Adaptive Curvature for Stochastic Optimization

by

Trevor Barron

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved February 2019 by the
Graduate Supervisory Committee:

Heni Ben Amor, Chair
Jingrui He
Martin Levihn

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

This thesis presents a family of adaptive curvature methods for gradient-based stochastic optimization. In particular, a general algorithmic framework is introduced along with a practical implementation that yields an efficient, adaptive curvature gradient descent algorithm. To this end, a theoretical and practical link between curvature matrix estimation and shrinkage methods for covariance matrices is established. The use of shrinkage improves estimation accuracy of the curvature matrix when data samples are scarce. This thesis also introduce several insights that result in data- and computation-efficient update equations. Empirical results suggest that the proposed method compares favorably with existing second-order techniques based on the Fisher or Gauss-Newton and with adaptive stochastic gradient descent methods on both supervised and reinforcement learning tasks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Adaptive gradient descent methods have a long tradition in machine learning (Becker *et al.*, 1988; Duchi *et al.*, 2011). The basic principle of these methods is to leverage information from previous iterations of the stochastic optimization process, e.g., the last t estimates of the gradient, to improve the convergence rate. Adam (Kingma and Ba, 2014), a particularly popular choice for training deep neural networks (Goodfellow *et al.*, 2016), is an adaptive gradient descent method that has been shown to yield superior convergence performance in practice, while also having a number of other beneficial properties, e.g., intuitively interpretable hyper-parameters, the ability to cope with online and non-stationary settings, invariability to gradient scaling, and its applicability to problems with noisy and sparse gradients.

To date, adaptive methods such as Adam or Adagrad (Duchi *et al.*, 2011) mainly focus on estimating first-order gradients in Euclidean space. An appealing extension is to estimate the local curvature of the objective function. Examples of curvatures estimates include the Fisher matrix, $\boldsymbol{F}$, and the Gauss-Newton matrix, $\boldsymbol{G}$. The Fisher defines the transformation from the Euclidean gradient to the natural gradient (Amari, 1998), which is invariant to smooth and invertible reparameterization of a probabilistic model and is therefore less susceptible to issues regarding scaling of feature dimensions or parameterizations of a task. The Gauss-Newton is an approximation to the Hessian matrix and, while equivalent to the Fisher for probabilistic models, is directly applicable to a wider range of objective functions. In practice, second-order descent methods often outperform their first-order alternatives. Successful application domains include reinforcement learning (Kakade, 2002; Peters and Schaal, 2008; Schulman *et al.*, 2015;

Rajeswaran *et al.*, 2017), blind source separation (Zhang *et al.*, 1999), low-rank matrix factorization (Buchanan and Fitzgibbon, 2005), and supervised training of deep neural networks (Pascanu and Bengio, 2013).

In this paper, we investigate whether second-order gradient approaches benefit from adaptive schemes for the *curvature matrix*. First, we discuss a general view of adaptive curvature and discuss interesting theoretical and practical insights and challenges thereof. Then, we introduce a practical algorithmic framework for data- and computation-efficient adaptive curvature optimization method, called AdaCurv. AdaCurv combines the benefits of adaptive methods, such as the ability to handle sparse gradients, with the benefits of second-order methods, such as invariance to the model parameterization. AdaCurv is applicable to any optimization problem with a differentiable model. The main contributions of this paper are:

1. A unified, general view of adaptive curvature gradient descent algorithms and practical challenges.

2. A practical, efficient algorithmic framework from which a number of different adaptive natural gradient descent algorithms can be instantiated. The examined instances compare favorably with existing state-of-the-art second-order and adaptive approaches.

3. Approximate update rules for adaptive curvature gradient descent that are computationally efficient with cost equal to a standard, conjugate gradient–based, Hessian-free-style method.

4. A data-efficient approach to the estimation of curvature matrices that establishes novel theoretical and practical links to shrinkage of covariance matrices.

We lay the foundations for our discussion of adaptive curvature in Chapter 2, highlighting theoretical and practical insights. The main contributions of this thesis are in Chapter 3, which describes the AdaCurv framework. The convergence guarantees for a particular instantiation of our method are described in Chapter 3.6. Chapters 4.1 and 4.2 cover important computational issues related to Hessian-free style optimization and determining the shrinkage factor. Chapter 5 provides experimental results on a number of machine learning tasks. Finally, we discuss prior work in Chapter 6.

Chapter 2

A GENERAL VIEW OF ADAPTIVE CURVATURE

To derive a general version of adaptive curvature gradient descent, we start with Adam as a representative sample of adaptive methods and introduce abstractions that allow for a variety of alternative instantiations. Adam maintains an exponential moving average of the mean and variance of the gradient estimate and adjusts the step size per parameter based on the variance. An idealized adaptive curvature gradient algorithm based on the Adam methodology may result in an update step of the form:

$$\begin{aligned}
\boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\hat{\boldsymbol{g}}_t \\
\boldsymbol{B}_t(\boldsymbol{\theta}_t) &\approx \nabla^2_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_t) \\
\boldsymbol{C}_t &= \beta_2 \boldsymbol{C}_{t-1} + (1-\beta_2)\boldsymbol{B}_t(\theta_t) \\
\boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \eta \boldsymbol{C}_t^{-1}\boldsymbol{m}_t
\end{aligned}$$

where β_1 and β_2 are scalars in $[0,1)$, $\boldsymbol{g}_t \in \mathbb{R}^p$ is the gradient, $\boldsymbol{B}_t \in \mathbb{R}^{p\times p}$ is an approximation of the local curvature of the objective, such as the Hessian, Gauss-Newton, or Fisher, and $\boldsymbol{\theta}_t \in \mathbb{R}^p$ is the parameter vector. The vector $\boldsymbol{m}_t$ and matrix $\boldsymbol{C}_t$ represent the averaged estimates of the gradient and curvature matrix. When clear we omit the explicit dependence of $\boldsymbol{B}_t$ on $\boldsymbol{\theta}_t$. A concrete instantiation based on the Fisher matrix would define the curvature matrix to be $\boldsymbol{B}_t(\boldsymbol{\theta}_t) = \mathbb{E}[\nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta}_t)(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta}))^\top]$, while an instantiation based on the Gauss-Newton matrix would employ, $\boldsymbol{B}_t(\boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)^\top \boldsymbol{H}_e \nabla_{\boldsymbol{\theta}_t} f(\boldsymbol{\theta}_t)$, where $\boldsymbol{H}_e$ is the Hessian of the error function.

Reddi *et al.* (2018) observe that the core component that distinguishes adaptive gradient descent algorithms is the *averaging function*. Changing the averaging function from an exponential moving average to a different update scheme yields algorithms that are similar to other previously reported adaptive gradient descent methods, e.g., Adagrad by Duchi *et al.* (2011). To extend the definition in Reddi *et al.* (2018) towards a more general framework for adaptive curvature gradients, we introduce two abstract functions ϕ_t and ψ_t, which represent the averaging functions for the gradient mean and curvature matrix, respectively. These functions take the history of gradients and curvature matrices up to time t and return a composite estimate, e.g., the average, of the gradient and curvature matrix through the current time step.

Our general algorithm (Algorithm 1) takes the gradient $\boldsymbol{g}_t$ and the curvature matrix $\boldsymbol{B}_t$ at each time step, generates composite estimates of the gradient $\boldsymbol{m}_t$ and the curvature matrix $\boldsymbol{C}_t$ by applying the averaging functions ϕ and ψ, and produces a descent direction from the calculated statistics. While a variety of different averaging functions can be found in the literature, all of which can be considered as a choice for ϕ and ψ, we will focus on three variants. The subsequent definitions describe modified, "curvature" versions of ϕ and ψ. The naming follows the original algorithms in which these averaging functions have been introduced.

a.) Adagrad (Duchi *et al.*, 2011) The averaging functions for a curvature Adagrad version can be expressed as:

$$\phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t) = \boldsymbol{g}_t,$$
$$\psi_t(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_t) = \frac{\sum_{i=1}^{t} \boldsymbol{B}_i}{t}.$$

b.) Adam (Kingma and Ba, 2014) The averaging functions for a curvature Adam version can be expressed as:

$$\phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t) = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} \boldsymbol{g}_i,$$
$$\psi_t(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_t) = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{B}_i.$$

Algorithm 1 A generalized adaptive curvature gradient descent algorithm. Parameters ϕ and ψ represent averaging functions.

```
Require: $\eta$: step size
Require: $\phi_t$: averaging function for gradient
Require: $\psi_t$: averaging function for curvature matrix
Require: $f(\boldsymbol{\theta})$: stochastic objective with parameters $\theta$
1: while not converged do
2:     $t \leftarrow t + 1$
3:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
4:     $\boldsymbol{m}_t \leftarrow \phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t)$
5:     $\boldsymbol{B}_t(\boldsymbol{\theta}_t) \approx \nabla^2_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_t)$
6:     $\boldsymbol{C}_t \leftarrow \psi_t(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_t)$
7:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \boldsymbol{C}_t^{-1} \boldsymbol{m}_t$
8: end while
9: return $\boldsymbol{\theta}_t$
```

This update can also be expressed as a recursion where $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_t)\boldsymbol{g}_t$ and $\boldsymbol{C}_t = \beta_2 \boldsymbol{C}_{t-1} + (1 - \beta_2)\boldsymbol{B}_t$.

c.) AMSGrad (Reddi *et al.*, 2018) The averaging functions for a curvature AMSGrad version are expressed most easily as a recursion since a max function is involved. In this case, $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_t)\boldsymbol{g}_t$ and with $\hat{\boldsymbol{C}}_t = \beta_2 \boldsymbol{C}_{t-1} + (1 - \beta_2)\boldsymbol{B}_t$,

$$\boldsymbol{C}_t = \begin{cases} \hat{\boldsymbol{C}}_t & \text{if } \rho(\hat{\boldsymbol{C}}_t) > \rho(\boldsymbol{C}_{t-1}) \\ \boldsymbol{C}_{t-1} & \text{otherwise.} \end{cases}$$

In AMSGrad the comparison between $\hat{\boldsymbol{C}}_t$ and $\boldsymbol{C}_{t-1}$ is based on the spectral norm of each matrix. Since the eigenvalues of the preconditioning matrix of the gradient control the magnitude of change in each dimension, this choice ensures that the effective learning rate is non-increasing.

While the averaging function over the gradient ϕ_t is easily computed, a straightforward attempt to implement Algorithm 1 leads to a number of practical challenges regarding data efficiency and computational costs while computing the averaging function over the curvature matrix ψ_t. Note, for example, that Algorithm 1 requires forming the matrix (and its inverse) explicitly, which may be intractable for some models, e.g., large neural networks. In such situations, the second-order approximations are often computed using conjugate gradient descent and Hessian-vector products (Martens, 2010; Schulman *et al.*, 2015; Rajeswaran *et al.*, 2017).

We employ this style of matrix-free optimization, which reduces the problem of estimating the descent direction to solving $\boldsymbol{C}_t\boldsymbol{v} = \boldsymbol{g}$ but makes computing the exponential moving average more difficult, since $\boldsymbol{C}_t\boldsymbol{v}$ is given by

$$\begin{aligned}\boldsymbol{C}_t\boldsymbol{v} &= (\beta\boldsymbol{C}_{t-1} + (1-\beta)\boldsymbol{B}_t)\boldsymbol{v}\\ &= \beta\boldsymbol{C}_{t-1}\boldsymbol{v} + (1-\beta)\boldsymbol{B}_t\boldsymbol{v}.\end{aligned}$$

The above approach leads to a number of challenges in practice. While the second term $(1-\beta)\boldsymbol{B}_t\boldsymbol{v}$ is easily computed using the current model, the first term $\beta\boldsymbol{C}_{t-1}\boldsymbol{v}$ depends recursively on all previous iterates and cannot directly be computed without access to all previous model instances. Addressing the aforementioned challenges is non-trivial and requires additional theoretical insights and computational tools. In Sec. 3, we will discuss a practical and efficient framework that eschews these challenges but is still able to incorporate different averaging functions.

In this chapter, we introduced a generalized view of adaptive curvature gradient descent and, extending the methodology in Reddi *et al.* (2018), discussed an algorithm that allows for different instantiations of the averaging functions. However, to be useful in practice, important challenges must be resolved, e.g., the recursive dependence on all previous model instances.

Chapter 3

A PRACTICAL FRAMEWORK FOR ADAPTIVE CURVATURE GRADIENT DESCENT

Building upon the preceding discussion, we will now discuss a practical algorithmic framework that resolves introduced challenges through a combination of theoretical and computational insights, while still maintaining the beneficial properties discussed so far. First, we discuss removing the dependence on previous iterations. In turn, we introduce a theoretical connection between curvature matrix estimation and shrinkage methods for covariance matrices. Finally, we provide computationally efficient formulations of the update equations.

3.1 Computing the Curvature-Vector Products

In the natural gradient case, the Fisher can also be interpreted as the expected Hessian of the log-likelihood,

$$\begin{aligned}\boldsymbol{B}_t(\boldsymbol{\theta}_t) &= \mathbb{E}[\nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta}_t)(\nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta}_t))^\top] \\ &= -\mathbb{E}[\boldsymbol{H}_{\log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta}_t)}].\end{aligned}$$

Thus, the Fisher-vector product can be computed at the cost of two backpropagations since $\frac{\partial^2 f}{\partial \boldsymbol{x}} \boldsymbol{v} = \frac{\partial}{\partial \boldsymbol{x}} (\frac{\partial f}{\partial \boldsymbol{x}} \boldsymbol{v})$.

Note, however, that $\nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta}_t) = \boldsymbol{J}^\top \nabla_{\boldsymbol{z}} \log r(\boldsymbol{y} \mid \boldsymbol{z})$ for an output distribution r with $\boldsymbol{z} = f(\boldsymbol{x})$, so the Fisher can be rewritten as, $\boldsymbol{F} = \mathbb{E}[\boldsymbol{J}^\top \boldsymbol{F}_r \boldsymbol{J}]$, where $\boldsymbol{F}_r$ is the Fisher of the distribution r (Martens, 2014; Park *et al.*, 2000). This shows that the Fisher is, in fact, equivalent to the Gauss-Newton for output distribution from the exponential family (Martens, 2014). Moreover, in this form, the Fisher of the output distribution and Hessian of the loss are generally tractable and can be computed directly.

