorithm

In this section we present the complete algorithm of Gadam in Algorithm 1.

Algorithm 1 Gadam

Require: initial weights θ₀; learning rate scheduler αₜ = α(t); momentum parameters {β₁, β₂} (Default to {0.9, 0.999} respectively); decoupled weight decay λ; averaging starting point T_avg; tolerance ε (default to 10⁻⁸)

Ensure: Optimised weights ˜θ

Set m₀ = 0, v₀ = 0, ˚v₀ = 0, n_models = 0.

for t = 1, ... T do
  αₜ = α(t)
  gₜ = ∇fₜ(θₜ)
  mₜ = β₁mₜ₋₁ + (1 − β₁)gₜ/(1 − β₁)
  vₜ = β₂vₜ₋₁ + (1 − β₂)gₜ²/(1 − β₂)
  ˚vₜ = max(˚vₜ₋₁, ˚vₜ) (If using Amsgrad)
  θₜ = (1 − αₜλ)θₜ₋₁ − αₜmₜ/(˚vₜ + ε)
  if T ≥ T_avg then
    n_models = n_models + 1
    θ_avg = θ_avg / n_models + θₜ
  else
    θ_avg = θₜ
  end if
end for

return ˜θ = θ_avg

B Supplementary Materials of Gadam

B.1 Gadam and Lookahead

Lookahead [14] is a very recent attempt that also features weight space averaging in order to achieve optimisation and generalisation benefits. However, instead of using simple averaging in our proposed algorithms, Lookahead maintains different update rules for the ‘fast’ and ‘slow’ weights, and uses exponentially moving average to update the parameters. In this section, we both comment on the key theoretical differences between Gadam and Lookahead and make some preliminary practical comparisons. We also offer an attempt to bring together the optimisation benefit of Lookahead and the generalisation benefit of Gadam, with promising preliminary results.

Major Differences between Gadam and Lookahead

Averaging Method  Lookahead opts for a more complicated averaging scheme: they determine the ‘fast’- and ‘slow’-varying weights during optimisation, and maintains an EMA to average the weight. On the other hand, Gadam uses a more straightforward simple average. As we discussed in the main text, EMA is more theoretically justified during the initial rather than later stage of training. This can also be argued from a Bayesian viewpoint following [13], who argued that iterates are simply the draws from the posterior predictive distribution of the neural network, where as averaging leads to a rough estimation of its posterior mean. It is apparent that if the draws from this distribution are equally good (which is likely to be the case if we start averaging only if validation metrics stop improving), assigning the iterates with an exponential weight just based on when they are drawn constitutes a rather arbitrary prior in Bayesian sense.

Averaging Frequency  Lookahead averages every iteration whereas in Gadam, while possible to do so as well, by default averages much less frequently. We detail our rationale for this in App. B.2.

Starting Point of Averaging  While Lookahead starts averaging at the beginning of the training, Gadam starts averaging either from a pre-set starting point or an automatic trigger (for GadamAuto).
While authors of Lookahead [14] argue that starting averaging eliminates the hyperparameter on when to start averaging, it is worth noting that Lookahead also introduces two additional hyperparameters $\alpha$ and $k$, which are non-trivially determined from grid search (although the authors argue that the final result is not very sensitive to them).

We believe the difference here is caused by the different design philosophies of Gadam and Lookahead: by using EMA and starting averaging from the beginning, Lookahead benefits from faster convergence and some generalisation improvement whereas in Gadam, since the averages of iterates are not used during training to promote independence between iterates, Gadam does not additionally accelerate optimisation but, by our theory, should generalise better. As we will see in the next section, this theoretical insight is validated by the experiments and leads to combinable benefits.

**Empirical Comparison**

We make some empirical evaluations on CIFAR-100 data-set with different network architectures, and we use different base optimiser for Lookahead. For all experiments, we use the author-recommended default values of $k = 5$ (number of lookahead steps) and $\alpha = 0.5$. We focus on the combination of Lookahead and adaptive optimisers, as this is the key focus of this paper, although we do include results with Lookahead with SGD as the base optimiser.

We first test AdamW and SGD with and without Lookahead and the results are in Fig. 5. Whilst SGD + LH outperforms SGD in final test accuracy by a rather significant margin in both architectures, Lookahead does not always lead to better final test accuracy in AdamW (although it does improve the convergence speed and reduce fluctuations in test error during training, which is unsurprising as EMA shares similar characteristics with IA in reducing sensitivity to gradient noise). On the other hand, it is clear that Gadam delivers both more significant and more consistent improvements over AdamW, both here and in the rest of the paper.

