An adaptive low dimensional quasi-Newton sum of functions optimizer

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Ben Poole POOLE@CS.STANFORD.EDU
Surya Ganguli SGANGULI@STANFORD.EDU

Abstract

We present an algorithm for minimizing a sum of functions that combines the computational efficiency of stochastic gradient descent (SGD) with the second order curvature information accessible by quasi-Newton methods. We unify these disparate approaches by maintaining an independent Hessian approximation for each contributing function in the sum. We maintain computational tractability even for high dimensional optimization problems by developing an adaptive scheme to store and manipulate these quadratic approximations in a shared, time evolving low dimensional subspace, determined by the recent history of gradient evaluations. This algorithm contrasts with earlier stochastic second order techniques, which treat the Hessian of each contributing function only as a noisy approximation to the full Hessian, rather than as a target for direct estimation. Our approach reaps the benefits of both SGD and quasi-Newton methods; each update step requires only a single subfunction evaluation (like SGD but unlike previous stochastic second order methods), while little to no adjustment of hyperparameters is required (as is typical for quasi-Newton methods but not for SGD). For convex problems the convergence rate of the proposed technique is at least linear. We demonstrate improved convergence on five diverse optimization problems.

1. Introduction

A common problem in computer science is to find a vector $\mathbf{x}^* \in \mathcal{R}^M$ which minimizes a function $F(\mathbf{x})$, where $F(\mathbf{x})$ is a sum of N computationally cheaper differentiable subfunctions $f_i(\mathbf{x})$,

$$F(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}), \tag{1}$$

$$\mathbf{x}^* = \underset{x}{\operatorname{argmin}} F(\mathbf{x}). \tag{2}$$

Many optimization tasks fit this form (Boyd & Vandenberghe, 2004), including training of autoencoders, support vector machines, and logistic regression algorithms, as well as parameter estimation in probabilistic models (Sohl-Dickstein et al., 2011). In statistics, x^* as defined by Equations 1 and 2 is referred to as an M-estimator (Huber, 1981).

There are two general approaches to efficiently optimizing a function of this form. The first is to use a quasi-Newton method (Dennis Jr & Moré, 1977), of which BFGS (Dennis Jr & Moré, 1977) or LBFGS (Liu & Nocedal, 1989) are the most common choice. Quasi-Newton methods use the history of gradient evaluations to build up an approximation to the inverse Hessian of the objective function $F(\mathbf{x})$. By making descent steps which are scaled by the approximate inverse Hessian, and which are therefore longer in directions of shallow curvature and shorter in directions of steep curvature, quasi-Newton methods can be orders of magnitude faster than steepest descent. Additionally, quasi-Newton techniques typically require adjusting few or no hyperparameters, because they use the measured curvature of the objective function to set step lengths and directions. However, direct application of quasi-Newton methods requires calculating the gradient of the full objective function $F(\mathbf{x})$ at every proposed parameter setting $\mathbf{x}$, which can be very computationally expensive, especially for large N.

The second approach is to use a variant of Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951; Bottou, 1991). In SGD, only one subfunction's gradient is evaluated per update step, and a small step is taken in the negative gradient direction. More recent descent techniques like IAG (Blatt et al., 2007), SAG (Roux et al., 2012), and MISO (Mairal, 2013) instead take update steps in the average gradient direction. For each update step, they evaluate the gradient of one subfunction, and update the average gradient using its new value. If the subfunctions are similar, then SGD can also be orders of magnitude faster than steepest descent on the full batch. However, because a different subfunction is evaluated for each update step, the gradients for each update step cannot be combined in a straightforward way to estimate the inverse Hessian of the full objective function. Additionally, efficient optimization with SGD typically involves tuning a number of hyperpa-

rameters, which can be a painstaking and frustrating process. (Ngiam & Coates, 2011) compares the performance of stochastic gradient and quasi-Newton methods on neural network training, and finds both to be competitive.