Using these forms, one can obtain improved matrix-vector multiplication performance by combining the $\mathcal{R}$-operator and $\mathcal{L}$-operator (Pearlmutter, 1994), commonly known in the machine learning community as forward and reverse-mode autodifferentiation. The operator $\mathcal{R}_{\boldsymbol{v}}(f)$ computes the Jacobian of f multiplied on the *right* by the vector $\boldsymbol{v}$ while the operator $\mathcal{L}_{\boldsymbol{v}}(f)$ computes the Jacobian of f multiplied on the *left* by the vector $\boldsymbol{v}$. Thus one can compute $\boldsymbol{F}\boldsymbol{v}$ by computing $\boldsymbol{J}\boldsymbol{v} = \mathcal{R}_{\boldsymbol{v}}(f)$, directly multiplying by the, now tractable, Fisher or Hessian, $\boldsymbol{F}_r\boldsymbol{J}\boldsymbol{v}$, and finally computing $\boldsymbol{J}^\top\boldsymbol{F}_r\boldsymbol{J}\boldsymbol{v} = \mathcal{L}_{\boldsymbol{F}_r\boldsymbol{J}\boldsymbol{v}}(f)$. This gives an algorithm to compute the Fisher or Gauss-Newton-vector product with one forward and one backward pass through the model.

3.2 Removing Dependence on Previous Iterations

To remove the dependence on previous iterates, we propose incorporating an inner optimization that finds parameters, $\boldsymbol{\theta}^-$, such that $\boldsymbol{C}_t(\boldsymbol{\theta}^-)\boldsymbol{v} = (\beta\boldsymbol{C}_{t-1} + (1-\beta)\boldsymbol{B}_t)\boldsymbol{v}$. In general, this is a highly non-linear and ill-posed optimization problem. That is, there may be several parameter instances that represent the desired curvature. To simplify this optimization we note a few significant properties, assuming that the gradient changes smoothly along the parameter space:

1. Given parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ over a small domain, there exists $\boldsymbol{\theta}^-$ such that $\boldsymbol{C}_2(\boldsymbol{\theta}^-) = \beta\boldsymbol{C}_1(\boldsymbol{\theta}_1) + (1-\beta)\boldsymbol{B}_2(\boldsymbol{\theta}_2)$, where $\boldsymbol{\theta}^- = \alpha\boldsymbol{\theta}_1 + (1-\alpha)\boldsymbol{\theta}_2$ with $\alpha \in [0,1]$. This reduces the problem to a line search between $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$.

2. To avoid computing the curvature matrix directly, we continue to operate in the space of matrix-vector products and solve $\boldsymbol{C}_2(\boldsymbol{\theta}^-)\boldsymbol{v} = (\beta\boldsymbol{C}_1(\boldsymbol{\theta}_1)+(1-\beta)\boldsymbol{B}_2(\boldsymbol{\theta}_2))\boldsymbol{v}$,

with α as defined above and $\boldsymbol{v}$ equal to the estimate descent direction at $\boldsymbol{\theta}_2$. Assuming that the standard gradient $\boldsymbol{g}$ is unique and smooth between $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ (a benign assumption over a small domain), then the matrix $\boldsymbol{C}(\boldsymbol{\theta}^-)$ is unique in this domain and, accordingly, the projection $\boldsymbol{C}(\boldsymbol{\theta}^-)\boldsymbol{v}$ is also unique.

3. Over long distances in parameter space, this optimization need not be convex, though we assume the optimization is locally convex and that this holds in the limit of small step sizes. This same step size constraint arises in the theory of NGD (Amari, 1998).

Our practical implementation of this algorithm incorporates a line search (LS) to find $\boldsymbol{\theta}^-$ (Wright and Nocedal, 1999). The resulting adaptive curvature algorithm is both compute and memory efficient since the gradient is computed without constructing or inverting the curvature matrix. Also, performing the line search before the CG optimization minimizes the number of matrix-vector products. In this form, the line search requires $k+2$ matrix-vector products and the CG optimization will require only j, where k and j are the number of iterations in the line search and CG, respectively. A standard CG-based, Hessian-free optimization algorithm requires j matrix-vector products. The computation necessary to estimate the gradient direction is on the order of $O(j+k)$ for the adaptive curvature gradient with a line search and $O(j)$ for a standard Hessian-free method.

Expressions for each of the three studied averaging functions can be found in Table 3.1. Now, ψ still represents the averaging function over the curvature matrix though, for computational reasons, it is now defined by $\boldsymbol{\theta}_{t-1}^-$ and $\boldsymbol{\theta}_{t-1}$. These averaging functions use a line search to compute $\boldsymbol{\theta}_t^-$.

3.3 Damping and Shrinking the Curvature Matrix

Although the Fisher matrix can be computed as the Hessian of the log-likelihood with respect to the model parameters, the Fisher can also be interpreted as the covariance of the gradient of the log-likelihood. Likewise, because the Gauss-Newton matrix is guaranteed to be positive semi-definite, it too can be interpreted as a covariance matrix. Ensuring positivity is a valuable property of a curvature matrix and is one reason why the Fisher and Gauss-Newton are often preferred over the Hessian. Note, that as long as the curvature matrix is guaranteed to be positive semi-definite that it can be interpreted as a covariance matrix.

The interpretation as a covariance allows one to apply techniques commonly applied to covariance estimation. One such technique, which has become popular in finance where one often deals with many more dimensions than data samples, is shrinkage (Ledoit and Wolf, 2004). Shrunk covariance estimates find the optimal interpolation between the sample and diagonal covariance. The optimal shrinkage depends on the number of samples available and potentially some assumed properties of the data (e.g. Gaussianity) (Chen *et al.*, 2010). It has been shown that, with small data, the shrunk covariance is a more accurate estimate of the true covariance than the sample covariance itself. Moreover, the shrunk covariance is often non-singular; some shrinkage estimators even guarantee this property (Ledoit and Wolf, 2004).

Following the work by Chen *et al.* (2010), we compute a shrunk curvature matrix as follows:

$$\begin{aligned}
\boldsymbol{D}_t &= \frac{\mathrm{Tr}(\boldsymbol{B}_t)}{p}\boldsymbol{I} \\
\rho &= \min\left(\frac{(1-2/p)\,\mathrm{Tr}(\boldsymbol{B}_t^2) + \mathrm{Tr}^2(\boldsymbol{B}_t)}{(n+1-2/p)[\mathrm{Tr}(\boldsymbol{B}_t^2) + \mathrm{Tr}^2(\boldsymbol{B}_t)/p]}, 1\right) \\
\boldsymbol{B}_{t,\text{shrunk}} &= (1-\rho)\boldsymbol{B}_t + \rho\boldsymbol{D}_t
\end{aligned}$$

where $\boldsymbol{D}_t$ is a diagonal covariance estimate, ρ is the shrinkage factor, p is the number of parameters, and n is the number of samples.

A careful look at the quantities needed to compute the shrinkage factor, ρ, reveals that both $\text{Tr}^2(\boldsymbol{B}_t)$ and $\text{Tr}(\boldsymbol{B}_t)^2$ are required. Methods exist to estimate the diagonal of the Hessian in large models (Martens *et al.*, 2012) but we are not aware of any method to approximate the diagonal of the square of the Hessian. To approximate these quantities without forming the curvature matrix explicitly, we note an eigenvalue relationship between $\text{Tr}^2(\boldsymbol{A})$ and $\text{Tr}(\boldsymbol{A})^2$. For a symmetric matrix $\boldsymbol{A}$, $\text{Tr}(\boldsymbol{A}^2) = \sum_i^p \sum_j^p \boldsymbol{A}_{ij}^2 = \sum_i^p \lambda_i^2$. This reduces the problem of computing the shrinkage to an eigenvalue problem.

To compute eigenvalues, we propose to use Lanczos' method (Lanczos, 1950). This method permits computing eigenvalues of $\boldsymbol{B}_t$ while only requiring the matrix-vector product of $\boldsymbol{B}_t$ with some vector $\boldsymbol{v}$. Given k eigenvalues of $\boldsymbol{B}_t$, where often $k \ll p$, $\boldsymbol{D}_t$ is approximated using $\text{Tr}(\boldsymbol{B}_t) = \sum_{i=1}^k \lambda_i$. Once the shrinkage factor, ρ, has been computed, it can be incorporated as a damping factor during the conjugate gradient iteration in a straightforward manner since the Fisher vector product becomes $(\boldsymbol{B}_t + \epsilon \boldsymbol{I})\boldsymbol{v} = \boldsymbol{G}_t \boldsymbol{v} + \epsilon \boldsymbol{v}$ where ϵ is the damping factor.

Moreover, this approach to damping can be combined with adaptive curvature gradient estimators. Instead of computing the shrinkage factor ρ based on the current curvature $\boldsymbol{B}_t$, we instead compute the shrinkage for $\beta_2 \boldsymbol{C}_{t-1} + (1-\beta_2)\boldsymbol{B}_t$. This generates the optimal shrinkage for the adaptive curvature estimate.

The outlined shrinkage approach is reminiscent of Tikhonov regularization, which generally adds a small multiple of the identity to ensure non-degeneracy and is commonly used by existing gradient-based methods that employ CG optimization. In previous work, the damping factor is typically either heuristically determined (Martens, 2010) or set at a constant value (Schulman *et al.*, 2015; Rajeswaran *et al.*, 2017).

Martens (2010) notes that using a constant damping is not desirable as it limits the algorithm's ability to "jump" over plateaus. In contrast, our method can be viewed as an automatic means to estimate the damping factor based on the data available without resorting to heuristics or constants.

Variant	Optimal adaptive update rule	Approximate adaptive update rule
AdaCurv-Adagrad	$\psi_t^*(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1}) =$ $\texttt{LS}\Big[\min_{\boldsymbol{\theta}} \lVert(\boldsymbol{C}(\boldsymbol{\theta}) - \Big((t-1)\boldsymbol{C}(\boldsymbol{\theta}_{t-1}^-) + \boldsymbol{B}(\boldsymbol{\theta}_{t-1})\Big)/t)\boldsymbol{\nu}_{t-1}\rVert\Big]$	$\hat{\psi_t}(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1}) =$ $\Big((t-1)\boldsymbol{\theta}_{t-1}^- + \boldsymbol{\theta}_{t-1}\Big)/t$
AdaCurv-Adam	$\psi_t^*(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1}) =$ $\texttt{LS}\Big[\min_{\boldsymbol{\theta}} \lVert(\boldsymbol{C}(\boldsymbol{\theta}) - \beta_2\boldsymbol{C}(\boldsymbol{\theta}_{t-1}^-) - (1-\beta_2)\boldsymbol{B}(\boldsymbol{\theta}_{t-1}))\boldsymbol{\nu}_{t-1}\rVert\Big]$	$\hat{\psi_t}(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1}) =$ $\beta_2\boldsymbol{\theta}_{t-1}^- + (1-\beta_2)\boldsymbol{\theta}_{t-1}$
AdaCurv-AMSGrad	$\psi_t^*(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1}) =$ $\texttt{LS}\Big[\min_{\boldsymbol{\theta}} \lVert(\boldsymbol{C}(\boldsymbol{\theta}) - \beta_2\boldsymbol{C}(\boldsymbol{\theta}_{t-1}^-) - (1-\beta_2)\boldsymbol{B}(\boldsymbol{\theta}_{t-1}))\boldsymbol{\nu}_{t-1}\rVert\Big]$ if $\rho(\boldsymbol{C}_t) > \rho(\boldsymbol{C}_{t-1})$	$\hat{\psi_t}(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1}) =$ $\beta_2\boldsymbol{\theta}_{t-1}^- + (1-\beta_2)\boldsymbol{\theta}_{t-1}$ if $\rho(\boldsymbol{C}_t) > \rho(\boldsymbol{C}_{t-1})$

Table 3.1: Optimal and Approximate Adaptive Update Expressions for $\boldsymbol{\theta}^-$. The Optimal Updates Perform a Line Search over $\boldsymbol{\theta}$ With the Constraint That $\boldsymbol{\theta} = \gamma\boldsymbol{\theta}_{t-1}^- + (1-\gamma)\boldsymbol{\theta}_{t-1}$ and $\gamma \in [0, 1]$. The Approximate Update Rule Assumes the Curvature is Locally Linear Around the Current Parameters. The Comparison Needed for AMSGrad is Computed by Estimating the Largest Eigenvalues of $\boldsymbol{C}_t$ and $\boldsymbol{C}_{t-1}$ Using Lanczos' Method and Matrix-Vector Products.

3.4 An Approximate, No-Overhead Adaptive Update Step

One can reduce the computational complexity of our approach further by assuming that the curvature matrix changes linearly with respect to the parameters in the region between $\boldsymbol{\theta}^-$ and $\boldsymbol{\theta}$. Empirically, we observe that for a given β_2 the optimal lagged parameter tends to be approximately $\beta_2\boldsymbol{\theta}^- + (1-\beta_2)\boldsymbol{\theta}$ (this implies that the change in the curvature is locally linear with respect to $\boldsymbol{\theta}$). Making this assumption, we propose approximate, adaptive curvature gradient update rules with the same complexity as existing Hessian-free approaches. That is, instead of optimizing the value of $\boldsymbol{\theta}^-$ at each iteration we update $\boldsymbol{\theta}^-$ assuming local linearity as shown in Table 3.1. Using these approximate updates yields good empirical performance at no additional cost relative to a standard Hessian-free method.

Algorithm 2 AdaCurv-{*algorithm*}*: a data- and computation-efficient template of the general adaptive curvature gradient descent framework.

```
Require: $\eta$: step size
Require: $\phi_t$: averaging function for gradient
Require: $\psi_t$: averaging function for curvature matrix
Require: $\boldsymbol{\theta}_0$: initial parameter vector
Require: $\boldsymbol{\theta}_0^-$: initial lagged parameter vector
Require: $f(\boldsymbol{\theta})$: stochastic objective with parameters $\theta$
 1: $\boldsymbol{m}_0 \leftarrow 0$
 2: $t \leftarrow 0$
 3: while not converged do
 4:     $t \leftarrow t + 1$
 5:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
 6:     $\boldsymbol{m}_t \leftarrow \phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t)$
 7:     $\boldsymbol{\theta}_t^- \leftarrow \psi_t(\boldsymbol{\theta}_{t-1}^-, \boldsymbol{\theta}_{t-1})$
 8:     $\boldsymbol{D}_t, \rho \leftarrow \texttt{SHRINKAGE}\left(\boldsymbol{C}(\boldsymbol{\theta}_t^-)\right)$          (optionally, else $\rho = $ constant)
 9:     $\boldsymbol{\nu}_t \leftarrow \texttt{CG}\left(((1-\rho)\boldsymbol{C}(\boldsymbol{\theta}_t^-) + \rho\boldsymbol{D}_t)\boldsymbol{\nu} = \boldsymbol{m}_t\right)$
10:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta\boldsymbol{\nu}_t$
11: end while
12: return $\boldsymbol{\theta}_t$
```

3.5 AdaCurv: Adaptive Curvature Gradient Descent

Taking all the above insights together results in a general yet practical framework for adaptive curvature gradient descent, which can be found in Algorithm 2. The algorithm, which we will henceforth refer to as Adaptive Curvature gradient descent (AdaCurv), represents a family of methods and is parameterized by the averaging functions ϕ_t

and ψ_t. Rather than a specific algorithm it can be regarded as a template from which different instances can be derived. The mechanics of many existing algorithms for adaptive gradient descent can be incorporated into AdaCurv by specifying appropriate functions for ϕ_t and ψ_t. As previously introduced, Table 3.1 describes both optimal and approximate adaptive rules for the popular choices of ϕ_t and ψ_t. In the remainder of the paper, we will refer to instantiations of AdaCurv as AdaCurv-$\{$*algorithm*$\}$, meaning that we perform the adaptive curvature gradient update using the analogous averaging functions from the designated algorithm. We use superscripts to denote variations of AdaCurv: * = optimal adaptive update, $\hat{}$ = approximate adaptive update, and $^{(s)}$ = shrinkage.

