Efficient Full-Matrix Adaptive Regularization

Naman Agarwal [1] **Brian Bullins** [1 2] **Xinyi Chen** [1] **Elad Hazan** [1 2] **Karan Singh** [1 2] **Cyril Zhang** [1 2] **Yi Zhang** [1 2]

Abstract

Adaptive regularization methods pre-multiply a descent direction by a preconditioning matrix. Due to the large number of parameters of machine learning problems, full-matrix preconditioning methods are prohibitively expensive. We show how to modify full-matrix adaptive regularization in order to make it practical and effective. We also provide a novel theoretical analysis for adaptive regularization in *non-convex* optimization settings. The core of our algorithm, termed GGT, consists of the efficient computation of the inverse square root of a low-rank matrix. Our preliminary experiments show improved iteration-wise convergence rates across synthetic tasks and standard deep learning benchmarks, and that the more carefully-preconditioned steps sometimes lead to a better solution.

1. Introduction

Stochastic gradient descent is the workhorse behind the recent deep learning revolution. This simple and age-old algorithm has been supplemented with a variety of enhancements to improve its practical performance, and sometimes its theoretical guarantees.

Amongst the acceleration methods there are three main categories: momentum, adaptive regularization, and variance reduction. Momentum (in its various incarnations, like heavy-ball or Nesterov acceleration) is the oldest enhancement. It has a well-developed theory, and is known to improve practical convergence in a variety of tasks, small and large. It is also easy to implement. Variance reduction is the most recent advancement; in theory and practice, it is mostly applicable to convex optimization, and is thus less influential in deep learning.

[1]Google AI Princeton [2]Department of Computer Science, Princeton University. Correspondence to: Cyril Zhang <cyril.zhang@cs.princeton.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

This brings us to adaptive regularization: the most sophisticated, hard to implement, and widely-debated acceleration method. While state-of-the-art optimizers such as Adam and AdaGrad (Kingma & Ba, 2014; Duchi et al., 2011) do use adaptive regularization, they exclusively restrict the preconditioner to a diagonal matrix; as such, they often marketed as per-coordinate "adaptive learning-rate" methods. Despite solid theoretical guarantees, the practical value of diagonal adaptive regularization as compared to "vanilla" SGD has been the subject of much debate (Wilson et al., 2017). However, the efficacy of full-matrix adaptive regularization has been relatively unexplored. This is due to the prohibitive computational cost associated with full-matrix operations: full AdaGrad requires taking the inverse square root of a large matrix.

In this paper, we present GGT, a practical solution to the computational problems plaguing full-matrix adaptive regularization, making this technique scalable for modern deep models. At the heart of our method is a simple, GPU-friendly way to apply the inverse square root of the low-rank second-moment matrix of recent gradients; see Figure 1. GGT's running time is comparable to state-of-the-art optimizers.

We proceed to show that full-matrix preconditioning allows for much better exploitation of anisotropic curvature in loss landscapes. First, we show synthetic experiments which demonstate clear benefits of GGT over baselines, especially when the problem is ill-conditioned. Then, we implement GGT at scale, and show that the benefits translate to faster training on standard deep learning benchmarks. Our improvement is most salient in complicated landscapes like RNN training.

Our algorithm comes with theoretical guarantees. We give the first proof of convergence to first-order critical points for an algorithm with adaptive regularization in a stochastic non-convex setting, featuring a rate which is dependent on an *adaptive ratio*. We show examples where our bound is stronger than that for SGD, providing some theoretical basis for our empirical findings.

1.1. Related work

Since the introduction of AdaGrad (Duchi et al., 2011), diagonal adaptive regularization has been a mainstay in the

machine learning practitioner's toolbox. A quick perusal of the literature shows that these methods have continued to thrive in the deep learning era, and appear in all major frameworks (Abadi et al., 2016; Paszke et al., 2017; Chen et al., 2015). By citation count (or GitHub search hits), Adam (Kingma & Ba, 2014) is by far the most popular adaptive optimizer for training a variety of modern deep models. For this reason, this paper's exposition is targeted towards a full-matrix drop-in replacement for Adam; however, our techniques extend straightforwardly to a plethora of variants, like RMSprop (Tieleman & Hinton, 2012), Adadelta (Zeiler, 2012), Nadam (Dozat, 2016), etc.

Full-matrix adaptive regularization has existed alongside the more commonly used diagonal-matrix manifestation since their common inception in (Duchi et al., 2011); however, a major obstacle to the scalability of these methods is the need for the storage and inversion of square matrices in the model dimension. This becomes prohibitively expensive in dimension greater than 10^4, while state-of-the-art models regularly exceed 10^7 parameters.