Combining quasi-Newton and stochastic gradient methods could improve optimization time, and reduce the need to tweak optimization hyperparameters. This problem has been approached from a number of directions. In (Schraudolph et al., 2007; Sunehag et al., 2009) a stochastic variant of LBFGS is proposed. In (Martens, 2010), (Byrd et al., 2011), and (Vinyals & Povey, 2011) stochastic versions of Hessian-free optimization are implemented and applied to optimization of deep networks. In (Lin et al., 2008) a trust region Newton method is used to train logistic regression and linear SVMs using minibatches. Stochastic meta-descent (Schraudolph, 1999), AdaGrad (Duchi et al., 2010), and SGD-QN (Bordes et al., 2009) rescale the gradient independently for each dimension, and can be viewed as accumulating something similar to a diagonal approximation to the Hessian. All of these techniques treat the Hessian on a subset of the data as a noisy approximation to the full Hessian. To reduce noise, they rely on regularization and large minibatches to descend $F(\mathbf{x})$ despite these noisy Hessian observations. Thus, unfortunately each update step requires the evaluation of many subfunctions and/or yields a highly regularized (i.e. diagonal) approximation to the full Hessian.

We develop a novel second-order quasi-Newton technique that only requires the evaluation of *a single* subfunction per update step. In order to achieve this substantial simplification, we treat the full Hessian of each subfunction as a direct target for estimation, thereby maintaining a separate quadratic approximation of each subfunction. This approach differs from all previous work, which in contrast treats the Hessian of each subfunction as a noisy approximation to the full Hessian. Our approach allows us to combine Hessian information from multiple subfunctions in a much more natural and efficient way than previous work, and avoid the use of large minibatches per update step to accurately estimate the full Hessian. Moreover, we develop a novel method to maintain computational tractability of this quasi-Newton method in the face of high dimensional optimization problems (large M), by storing and manipulating the subfunctions in a shared, adaptive low dimensional subspace, determined by the recent history of the gradients and positions.

Thus our optimization method can usefully estimate and utilize powerful second-order information inherent in the total function $F(\mathbf{x})$ while simultaneously combatting two potential sources of computational intractability: large numbers of subfunctions (large N) and a high-dimensional optimization domain (large M). Moreover, the use of a second order approximation means that minimal or no adjustment of hyperparameters is required. We refer to the resulting algorithm as Sum of Functions Optimizer (SFO). We demonstrate that the combination of techniques and new ideas inherent in SFO results in faster optimization on five disparate example problems. Finally, we release the optimizer and the test suite as an open source Python package.

2. Algorithm

Our goal is to combine the benefits of stochastic and quasi-Newton optimization techniques. We first describe the general procedure by which we optimize the parameters $\mathbf{x}$. We then describe the process by which an independent online Hessian approximation is maintained for each subfunction. This is followed by an explanation of the construction of the shared low dimensional subspace which makes the algorithm tractable for large problems. Finally, we end this section with a review of implementation details.

2.1. Approximating Functions

We define a series of functions $G^t(\mathbf{x})$ intended to approximate $F(\mathbf{x})$,

$$G^t(\mathbf{x}) = \sum_{i=1}^{N} g_i^t(\mathbf{x}), \tag{3}$$

where the superscript t indicates the learning iteration. The functions $g_i^t(\mathbf{x})$ will be stored, and one of them will be updated per learning step. Each $g_i^t(\mathbf{x})$ serves as a quadratic approximation to the corresponding $f_i(\mathbf{x})$.

2.2. Update Steps

As is illustrated in Figure 1, optimization is performed by repeating the steps:

1. Choose a vector $\mathbf{x}^t$ by minimizing the approximating objective function $G^{t-1}(\mathbf{x})$,

$$\mathbf{x}^t = \operatorname*{argmin}_{\mathbf{x}} G^{t-1}(\mathbf{x}). \tag{4}$$

   Note that $G^{t-1}(\mathbf{x})$ can be minimized in closed form, since it is a sum of quadratic functions $g_i^{t-1}(\mathbf{x})$.

2. Choose an index $j \in \{1...N\}$, and update the corresponding approximating subfunction $g_i^t(\mathbf{x})$ using a second order power series around $\mathbf{x}^t$, while leaving all other subfunctions unchanged,