3.6 Convergence of AdaCurv

We study the convergence of a particular instantiation of AdaCurv in the case of a strongly convex function. Specifically, we consider averaging functions $\phi(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t) = \boldsymbol{g}_t$ and $\psi(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_t) = \frac{1-\beta}{1-\beta^t} \sum_{i=1}^{t} \beta^{t-i} \boldsymbol{H}_i$, where $\boldsymbol{g}_i = \nabla f(\boldsymbol{x}_i)$ and $\boldsymbol{H}_i = \nabla^2 f(\boldsymbol{x}_i)$. Intuitively, this is similar to Newton's method with bias-corrected Adam-style averaging only over the Hessian, but not the gradient, which we will denote AdaCurv-Newton. Our analysis, which mirrors that of Boyd and Vandenberghe (2004, Chap. 9.5), shows that AdaCurv-Newton enjoys the same convergence guarantees as Newton's method. Analogous to classical Newton's method, AdaCurv-Newton converges in two phases, traditionally referred to as the *damped* and *true* Newton phases. The true Newton phase exhibits extremely fast, quadratic convergence. For a function continuous, twice-differentiable function f with $\nabla^2 f(\boldsymbol{x}) \succcurlyeq m\boldsymbol{I}$, $\nabla^2 f(\boldsymbol{x}) \preccurlyeq M\boldsymbol{I}$, and $\|\nabla^2 f(\boldsymbol{x}) - \nabla^2 f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2$ (Lipschitz constant L), AdaCurv-Newton converges to an *extremely good* [1] solution in at most

$$6 + \frac{M^2 L^2 / m^5}{\alpha\beta \min\{1, 9(1-2\alpha)^2\}} \left(f(\boldsymbol{x}_0) - \boldsymbol{x}^*\right)$$

iterations, where α and β are parameters of the backtracking line search. A detailed analysis is in Appendix A. Despite the fact that the convergence theory is the same for both AdaCurv-Newton and classical Newton's method, we find that adaptive curvature provides substantial benefits in practice as we show in the experiments (Chapter 5).

[1] Convergence is to an *extremely good* solution because the the quadratically convergent phase is approximated by 6 iterations, after which the error is approximately $5 * 10^{-20} e_0$ (Boyd and Vandenberghe, 2004, Chap. 9.5.3).

Chapter 4

COMPUTATIONAL DETAILS

Approximate second-order optimization methods tend to be computationally intensive. Accordingly, we specify in detail here how one can efficiently compute the necessary approximations. We focus on computing the descent direction as the solution to $\boldsymbol{B}\boldsymbol{v} = \boldsymbol{g}$ and computing the shrinkage factor for $\boldsymbol{B}$.

4.1 Efficiently Solving the Linear System

Newton-style methods require solving $\boldsymbol{C}\boldsymbol{v} = \boldsymbol{g}$ at every iteration. This inner optimization is generally the most compute intensive portion of the optimizer so it is important to perform this step efficiently. We discuss three different techniques that improve the computational efficiency of this inner optimization. In practice, we find that these optimizations provide excellent performance and that, especially later in training when nearing a local optimum, the CG solver sometimes terminates in less than 10 iterations.

4.1.1 Truncated CG

One common approach to reduce the computational needs of the CG solver is the truncate the number of iterations and terminate before the minimum is reached. This permits finding an approximate descent direction and is well-justified theoretically because CG converges rapidly over the initial iterations (Shewchuk *et al.*, 1994, Sec. 9). We use this technique in all tested AdaCurv variants. Moreover, by making the assumption that final direction computed by CG in the previous iteration is close to optimal for the current iteration we can initialize the CG solver with the previous solution. This results in a form of "momentum" that improves convergence of the truncated CG. We find that 10 CG iterations and a CG initialization of $\boldsymbol{x}_0 = 0.5\boldsymbol{v}_{t-1}$ perform well.

4.1.2 Preconditioned CG

The use of a preconditioner that is easily computed and inverted can be used to improve the conditioning of the curvature matrix and accelerate the convergence of CG by transforming the quadratic to be approximately spherical. Following the work by (Martens, 2010) we use a regularized version of the diagonal of *empirical* Fisher as a preconditioner. For the non-adaptive case this results in a diagonal matrix, $\boldsymbol{V}_t = \mathrm{diag}(\boldsymbol{g}_t \odot \boldsymbol{g}_t + \lambda \boldsymbol{I})^d$, where λ is a regularization parameter and d is an exponent less than one used to dampen the effect of extreme values. In the adaptive case we form an adaptive preconditioner based on the same averaging functions used in the core algorithm. For example, if performing adaptive curvature based on the Adam averaging functions, the preconditioner is defined to be,

$$\hat{\boldsymbol{M}}_t = \beta_2 \boldsymbol{M}_{t-1} + (1 - \beta_2)\boldsymbol{V}_t$$
$$\boldsymbol{V}_t = \hat{\boldsymbol{M}}_t / (1 - \beta_2^t).$$

Because this matrix is diagonal it is easy to invert an apply during the CG iteration. Since the *empirical* Fisher may not accurately reflect the local curvature (Martens, 2014) this preconditioner likely does not result in a well-conditioned curvature matrix for many cases but, as is often the case, is a balance between computation and performance. More accurate, and compute intensive, preconditioners have been proposed (Chapelle and Erhan, 2011) but we do not investigate any additional preconditioners in this work.

4.1.3 Block-Diagonal CG

There exists empirical evidence that the true curvature of a neural network is approximately block-diagonal by layer or block-tridiagonal spanning adjacent layers (Martens and Grosse, 2015). Previous work proposed solving $\boldsymbol{C}_l \boldsymbol{v}_l = \boldsymbol{g}_l$ by layer where $\boldsymbol{C}_l$ and $\boldsymbol{g}_l$ represent the curvature matrix and gradient corresponding to the weights of layer l (Zhang *et al.*, 2017). This set of systems of equations can be solved in parallel.

We investigate this approach and find that it does not significantly alter final test performance but were unable to achieve a notable computational improvement. It is not clear, without substantial parallel compute resources, how such an approach can reduce computation time since solving $\boldsymbol{C}_1 \boldsymbol{v}_1 = \boldsymbol{g}_1$ still requires one to propagate forward and backward through all the subsequent layers, an inherently sequential process.

4.2 Efficiently Computing the Shrinkage

Like CG, computing the shrinkage factor by Lanczos is a potential computation bottleneck. We propose two approximations that make computing this component more efficient, both of which assume that the damping factor will not change significantly over the course of a few iterations:

1. Amortize the computational cost of Lanczos shrinkage over k iterations.

2. Compute a lagged shrinkage based on the previous iteration by extracting eigenvalue estimates from CG solver.

The first approximation computes the shrinkage factor based on Lanczos' method only every k steps of the optimization. This keeps the cost of computing eigenvalues low relative to the cost of computing the gradient and solving $\boldsymbol{C}\boldsymbol{v} = \boldsymbol{g}$ but assumes the shrinkage factor will remain approximately constant over k optimization steps. In practice we find this assumption to hold extremely well, however, if this is not expected to be the case for a particular task, a heuristic update could be used as in Martens (2010).

The second approximation computes the shrinkage based on the eigenvalues from the previous timestep. These eigenvalues can be computed by recovering the tridiagnonal Lanczos matrix from the iterations of conjugate gradient (Saad, 2003, chap. 6.7.3),

$$T = \begin{bmatrix} \frac{1}{\alpha_0} & \frac{\sqrt{\beta_0}}{\alpha_0} & & & \\ \frac{\sqrt{\beta_0}}{\alpha_0} & \frac{1}{\alpha_1} + \frac{\beta_0}{\alpha_0} & \frac{\sqrt{\beta_1}}{\alpha_1} & & \\ & \frac{\sqrt{\beta_1}}{\alpha_1} & \cdot & \cdot & \\ & & \cdot & \cdot & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} \\ & & & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} & \frac{1}{\alpha_{m-1}} + \frac{\beta_{m-2}}{\alpha_{m-2}} \end{bmatrix}$$

where $\alpha_i = \frac{\boldsymbol{r}_i^\top \boldsymbol{r}_i}{\boldsymbol{d}_i^\top \boldsymbol{F} \boldsymbol{d}_i}$ and $\beta_i = \frac{\boldsymbol{r}_{i+1}^\top \boldsymbol{r}_{i+1}}{\boldsymbol{r}_i^\top \boldsymbol{r}_i}$ for residual, r_i, and descent direction, d_i. This method has negligible overhead as it it does not add any additional matrix-vector products and computing the eigenvalues of a small tridiagonal matrix is fast, but the shrinkage factor will lag behind the matrix for a single optimization step and has the additional constraint that the number of eigenvalues estimated must equal the number of CG iterations. We observe no reduction performance using this versus computing the shrinkage by Lanczos at every iteration. Note that when using a preconditioner, $\boldsymbol{M}$, this method computes the eigenvalues of $\boldsymbol{M}^{-1}\boldsymbol{F}$ instead of $\boldsymbol{F}$. This is acceptable, however, because in this case one cares about the variance of $\boldsymbol{M}^{-1}\boldsymbol{F}$ rather than $\boldsymbol{F}$.

Chapter 5

EXPERIMENTS

We now investigate the performance of the introduced AdaCurv framework and specific instances thereof, when compared to existing adaptive and non-adaptive methods. [1]

5.1 Application to Noisy, Ill-Conditioned, and Sparse Problems

To begin, we present an intuitive perspective on our algorithm on an informative but simple problem to optimizer a stochastic quadratic. We adapt this experiment from Balles and Hennig (2018).

We define the loss, $\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{x}) = 0.5(\boldsymbol{\theta} - \boldsymbol{x})^\top \boldsymbol{Q}(\boldsymbol{\theta} - \boldsymbol{x})$ for a symmetric, positive semi-definite matrix $\boldsymbol{Q}$. Data is sampled from the distribution $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}^*, \nu^2 \boldsymbol{I})$. We analyze problems of dimension 100 varying the noise, the conditioning of the quadratic and its rotation, along with the sparsity of the gradients. A random quadratic with the desired conditioning is generated by first generating a diagonal matrix Λ with the desired eigenvalues and then transforming it with a rotation matrix uniformly sampled from the set of all rotation matrices (Diaconis and Shahshahani, 1987). The gradient sparsity parameter signifies that each dimension of the gradient g_i is set to zero with the given probability. At every time step 10 samples are drawn from the, possibly noisy, data distribution and the optimization is run for 100 steps.

We observe (Figure 5.1) that the adaptive variant outperforms the classical formulation when the gradients are noisy and sparse. Rotation does not appear to have a significant effect on convergence. When there is no noise and the problem is

[1] Code available: `https://github.com/tpbarron/adacurv`

well-conditioned, the non-adaptive version performs well, which is arguably a case that does not exist in practice.

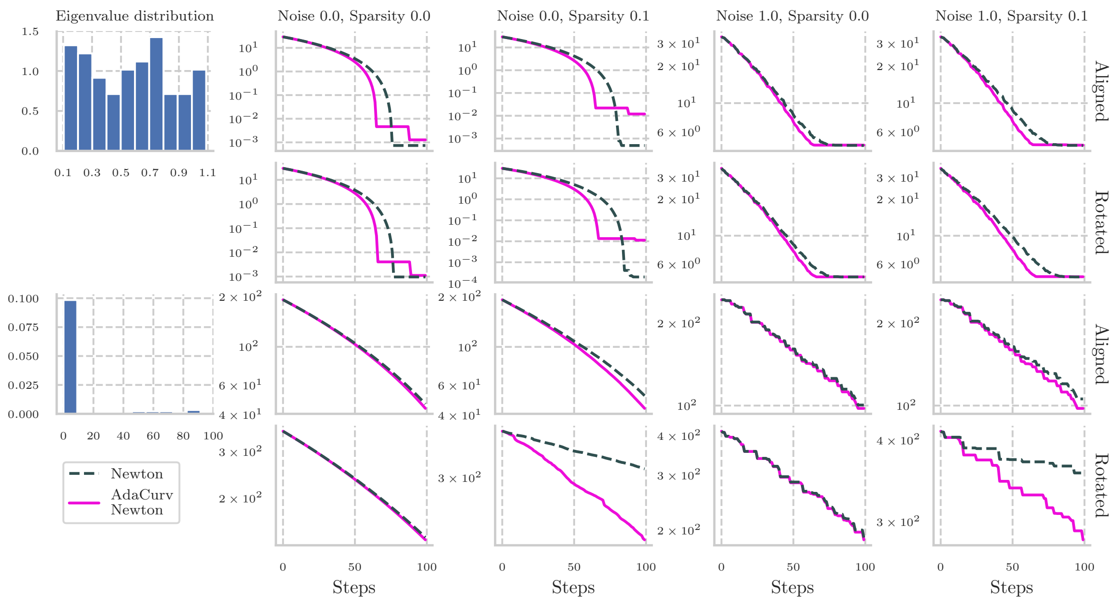
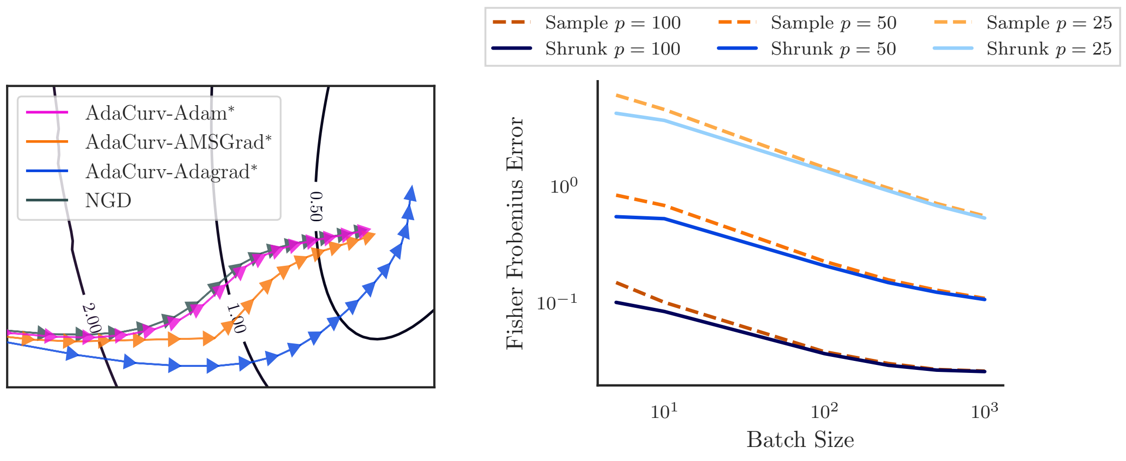


Figure 5.1: Optimization of a Noisy and Sparse Quadratic. AdaCurv Outperforms Classical Newton's Method When the Data is Noisy and the Gradient is Sparse.