Matrix sketching has been employed to approximate the AdaGrad preconditioner (Krummenacher et al., 2016; Mehta et al., 2016); however, the sketched estimate for the matrix inverse can be sensitive to noise. Furthermore, there is a sizeable performance gap which renders these methods unsuitable for large-scale implementation. For example, in the former reference, the authors report a 5-10$\times$ overhead over AdaGrad, even with $< 10^5$ model parameters; we could not find a usable GPU implementation for their requisite rank-1 QR update. Rather than sketching the preconditioner, our computations are exact, and efficient in practice.

(Gupta et al., 2018) propose a way to do AdaGrad with Kronecker products of full-matrix preconditioners, a more limited setting which requires knowledge of the model's structure. Finally, as we argue in Section 3.1, there is intrinsic value of "forgetting" past curvature using an exponential window. With this, a low-rank preconditioning matrix naturally arises, allowing us to bypass the computational need for matrix sketching in the model dimension or architecture-dependent restriction of the preconditioner.

Our algorithm bears a superficial resemblance to L-BFGS (Liu & Nocedal, 1989): both compute a preconditioner using a sliding window of gradient history. However, L-BFGS uses differences of gradients to estimate the Hessian, while we use the gradients to keep an exact (exponentially decayed) gradient Gram matrix. In the former, estimating the Hessian with *stochastic* gradients is very unstable. In a similar vein, stochastic second-order methods (e.g. Erdogdu & Montanari, 2015; Agarwal et al., 2017b; Luo et al., 2016; Hazan et al., 2007; Agarwal et al., 2017a; Carmon et al., 2017) have seen limited adoption in deep learning. This is partly due to prohibitive overhead costs (despite asymptotic efficiency); however, more fundamentally, they exhibit very different training dynamics than the typical family of optimizers in deep learning.

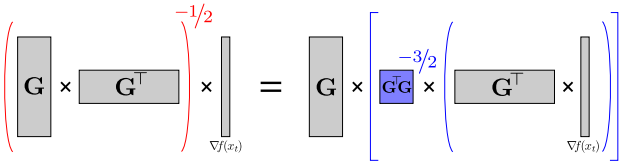

Figure 1. Sketch of how GGT performs fast full-matrix preconditioning. Note that the inverse matrices are understood here to be Moore-Penrose pseudoinverses; see Section 2.1 for a full treatment.

Natural gradient descent (Amari, 1998) and K-FAC (e.g. Martens & Grosse, 2015; Martens et al., 2018) precondition the gradient by the inverse (rather than inverse square root) of the gradient Gram matrix, or an efficiently invertible approximation thereof. Like in the above discussion, the training dynamics arising from these methods diverge from those of standard adaptive optimizers, as well as the underlying theoretical motivations.

Several recent works (Li & Orabona, 2018; Zou & Shen, 2018; Ward et al., 2018; Chen et al., 2018a;b; Zhou et al., 2018) have studied the convergence of adaptive methods for non-convex optimization, matching the asymptotic iteration complexity of SGD. (Staib et al., 2019) show that adaptive methods escape saddle points. Apart from our algorithmic contribution, our work is (to our knowledge) the first attempt to characterize the *advantage* of adaptivity in terms of the dimension and geometry of the optimization problem.

2. The GGT algorithm

Our main algorithmic contribution is GGT, an efficient first-order algorithm for full-matrix adaptive preconditioning. In brief, GGT uses the preconditioner from full-matrix AdaGrad, with gradient history attenuated exponentially as in Adam, and truncated to a window parameter r. The name GGT acts as a convenient mnemonic for the gradient second-moment matrix $\mathbf{G}\mathbf{G}^\top$ maintained by full-matrix AdaGrad, even though *we never compute this matrix*.

The mathematical specification of GGT is given in Algorithm 1, in the usual model of stochastic optimization (see Section 4), with gradients $\widetilde{\nabla} f(x)$. Notice that the coordinate-wise scaling of Adam is recovered by zeroing out the off-diagonal entries of $\mathbf{G}\mathbf{G}^\top$.