![Figure 5: Test accuracy of Lookahead in CIFAR-100 against number of epochs.](a) VGG-16  
(b) PRN-110

Nonetheless, we believe that Lookahead, being an easy-to-use plug-in optimiser that clearly improves convergence speed, offers significant combinable potential with Gadam, which focuses on generalisation. Indeed, by using Lookahead before the 161st epoch where we start IA, and switching to IA after the starting point, we successfully combine Gadam and LH into a new optimiser which we term Gadam + LH. With reference to Fig. 5 in VGG-16, Gadam + LH both converges at the fastest speed in all the optimisers tested and achieves a final test accuracy only marginally worse than Gadam (but still stronger than all others). On the other hand, in PRN-110, perhaps due to the specific architecture choice, the initial difference in convergence speed of all optimisers is minimal, but Gadam + LH clearly performs very promisingly in the end: it is not only stronger than our result without Lookahead in Fig. 5(b), but also, by visual inspection, significantly stronger than the SGD + LH results on the same data-set and using the same architecture reported in the original Lookahead paper [14].

Due to the constraint on computational resources, we have not been able to fully test Gadam + LH on a wider range of problems. Nonetheless, we believe that the results obtained here are encouraging, and should merit more in-depth investigations in the future works.

**B.2 Effect of Frequency of Averaging**

While we derive the theoretical bounds using Polyak-style averaging on every iteration, practically we average much less: we either average once per epoch similar to [12], or select a rather arbitrary value
such as averaging once per 100 iterations. The reason is both practical and theoretical: averaging much less leads to significant computational savings, and at the same time as we argued, on more independent iterates the benefit from averaging is better, because our theoretical assumptions on independence are more likely met in these situations. In this case, averaging less causes the iterates to be further apart and more independent, and thus fewer number of iterates is required to achieve the similar level of performance if less independent iterates are used. We verify this both on the language and the vision experiments using the identical setup as the main text. With reference to Fig. 6(a), not only is the final perplexity very insensitive to averaging frequency (note that the y-axis scale is very small), it is also interesting that averaging less actually leads to a slightly better validation perplexity compared to schemes that, say, average every iteration. We see a similar picture emerges in Fig. 6(b), where the despite of following very close trajectories, averaging every iteration gives a slightly worse testing performance compared to once an epoch and is also significantly more expensive (with a NVIDIA GeForce RTX 2080 Ti GPU, each epoch of training takes around 10s if we average once per epoch but averaging every iteration takes around 20s).

Figure 6: Effect of different averaging frequencies on validation perplexity of Gadam on representative (a) Language and (b) Image classification tasks. \( \text{Freq}=n \) suggests averaging once per \( n \) iterations. \( \text{Freq}=350 \) in (b) is equivalently averaging once per epoch.

### B.3 Effect of \( T_{\text{avg}} \)

In Gadam(X), we need to determine when to start averaging (\( T_{\text{avg}} \) in Algorithm 1), and here we investigate the sensitivity of Gadam(X) to this hyperparameter. We use a range of \( T_{\text{avg}} \) for a number of different tasks and architectures (Fig. 7 and Table 4), including extreme choices such as \( T_{\text{avg}} = 0 \) (start averaging at the beginning). We observe that for any reasonable \( T_{\text{avg}} \), Gadam(X) always outperform their base optimisers with standard learning rate decay, and tuning \( T_{\text{avg}} \) yields even more improvements over the heuristics employed in the main text, even if selecting any sensible \( T_{\text{avg}} \) already can lead to a promising performance over standard learning rate decay.

Figure 7: Effect of different \( T_{\text{avg}} \) on the performance of various tasks and architectures. †: In (a), dashed lines denote validation and solid lines denote test perplexities.

Here we also conduct preliminary experiments on Gadam with automatic determination of \( T_{\text{avg}} \) and training termination (we term this approach GadamAuto) - this is possible given the insensitivity of the end-results towards \( T_{\text{avg}} \) as shown above, and is desirable as the optimiser both has fewer hyperparameters to tune and trains faster. We use VGG-16 network on CIFAR-100. For all experiments,
we simply use a flat learning rate schedule. The results are shown in Table 5. We use a patience of 10 for both the determination of the averaging activation and early termination. We also include SWA experiments with SGD iterates.

<table>
<thead>
<tr>
<th>Optimiser</th>
<th>Data-set</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadam-Auto</td>
<td>CIFAR-100</td>
<td>75.39</td>
</tr>
<tr>
<td>SWA-Auto</td>
<td>CIFAR-100</td>
<td>73.93</td>
</tr>
</tbody>
</table>

It can be seen that while automatic determination for averaging trigger and early termination work well for Gadam (GadamAuto posts a performance only marginally worse than the manually tuned Gadam), they lead to a rather significant deterioration in test in SWA (SWA-Auto performs worse than tuned SWA, and even worse than tuned SGD. See Table 3). This highlights the benefit of using adaptive optimiser as the base optimiser in IA, as the poor performance in SWA-Auto is likely attributed to the fact that SGD is much more hyperparameter-sensitive (to initial learning rate and learning rate schedule, for example. SWA-Auto uses a constant schedule, which is sub-optimal for SGD), and that validation performance often fluctuates more during training for SGD: SWA-Auto determines averaging point based on the number of epochs of validation accuracy stagnation. For a noisy training curve, averaging might be triggered too early; while this can be ameliorated by setting a higher patience, doing so will eventually defeat the purpose of using an automatic trigger. Both issues highlighted here are less serious in adaptive optimisation, which likely leads to the better performance of GadamAuto.