5.2 Logistic Regression and Shrinkage Analysis

We fit a 2-dimensional logistic regression model on a randomly generated dataset with 500 samples and two classes and observed the convergence of various algorithms over the first 25 iterations. Figure 5.2a shows the behavior of AdaCurv variants and NGD over the contour plot of the objective function. Within the given limited budget of iterations, AdaCurv variants make faster progress toward the local minimum even on a very low dimensional task.

Figure 5.2b confirms empirically that shrinkage improves the accuracy of the Fisher matrix, which for this problem is equivalent to the Gauss-Newton matrix. Note that p indicates the dimensionality of the task. When the batch size is small, the shrinkage estimator significantly improves the accuracy relative to the sample Fisher. As the



(a) (b)

Figure 5.2: (a): Convergence Behavior for Each Variant on a Logistic Regression Problem over the First 25 Iterations. (b): Error from the True Fisher According to the Spectral Norm for Various Numbers of Parameters, p, and Batch Sizes, with and Without Shrinkage.

batch size increases both estimators improve in accuracy, though even at a batch size of 1,000, shrinkage improves the accuracy of the Fisher.

5.3 Supervised Learning on MNIST and SVHN

MNIST	AdaCurv-Adam*	AdaCurv-AMSGrad*	AdaCurv-Adagrad*	NGD	Adam	AMSGrad	Adagrad	SGD
125	97.41±0.06	97.79±0.09	**97.95±0.02**	96.98±0.13	97.83±0.04	97.79±0.03	91.81±0.12	93.12±0.05
250	97.65±0.1	**98.1±0.03**	98.01±0.07	97.12±0.11	97.55±0.13	97.56±0.14	91.53±0.06	91.61±0.11
500	97.7±0.04	**98.12±0.05**	97.99±0.07	97.21±0.07	96.93±0.15	96.9±0.16	91.14±0.08	89.97±0.19
1000	97.6±0.06	**97.94±0.04**	97.89±0.06	97.33±0.07	95.87±0.11	95.85±0.1	90.36±0.15	87.67±0.51

Table 5.1: Comparison of AdaCurv Variants with NGD and Adaptive SGD Methods on the MNIST Dataset Averaged over 5 Seeds.

We compared several adaptive SGD methods, both standard natural gradient descent, and AdaCurv variants applied to the Fisher matrix and examined the effect of shrinkage and approximate updates. The following experiments were performed on the MNIST dataset using a two-layer MLP (784-100-10). The learning rate was

MNIST	AdaCurv-Adam*	AdaCurv-AMSGrad*	AdaCurv-Adagrad*	AdaCurv-Adam$^{*(s)}$	AdaCurv-AMSGrad$^{*(s)}$	AdaCurv-Adagrad$^{*(s)}$	AdaCurv-Adamˆ	AdaCurv-AMSGradˆ	AdaCurv-Adagradˆ
125	97.45	97.88	98.02	97.49	**98.04**	98.01	97.51	97.74	97.27
250	97.61	98.07	98.01	97.67	**98.13**	98.09	97.68	98.1	97.65
500	97.71	**98.1**	97.91	97.7	98.03	97.95	97.68	98.07	97.81
1000	97.63	**98.06**	97.92	-	-	-	97.66	98.0	97.63

Table 5.2: Comparison of Optimal and Approximate Updates and the Effect of Shrinkage. The Best Value from Each Variant is Reported.

SVHN	AdaCurv-Adam*	AdaCurv-AMSGrad*	AdaCurv-Adagrad*	NGD	Adam	AMSGrad	Adagrad	SGD
125	**83.68**	83.55	82.91	79.65	83.27	83.34	66.65	68.92
250	**82.61**	82.15	82.13	80.33	82.86	82.86	65.98	62.62
500	82.53	82.67	**82.86**	80.02	82.01	82.1	64.78	49.9
1000	**82.88**	82.61	80.92	79.0	80.0	80.01	63.41	34.53

Table 5.3: Comparison of AdaCurv Variants with NGD and Adaptive SGD Methods on the SVHN Dataset.

held constant for all variants at 0.001 with the exception of batch sizes 125 and 250 of AdaCurv-Adam and AdaCurv-AMSGrad for which it was set at 0.0001 and 0.0005, respectively. The training was performed for 10 epochs and the data is accumulated over 5 random seeds. Additional hyperparameters are noted in the supplemental material. Figure 5.3 shows how performance varies with the batch size. We find that the AdaCurv variants converge at a faster rate and are remarkably consistent across batch sizes, while adaptive SGD algorithms tend to degrade in performance as the batch size increases and standard NGD performs poorly with small batch sizes. All AdaCurv variants outperform NGD at every batch size, and the best AdaCurv variant outperforms all adaptive SGD methods at every batch size. Table 5.1 summarizes each method's performance with the mean and standard deviation over 5 seeds.

Table 5.2 shows how the performance of our method varies with shrinkage and the approximate update step. We find that the shrinkage estimate improves performance,

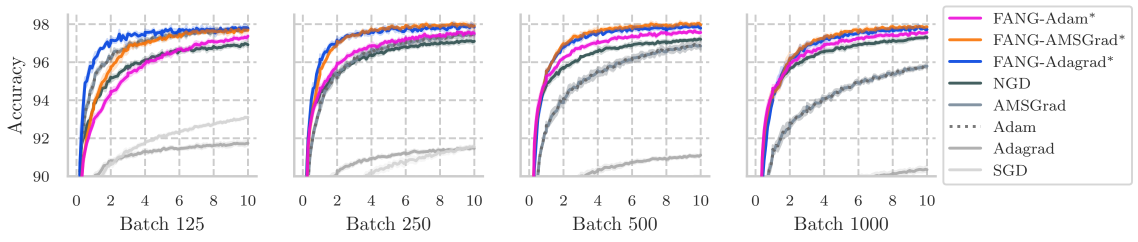


Figure 5.3: Performance on MNIST Across Different Batch Sizes. Note That the Performance of Adam Degrades as the Batch Size Increases but the Performance of AdaCurv Variants are Within 0.1% Accuracy Across All Batch Sizes.

particularly when the batch size is small. For batch sizes 125 and 250, AdaCurv-AMSGrad$^{*(s)}$ outperforms all other tested optimizers. We observe that using the approximate AdaCurv update (Table 3.1) results in a slight performance decrease at the benefit of reduced computational cost. The performance hit is most significant for AdaCurv-Adagrad at approximately 0.75%. Approximate versions of AdaCurv-Adam and AdaCurv-AMSGrad perform very closely to the optimal versions. This aligns with our intuition regarding when the Fisher matrix will change linearly. In AdaCurv-Adam and AdaCurv-AMSGrad the lagged parameters trail the true parameters, while AdaCurv-Adagrad averages all previous parameters. Thus, the space AdaCurv-Adagrad averages over is likely *not* linear, while the linearity assumption is more likely to hold for the space averaged by AdaCurv-Adam and AdaCurv-AMSGrad.

We perform a similar experiment using the SVHN dataset (Netzer *et al.*, 2011). In this experiment we classify grayscale images with a feed-forward neural network with structure MLP(1024-265-10). In this task, we again find that AdaCurv variants applied to the Fisher matrix outperform both NGD and adaptive SGD methods (see Table 5.3). It is possible to achieve higher performance on this task with a convolutional model but the relative performance with an MLP should be representative.

Finally, we have performed additional experiments comparing K-FAC on the Fashion MNIST dataset using a 4-layer, convolutional model and found that, whereas

K-FAC initially converged slightly faster (still, both methods reach 90% test accuracy in the first 2-3 epochs), AdaCurv-Adamˆ reached a slightly better test accuracy over 10 epochs (AdaCurv: 91.0%, K-FAC: 90.95%; not a statistically significant result given the limited number of seeds, one in this case). The K-FAC optimizer used default hyperparameters from the `tensorflow-kfac` implementation, and both methods used a learning rate of 0.001 and batch size of 250. Both methods outperformed Adam and NGD, which achieved test accuracies of approximately 90.75% and 88.31%, respectively, over the same number of iterations. We emphasize that unlike K-FAC, AdaCurv does not assume block-diagonal or block-tridiagonal structure in the Fisher matrix, does not make the independence assumption of activations and gradients required by K-FAC, and instead approximates the transformation by the true Fisher.

5.4 Reinforcement Learning and Robotics

In addition to supervised learning experiments, we also apply our method to continuous control tasks with reinforcement learning. We perform experiments on three different simulated control tasks in the PyBullet simulator (Coumans and Bai, 2018), comparing adaptive variants of natural policy gradient (Rajeswaran *et al.*, 2017) and Trust Region Policy Optimization (Schulman *et al.*, 2015). The goal in these tasks is to control a simulated robot to walk forward as quickly as possible. We are able to directly apply our method to enhance TRPO by computing the update *direction* using AdaCurv but computing the *step size* using the KL-divergence bound from TRPO. On all three tasks the AdaCurv variant with the Adam averaging functions and a shrunk Fisher estimate applied to TRPO outperforms all methods (see Figure 5.4). The same AdaCurv variant applied to NPG outperforms "vanilla" NPG.

We also apply our method to a more practical robot control application where a simulated bi-manual robot is tasked with throwing a ball into a hoop. Figure 5.5a

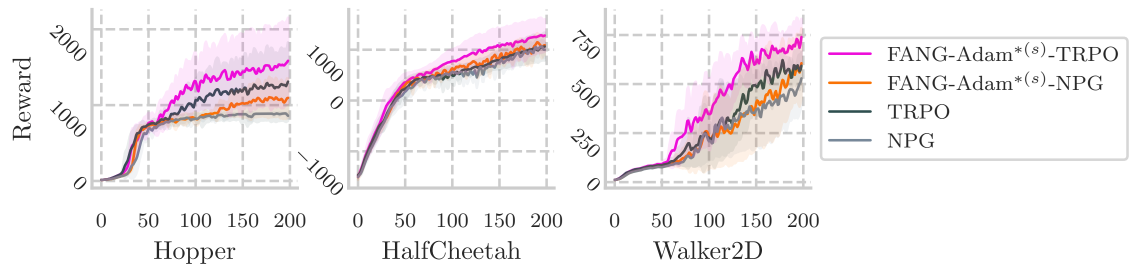
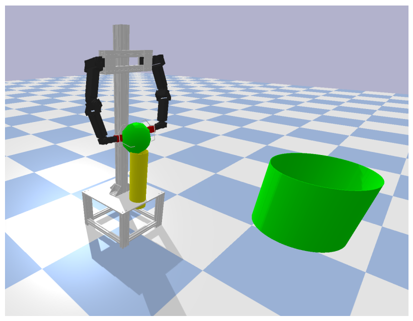


Figure 5.4: Mean and Standard Deviation Performance on Simulated Robotic Control Tasks Across 5 Seeds. AdaCurv-Adam$^{*(s)}$-$\{$*rl-algo*$\}$ Signifies That AdaCurv was Applied to the Designated RL Algorithm.

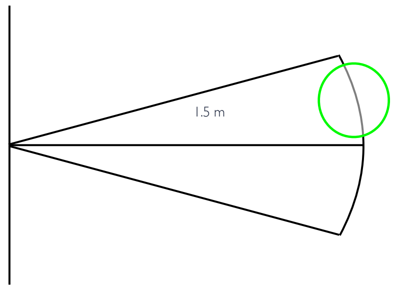
shows a render of the simulation environment. This robot is a high-fidelity model of a physical robot in our lab.

The hoop position is randomized within a 30 degree angle of the robot's orientation. At each step the robot receives the environment state as input, which includes the joint angles and velocities of both arms, the position and velocity of the ball along with the angle to the hoop. Given the input state the policy generates a control output specifying the joint velocities for each arm individually. The robot also receives a reward at each time step, which we define to be the negative Euclidean norm between the current ball velocity vector and that which would land the ball in the hoop. Intuitively, the robot is rewarded for moving the ball such that if it were to let go immediately, the ball would follow the proper trajectory into the hoop. In our experimental analysis we find AdaCurv results in as much as 50 percent improvement in learning rate.

Figure 5.6 provides some analysis of our results. Figure 5.6a shows the low dimensional space carved out by the learned controller. That is, if one samples trajectories across the possible range of hoop placements and fits a 2D embeddeding with PCA, it becomes clear the dimensions of most significance relative to the robot are left-right and in the axis of the hoop. This is not at all surprising and is, in fact, what one would intuitively expect for this task. Figure 5.6b shows learning curves for various instantiations of TRPO with and without AdaCurv. As expected, when the batch size is large the benefit of AdaCurv is relatively low since the curvature matrix can be estimated accurately with the available samples. As the batch size is decreased, the adaptive curvature estimate becomes particularly useful for fast convergence.