GGT provides the power of full-matrix adaptive regularization at a cost not much larger than SGD. This crucially exploits the fact only a small window of historical gradi-

Algorithm 1 GGT adaptive optimizer

Input: initializer x_1, window size r, learning rate schedule $\{\eta_t\}$, $\beta_2 \leq 1, \varepsilon > 0$.
for $t = 1, \ldots, T$ **do**
  Receive stochastic gradient $\widetilde{\nabla} f(x_t)$.
  Let $\mathbf{G}_t = [g_t \; g_{t-1} \; \ldots \; g_{t-r+1}]$, where $g_{t-k} := \beta_2^k \widetilde{\nabla} f(x_{t-k})$, or $\mathbf{0}$ if $k \geq t$.
  $x_{t+1} \leftarrow x_t - \eta_t \cdot [(\mathbf{G}_t \mathbf{G}_t^\top)^{1/2} + \varepsilon \mathbf{I}]^{-1} \widetilde{\nabla} f(x_t)$.
end for

ents are used for preconditioning. The intuition for using a small window, as opposed to the entire history, is clear (and time-tested, by the ubiquity of Adam): the curvature of the loss surface changes, rendering previous gradient information obsolete. We expand on the benefits of forgetting gradients in section 3.1.

The fact that the preconditioning matrix is based on a small window of gradients implies that it has low rank. GGT exploits this fact by computing the inverse square root of the empirical covariance matrix indirectly, as outlined in Figure 1. In effect, instead of inverting a full matrix in the dimension of parameters, using the special matrix structure GGT inverts a matrix of dimension window-size. The remainder of this section will discuss efficient implementation and some heuristics.

GGT has provable guarantees even for non-convex optimization: it is guaranteed to converge to a first-order critical point. Its rate of convergence is never significantly slower than that of SGD, and in some favorable geometric conditions, can be significantly faster. These theoretical bounds are made precise in Section 4.

2.1. Fast low-rank preconditioning

The window parameter r should be roughly the number of copies of the model that fit in RAM; in our large-scale experiments, we use $r = 200$. A pessimistic but principled choice is $r = \Theta(1/(1-\beta_2))$, which truncates on the time scale of the exponential attenuation. Our key observation, highlighted in Figure 1, is that the inversion of the large low-rank matrix $\mathbf{G}\mathbf{G}^\top$ can be performed by diagonalizing the small matrix $\mathbf{G}^\top\mathbf{G}$, along with some extremely GPU-friendly matrix-vector operations.

The basic intuition is contained in Figure 1, but it remains to include the $\varepsilon\mathbf{I}$ term. We derive the full update here. Let $\mathbf{G} \in \mathbb{R}^{d\times r}$, $v \in \mathbb{R}^d$ be arbitrary, with $r \leq d$. Write the singular value decomposition $\mathbf{G} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, with $\mathbf{U} \in \mathbb{R}^{d\times d}, \mathbf{\Sigma} \in \mathbb{R}^{d\times r}, \mathbf{V} \in \mathbb{R}^{r\times r}$. Let $\mathbf{\Sigma}_d \in \mathbb{R}^{d\times d} := [\mathbf{\Sigma} \; 0]$, and let $\mathbf{\Sigma}_r \in \mathbb{R}^{r\times r}$ be its top left block. Let $\mathbf{U} =: [\mathbf{U}_r \; \mathbf{U}_{d-r}]$, so that the columns of $\mathbf{U}_r \in \mathbb{R}^{d\times r}$ are an orthonormal basis for the column space of $\mathbf{G}$, and $\mathbf{U}_{d-r} \in \mathbb{R}^{d\times(d-r)}$ its orthogonal component, noting that $\mathbf{U}_r\mathbf{U}_r^\top + \mathbf{U}_{d-r}\mathbf{U}_{d-r}^\top = \mathbf{I}_d$. Then, we have

$$\begin{aligned}
\left[(\mathbf{G}\mathbf{G}^\top)^{1/2} + \varepsilon\mathbf{I}\right]^{-1} v &= \left[(\mathbf{U}\mathbf{\Sigma}_d^2\mathbf{U}^\top)^{1/2} + \varepsilon\mathbf{U}\mathbf{U}^\top\right]^{-1} v \\
&= \mathbf{U}(\mathbf{\Sigma}_d + \varepsilon\mathbf{I})^{-1}\mathbf{U}^\top v \\
&= \left[\mathbf{U}_r(\mathbf{\Sigma}_r + \varepsilon\mathbf{I}_r)^{-1}\mathbf{U}_r^\top + \mathbf{U}_{d-r}(\varepsilon\mathbf{I}_{d-r})^{-1}\mathbf{U}_{d-r}^\top\right] v \\
&= \mathbf{U}_r(\mathbf{\Sigma}_r + \varepsilon\mathbf{I}_r)^{-1}\mathbf{U}_r^\top v + \frac{1}{\varepsilon}(\mathbf{I}_d - \mathbf{U}_r\mathbf{U}_r^\top)v \\
&= \frac{1}{\varepsilon}v + \mathbf{U}_r\left[(\mathbf{\Sigma}_r + \varepsilon\mathbf{I}_r)^{-1} - \frac{1}{\varepsilon}\mathbf{I}_r\right]\mathbf{U}_r^\top v.
\end{aligned}$$

The first term is none other than an SGD update step. The rest can be computed by taking the eigendecomposition $\mathbf{G}^\top\mathbf{G} = \mathbf{V}\mathbf{\Sigma}_r^2\mathbf{V}^\top$, giving $\mathbf{U}_r = \mathbf{G}\mathbf{V}\sqrt{\mathbf{\Sigma}_r}^\dagger$. We prefer this to taking the direct SVD of $\mathbf{G}$, which is > 10 times slower on GPU.