Nonetheless, the fact that scheduled Gadam still outperforms GadamAuto suggests that there is still ample room of improvement to develop a truly automatic optimiser that performs as strong as or even stronger than tuned ones. One desirable alternative we propose for the future work is the integration of Rectified Adam [29], which is shown to be much more insensitive to choice of hyperparameter even compared to Adam.

### C Experiment Setup

Unless otherwise stated, all experiments are run with PyTorch 1.1 on Python 3.7 Anaconda environment with GPU acceleration. We use one of the three possible GPUs for our experiment: NVIDIA GeForce GTX 1080 Ti, GeForce RTX 2080 Ti or Tesla V100. We always use a single GPU for any single run of experiment.

#### C.1 Validating Experiments

**VGG-16 on CIFAR-100** In this expository experiment, we use the original VGG-16. We select VGG-16 as it is both a very classical network and is also rather unique in a sense that its variants both with and without BN are widely used (more recent networks such as ResNets almost invariably have BN built-in, and their counterparts without BN are not commonly used). We conduct all experiments with initial learning rate 0.03. For fair comparison to previous literature, we use the linear decay schedules advocated in [12], for both SGD and IA. For IA we run the set of terminal learning rates
we set $\alpha$ where $\alpha$ is the initial learning rate. In the motivating logistic regression experiments on MNIST, we used $T = 50$. $T = 300$ is the total number of epochs budgeted for all CIFAR experiments, whereas we used $T = 200$ and $50$ respectively for PRN-110 and WideResNet 28x10 in ImageNet. We set $r = 0.01$ for all experiments. For experiments with iterate averaging, we use the following learning rate schedule instead:

$$
\alpha_t = \begin{cases} 
\alpha_0, & \text{if } t \leq \frac{T}{T_{avg}} \leq 0.5 \\
\alpha_0 \frac{(0.5 - 0.5)}{0.4} & \text{if } 0.5 < \frac{t}{T_{avg}} \leq 0.9 \\
\alpha_0 & \text{otherwise}
\end{cases}
$$

where $\alpha_0$ is the initial learning rate. In the motivating logistic regression experiments on MNIST, we used $T = 50$. $T = 300$ is the total number of epochs budgeted for all CIFAR experiments, whereas we used $T = 200$ and $50$ respectively for PRN-110 and WideResNet 28x10 in ImageNet. We set $r = 0.01$ for all experiments. For experiments with iterate averaging, we use the following learning rate schedule instead:

$$
\alpha_t = \begin{cases} 
\alpha_0, & \text{if } t \leq \frac{T}{T_{avg}} \leq 0.5 \\
\alpha_0 \frac{(0.5 - 0.5)}{0.4} & \text{if } 0.5 < \frac{t}{T_{avg}} \leq 0.9 \\
\alpha_0 & \text{otherwise}
\end{cases}
$$

where $\alpha_{avg}$ refers to the (constant) learning rate after iterate averaging activation, and in this paper we set $\alpha_{avg} = \frac{1}{2} \alpha_0$. $T_{avg}$ is the epoch after which iterate averaging is activated, and the methods
### Table 6: Baseline Results from Previous Works

<table>
<thead>
<tr>
<th>Network</th>
<th>Optimiser</th>
<th>Accuracy/Perplexity</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG-16</td>
<td>SGD</td>
<td>73.80</td>
<td>[50]</td>
</tr>
<tr>
<td>VGG-16</td>
<td>FGE</td>
<td>74.26</td>
<td>[12]</td>
</tr>
<tr>
<td>PRN-164</td>
<td>SGD</td>
<td>75.67</td>
<td>[36]</td>
</tr>
<tr>
<td>PRN-110</td>
<td>SGD</td>
<td>76.35</td>
<td>online repository**</td>
</tr>
<tr>
<td>ResNet-164</td>
<td>FGE</td>
<td>79.84</td>
<td>[12]</td>
</tr>
<tr>
<td>ResNeXt-29</td>
<td>SGD</td>
<td>82.20</td>
<td>[39]</td>
</tr>
<tr>
<td>ResNeXt-29</td>
<td>SGD</td>
<td>81.47</td>
<td>[39]</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG-19</td>
<td>SGD</td>
<td>93.34</td>
<td>online repository**</td>
</tr>
<tr>
<td>VGG-16</td>
<td>SGD</td>
<td>93.90</td>
<td>[50]</td>
</tr>
<tr>
<td>PRN-110</td>
<td>SGD</td>
<td>93.63</td>
<td>[36]</td>
</tr>
<tr>
<td>PRN-110</td>
<td>SGD</td>
<td>95.06</td>
<td>online repository**</td>
</tr>
<tr>
<td>ImageNet 32×32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRN-28-10</td>
<td>SGD</td>
<td>59.04/81.13*</td>
<td>[10]</td>
</tr>
<tr>
<td>Modified WRN</td>
<td>SGD</td>
<td>60.04/82.11*</td>
<td>[41]</td>
</tr>
<tr>
<td>PTB</td>
<td>LSTM 3-layer</td>
<td>NT-ASGD</td>
<td>61.2/58.8***</td>
</tr>
</tbody>
</table>