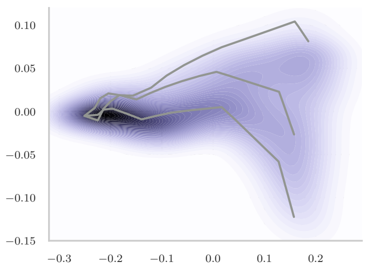


(a)

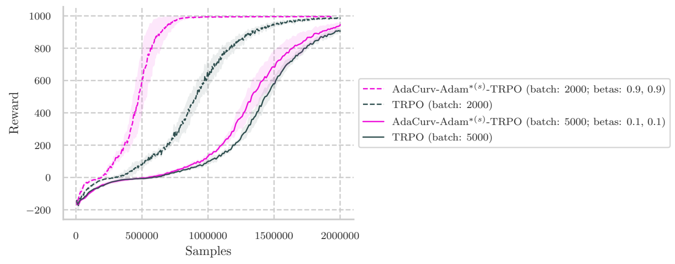


(b)

Figure 5.5: (a): Render of the Basketball Robot Simulation. (b): Diagram of the Basketball Setup. The Robot is 1.5 Meters from the Hoop and the Hoop is Randomized Within a 30 Degree Arc.



(a)



(b)

Figure 5.6: (a): A 2D Density Plot of the Latent Space Represented by the Robot Policy. The Data is Collected over 100 Trajectories with the Hoop Spaced At Equal Intervals Between the Left and Right Bounds. The Gray Lines Represent a Trajectories with the Hoop Placed on the Left, Middle, and Right. Clearly, the Two Most Significant Dimensions are the Coordinate Axes Representing Lateral and Longitudinal Movement Relative to the Hoop. (b): Learning Curves for the TRPO and AdaCurv-TRPO with Adam-Style Averaging Applied to Fisher Matrix Estimation.

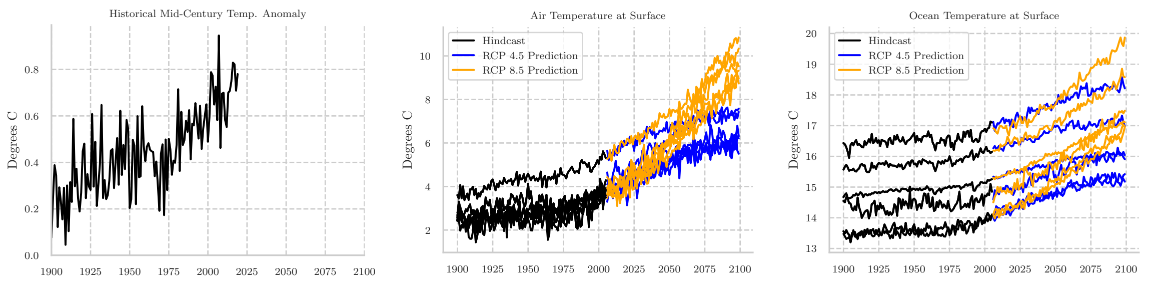
In the RL tasks, AdaCurv-Adam* applied to TRPO is about 1.15x slower than the benchmarked TRPO algorithm. The addition of shrinkage computation by CG has negligable overhead, while computing shrinkage by Lanczos adds approximately 10% additional overhead. As expected, the approximate adaptive update, AdaCurv-Adamˆ, has no additional cost.

5.5 Low-Rank Matrix Completion to Estimate Natural Disaster Trends

Matrix completion tasks aim to estimate the unknown entries of a matrix $\boldsymbol{R}$ that is hypothesized to be represented as the product of two smaller matrices, $\boldsymbol{R} \approx \boldsymbol{A}\boldsymbol{B}^\top$, where $\boldsymbol{R} \in \mathcal{R}^{n\times m}$, $\boldsymbol{A} \in \mathcal{R}^{n\times r}$, and $\boldsymbol{B} \in \mathcal{R}^{m\times r}$. That is, the matrix $\boldsymbol{R}$ is assumed to have rank r. While there are various approaches to solve this problem, one is to directly optimize the (non-linear) objective, $\mathcal{L} = \|\boldsymbol{W}(\boldsymbol{R} - \boldsymbol{A}\boldsymbol{B}^\top)\|_2^2 + \lambda_1\|\boldsymbol{A}\|_2^2 + \lambda_2\|\boldsymbol{B}\|_2^2$, where $\boldsymbol{W}$ is a matrix the same size as $\boldsymbol{R}$ that acts as a mask so that the loss is only computed over the observed values. The use of a Newton-style optimizer has been found to be effective in this context (Buchanan and Fitzgibbon, 2005).

With inspiration from Ghafarianzadeh and Monteleoni (2013) we aim to use matrix completion for climate modeling but instead of estimating future temperature trends we aim to look for relationships between temperature and the economic and social costs of natural disasters. Using model output from various institutions contributing data to the World Climate Research Program's Climate Model Intercomparison Project (CMIP) combined with natural disaster data from the NOAA (NOAA, 2019) we aim to estimate the economic cost and number of fatalities resulting from natural disasters over the next century. The CMIP5 project includes model output for various representative concentration pathways (RCP), which describe different possible future outcomes depending on the level of greenhouse gas emissions. We focus on two of these variants: RCP 4.5, which models a medium-emissions scenario, and RCP 8.5, which models a high-emissions scenario. Using this data, the matrix $\boldsymbol{R}$ is constructed. To provide some intuition, Figure 5.7 shows the data used to build the sparse matrix.

$$\boldsymbol{R} = \begin{bmatrix} \text{Historical temperature anomalies} & - \\ 6 \times \text{RCP Temp. air surface hindcast} & 6 \times \text{RCP temp. air surface forecast} \\ 6 \times \text{RCP temp. ocean surface hindcast} & 6 \times \text{RCP temp. ocean surface forecast} \\ \text{NOAA natural disaster costs} & - \\ \text{NOAA natural disaster fatalities} & - \end{bmatrix}$$



(a) (b) (c)

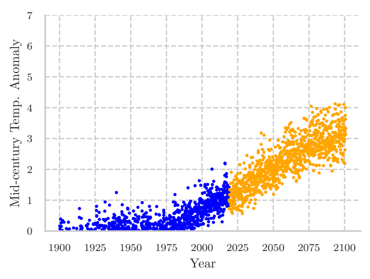
Figure 5.7: (a): The Historical Temperature Anomalies by Month Since 1900. (b): Hindcasts (black) and Forecasts (blue: RCP4.5, Orange: RCP8.5) for the Air Surface Temperature. (c): Hindcasts (black) and Forecasts (blue: RCP4.5, Orange: RCP8.5) for the Sea Surface Temperature.

We optimize the objective, $\mathcal{L}$, with AdaCurv-Adamˆ using mini-batches. That is, at each iteration, a subset of observations is used to compute the loss and the gradient. We found that, in this setting, adaptive estimates are particularly useful since, when using mini-batches, the gradient is explicitly sparse for many entries of the elements.

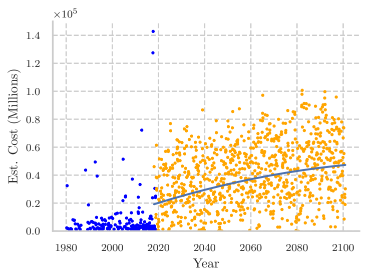
Figure 5.8 shows the estimates for each statistic from the completed matrix. Interestingly, in the RCP 4.5 case the rate of increase in cost and fatalities decreases over time, whereas in the RCP 8.5 scenario the rate continues to increase exponentially.

5.6 Runtime Comparison

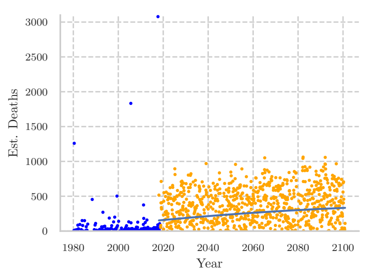
Computational time is often a concern with second-order approximations, especially those that employ truncated Newton approximations. However, we have found that



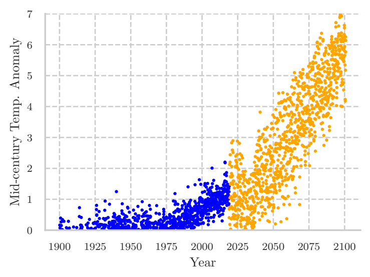
(a) RCP 4.5 Pred. Temp.



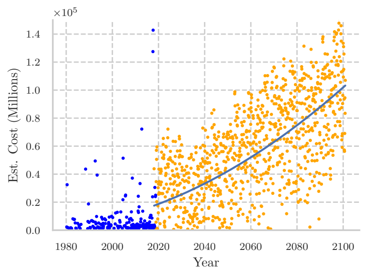
(b) RCP 4.5 Pred. Cost



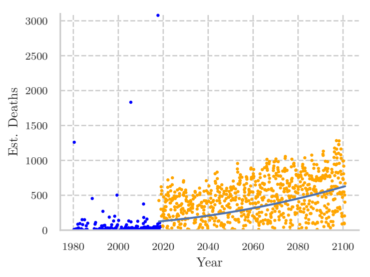
(c) RCP 4.5 Pred. Fatalities



(d) RCP 8.5 Pred. Temp.



(e) RCP 8.5 Pred. Cost



(f) RCP 8.5 Pred. Fatalities

Figure 5.8: (a,b,c): Predictions Based on the RCP 4.5 Scenario. (d,e,f): Predictions Based on the RCP 8.5 Scenario.

compared to the original Hessian-free work, we can significantly reduce the number of CG iterations at each step by using the adaptive estimates and that in some cases CG converges to the residual tolerance in fewer than 10 iterations.

In supervised tasks, an optimized implementation of AdaCurv with the approximate adaptive update takes about 1.25x the wall-clock time of the best block diagonal methods (e.g., K-FAC) for a 4-layer convolutional model, and this margin could likely be reduced by solving for the natural gradient in a layer-wise, block-diagonal fashion.

In the RL tasks, AdaCurv-Adam* applied to TRPO is about 1.15x slower than the benchmarked TRPO algorithm. The addition of shrinkage computation by CG has negligible overhead, while computing shrinkage by Lanczos adds approximately 10% additional overhead. As expected, the approximate adaptive update, AdaCurv-Adamˆ, has no additional cost.

Chapter 6

RELATED WORK

The seminal work on natural gradients comes from Amari (1998), who discusses its application to a variety of tasks and shows that it is Fisher-efficient. Since then, significant work has gone into reducing the computational complexity, resulting in incremental algorithms, as proposed by Park *et al.* (2000) and Roux *et al.* (2008). These methods form the estimate of the Fisher matrix at the current time step using the previous Fisher and integrating a rank-1 estimate, usually via the Woodbury matrix identity. These methods fit in our framework if viewed as an online adaptive natural gradient with batch size one.

Martens (2010) proposed Hessian-free learning for neural networks, which eliminates the need to construct the Hessian (or Fisher or Gauss-Newton) explicitly and instead solves $\boldsymbol{H}\boldsymbol{v} = \boldsymbol{g}$ with CG. Pascanu and Bengio (2013) examined the effectiveness of NGD for deep learning applications. More recently, Martens and Grosse (2015) proposed K-FAC, which uses an assumption of independence between the activations and gradients of a layer to formulate a factored, block-diagonal approximation to the Fisher. Botev *et al.* (2017) extend this framework to estimate the Gauss-Newton matrix. Although we have framed our method in the Hessian-free style, it extends in a straightforward manner to algorithms that employ explicit block-diagonal approximations. Note, however, that there are applications, such as matrix completion, where clear block definitions are not obvious and block-diagonal approximation may remain intractable, whereas our enhancements remains advantageous.

The need for sample-efficiency in reinforcement learning (RL) has also produced a large number of policy gradient methods based on the natural gradient beginning with Kakade (2002). Since then, the natural policy gradient algorithm has become a staple of RL research and has been repeatedly improved to include a critic (Peters and Schaal, 2008), bounded and safe step size (Schulman *et al.*, 2015), and lower computational requirements Wu *et al.* (2017). As we will see in the experiments, even these RL methods benefit from the introduced AdaCurv framework.

The large number of proposed adaptive SGD methods within the last decade (Duchi *et al.*, 2011; Zeiler, 2012; Hinton *et al.*, 2012; Kingma and Ba, 2014; Reddi *et al.*, 2018) aim to estimate the 2nd-order characteristics of the optimization landscape focusing primarily on diagonal approximations of gradient variance and do not consider adaptive estimates of the Hessian or Fisher. Martens (2014) gives a survey of natural gradient techniques along with local convergence analysis and some intuition behind why the Fisher, as an approximation to the Hessian, works well as a preconditioner. Martens also notes that use of the *empirical* Fisher as a scaling matrix, the diagonal of which is often used by existing adaptive SGD methods, is a questionable choice since it is not guaranteed that the empirical Fisher accurately describes the local properties of the model. Our work integrates the promising components from adaptive SGD methods based on a diagonal scaling to an adaptive curvature method based on the improved curvature estimates.

Chapter 7

CONCLUSION

After a discussion of a general and unified view of adaptive curvature gradient descent, we proposed AdaCurv, a practical and efficient algorithmic framework that encapsulates the update mechanics of a number of existing methods. Our framework introduces a variety of innovations including approximate update rules and sample-efficient shrinkage estimation of the curvature matrix. We found on a variety of benchmarks that AdaCurv provides excellent empirical performance. An interesting future research direction would be to investigate the application of this framework to block-diagonal curvature estimation methods.

REFERENCES

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

Balles, L. and P. Hennig, “Dissecting adam: The sign, magnitude and variance of stochastic gradients”, in “Proceedings of the 35th International Conference on Machine Learning”, edited by J. Dy and A. Krause, vol. 80 of *Proceedings of Machine Learning Research*, pp. 404–413 (PMLR, Stockholmsmssan, Stockholm Sweden, 2018), URL `http://proceedings.mlr.press/v80/balles18a.html`.

Becker, S., Y. Le Cun *et al.*, “Improving the convergence of back-propagation learning with second order methods”, in “Connectionist Models Summer School”, pp. 29–37 (IEEE, 1988).

Botev, A., H. Ritter and D. Barber, “Practical gauss-newton optimisation for deep learning”, in “Proceedings of the 34th International Conference on Machine Learning-Volume 70”, pp. 557–565 (JMLR.org, 2017).

Boyd, S. and L. Vandenberghe, *Convex optimization* (Cambridge university press, 2004).

Buchanan, A. M. and A. W. Fitzgibbon, “Damped newton algorithms for matrix factorization with missing data”, in “Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on”, vol. 2, pp. 316–322 (IEEE, 2005).

Chapelle, O. and D. Erhan, “Improved preconditioner for hessian free optimization”, in “In NIPS Workshop on Deep Learning and Unsupervised Feature Learning”, (Citeseer, 2011).