Using a cyclic buffer to store and update $\mathbf{G}_t$, the algorithm takes $O(dr^2 + r^3)$ (sequential) time per iteration, and $O(dr)$ memory in total. Iterating over the model parameters to update $\mathbf{G}_t$ incurs the same overhead cost as usual adaptive optimizers. The $r \times d$ matrix multiplication and $r \times r$ SVD operations benefit from decades of extensive hardware-level optimizations.

In the experiments in Section 3, we observed a $\sim 1.3\times$ (CNN) and $\sim 2\times$ (RNN) running-time overhead compared to SGD; we note that this ratio could be even smaller in reinforcement learning (where the environment causes the time bottleneck), or universally with a more optimized implementation.

2.2. Tweaks for GGT on deep models

Below, we list some practical suggestions for applying GGT to training large-scale models.

Momentum. In order to bring GGT closer to a drop-in replacement for Adam, we can add momentum to the gradient steps: let $v_t \leftarrow \beta_1 v_{t-1} + \widetilde{\nabla} f(x_t)$, and apply the preconditioner to v_t to compute the update step. We use momentum in all large-scale experiments, with the standard $\beta_1 = 0.9$. We also get a small performance boost by using v_t instead of the gradients to update $\mathbf{G}_t$. On the other hand, as long as $r \ll T$, it makes little difference to choose $\beta_2 = 1$, letting the window (rather than exponential attenuation) forget stale gradient information.

Interpolation with SGD. We note the possibility of decoupling the scalars ε and $1/\varepsilon$ which appear in the efficient update step. Appealingly, this allows the user to tune GGT's behavior to be arbitrarily close to that of SGD.

Numerical concerns. For greater numerical stability, it is possible to add a small multiple of the identity matrix (we suggest 10^{-6}) to $\mathbf{G}^\top\mathbf{G}$ before computing its eigendecom-

position, without noticeable differences in training.

3. Experiments

In this section, we present an empirical study of GGT. We begin with some simple experiments, showing that adaptive methods help in the presence of ill-conditioned optimization problems, as well as the value of limited gradient memory. Next, we evaluate the performance of GGT on larger-scale deep learning tasks (and provide some additional such experiments in Appendix B). Finally, we present some interesting empirical insights on the training dynamics in deep learning models. Our visualizations of gradient spectra suggest that adaptive optimizers are indeed correcting for changing anisotropic curvature in the loss landscape.

3.1. Synthetic data: when do adaptivity and forgetfulness help?

The original theorems on the behavior of adaptive first-order methods are established from the perspective of online convex optimization (Duchi et al., 2011). The dynamics are less understood on realistic loss landscapes in stochastic optimization. For this reason, we begin our experimental section with some simple empirical comparisons between full- and diagonal-matrix adaptive optimizers and SGD. Figure 2 summarizes our findings.

In each synthetic experiment, we generated an ill-conditioned landscape, and compared SGD with adaptive optimizers, excluding the typical accompanying heuristics (i.e. no momentum, regularization, or learning rate schedule). We tested diagonal-matrix preconditioners with and without exponential gradient attenuation (like Adam and AdaGrad, respectively), and their full-matrix analogues. The experiments were robust with respect to the choice of ε (we used 10^{-4}) and batch size.

In the first synthetic experiment *(left)*, we exhibit an instance of logistic regression in dimension 10, with 10^3 samples generated from an extremely anisotropic ($\sigma^2_{\max}/\sigma^2_{\min} \approx 10^4$) Gaussian distribution, and binary labels determined by a random hyperplane. SGD converges the slowest, and diagonal AdaGrad consistently accelerates optimization. Finally, full-matrix preconditioning (using cubic-time matrix inversion) converges the fastest. In this setting, adding a window improved convergence, but not drastically; we elaborate below.

Next, we show an optimization problem *(right)* which accentuates the utility of exponentially decaying gradient memory. We consider the problem of minimizing the logarithmic barrier function of a randomly generated anisotropic polytope, otherwise known as finding its *analytic center*: this replaces the logistic loss terms with $f_i(w) = -\log(w^\top x_i + c_i)$, with x_i generated the same way as above, and c_i generated uniformly from $[0, 1]$. We observed the same ranking of convergence rates as in the first experiment, but the improvement afforded by the window was much clearer.