Notes:

* Top-1/Top-5 Accuracy  
** Link: [https://github.com/bearpaw/pytorch-classification](https://github.com/bearpaw/pytorch-classification)  
*** Validation/Test Perplexity

to determine $T_{avg}$ was described in the main text. This schedule allows us to adjust learning rate smoothly in the epochs leading up to iterate averaging activation through a similar linear decay mechanism in the experiments without iterate averaging, as described above.

The only exception is the WRN experiments on ImageNet 32×32, where we only run 50 epochs of training and start averaging from 30th epoch. We found that when using the schedule described above for the IA schedules (SWA/Gadam/GadamX), we start decay the learning rate too early and the final result is not satisfactory. Therefore, for this particular set of experiments, we use the same learning rate schedule for both averaged and normal optimisers. The only difference is that for IA experiments, we decay the learning rate until the 30th epoch and keep it fixed for the rest of the training.

### C.3 Language Modelling Experiments

In language modelling experiments, we use the codebase provided by [https://github.com/salesforce/awd-lstm-lm](https://github.com/salesforce/awd-lstm-lm). For ASGD, we use the hyperparameters recommended by [18] and set the initial learning rate to be 30. Note that in language experiments, consistent with other findings decoupled weight decay seems to be not as effective $L_2$, possibly due to LSTM could be more well-regularised already, and that BN, which we argue to be central to the efficacy of decoupled weight decay, is not used in LSTM. Thus, for this set of experiments we simply use Adam and Padam as the iterates for Gadam and GadamX. For Adam/Gadam, we tune the learning rate by searching initial learning rate in the range of {0.0003, 0.001, 0.003, 0.01} and for Padam and GadamX, we set the initial learning rate to be 1 and partially adaptive parameter $p = 0.2$, as recommended by the authors [10]. We further set the weight decay to be their recommended value of $1.2E−6$. For the learning rate schedule, we again follow [18] for a piecewise constant schedule, and decay the learning rate by a factor of 10 at the {100, 150}-th epochs for all experiments without using iterate averaging. For experiments with iterate averaging, instead of decaying the learning rate by half before averaging starts, we keep the learning rate constant throughout to make our experiment comparable with the ASGD schedule. We run all experiments for 200 (instead of 500 in [18]) epochs.

**Learning Rate Schedule**  
As discussed in the main text, the experiments shown in Table 3 and Fig. 4c are run with constant schedules (except for Padam). Padam runs with a step decay of factor of 10 at {100, 150}-th epochs. However, often even the adaptive methods such as Adam are scheduled with learning rate decay for enhanced performance. Therefore, we also conduct additional scheduled experiments with Adam, where we follow the same schedule of Padam. The results are shown in App. D.2.
C.4 Experiment Baselines

To validate the results we obtain and to make sure that any baseline algorithms we use are properly and fairly tuned, we also survey the previous literature for baseline results where the authors use same (or similar) network architectures on the same image classification/language tasks, and the comparison of our results against theirs is presented in Table 6. It is clear that for most of the settings, our baseline results achieve similar or better performance compared to the previous work for comparable methods; this validates the rigour of our tuning process.

D Additional Experimental Results

D.1 Testing Performance of CIFAR-10

We report the testing performance of VGG-16 and PRN-110 on CIFAR-10 in Fig. 8 and Table 7. Perhaps due to the fact that CIFAR-10 poses a simpler problem compared to CIFAR-100 and ImageNet in the main text, the convergence speeds of the optimisers differ rather minimally. Nonetheless, we find that GadamX still outperforms all other optimisers by a non-trivial margin in terms of final test accuracy.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>optimiser</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>SGD</td>
<td>94.14±0.37</td>
</tr>
<tr>
<td></td>
<td>SWA</td>
<td>94.69±0.36</td>
</tr>
<tr>
<td></td>
<td>Adam(W)</td>
<td>93.90±0.11</td>
</tr>
<tr>
<td></td>
<td>Padam(W)</td>
<td>94.13±0.06</td>
</tr>
<tr>
<td></td>
<td>Gadam</td>
<td>94.62±0.15</td>
</tr>
<tr>
<td></td>
<td>GadamX</td>
<td><strong>94.88±0.03</strong></td>
</tr>
<tr>
<td>PRN-110</td>
<td>SGD</td>
<td>95.40±0.25</td>
</tr>
<tr>
<td></td>
<td>SWA</td>
<td>95.55±0.12</td>
</tr>
<tr>
<td></td>
<td>Adam(W)</td>
<td>94.69±0.14</td>
</tr>
<tr>
<td></td>
<td>Padam(W)</td>
<td>95.28±0.13</td>
</tr>
<tr>
<td></td>
<td>Gadam</td>
<td>95.27±0.02</td>
</tr>
<tr>
<td></td>
<td>GadamX</td>
<td><strong>95.95±0.06</strong></td>
</tr>
</tbody>
</table>