Chen, Y., A. Wiesel, Y. C. Eldar and A. O. Hero, “Shrinkage algorithms for mmse covariance estimation”, IEEE Transactions on Signal Processing **58**, 10, 5016–5029 (2010).

Coumans, E. and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning”, `http://pybullet.org` (2016–2018).

Diaconis, P. and M. Shahshahani, “The subgroup algorithm for generating uniform random variables”, Probability in the engineering and informational sciences **1**, 1, 15–32 (1987).

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

Ghafarianzadeh, M. and C. Monteleoni, “Climate prediction via matrix completion”, in “Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence”, (2013).

Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning* (MIT Press, 2016), `http://www.deeplearningbook.org`.

Hinton, G., N. Srivastava and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent", (2012).

Hutchinson, T., *Essentials of statistical methods, in 41 pages* (Rumsby Scientific Pub., 1993).

Kakade, S. M., "A natural policy gradient", in "Advances in neural information processing systems", pp. 1531–1538 (2002).

Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization", (2014).

Lanczos, C., "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators", Journal of the Research of the National Bureau of Standards **45**, 4 (1950).

Ledoit, O. and M. Wolf, "Honey, i shrunk the sample covariance matrix", The Journal of Portfolio Management **30**, 4, 110–119, URL `https://jpm.iijournals.com/content/30/4/110` (2004).

Martens, J., "Deep learning via hessian-free optimization", in "Proceedings of the 27th International Conference on Machine Learning (ICML-10)", edited by J. Fürnkranz and T. Joachims, pp. 735–742 (Omnipress, Haifa, Israel, 2010), URL `http://www.icml2010.org/papers/458.pdf`.

Martens, J., "New insights and perspectives on the natural gradient method", arXiv preprint arXiv:1412.1193 (2014).

Martens, J. and R. B. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature", in "Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015", pp. 2408–2417 (2015), URL `http://jmlr.org/proceedings/papers/v37/martens15.html`.

Martens, J., I. Sutskever and K. Swersky, "Estimating the hessian by back-propagating curvature", in "Proceedings of the 29th International Conference on Machine Learning (ICML-12)", edited by J. Langford and J. Pineau, ICML '12, pp. 1783–1790 (Omnipress, New York, NY, USA, 2012).

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", in "NIPS workshop on deep learning and unsupervised feature learning", (2011).

NOAA, "Noaa national centers for environmental information (ncei) u.s. billion-dollar weather and climate disasters", URL `https://www.ncdc.noaa.gov/billions/` (2019).

Park, H., S.-I. Amari and K. Fukumizu, "Adaptive natural gradient learning algorithms for various stochastic models", Neural Networks **13**, 7, 755–764 (2000).

Pascanu, R. and Y. Bengio, “Natural gradient revisited”, CoRR **abs/1301.3584**, URL `http://arxiv.org/abs/1301.3584` (2013).

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, “Automatic differentiation in pytorch”, in “NIPS workshop on autodiff”, (2017).

Pearlmutter, B. A., “Fast exact multiplication by the hessian”, Neural computation **6**, 1, 147–160 (1994).

Peters, J. and S. Schaal, “Natural actor-critic”, Neurocomputing **71**, 7-9, 1180–1190 (2008).

Rajeswaran, A., K. Lowrey, E. V. Todorov and S. M. Kakade, “Towards generalization and simplicity in continuous control”, in “Advances in Neural Information Processing Systems”, pp. 6550–6561 (2017).

Reddi, S. J., S. Kale and S. Kumar, “On the convergence of adam and beyond”, in “International Conference on Learning Representations”, (2018), URL `https://openreview.net/forum?id=ryQu7f-RZ`.

Roux, N. L., P.-A. Manzagol and Y. Bengio, “Topmoumoute online natural gradient algorithm”, in “Advances in neural information processing systems”, pp. 849–856 (2008).

Saad, Y., *Iterative methods for sparse linear systems*, vol. 82 (siam, 2003).

Schulman, J., S. Levine, P. Abbeel, M. Jordan and P. Moritz, “Trust region policy optimization”, in “International Conference on Machine Learning”, pp. 1889–1897 (2015).

Shewchuk, J. R. *et al.*, “An introduction to the conjugate gradient method without the agonizing pain”, (1994).

Todorov, E., T. Erez and Y. Tassa, “Mujoco: A physics engine for model-based control”, in “Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on”, pp. 5026–5033 (IEEE, 2012).

Wright, S. and J. Nocedal, “Numerical optimization”, Springer Science **35**, 67-68, 7 (1999).

Wu, Y., E. Mansimov, R. B. Grosse, S. Liao and J. Ba, “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation”, in “Advances in neural information processing systems”, pp. 5279–5288 (2017).

Zeiler, M. D., “ADADELTA: an adaptive learning rate method”, CoRR **abs/1212.5701**, URL `http://arxiv.org/abs/1212.5701` (2012).

Zhang, H., C. Xiong, J. Bradbury and R. Socher, “Block-diagonal hessian-free optimization for training neural networks”, CoRR **abs/1712.07296**, URL `http://arxiv.org/abs/1712.07296` (2017).

Zhang, L.-Q., A. Cichocki and S.-i. Amari, “Natural gradient algorithm for blind separation of overdetermined mixture with additive noise”, IEEE Signal Processing Letters **6**, 11, 293–295 (1999).

APPENDIX A

CONVERGENCE OF ADACURV-NEWTON

Proof of convergence of adaptive Newton's method on strongly convex functions

Our analysis closely follows the exposition by Boyd and Vandenberghe (2004, chap. 9.5). We emphasize the differences and explicitly note assumptions that must hold. We assume the function to be optimized, f, is twice differentiable and strongly convex and there exists constant m such that $\nabla^2 f(x) \succcurlyeq m\boldsymbol{I}$, which also implies there exists M such that $\nabla^2 f(x) \preccurlyeq M\boldsymbol{I}$. We also assume the Hessian is Lipschitz continuous with constant L,

$$\|\nabla^2 f(\boldsymbol{x}) - \nabla^2 f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

and that a backtracking line search is used with parameters $\alpha \in (0.0, 1.0)$ and $\beta \in (0.0, 0.5)$. As in the analysis by Boyd and Vandenberghe (2004), we analyze the convergence in two cases, which depend on the norm of the gradient, and affect the rate of convergence. For $0 < \eta < m^2/L$ and $\gamma > 0$ either:

1. $\|\nabla f(\boldsymbol{x}_k)\|_2 \geq \eta$ and $f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \leq -\gamma$ or,

2. $\|\nabla f(\boldsymbol{x}_k)\|_2 < \eta$ and $\frac{L}{2m^2}\|\nabla f(\boldsymbol{x}_{k+1})\|_2 \leq \left(\frac{L}{2m^2}\|\nabla f(\boldsymbol{x}_k)\|_2\right)^2$.

If the second condition is satisfied the line search selects a step size $t = 1$ and, moreover, once it has been satisfied at a given step, k, it will be satisfied for all future iterations. Define the Hessian at step k be $\boldsymbol{H}_k = \nabla^2 f(\boldsymbol{x}_k)$ and the adaptive Hessian, $\hat{\boldsymbol{H}}_k = \frac{1-\beta}{1-\beta^k}\sum_{i=1}^{k} \beta^{k-i}\boldsymbol{H}_i$. Then the AdaCurv-Newton step is $\Delta\boldsymbol{x}_{ant,k} = -\hat{\boldsymbol{H}}_k^{-1}\nabla f(\boldsymbol{x}_k)$ and the AdaCurv-Newton decrement is $\lambda(\boldsymbol{x}_k) = (\Delta\boldsymbol{x}_{ant,k}^\top \hat{\boldsymbol{H}}_k \Delta\boldsymbol{x}_{ant,k})^{1/2}$. We note that the AdaCurv-Newton step, $\Delta\boldsymbol{x}_{ant,k}$ is guaranteed to be a descent direction since $\nabla f(\boldsymbol{x}_k)\Delta\boldsymbol{x}_{ant,k} = -\nabla f(\boldsymbol{x}_k)^\top \hat{\boldsymbol{H}}_k^{-1}\nabla f(\boldsymbol{x}_k)$, the gradient must be positive, and the RHS is negative since $\hat{\boldsymbol{H}}_k^{-1}$ is PSD as it is inverse of the sum of PSD matrices.

The damped Newton phase In this phase of convergence we require that there exist γ such that $f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \leq -\gamma$.

$$\begin{aligned} f(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k}) &\leq f(\boldsymbol{x}_k) + t\nabla f(\boldsymbol{x}_k)^\top \Delta\boldsymbol{x}_{ant,k} + \frac{M\|\Delta\boldsymbol{x}_{ant,k}\|_2^2}{2}t^2 \\ &\leq f(\boldsymbol{x}_k) - t\lambda(\boldsymbol{x}_k)^2 + \frac{M}{2m}t^2\lambda(\boldsymbol{x}_k)^2 \end{aligned}$$

This last step relies on $\lambda(\boldsymbol{x}_k)^2 = \Delta\boldsymbol{x}_{ant,k}^\top \hat{\boldsymbol{H}}_k \Delta\boldsymbol{x}_{ant,k} \geq m\|\Delta\boldsymbol{x}_{ant,k}\|_2^2$. This can be seen since the minimum eigenvalue of $\hat{\boldsymbol{H}}_k$ must be greater than or equal to m. Similarly, $\lambda(\boldsymbol{x}_k)^2 = \nabla f(\boldsymbol{x}_k)^\top \hat{\boldsymbol{H}}_k^{-1}\nabla f(\boldsymbol{x}_k) \geq (1/M)\|\nabla f(\boldsymbol{x}_k)\|_2^2$ because the maximum eigenvalue of the sum will be less than or equal to the sum of the maximum eigenvalues. Hence we have,

$$f(\boldsymbol{x}_k^+) - f(\boldsymbol{x}_k) \leq -\alpha t\lambda(\boldsymbol{x}_k)^2.$$

Noting that $t = m/M$ satisfies the line search condition,

$$\leq -\alpha\beta\frac{m}{M}\lambda(\boldsymbol{x}_k)^2$$
$$\leq -\alpha\beta\eta^2\frac{m}{M^2}.$$

So the first condition is satisfied with $\gamma = \alpha\beta\eta^2\frac{m}{M^2}$.

The true Newton phase The second case uses the bound on the Lipschitz constant of the Hessian, which in general implies that,

$$\|\nabla^2 f(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k}) - \nabla^2 f(\boldsymbol{x}_k)\|_2 \leq tL\|\Delta\boldsymbol{x}_{ant,k}\|_2.$$

With an adaptive Hessian estimate this bound becomes a bit more complicated because,

$$\hat{\boldsymbol{H}}_k(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k}) = \frac{1}{1-\beta^k}\Big(\beta^{k-1}(1-\beta)\boldsymbol{H}_1(\boldsymbol{x}_1 + t\Delta\boldsymbol{x}_{ant,1}) + \beta^{k-2}(1-\beta)\boldsymbol{H}_2(\boldsymbol{x}_2 + t\Delta\boldsymbol{x}_{ant,2}) + \cdots + (1-\beta)\boldsymbol{H}_k(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k})\Big).$$

One can observe the Lipschitz bound for each of these matrices individually as it is clear that for each i the following holds:

$$\frac{\beta^{k-i}(1-\beta)}{1-\beta^k}\|\boldsymbol{H}_i(\boldsymbol{x}_i + t\Delta\boldsymbol{x}_{ant,i}) - \boldsymbol{H}_i(\boldsymbol{x}_i)\|_2 \leq tL\frac{\beta^{k-i}(1-\beta)}{1-\beta^k}\|\Delta\boldsymbol{x}_{ant,i}\|_2.$$

Then, the sum over i from 1 to k yields the desired relationship,

$$\|\hat{\boldsymbol{H}}_k(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k}) - \hat{\boldsymbol{H}}_k(\boldsymbol{x}_k)\|_2 \leq tL\|\Delta\boldsymbol{x}_{ant,k}\|_2,$$

where the sum over the Euclidean norms holds by the generalized triangle inequality.

The remainder of the proof follows identically to the traditional proof of Newton's method presented in Boyd and Vandenberghe (2004, Chap. 9.5), which we reproduce here for completeness. The above inequality implies that,

$$\left|\Delta\boldsymbol{x}_{ant,k}^\top\left(\hat{\boldsymbol{H}}_k(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k}) - \hat{\boldsymbol{H}}_k(\boldsymbol{x}_k)\right)\Delta\boldsymbol{x}_{ant,k}\right| \leq tL\|\Delta\boldsymbol{x}_{ant,k}\|_2^3.$$

Defining $\tilde{f}_k(t) = f_k(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k})$ and its second derivative $\tilde{f}_k''(t) = \Delta\boldsymbol{x}_{ant,k}^\top\hat{\boldsymbol{H}}_k(\boldsymbol{x}_k + t\Delta\boldsymbol{x}_{ant,k})\Delta\boldsymbol{x}_{ant,k}$ the above inequality becomes,

$$\left|\tilde{f}''(t) - \tilde{f}''(0)\right| \leq tL\|\Delta\boldsymbol{x}_{ant,k}\|_2^3.$$

So, in turn,

$$\tilde{f}''(t) \leq \tilde{f}''(0) + tL\|\Delta\boldsymbol{x}_{ant}\|_2^3.$$

This will be used to bound $\tilde{f}(t)$. Using the fact that $\tilde{f}''(0) = \lambda(\boldsymbol{x}_k)^2$ and $\lambda(\boldsymbol{x}_k)^3 \geq m^{3/2}\|\Delta \boldsymbol{x}_{ant,k}\|_2^3$,

$$\tilde{f}''(t) \leq \lambda(\boldsymbol{x}_k)^2 + \frac{tL}{m^{3/2}}\lambda(\boldsymbol{x}_k)^3.$$

Integrating the inequality twice and using the fact that $\tilde{f}'(0) = -\lambda(\boldsymbol{x}_k)^2$,

$$\tilde{f}(t) \leq \tilde{f}(0) - t\lambda(\boldsymbol{x}_k)^2 + t^2\frac{1}{2}\lambda(\boldsymbol{x}_k)^2 + t^3\frac{L}{6m^{3/2}}\lambda(\boldsymbol{x}_k)^3.$$

Substituting $t = 1$ (a full step size),

$$f(\boldsymbol{x}_k + \Delta \boldsymbol{x}_{ant,k}) \leq f(\boldsymbol{x}_k) - \frac{1}{2}\lambda(\boldsymbol{x}_k)^2 + \frac{L}{6m^{3/2}}\lambda(\boldsymbol{x}_k)^3.$$

Now, assume $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \eta \leq 3(1-2\alpha)m^2/L$. Because $\lambda(\boldsymbol{x}_k)^2 = \nabla f(\boldsymbol{x}_k)^\top \hat{\boldsymbol{H}}_k^{-1}\nabla f(\boldsymbol{x}_k) \leq 1/m\|\nabla f(\boldsymbol{x}_k)\|_2^2$, it follows that $\lambda(\boldsymbol{x}_k) \leq 3(1-2\alpha)m^{3/2}/L$. Substituting back into the previous inequality, we have

$$f(\boldsymbol{x}_k + \Delta \boldsymbol{x}_{ant,k}) \leq f(\boldsymbol{x}_k) - \lambda(\boldsymbol{x}_k)^2\left(\frac{1}{2} - \frac{L\lambda(\boldsymbol{x}_k)}{6m^{3/2}}\right).$$

Noting that the coefficient on $\lambda(\boldsymbol{x}_k)^2$ equals α,

$$\begin{aligned} f(\boldsymbol{x}_k + \Delta \boldsymbol{x}_{ant,k}) &\leq f(\boldsymbol{x}_k) - \alpha\lambda(\boldsymbol{x}_k)^2 \\ &= f(\boldsymbol{x}_k) + \alpha\nabla f(\boldsymbol{x}_k)^\top \Delta \boldsymbol{x}_{ant,k}, \end{aligned}$$

which shows a step size $t = 1$ is selected. Applying the Lipschitz condition to the resulting iterate shows $\|\nabla f(\boldsymbol{x}_k^+)\|_2 \leq \frac{L}{2m^2}\|\nabla f(\boldsymbol{x}_k)\|_2^2$, which is the condition case 2, and implies that if $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \eta$ with $\eta = \min\{1, 3(1-2\alpha)\}\frac{m^2}{L}$.