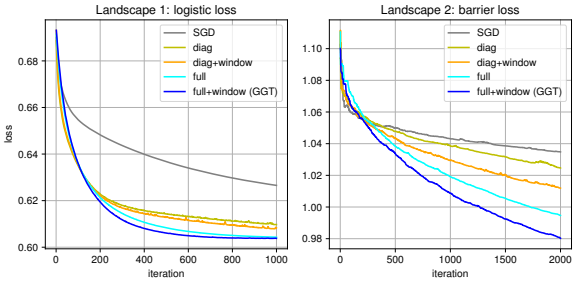

Figure 2. Synthetic experiments on convex loss functions, demonstrating the value of adaptive regularization and attenuation of gradient history. *Left:* An ill-conditioned instance of logistic regression. Adaptive regularization finds a good preconditioner, accelerating optimization. *Right:* Minimizing a barrier function, an example where the curvature changes with position. Optimization is further accelerated by *forgetting* outdated gradient information.

The primary conclusion of our synthetic experiments is to demonstrate some small-scale settings in which adaptive regularization ameliorates anisotropy in the optimization landscape. A subtler point is that the windowed variants can help with changing curvature, even for convex losses. Note that the curvature of the former landscape is constant (in that its Hessian matrix at different locations w only changes by a scalar factor). The latter setting, in contrast, features a changing curvature (its Hessians do not commute in general), necessitating "forgetfulness" in adaptive curvature estimation.

In Section 3.4, we will return to these proof-of-concept optimization instances, connecting them to an empirical study of curvature in more realistic landscapes.

3.2. GGT on deep convolutional models

We investigated the training dynamics of GGT on a typical deep architecture for computer vision. For this, we used a 26-layer 3-branch residual network with Shake-Shake regularization (Gastaldi, 2017). Aside from its ability to reach state-of-the-art classification accuracy, this architecture also features a relatively low parameter count ($\sim$3M), enabling the use of a large window parameter ($r = 200$).

In each experiment, we kept the cosine learning rate annealing schedule used in the paper, originally from (Loshchilov & Hutter, 2016); performance degraded consistently and significantly with a fixed learning rate. For both Adam and GGT, we chose the commonly used parameters $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$; for SGD, we

used momentum with parameter 0.9. With correctly tuned RMSprop and Adadelta, with the same window parameters, training curves were virtually identical to those for Adam. We used a batch size of 128, and the standard data augmentation techniques of 4-pixel padding + random cropping and horizontal flipping.

Our results are shown in Figure 3 *(top)*. In terms of training loss, GGT consistently dominated existing optimizers. We corroborate a number of observations from previous empirical studies of the generalization of optimizers. Most prominently, we found that SGD generalized slightly better than all others (Wilson et al., 2017; Keskar & Socher, 2017) towards the end of training, including ours. The gap ($< 0.2\%$) is less dramatic than that seen in (Wilson et al., 2017) for two reasons: we only show curves with a tuned and annealed learning rate; also, we use an architecture with powerful explicit regularization techniques which have gained attention since their publication. Our preliminary observation is that GGT shrinks this gap slightly (corroborated by another experiment in Appendix B), and expect that there is vastly more empirical work to be done concerning architectures synergistically tuned to existing optimizers.

We also verify the long-held empirical observation that the learning rate decay of AdaGrad is too aggressive (e.g. in (Zeiler, 2012)), resulting in convergence to a poor solution. Finally, in agreement with (Wilson et al., 2017), we find that using a sufficiently low learning rate for any optimizer can result in a better training loss curve, but not without significantly degrading generalization ($> 3\%$ worse).

3.3. GGT on recurrent models

Next, we move to recurrent architectures for language modeling. We train a 3-layer LSTM (Hochreiter & Schmidhuber, 1997) with ~ 5M parameters for character-level modeling of the Penn Treebank dataset (Marcus et al., 1994). This is the setting in which we observe the most striking improvement over baselines. The particularities of this optimization task, and why it might be especially amenable to full-matrix regularization, remain a fruitful research direction (Pascanu et al., 2013). Figure 3 *(bottom)* shows training and validation perplexities for the first 50 epochs; no optimizer makes significant progress afterwards.