D.2 Word Level Language Modelling with Learning Rate Schedules and Non-monotonic Trigger

Here we include additional results on word-level language modelling using scheduled Adam and NT-ASGD, where the point to start averaging is learned non-monotonically and automatically. Where scheduling further improves the Adam performance marginally, the automatically triggered ASGD actually does not perform as well as the manually triggered ASGD that starts averaging from 100th epoch onwards, as we discussed in the main text - this could be because that ASGD converges rather slowly, the 200-epoch budget is not sufficient, or the patience (we use patience = 10) requires further tuning. Otherwise, our proposed Gadam and GadamX without schedules still outperform the variants.
Table 8: Validation and Test Perplexity on PTB. The Gadam(X) results are lifted from Table 3

<table>
<thead>
<tr>
<th></th>
<th>PTB Validation</th>
<th>PTB Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT-ASGD</td>
<td>66.01</td>
<td>64.73</td>
</tr>
<tr>
<td>Scheduled Adam</td>
<td>63.99</td>
<td>61.51</td>
</tr>
<tr>
<td>Gadam (Ours)</td>
<td>61.35</td>
<td>58.77</td>
</tr>
<tr>
<td>GadamX (Ours)</td>
<td>63.49</td>
<td>60.45</td>
</tr>
</tbody>
</table>

tested here without careful learning rate scheduling. The results are summarised in Fig. 9 and Table 8.

D.3 Linear vs Step Scheduling

In this work, for the baseline methods in image classification tasks we use linear instead of the more conventionally employed step scheduling because we find linear scheduling to generally perform better in the experiments we conduct. In this section, we detail the results of these experiments, and in this section, ‘linear’ refers to the schedule introduced in App. C.2 and ‘step’ refers to the schedule that reduces the learning rate by a factor of 10 in \{150, 250\} epochs for 300-epoch experiments (CIFAR datasets), or in \{25, 40\} epochs for 50-epoch experiments (ImageNet dataset). The results are shown in Table 9.

Table 9: Testing performance of linear and step learning rate schedules on baseline methods.

<table>
<thead>
<tr>
<th>CIFAR-100</th>
<th>Optimiser</th>
<th>Step</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>SGD</td>
<td>73.28</td>
<td>74.15</td>
</tr>
<tr>
<td></td>
<td>AdamW</td>
<td>73.20</td>
<td>73.26</td>
</tr>
<tr>
<td></td>
<td>Padam</td>
<td>74.46</td>
<td>74.56</td>
</tr>
<tr>
<td>PRN-110</td>
<td>SGD</td>
<td>77.23</td>
<td>77.22</td>
</tr>
<tr>
<td></td>
<td>AdamW</td>
<td>75.27</td>
<td>75.47</td>
</tr>
<tr>
<td></td>
<td>Padam</td>
<td>73.95</td>
<td>77.30</td>
</tr>
</tbody>
</table>

D.4 Adam results on Image Problems

As discussed in the main text, we find that for the image classification problems Adam and Gadam with Adam (instead of AdamW) often underperform, and for the conciseness the results are not presented in the main text. Here we detail these results in Table 10. It is evident that in all cases here, the results here underperform the results in Table 5.

Table 10: Test Accuracy of Adam and Gadam based on Adam on Image Classification Problems

<table>
<thead>
<tr>
<th>CIFAR-100</th>
<th>Adam</th>
<th>Gadam (based on Adam)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>69.7</td>
<td>71.1</td>
</tr>
<tr>
<td>PRN-110</td>
<td>73.1</td>
<td>73.7</td>
</tr>
<tr>
<td>ResNeXt-29</td>
<td>76.3</td>
<td>77.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ImageNet 32 x 32</th>
<th>Adam Top-1</th>
<th>Adam Top-5</th>
<th>Gadam (based on Adam) Top-1</th>
<th>Gadam (based on Adam) Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRN-110</td>
<td>54.5</td>
<td>78.1</td>
<td>54.3</td>
<td>77.9</td>
</tr>
<tr>
<td>WRN-28-10</td>
<td>50.7</td>
<td>75.0</td>
<td>52.6</td>
<td>76.7</td>
</tr>
</tbody>
</table>