The number of iterations required to compute an extremely good approximation is bounded by $\frac{f(\boldsymbol{x}_0)-\boldsymbol{x}^*}{\gamma} + 6$, where the 6 iterations are from the quadratically convergent phase. Substituting this value for η into the value derived for gamma above and, finally, substituting into the aforementioned expression gives a bound on the number of iterations found to be,

$$6 + \frac{M^2L^2/m^5}{\alpha\beta\min\{1, 9(1-2\alpha)^2\}}(f(\boldsymbol{x}_0) - \boldsymbol{x}^*),$$

which completes the convergence analysis.

APPENDIX B

DETAILS OF SPECIFIC ADACURV VARIANTS

We present pseudocode for a AdaCurv-Adam and AdaCurv-Adagrad. AdaCurv-AMSGrad is a simple change from AdaCurv-Adam. All of the variants optionally use a shrinkage estimator to compute the damping factor. Algorithm 3 gives a formal algorithm for computing this shrinkage given the number of samples, the parameter vector, and the desired number of eigenvectors to compute. This function returns a diagonal matrix $\boldsymbol{D}$, which is the shrinkage target, and a factor $\rho \in [0, 1]$, which specifies how much shift the empirical estimate towards $\boldsymbol{D}$. The call to `LANCZOS` is a standard Lanczos procedure (Saad, 2003, chap. 6.7.1). The `CG` procedure is implemented as a standard conjugate gradient iteration (Shewchuk *et al.*, 1994) but the required α, β are stored in order to recover the tridiagonal Lanczos matrix.

Algorithm 3 Lanczos shrinkage estimation

```
Require: $n$: batch size
Require: $\boldsymbol{\theta}$: parameter vector representing $\boldsymbol{C}(\boldsymbol{\theta})$
Require: $k$: number of Lanczos iterations
1: $p \leftarrow \text{len}(\boldsymbol{\theta})$
2: $\lambda_i, \ldots \lambda_k \leftarrow$ LANCZOS($\boldsymbol{\theta}, k$)
3: $\tau \leftarrow \sum_i \lambda_i$                                  $\text{Tr}(\boldsymbol{C}(\boldsymbol{\theta}))$
4: $\gamma \leftarrow \tau^2 - 2\sum_{i<j} \lambda_i \lambda_j$                    $\text{Tr}(\boldsymbol{C}(\boldsymbol{\theta})^2)$
5: $\boldsymbol{D} \leftarrow \frac{\tau}{p} \boldsymbol{I}_n$
6: $\rho \leftarrow \left( \frac{(1-2/p)\gamma + \tau^2}{(n+1-2/p)[\gamma + \tau^2/p]}, 1 \right)$
7: return $\boldsymbol{D}, \rho$
```

AdaCurv-Adam

The Adam algorithm (Kingma and Ba, 2014) uses bias correction to correct for the zero initialization of the first and second moments. AdaCurv-Adam and AdaCurv-AMSGrad also incorporate bias correction but it is a bit nuanced. Because the curvature matrices are never constructed explicitly the bias correction must be incorporated into the `CG` optimization.

The bias correction for the gradient is exactly the same as Adam. The bias correction for the curvature matrix can be derived as follows,

$$\hat{\boldsymbol{C}}_t = \frac{\beta_2 \boldsymbol{C}_{t-1} + (1-\beta_2)\boldsymbol{B}_t}{(1-\beta_2^t)}$$

$$\hat{\boldsymbol{C}}_t \boldsymbol{v} = \frac{\beta_2 \boldsymbol{C}_{t-1} + (1-\beta_2)\boldsymbol{B}_t}{(1-\beta_2^t)} \boldsymbol{v}$$

This implies that while performing `CG`, in order to account for bias, one must divide the curvature-vector product by $1 - \beta_2^t$. This leads to AdaCurv-Adam* using the optimal update rule for $\boldsymbol{\theta}^-$ as shown in Algorithm 5.

We also note that on line 14 of Algorithm 5 we compute a normalized step size based on the current estimate of the Fisher. This transforms the step size onto the local metric space. This follows the form by Rajeswaran *et al.* (2017). We find that this normalized step size is sufficient for stable convergence.

Algorithm 4 CG shrinkage estimation

```
Require: $n$: batch size
Require: $\boldsymbol{\theta}$: parameter vector representing $\boldsymbol{C}(\boldsymbol{\theta})$
Require: $\boldsymbol{g}$: gradient vector to solve $\boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{v} = \boldsymbol{g}$
Require: $k$: number of conjugate gradient solver iterations
 1: $p \leftarrow \text{len}(\boldsymbol{\theta})$
 2: $\text{diag}_0 \leftarrow []$; $\text{diag}_1 \leftarrow []$
 3: for $i$ in $1 : k$ do
 4:     cg_status $\leftarrow \texttt{CG}(\boldsymbol{C}, \boldsymbol{v}, \boldsymbol{g})$
 5:     $\alpha_i \leftarrow \frac{\boldsymbol{r}_i^\top \boldsymbol{r}_i}{\boldsymbol{d}_i^\top \boldsymbol{F} \boldsymbol{d}_i}$                                CG iteration step size
 6:     $\beta_i \leftarrow \frac{\boldsymbol{r}_{i+1}^\top \boldsymbol{r}_{i+1}}{\boldsymbol{r}_i^\top \boldsymbol{r}_i}$                                CG residual step
 7:     $\text{diag}_0.\texttt{append}\left(\frac{1}{\alpha_i} + \frac{\beta_{i-1}}{\alpha_{i-1}}\right)$
 8:     if $i > 1$ then
 9:         $\text{diag}_1.\texttt{append}\left(\frac{\sqrt{\beta_{i-1}}}{\alpha_{i-1}}\right)$
10:     end if
11: end for
12: $\boldsymbol{x}_{\min} \leftarrow$ cg_status.solution
13: $\lambda_i, \ldots \lambda_k \leftarrow \texttt{eigvalsh}(\text{diag}_0, \text{diag}_1)$    Compute eigenvalues of tridiagonal matrix
14: $\tau \leftarrow \sum_i \lambda_i$                                $\text{Tr}(\boldsymbol{C}(\boldsymbol{\theta}))$
15: $\gamma \leftarrow \tau^2 - 2\sum_{i<j} \lambda_i \lambda_j$                                $\text{Tr}(\boldsymbol{C}(\boldsymbol{\theta})^2)$
16: $\boldsymbol{D} \leftarrow \frac{\tau}{p} \boldsymbol{I}_n$
17: $\rho \leftarrow \left(\frac{(1-2/p)\gamma + \tau^2}{(n+1-2/p)[\gamma + \tau^2/p]}, 1\right)$
18: return $\boldsymbol{x}_{\min}, \boldsymbol{D}, \rho$
```

In order to reduce complexity of the above algorithm, we experiment with replacing the line search in Algorithm 5 with an approximate version that does not incur any overhead. This version may be seen in Algorithm 6.

AdaCurv-AMSGrad

The AdaCurv variant with the AMSGrad averaging functions mirrors the Adam variant with one significant change. After the line search has return the new lagged parameters, $\boldsymbol{\theta}_t^-$, the largest eigenvalues $\lambda_t^{\max}$ and $\lambda_{t-1}^{\max}$ of $\boldsymbol{C}(\boldsymbol{\theta}_t^-)$ and $\boldsymbol{C}(\boldsymbol{\theta}_{t-1}^-)$ are estimated. If $\lambda_t^{\max}$ is not greater than $\lambda_{t-1}^{\max}$, then $\boldsymbol{\theta}_t^-$ is reset to be $\boldsymbol{\theta}_{t-1}^-$.

AdaCurv-Adagrad

The adaptive update performed in AdaCurv-Adagrad differs from that in AdaCurv-Adam and AdaCurv-AMSGrad. Adagrad maintains a cumulative average over the variance and does not average the gradient. AdaCurv-Adagrad performs a cumulative average over the curvature matrix. The optimal update, shown in Algorithm 7, uses the line search to find parameters that represent this cumulative average. The approximate

Algorithm 5 AdaCurv-Adam*

```
Require: $\eta$: normalized step size
Require: $\beta_1, \beta_2 \in [0, 1)$
Require: $\boldsymbol{\theta}_0$: initial parameter vector
Require: $\boldsymbol{\theta}_0^-$: initial lagged parameter vector
Require: $f(\boldsymbol{\theta})$: stochastic objective with parameters $\theta$
 1: $\boldsymbol{m}_0 \leftarrow 0$
 2: $t \leftarrow 0$
 3: while not converged do
 4:     $t \leftarrow t + 1$
 5:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
 6:     $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
 7:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
 8:     $\boldsymbol{\theta}_t^- \leftarrow \texttt{LS}\Big[\min_{\boldsymbol{\theta}} \big\| \big(\boldsymbol{C}(\boldsymbol{\theta}) - \beta_2 \boldsymbol{C}(\boldsymbol{\theta}_{t-1}^-) - (1 - \beta_2)\boldsymbol{B}(\boldsymbol{\theta}_{t-1})\big)\hat{\boldsymbol{\nu}}_{t-1} \big\| \Big]$
 9:         s.t.   $\boldsymbol{\theta} = \gamma \boldsymbol{\theta}_{t-1}^- + (1 - \gamma)\boldsymbol{\theta}_{t-1}$
10:                $\gamma \in [0, 1]$
11:     $\boldsymbol{D}_t, \rho \leftarrow \texttt{SHRINKAGE}\big(\boldsymbol{C}(\boldsymbol{\theta}_t^-)\big)$          (optionally, else $\rho =$ constant)
12:     $\boldsymbol{\nu}_t \leftarrow \texttt{CG}\left(\frac{((1-\rho)\boldsymbol{C}(\boldsymbol{\theta}_t^-) + \rho \boldsymbol{D}_t)\boldsymbol{\nu}}{1 - \beta_2^t} = \hat{\boldsymbol{m}}_t\right)$
13:     $\alpha \leftarrow \sqrt{\frac{\eta}{\hat{\boldsymbol{m}}_t^\top \boldsymbol{\nu}_t}}$
14:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \boldsymbol{\nu}_t$
15: end while
16: return $\boldsymbol{\theta}_t$
```

AdaCurv-Adagrad update is shown in Algorithm 8.

Algorithm 6 AdaCurv-Adam^

```
Require: $\eta$: normalized step size
Require: $\beta_1, \beta_2 \in [0, 1)$
Require: $\boldsymbol{\theta}_0$: initial parameter vector
Require: $\boldsymbol{\theta}_0^-$: initial lagged parameter vector
Require: $f(\boldsymbol{\theta})$: stochastic objective with parameters $\theta$
 1: $\boldsymbol{m}_0 \leftarrow 0$
 2: $t \leftarrow 0$
 3: while not converged do
 4:     $t \leftarrow t + 1$
 5:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
 6:     $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
 7:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
 8:     $\boldsymbol{\theta}_t^- \leftarrow \beta_2 \boldsymbol{\theta}_{t-1}^- + (1 - \beta_2)\boldsymbol{\theta}_{t-1}$
 9:     $\boldsymbol{D}_t, \rho \leftarrow \texttt{SHRINKAGE}\left(\boldsymbol{C}(\boldsymbol{\theta}_t^-)\right)$          (optionally, else $\rho =$ constant)
10:     $\boldsymbol{\nu}_t \leftarrow \texttt{CG}\left(\frac{((1-\rho)\boldsymbol{C}(\boldsymbol{\theta}_t^-)+\rho \boldsymbol{D}_t)\boldsymbol{\nu}}{1-\beta_2^t} = \hat{\boldsymbol{m}}_t\right)$
11:     $\alpha \leftarrow \sqrt{\frac{\eta}{\hat{\boldsymbol{m}}_t^\top \boldsymbol{\nu}_t}}$
12:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \boldsymbol{\nu}_t$
13: end while
14: return $\boldsymbol{\theta}_t$
```

Algorithm 7 AdaCurv-Adagrad*

```
Require: $\eta$: normalized step size
Require: $\boldsymbol{\theta}_0$: initial parameter vector
Require: $\boldsymbol{\theta}_0^-$: initial lagged parameter vector
Require: $f(\boldsymbol{\theta})$: stochastic objective with parameters $\theta$
 1: $t \leftarrow 0$
 2: while not converged do
 3:     $t \leftarrow t + 1$
 4:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
 5:     $\boldsymbol{\theta}_t^- \leftarrow \texttt{LS}\Big[\min_{\boldsymbol{\theta}} \left\|\left(\boldsymbol{C}(\boldsymbol{\theta}) - \left((t-1)\boldsymbol{C}(\boldsymbol{\theta}_{t-1}^-) + \boldsymbol{B}(\boldsymbol{\theta}_{t-1})\right)/t\right)\boldsymbol{\nu}_{t-1}\right\|\Big]$
 6:         s.t.   $\boldsymbol{\theta} = \gamma \boldsymbol{\theta}_{t-1}^- + (1 - \gamma)\boldsymbol{\theta}_{t-1}$
 7:                $\gamma \in [0, 1]$
 8:     $\boldsymbol{D}_t, \rho \leftarrow \texttt{SHRINKAGE}\left(\boldsymbol{C}(\boldsymbol{\theta}_t^-)\right)$          (optionally, else $\rho =$ constant)
 9:     $\boldsymbol{\nu}_t \leftarrow \texttt{CG}\left(\frac{((1-\rho)\boldsymbol{C}(\boldsymbol{\theta}_t^-)+\rho \boldsymbol{D}_t)\boldsymbol{\nu}}{1-\beta_2^t} = \boldsymbol{g}_t\right)$
10:     $\alpha \leftarrow \sqrt{\frac{\eta}{\boldsymbol{g}_t^\top \boldsymbol{\nu}_t}}$
11:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \boldsymbol{\nu}_t$
12: end while
13: return $\boldsymbol{\theta}_t$
```