The state of the art for character-level language modeling is less thoroughly documented than its word-level counterpart, though we note that our end-to-end result (validation perplexity 2.42 after 500 epochs) is competitive with those reported for recurrent models, like by (Krueger et al., 2016). In contrast, Adam, AdaGrad, and SGD reach 2.51, 2.65, and 2.76, respectively. Note that Adam is the *de facto* standard optimizer for language modeling (Melis et al., 2017). Even with iterations taking twice the time, we outperform all baselines in wall-clock time throughout training.

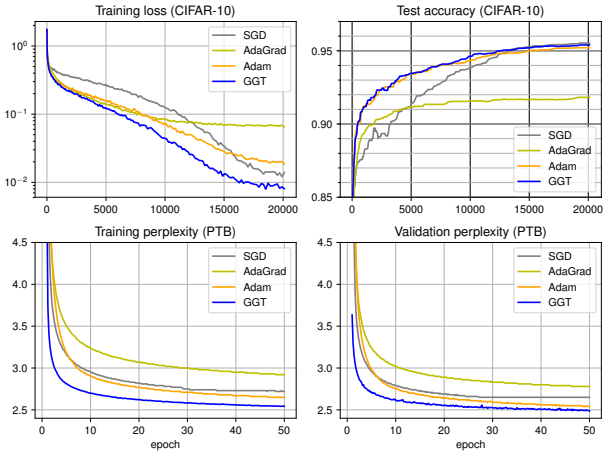

Figure 3. Results of CNN and RNN experiments. GGT dominates in training loss across both tasks, and generalizes better on the RNN task. *Top:* CIFAR-10 classification with a 3-branch ResNet. *Bottom:* PTB character-level language modeling with a 3-layer LSTM.

We also tried using GGT as a drop-in replacement for Adam in the state-of-the-art word-level language modeling code accompanying (Merity et al., 2017; 2018). Although we were competitive with Adam, we only observed an improvement in the first ~ 20 epochs. We hypothesize that the advantage of full-matrix regularization in this setting is more marginal, as the gradients in the embedding layers are naturally sparse in the vocabulary ("one-hot") basis. On a similar note, we found that Adam outperformed GGT on attention-based architectures for NLP; refer to Appendix B for an experiment and discussion.

3.3.1. Wall clock time comparisons

For those interested in end-to-end performance in terms of model training speed, we provide in Figure 4 an alternate visualization for the large-scale experiments, replacing the epoch count with total cumulative training time on the horizontal axis. On the LSTM task, GGT outperforms the baselines throughout training (and converges upon a better solution), even with the additional overhead running time.

The same unconditional improvement was not observed in the vision task, for training convergence nor generalization. We believe this is due to the interactions between modern convolutional architectures and the epoch-dependent learning rate schedule, which we have not attempted to re-tune. Indeed, recent state-of-the-art work in rapid training of convolutional neural nets is centered on a selection of learning rate and momentum schedules, rather than step directions.

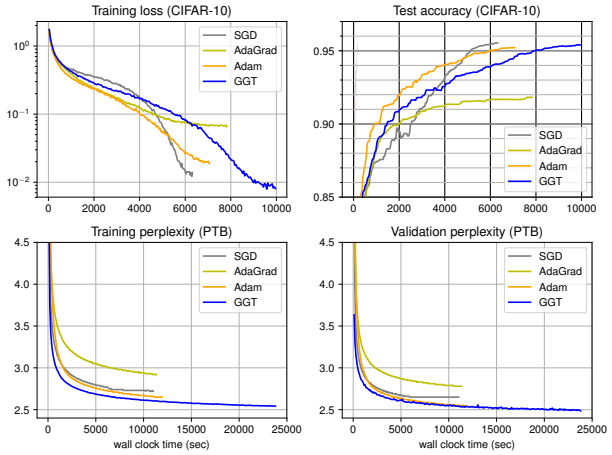

Figure 4. Alternate presentation of Figure 3, with wall clock time on the horizontal axis. *Top:* CIFAR-10 classification with a 3-branch ResNet. *Bottom:* PTB character-level language modeling with a 3-layer LSTM.

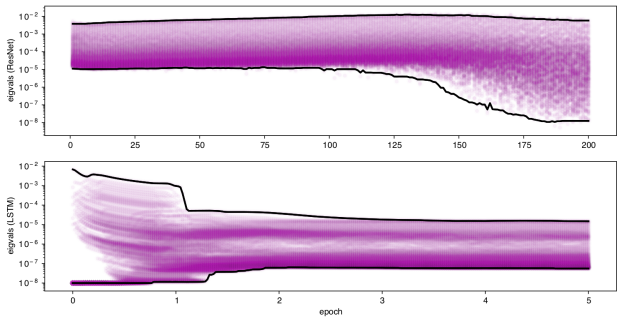

Figure 5. Evolution of the spectrum of the gradient matrix during training. Each vertical slice is a density heatmap of the eigenvalues of $\mathbf{G}_t^\top \mathbf{G}_t$. The black lines indicate the minimum and maximum eigenvalues, smoothed in time by a median filter. *Top:* CNN training. Approaching the end of training, the gradients become more anisotropic. *Bottom:* RNN training. Within the first few epochs, the gradients become more isotropic, then stabilize. (Truncated to 5 epochs; the density was visually stable for the remainder of training.)