E Derivations

E.1 Derivation of Theorem 1

Common schedules often involve decaying the learning rate and return the final iterate as the solution. In order to compare these against the IA, we consider the 1D quadratic function \( f(w) = \frac{1}{2}||w||^2 \) as a proxy of our true loss function, minimised using gradient descent with learning rate \( \alpha \). At each iteration the gradient is perturbed by some i.i.d. Gaussian noise, \( \epsilon \sim \mathcal{N}(0, \sigma^2) \). The final iterate, with a total training budget \( n \) is given by:

\[
    w_n \sim \mathcal{N}\left((1 - \alpha \lambda)^n w_0, \frac{\alpha \lambda^2 (1 - (1 - \alpha \lambda)^n - 2)}{\lambda}\right)
\]
where we have exploited the formulas for geometric sums and independence and Gaussianity of the noise. For the average iterate $w_{\text{avg}}$, the variance is,

$$\forall j \left( \sum_{i=1}^{n-1-j} (1 - \alpha \lambda)^i \alpha \epsilon_j \right) < \forall j \left( \sum_{i=1}^{n} (1 - \alpha \lambda)^i \alpha \epsilon_j \right),$$

and hence, as there are $n$ such sums we have

$$w_{\text{avg}} \sim N(\tilde{\mu}, \tilde{\sigma}^2) = \frac{[1 - (1 - \alpha \lambda)^{n-2}] w_0}{n \alpha \lambda}, \quad \tilde{\sigma}^2 < \frac{\alpha \sigma^2}{\lambda n}$$

Let us now assume that $n \to \infty$ and that $|\alpha \lambda| \ll 1$ Then the terms above simplify to

$$w_n \sim N(w_0 e^{-n \alpha \lambda}, \frac{\alpha \sigma^2}{\lambda}), \quad w_{\text{avg}} \sim N\left( w_0 \frac{n \alpha \lambda}{\alpha \sigma^2}, \frac{\alpha \sigma^2}{\lambda n} \right)$$

Whilst we attain exponential convergence in the mean for the end point, we do not control the noise in the gradient, whereas for the averaged iterates, although the convergence in the mean is worse (linear), the variance vanishes asymptotically. In higher dimensions, where $P$ is large, the iterates evolve independently along the eigenbasis of $H = \nabla \nabla L$ and assuming isotropic noise:

$$w_n \sim N\left( \sum_{j=1}^{P} \frac{w_{0,j}}{n \alpha \lambda}, \sum_{j=1}^{P} \frac{\alpha \sigma^2}{B \lambda_j n} \right), \quad w_{\text{avg}} \sim N\left( \sum_{j=1}^{P} \frac{w_{0,j} n \alpha \lambda}{\alpha \sigma^2}, \sum_{j=1}^{P} \frac{\alpha \sigma^2}{B \lambda_j n} \right)$$

where $B$ is the minibatch size. Following this, we present Theorem 1 and here we provide the proof sketch:

The basic idea, is that since

$$E \left\| w \right\|^2 = \sum_{i} E (w_i^2) = \sum_{i} \left( (E(w_i))^2 + V(w_i) \right)$$

we expect $\|w\|_2$ to be approximately the square root of this. Our proof follows very closely from [52] (p.51) except that the variables we consider are not zero-mean or unit-variance. We sketch the proof below:

**Proof Sketch:** To show that Theorem 1 is indeed true with high probability, we consider the centered, unit variance version of the random variables i.e $X_i = (\tilde{X}_i - \mu_i) / \sigma_i$

**Lemma 1** (Bernstein’s inequality): Let $\{X_1, \ldots, X_n\}$ be independent, zero mean, sub-exponential random variables. Then for every $t \geq 0$, we have

$$P \left\{ \frac{1}{N} \sum_{i} \sum_{i} X_i \geq t \right\} \leq 2 \exp \left\{ - \min \left( \frac{t^2}{K^2}, \frac{t}{K} \right) N \right\}$$

where $K = \arg \max_i ||X_i||_{\phi_1}$, and $||X||_{\phi_1} = \inf \{ t > 0 : E \exp |X|/t \leq 2 \}$.

**Proof.** The proof is standard and can be found in [52] p.45, essentially we multiply both sides of the inequality by $\lambda$, exponentiate, use Markov’s inequality and independence assumption. Then we bound the MGF of each $X_i$ and optimise for $\lambda$.

Let $X = (X_1, \ldots, X_n) \in \mathbb{R}^P$ be a random vector with independent sub-gaussian coordinates $X_i$, that satisfy $E X_i^2 = 1$. We then apply Bernstein’s deviation inequality (Lemma 1) for the normalized sum of independent, mean zero variables

$$\frac{1}{P} ||X||_2^2 - 1 = \frac{1}{n} \sum_{i} (X_i^2 - 1)$$

(E.3)
Since $X_i$ is sub-gaussian $X_i^2 - 1$ is sub-exponential and by centering and the boundedness of the MGF $||X_i^2 - 1||_{\phi_1} \leq CK^2$ hence assuming $K \geq 1$

$$\mathbb{P}\left\{ \left| \frac{1}{P} \|X\|_2^2 - 1 \right| \geq u \right\} \leq 2 \exp \left( - \frac{cP}{K^4} \min(u^2, u) \right) \quad (E.4)$$