Algorithm 8 AdaCurv-Adagradˆ

```
Require: $\eta$: normalized step size
Require: $\boldsymbol{\theta}_0$: initial parameter vector
Require: $\boldsymbol{\theta}_0^-$: initial lagged parameter vector
Require: $f(\boldsymbol{\theta})$: stochastic objective with parameters $\theta$
 1: $t \leftarrow 0$
 2: while not converged do
 3:     $t \leftarrow t + 1$
 4:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
 5:     $\boldsymbol{\theta}_t^- \leftarrow \big((t-1)\boldsymbol{\theta}_{t-1}^- + \boldsymbol{\theta}_{t-1}\big)/t$
 6:     $\boldsymbol{D}_t, \rho \leftarrow \texttt{SHRINKAGE}\big(\boldsymbol{C}(\boldsymbol{\theta}_t^-)\big)$                    (optionally, else $\rho =$ constant)
 7:     $\boldsymbol{\nu}_t \leftarrow \texttt{CG}\left(\frac{((1-\rho)\boldsymbol{C}(\boldsymbol{\theta}_t^-)+\rho\boldsymbol{D}_t)\boldsymbol{\nu}}{1-\beta_2^t} = \boldsymbol{g}_t\right)$
 8:     $\alpha \leftarrow \sqrt{\frac{\eta}{\boldsymbol{g}_t^\top \boldsymbol{\nu}_t}}$
 9:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha\boldsymbol{\nu}_t$
10: end while
11: return $\boldsymbol{\theta}_t$
```

APPENDIX C

ANALYSIS OF BIAS AND VARIANCE OF ADAPTIVE METHODS

We analyze, in the finite sample case, the relationship between the bias and variance of the adaptive and single-step gradient estimates for a general adaptive method. The adaptive estimates reduce error incurred by Monte Carlo sampling at the expense of bias.

To simplify the analysis, assume the gradient at time t is a normal random variable $x_t \sim \mathcal{N}(\mu_t, \sigma_t)$. The estimate of the mean $\bar{x}_t$ has Monte Carlo error on the order of $O(\sigma_t/\sqrt{n})$ for an estimate derived from n samples (Hutchinson, 1993). The mean and variance of an adaptive gradient estimate can be derived as follows. First, the mean, with $\delta_{\mu,t} = \mu_T - \mu_t$ as the difference between μ_T and μ_t, can be estimated according to:

$$\begin{aligned} E[\hat{\mu_t}] &= (1-\beta_1)\sum_{t=1}^{T} \beta_1^{T-t}\mu_t \\ &= (1-\beta_1)\sum_{t=1}^{T} \beta_1^{T-t}(\mu_T - \delta_{\mu,t}) \\ &= (1-\beta_1^T)\mu_T - (1-\beta_1)\sum_{t=1}^{T}\beta_1^{T-t}\delta_{\mu,t}. \end{aligned}$$

In turn, similarly letting $\delta_{\sigma,t} = \sigma_T - \sigma_t$ be the difference between σ_T and σ_t, the variance can be estimated according to:

$$V_T = \mathrm{Var}[\hat{\mu_t}] = \sum_{t=1}^{T} \left((1-\beta_2)\beta_2^{T-t}\right)^2 \sigma_t^2 + 2\sum_{1<i}\sum_{<j\le n}(1-\beta_2)\beta_2^{T-i}(1-\beta_2)\beta_2^{T-j}\rho\sigma_i\sigma_j.$$

Since the absolute value of the correlation $|\rho|$ is less than or equal to 1, we can write

$$V_T \le \sum_{t=1}^{T} \left((1-\beta_2)\beta_2^{T-t}\right)^2 \sigma_t^2 + 2\sum_{1<i}\sum_{<j\le n}(1-\beta_2)\beta_2^{T-i}(1-\beta_2)\beta_2^{T-j}\sigma_i\sigma_j.$$

Factoring this expression into the form $(x_1 + x_2 + \ldots + x_n)^2$ gives

$$V_T \le \left((1-\beta_2)\beta_2^{T-1}\sigma_1 + (1-\beta_2)\beta_2^{T-2}\sigma_2 + \ldots + (1-\beta_2)\sigma_T\right)^2.$$

Setting $\sigma_t = \sigma_T - \delta_{\sigma,t}$, we get

$$\begin{aligned} V_T &\le \left((1-\beta_2)\beta_2^{T-1}(\sigma_T - \delta_{\sigma,1}) + (1-\beta_2)\beta_2^{T-2}(\sigma_T - \delta_{\sigma,2}) + \ldots + (1-\beta_2)\sigma_T\right)^2 \\ &\le \left(\sigma_T - (1-\beta_2)\beta_2^{T-1}\delta_{\sigma,1} - (1-\beta_2)\beta_2^{T-2}\delta_{\sigma,2} - \ldots - (1-\beta_2)\beta_2\delta_{\sigma,T-1}\right)^2. \end{aligned}$$

Given these quantities, we compare the Monte Carlo sampling error of the adaptive and single-step estimates. Let

$$\begin{aligned} \Delta_{\sigma,T} &= -(1-\beta_2)\beta_2^{T-1}\delta_{\sigma,1} - (1-\beta_2)\beta_2^{T-2}\delta_{\sigma,2} - \ldots - (1-\beta_2)\beta_2\delta_{\sigma,T-1}, \\ \Delta_{\mu,T} &= (1-\beta_1)\sum_{t=1}^{T}\beta_1^{T-t}\delta_{\mu,t}. \end{aligned}$$

Comparing the Monte Carlo errors we get

$$\frac{\sqrt{V_T}}{\sqrt{Tn}} \leq \frac{\sigma_T}{\sqrt{n}}.$$

The above expression relates the Monte Carlo sampling error for adaptive methods (LHS) to the sampling error of single-step methods (RHS). We are particularly interested in the conditions under which the left-hand side is smaller than the right-hand side. Accordingly, we can rewrite the inequality as

$$\begin{aligned} \sqrt{V_T} &\leq \frac{\sigma_T \sqrt{Tn}}{\sqrt{n}} \\ \Delta_{\sigma,T} &\leq \frac{\sigma_T \sqrt{Tn}}{\sqrt{n}} - \sigma_T \\ &\leq \sigma_T(\sqrt{T} - 1). \end{aligned}$$

The above derivation implies that in cases where, $\Delta_{\sigma,T}$, the negative, discounted sum of $\delta_{\sigma,t}$, is less than $\sigma_T(\sqrt{T}-1)$, the adaptive estimate will have a smaller Monte Carlo error. This is a rather benign assumption since β_2 is often set close to 1 and, in general, $\delta_{\sigma,t}$ will be small. Hence, under reasonable conditions, the adaptive estimate has lower variance. Given the Monte Carlo errors, we can investigate how likely the adaptive estimate is to be closer to the truth when compared to the single-step estimate. To analyze this, we also need the bias of the adaptive estimate,

$$\begin{aligned} B_T &= \mu_T - ((1-\beta_1)\mu_T - \Delta_{\mu,T}) \\ &= \beta_1 \mu_T + \Delta_{\mu,T}. \end{aligned}$$

The bias of the single-step estimate is zero because $E[\bar{x}_t] = \mu_t$. Thus, if the bias of the adaptive estimate plus the expected error of the adaptive estimate is less than the expected error of the single-step estimate, then the adaptive estimate will be more accurate. Accordingly, we have the expected accuracy of the adaptive estimator, in comparison to the single-step estimate,

$$\begin{aligned} B_T + \frac{\sqrt{V_T}}{\sqrt{Tn}} &\leq \frac{\sigma_T}{\sqrt{n}} \\ \beta_1 \mu_T + \Delta_{\mu,T} &\leq \frac{\sigma_T}{\sqrt{n}} - \frac{\sqrt{V_T}}{\sqrt{Tn}}. \end{aligned}$$

We can see that the bias must be less than the difference in the variances. By assuming that $\Delta_{\sigma,T}$ has expected value zero, i.e., the variance of the gradients is constant, we can simplify further,

$$\begin{aligned} \beta_1 \mu_T + \Delta_{\mu,T} &\leq \frac{\sigma_T}{\sqrt{n}} - \frac{\sigma_T + \Delta_{\sigma,T}}{\sqrt{Tn}} \\ \beta_1 \mu_T + \Delta_{\mu,T} &\leq \frac{\sigma_T(\sqrt{T}-1)}{\sqrt{Tn}}. \end{aligned}$$

Assuming a value of β_1 close to one, the value of $\Delta_{\mu,T}$ will be close to zero. Accordingly, whether the above condition holds depends primarily on the magnitude of the gradient, μ_T, and its (constant) standard deviation, σ_T. Moreover, since the standard deviation is scaled by a factor approximately equal to $\frac{1}{\sqrt{n}}$, the gradient mean must decrease in magnitude.

APPENDIX D

EXPERIMENT HYPERPARAMETERS

This section details the hyperparameters used in our MNIST and reinforcement learning experiments. All experiments are run using the PyTorch framework (Paszke *et al.*, 2017). The learning rate for Adam, AMSGrad, Adagrad and SGD-momentum was initialized at 0.001 for all runs. The betas (β_1, β_2) in Adam and AMSGrad set set to the recommended defaults of 0.9 and 0.999. The momentum in SGD was set to 0.9.

Details of logistic regression task and Fisher error analysis

In the logistic regression experiment we generate random data from two classes using the `sklearn.datasets.make_regression` utility. The generated dataset contains 500 samples of two dimensions so the results can be easily interpreted.

We use a similar approach to compute the Fisher error for the sample and shrunk Fishers. In this case, we generate additional datasets using the sklearn utility of varying dimenion and batch size. When the model is small, as in this experiment, the curvature can be computed fully, as opposed to observed through curvature-vector products. The *true* curvature is computed by calculating the curvature over a very large data sample (we use 10^5 samples). Finally, the error is computed as:

$$\begin{aligned}
\lambda_1^{\text{true}} &= \rho(\boldsymbol{C}^{\text{true}}) \\
\lambda_1^{\text{est}} &= \rho(\boldsymbol{C}^{\text{est}}) \\
\mathit{error} &= |\lambda_1^{\text{true}} - \lambda_1^{\text{est}}|.
\end{aligned}$$

As shown in Section 5.2 of the paper, the shrinkage reduces the error in the curvature matrix.

MNIST and SVHN

Hyperparameter	Value
Learning rate	0.001 [1]
Learning rate decay factor	$1/\sqrt{\text{epoch}}$
Network structure	MLP(784, 100, 10)
Activation function	ReLU
Lanczos k (where applicable)	10
CG damping (when not shrunk)	0.0001
CG iters	10
β_1 (where applicable)	0.1
β_2 (where applicable)	0.1
Epochs	10
Batch size	[125, 250, 500, 1000]

Table D.1: MNIST Hyperparameters

The hyperparameters in the SVHN experiment are exactly the same as MNIST except that the model used was MLP(1024,256,10) and the images were transformed to grayscale.

[1]AdaCurv-Adam and AdaCurv-AMSGrad with batch sizes of 125 and 250 use learning rates of

Reinforcement Learning with PyBullet

The RL experiments are performed in the Bullet Physics Simulator (Coumans and Bai, 2018). Note that the tasks in this simulator are tuned to be more realistic than those crafted for MuJoCo (Todorov *et al.*, 2012) thus the performance of the learned policy is not directly comparable.

Hyperparameter	Value
Steps per policy update	5000
Policy network structure	MLP(*obs_d*, 64, *act_d*)
Policy activation function	Tanh
Policy learning rate	0.005
Critic structure	MLP(*obs_d*, 64, 64, 1)
Critic activation function	ReLU
Critic batch size	64
Critic L_2 regu. coeff.	0.001
Critic epochs / policy update	2
Critic learning rate	0.001
Total steps	1000000
Discount factor γ	0.995
Generalized advantage est. λ	0.97
Lanczos k (where applicable)	5
CG damping (when not shrunk)	0.0001
CG iters	10
β_1 (where applicable)	0.1
β_2 (where applicable)	0.1

Table D.2: RL Hyperparameters

Matrix Completion for Estimating Natural Disaster Trends

In our matrix completion experiment we use climate models from six international institutes. Table D.3 lists the models used in our experiment. There is data available for RCP 4.5 and 8.5 for each of these models.

The sparse matrix is of dimension 15 × 2412 The rows represent 1 historical data, 5 models for each of air and sea temperature, 1 NOAA billion dollar costs, 1 NOAA billion dollar fatalities. The matrix columns represent months from 1900 through 2100. We use a model of rank 5 and represent the matrix $\boldsymbol{R} \approx \boldsymbol{A}\boldsymbol{B}^\top$, where $\boldsymbol{R} \in \mathcal{R}^{15\times2412}, \boldsymbol{A} \in \mathcal{R}^{15\times5}$, and $\boldsymbol{B} \in \mathcal{R}^{2412\times5}$. We optimize the objective with batch sizes of 5000 and a learning rate of 0.01 that is set to decay when progress on the objective plateaus. Because this objective does not have a clear probabilistic interpretation, we employ Gauss-Newton curvature for this task.

Finally, we gratefully acknowledge the World Climate Research Programme's Working Group on Coupled Modelling, which is responsible for CMIP, and we thank

0.0001 and 0.0005, respectively.

the climate modeling groups (listed in Table D.3 of this paper) for producing and making available their model output.

Institute	Model
CCCma	CanESM2
CNRM	CNRM-CM5
NASA GISS	GISS-E2-H
NIMR/KMA	HadGEM2-A0
IPSL	IPSL-CM5A-MR
MPI-M	MPI-ESM-MR

Table D.3: Climate Models Used in Matrix Completion Experiment.