3.4. Empirical insights on the spectral decay

In this section, we unify the insights gleaned from the synthetic experiments and deep learning benchmarks. Along the way, we provide some interesting anecdotal observations on the evolution of the preconditioner matrices' singular values.

We plot the density of the spectrum of the low-rank preconditioner $\mathbf{G}_t\mathbf{G}_t^\top$ as training progresses. Since the fast implementation of GGT takes an eigendecomposition of $\mathbf{G}_t^\top\mathbf{G}_t$, we can read off the distribution of eigenvalues during training at no additional computational cost. Figure 5 visualizes the result of this experiment for the CNN and RNN training settings from the previous two sections. In each case, we observe that $\mathbf{G}_t^\top\mathbf{G}_t$ has a condition number of $\sim 10^3$, noting that this can be visualized as the vertical range in the logarithmic plot.

This visualization affords a new way to see how CNN and RNN landscapes are fundamentally different: their gradient spectra evolve in very distinct ways over the course of training. Interestingly, the condition number of the CNN landscape surges near the end, which may be related to the the low-rank structure of well-trained nets noted by (Arora et al., 2018), who derive rank-dependent generalization bounds for neural networks. On recurrent models, the rapidly evolving spectral structure at the early stage of training indicates a possibly more complex landscape. Intriguingly, the enormous condition number ($\sim 10^6$) correlates with the massive lead of GGT over the others, confirming our intuition that full-matrix preconditioning ameliorates anisotropy.

To our knowledge, this is the first empirical study of this kind, using the covariance matrix of recent gradients as a surrogate to examining the changing curvature of the loss landscape. In the spirit of recent empirical lenses of this flavor (Raghu et al., 2017; Li et al., 2017), we leave this as a way to visualize deep learning dynamics, possibly of independent exploratory interest.

4. A convergence rate analysis with adaptivity

In this section we outline our analysis of GGT, for which we show convergence to an approximate first-order critical point, in some settings faster than SGD. To obtain the strongest theory, we analyze GGT with a "hard window" instead of exponentially decaying gradient memory, explained in Section A.2.

We work in the usual theoretical framework of stochastic optimization of a differentiable non-convex function $f(\cdot)$, equipped with an unbiased variance-bounded stochastic gradient oracle $\widetilde{\nabla} f(\cdot)$. The objective, as is standard in the literature (see, e.g. (Ghadimi & Lan, 2013; Allen-Zhu & Hazan, 2016)), is to find an ε-approximate stationary point x; that is, $\|\nabla f(x)\| \le \varepsilon$.

4.1. The adaptive ratio

We quantify the improvement of adaptive regularization by its advantage over the usual worst-case bound of SGD. To this end, we define the *adaptive ratio* μ of an algorithm $\mathcal{A}$ as

$$\mu \stackrel{\text{def}}{=} \frac{f(x_{\mathcal{A}}) - f(x^*)}{\|x_1 - x^*\|_2 \cdot \frac{\sigma}{\sqrt{T}}},$$

where $x_{\mathcal{A}}$ is the output of the $\mathcal{A}$, and x^* is a comparator. For convex optimization problems x^* is naturally the global minimum. For non-convex optimization it is a subtler choice, which we detail in Appendix A.

This ratio for the AdaGrad algorithm was shown in (Duchi et al., 2011) to be always bounded by a quantity independent of T, and potentially much smaller. Specifically, it was shown to be inversely proportional to the dimension in certain convex optimization problems, providing a theoretical justification for the speedup of adaptive optimizers. In Section A.4, we show a new, simple, and natural setting illustrating adaptive speedup, even for a *strongly convex* function f.

4.2. Adaptive convergence rate guarantee

We informally state the main theorem below. We defer the full bound without suppressed smoothness constants, as well as all technical proofs, to Appendix A.

Theorem 4.1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a bounded, Lipschitz, and smooth function with stochastic gradient oracle $\widetilde{\nabla} f(\cdot)$, whose variance is at most σ^2. In expectation, Algorithm 2 outputs an ε-approximate critical point of f, with $\widetilde{O}\left(\frac{\mu^2\sigma^2}{\varepsilon^4}\right)$ calls to $\widetilde{\nabla} f(\cdot)$.*

This theorem matches and potentially improves the known analysis for stochastic gradient descent with the introduction of the data-dependent adaptivity constant μ into the leading-order term governing the rate of convergence. Since (Duchi et al., 2011) bounded μ by a quantity independent of T, our theorem matches the classic $O\left(\varepsilon^{-4}\right)$ rate of convergence.