Then using $|z - 1| \geq \delta$ implies $|z^2 - 1| \geq \max(\delta, \delta^2)$

$$\mathbb{P}\left\{ \left| \frac{1}{P} \|X\|_2 - 1 \right| \geq \delta \right\} \leq \mathbb{P}\left\{ \left| \frac{1}{P} \|X\|_2^2 - 1 \right| \geq \max(\delta, \delta^2) \right\} \quad (E.5)$$

changing variables to $t = \delta \sqrt{P}$ we obtain

$$\mathbb{P}\{\|X\|_2 - \sqrt{P} \geq t\} \leq 2 \exp - \frac{ct^2}{K^4} \quad (E.6)$$

for all $t \geq 0$ Our proofs follows by noting that the significance of the 1 in equation $E.5$ is simply the mean of the square and hence by replacing it by the mean squared plus variance we obtain Theorem 1.

E.2 Proof of Theorem 2

The proof of this Theorem is a simple application of the Cauchy-Schwartz inequality. For completeness here we simply show by induction: For $T = 2$

$$\frac{(w_1 + w_2)^2}{4} \leq \frac{w_1^2 + w_2^2}{2} \quad (E.7)$$

$$0 \leq (w_1 - w_2)^2 \quad (E.8)$$

Assume for $T = n$, for $T = n + 1$ i.e. $\left( \frac{1}{n} \sum_{i=1}^{n} w_i \right)^2 \leq \frac{1}{n} \sum_{i=1}^{n} w_i^2$

$$\left( \frac{1}{n + 1} \sum_{i=1}^{n+1} w_i \right)^2 \leq \frac{1}{n} \sum_{i=1}^{n+1} w_i^2 + \sum_{i=1}^{n} w_i^2 \quad (E.9)$$

$$\left( \sum_{i=1}^{n} w_i \right)^2 + 2w_{n+1} \sum_{i=1}^{n} w_i + w_{n+1}^2 \leq n \sum_{i=1}^{n} w_i^2 + w_{n+1}^2 \quad (E.10)$$

$$\leq n \sum_{i=1}^{n} w_i^2 + \sum_{i=1}^{n} w_i^2 + (n + 1)w_{n+1}^2$$

$$0 \leq \sum_{i=1}^{n} (w_i - w_{n+1})^2 \quad (E.11)$$

and by mathematical induction, for any $T = n \geq 2$ and that $n \in \mathbb{N}^+$, we have:

$$\left( \frac{1}{T} \sum_{i=1}^{T} w_i \right)^2 \leq \frac{1}{T} \sum_{i=1}^{T} w_i^2 \quad (E.12)$$

Assuming that $\{ w_i \}$ are drawn from a uniform distribution, Theorem 2 follows.
E.3 Derivation of the Claim that Adam with IA has better expected loss

We argue that with some mild assumptions, in some circumstances averaging iterates of adaptive methods lead to better expected loss bound during training, and now we derive this claim. Formally, from [23], suppose we start averaging at a point \( w_0 \) in the weight space that is different from minimum \( w^* \), the difference between the expected loss at the average of iterates \( E(w_{avg,n}) \) and the loss at minimum \( L(w^*) \) is given by:

\[
E \left( L(w_{avg,n}) \right) - L(w^*) \\
\leq \min \left( \frac{1}{n+1} \text{Tr}(H^{-1} \Sigma_g(w^*)) + \frac{\alpha}{2} \text{Tr}(B^{-1} \Sigma_g(w^*)) \right) \\
+ \min \left( \frac{1}{(n+1)^2 \alpha^2} \|H^{-1/2} B(w_0 - w^*)\|^2, \frac{1}{(n+1) \alpha} \|B^{1/2}(w_0 - w^*)\|^2, 3L(w_0) \right) 
\] (E.13)

For SGD, we have \( B = I \), and thus Equation\[E.13\] reduces to

\[
E \left( L(w_{avg,n}) \right) - L(w^*) \\
\leq \frac{\text{Tr}(H^{-1} \Sigma_g(w^*))}{n+1} + \frac{\|H^{-1/2}(w_0^{\text{SGD}} - w^*)\|^2}{(n+1)^2 \alpha^2} 
\] (E.14)

On the other hand, adaptive methods have general update rule \( w_{n+1} \leftarrow w - \alpha B^{-1} \nabla L_w \), where the inverse of the pre-conditioning matrix \( B^{-1} \) is generally intractable and is often approximated by \( \text{diag}(B^{-1}) \). [1] argues that for Adam, the pre-conditioning matrix approximates the square root of the diagonals of the Fisher Information Matrix, a positive-semidefinite surrogate of Hessian \( H \). Therefore, along these arguments, we assume that in Adam \( B \approx H^{1/2} \). This yields,

\[
E \left( L(w_{avg,n}) \right) - L(w^*) \\
\leq \frac{\text{Tr}(H^{-1} \Sigma_g(w^*))}{n+1} + \frac{\|w_0^{\text{Adam}} - w^*\|^2}{(n+1)^2 \alpha^2} 
\] (E.15)

and therefore, the difference compared to the SGD regret bound is on the noise-independent term. With same number of iterations \( n \), we expect \( \|w_0^{\text{Adam}} - w^*\| < \|w_0^{\text{SGD}} - w^*\| \) due to the faster convergence of Adam (In particular, as argued in the original Adam paper [1], for sparse-bounded gradients, the regret and hence convergence bound derived in App. [E] is reduced from \( \mathcal{O}(\sqrt{n}) \) to \( \mathcal{O}(\log P \sqrt{n}) \)). Furthermore, as argued by [23], when \( H \) is ill-conditioned, \( \|w_0^{\text{SGD}} - w^*\| \) is likely to have large component in small eigenvalue directions, and the pre-conditioning by \( H^{-1/2} \) could lead to \( \|w_0^{\text{Adam}} - w^*\| \ll \|H^{-1/2}(w_0^{\text{SGD}} - w^*)\| \) and as a result, \( E \left( L(w_{avg,n}) \right) < E \left( L(w_{avg,n})^{\text{SGD}} \right) \).

F Importance of Iterate Averaging for Convergence

We argue that despite of the universal practical use of the final iterate of optimisation, it is heuristically motivated and in most proofs of convergence, some form of iterative averaging is required and used implicitly to derive the theoretical bounds. For \( \beta \)-Lipschitz, convex empirical risks, denoted the (overall) loss \( L \). The difference between the \( t + 1 \)th iterate and the optimal solution \( L^*_w \) can be bounded. The sum of differences along the trajectory (known as the regret) telescopes, hence resulting in a convergence rate for the average regret which is an upper bound for the loss of the average point
\[ \delta L = L_{w_{t+1}} - L^* \leq \nabla L_{w_{t}}(w_{t+1} - w^*) + \frac{\beta}{2} \|w_{t+1} - w_t\|^2 \]  

\[ \mathbb{E}(\delta L) \leq \hat{\nabla} L_{w_{t}}(w_t - w^*) - (\alpha - \frac{\beta \alpha^2}{2})\|\hat{\nabla} L_{w_{t}}\|^2 + \alpha \sigma_t^2 \]  

where \( \hat{\nabla} L_{w_{t}} \) is the noisy gradient at \( w_t \) and \( \sigma_t^2 \) is its variance: \( \text{Var}(\hat{\nabla} L_{w_{t}}) \). Noting that \( w_{t+1} = w_t - \alpha \hat{\nabla} L_{w_{t}} \):

\[ \frac{R}{T} = \mathbb{E}\left[ \frac{1}{T} \sum_{t=1}^{T-1} L_{w_{t+1}} - L^* \right] \]  

Using Jensen’s inequality, we have:

\[ \frac{R}{T} \leq \frac{1}{T} \sum_{t=0}^{T-1} \frac{\|w_t - w^*\|^2 - \|w_{t+1} - w^*\|^2}{2\alpha} + \alpha \sigma_t^2 \]  

\[ \mathbb{E}[L_T \sum_{t=0}^{T-1} w_{t+1} - L^*] \leq \frac{R}{T} \leq \frac{\|w_0 - w^*\|^2}{2\alpha T} + \alpha \sigma_m^2 \]  

where \( \sigma_m^2 = \arg\max_{w_0} \mathbb{E}[\|\nabla L_{w_{T}} - \nabla L_{w_{t}}\|^2] \), and \( R \) is the regret. Setting \( \alpha = (\beta + \frac{\sigma \sqrt{T}}{D})^{-1} \) in equation \( \text{(F.2)} \) gives us the optimal convergence rate. Similar convergence results can be given for a decreasing step size \( \alpha_t \propto t^{-1/2} \alpha_0 \). For adaptive optimisers, the noisy gradient is preconditioned by some non-identity matrix \( \bar{B}^{-1} \):

\[ w_{k+1} \leftarrow w_k - \alpha \bar{B}^{-1} \nabla L_k(w) \]  

Methods of proof \[54, 55\] rely on bounding the regret \( O(\sqrt{T}) \) and showing that the average regret \( \frac{R}{T} \to 0 \) and Equation \( \text{(F.2)} \) explicitly demonstrates that the average regret is an upper bound on the expected loss for the average point in the trajectory. Hence existing convergence results in the literature prove convergence for the iterate average, but not the final iterate.

**Optimal Learning Rates** Setting \( \alpha = (\beta + \frac{\sigma \sqrt{T}}{D})^{-1} \) gives us the optimal convergence rate of \( \frac{\beta R^2}{T} + \frac{\sigma D}{\sqrt{T}} \). Similar convergence results can be given for a decreasing step size \( \alpha_t \propto t^{-1/2} \alpha_0 \) \[53\] when the number of iterations \( T \) is not known in advance. Given the use of both iterate averaging and learning rate schedule in the proofs, it is difficult to understand the relative importance of the two and how this compares with the typical heuristic of using the final point.