5. Conclusion

This work investigates full-matrix adaptive regularization: our main contribution is to make this technique viable for large-scale optimization, by a method for efficient multiplication by the inverse square root of a full second-moment matrix over a short window of gradients. This leads to a new algorithm, GGT, a truly scalable optimization algorithm with full-matrix adaptive preconditioning.

Through synthetic experiments, we have shown that GGT accelerates optimization in ill-conditioned loss landscapes; this is supported by accompanying adaptive convergence guarantees. Preliminary experiments show accelerated convergence on standard deep learning benchmarks, with very different training dynamics from existing diagonal adaptive methods. We accompany our algorithm and experiments with the first theoretical characterization of the benefits of adaptive regularization in a non-convex setting. We hope that GGT will be the first of a new class of algorithms for the modern large-scale optimization toolbox, and to foster new discussion towards an ever-elusive understanding of loss landscapes in deep learning.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding approximate local minima faster than gradient descent. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1195–1199. ACM, 2017a.

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18 (1):4148–4187, 2017b.

Allen-Zhu, Z. and Hazan, E. Variance reduction for faster non-convex optimization. In *International Conference on Machine Learning*, pp. 699–707, 2016.

Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. "convex until proven guilty": Dimension-free acceleration of gradient descent on non-convex functions. In *International Conference on Machine Learning*, pp. 654–663, 2017.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. MxNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Chen, X., Liu, S., Sun, R., and Hong, M. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*, 2018a.

Chen, Z., Yang, T., Yi, J., Zhou, B., and Chen, E. Universal stagewise learning for non-convex problems with convergence on averaged solutions. *arXiv preprint arXiv:1808.06296*, 2018b.

Dozat, T. Incorporating Nesterov momentum into Adam. 2016.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Erdogdu, M. A. and Montanari, A. Convergence rates of sub-sampled newton methods. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pp. 3052–3060. MIT Press, 2015.

Gastaldi, X. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

Ghadimi, S. and Lan, G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Gupta, V., Koren, T., and Singer, Y. Shampoo: Preconditioned stochastic tensor optimization. *arXiv preprint arXiv:1802.09568*, 2018.

Hazan, E., Agarwal, A., and Kale, S. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Keskar, N. S. and Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Krummenacher, G., McWilliams, B., Kilcher, Y., Buhmann, J. M., and Meinshausen, N. Scalable adaptive stochastic optimization using random projections. In *Advances in Neural Information Processing Systems*, pp. 1750–1758, 2016.

Li, H., Xu, Z., Taylor, G., and Goldstein, T. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

Li, X. and Orabona, F. On the convergence of stochastic gradient descent with adaptive stepsizes. *arXiv preprint arXiv:1805.08114*, 2018.

Liu, D. C. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

Loshchilov, I. and Hutter, F. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Luo, H., Agarwal, A., Cesa-Bianchi, N., and Langford, J. Efficient second order online learning by sketching. In *Advances in Neural Information Processing Systems*, pp. 902–910, 2016.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pp. 114–119. Association for Computational Linguistics, 1994.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pp. 2408–2417, 2015.

Martens, J., Ba, J., and Johnson, M. Kronecker-factored curvature approximations for recurrent neural networks. 2018.

Mehta, N. A., Rendell, A., Varghese, A., and Webers, C. Compadagrad: A compressed, complementary, computationally-efficient adaptive gradient method. *arXiv preprint arXiv:1609.03319*, 2016.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Merity, S., Keskar, N. S., and Socher, R. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. Svcca: Singular vector canonical correlation analysis for deep understanding and improvement. *arXiv preprint arXiv:1706.05806*, 2017.

Staib, M., Reddi, S. J., Kale, S., Kumar, S., and Sra, S. Escaping saddle points with adaptive gradient methods. *arXiv preprint arXiv:1901.09149*, 2019.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Ward, R., Wu, X., and Bottou, L. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4151–4161, 2017.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhou, D., Tang, Y., Yang, Z., Cao, Y., and Gu, Q. On the convergence of adaptive gradient methods for nonconvex optimization. *arXiv preprint arXiv:1808.05671*, 2018.

Zou, F. and Shen, L. On the convergence of adagrad with momentum for training deep neural networks. *arXiv preprint arXiv:1808.03408*, 2018.