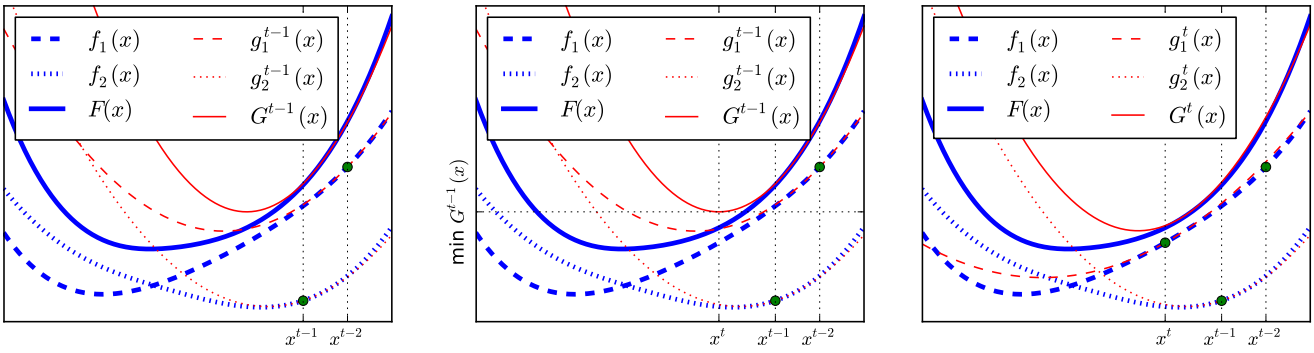

Figure 1. A cartoon illustrating the proposed optimization technique. *(a)* The objective function $F(x)$ (solid blue line) consists of a sum of two subfunctions (dashed blue lines), $F(x) = f_1(x) + f_2(x)$. At learning step $t-1$, $f_1(x)$ and $f_2(x)$ are approximated by quadratic functions $g_1^{t-1}(x)$ and $g_2^{t-1}(x)$ (red dashed lines). The sum of the approximating functions $G^{t-1}(x)$ (solid red line) approximates the full objective $F(x)$. *(b)* The next parameter setting x^t is chosen by minimizing the approximating function $G^{t-1}(x)$ from the prior update step. See Equation 4. *(c)* After each parameter update, the quadratic approximation for one of the subfunctions is updated using a second order expansion around the new parameter vector x^t. See Equation 5. The constant and first order term in the expansion are evaluated exactly, and the second order term is estimated by performing BFGS on the subfunction's history. In this case the approximating subfunction $g_1^t(x)$ is updated (long dashed red line). This update is also reflected in the full approximating function $G^t(x)$ (solid red line). Optimization proceeds by repeating these two illustrated update steps.

$$g_i^t(\mathbf{x}) = \begin{cases} g_i^{t-1}(\mathbf{x}) & i \neq j \\ \left[\begin{array}{l} f_i(\mathbf{x}^t) \\ \quad + (\mathbf{x}-\mathbf{x}^t)^T f_i'(\mathbf{x}^t) \\ \quad + \frac{1}{2}(\mathbf{x}-\mathbf{x}^t)^T \mathbf{H}_i^t (\mathbf{x}-\mathbf{x}^t) \end{array}\right] & i = j \end{cases} \tag{5}$$

2.3. Online Hessian Approximation

The constant and first order term in Equation 5 are set by evaluating the subfunction and gradient, $f_j(\mathbf{x}^t)$ and $f_j'(\mathbf{x}^t)$. We set the quadratic term $\mathbf{H}_j^t$ by using the BFGS (Dennis Jr & Moré, 1977) algorithm to generate an online approximation to the true Hessian of subfunction j based on its history of gradient evalutaions[1].

For the subfunction j, we construct two matrices, $\Delta f'$ and $\Delta \mathbf{x}$. Each column of $\Delta f'$ holds the change in the gradient of subfunction j between successive evaluations of that subfunction, including all evaluations up until the present time. Each column of $\Delta \mathbf{x}$ holds the corresponding change in the position $\mathbf{x}$ between successive evaluations. Both matrices are truncated after a number of columns L, meaning that they include information from only the prior $L+1$ gradient evaluations for each subfunction. For all results in this paper, $L = 10$ (identical to the default history length for the LBFGS implementation used in Section 4).

[1]We additionally experimented with Symmetric Rank 1 (Dennis Jr & Moré, 1977) updates to the approximate Hessian, but found they performed consistently worse than BFGS.

2.3.1. BFGS Updates

The BFGS algorithm functions by iterating through the columns in $\Delta f'$ and $\Delta \mathbf{x}$, from oldest to most recent. Let s be the column index, and $\mathbf{B}_s$ be the approximate Hessian for subfunction j after processing column s. For each s, the approximate Hessian matrix $\mathbf{B}_s$ is set so that it obeys the secant equation $\Delta f'_s = \mathbf{B}_s \Delta \mathbf{x}_s$ for the corresponding columns, where $\Delta f'_s$ and $\Delta \mathbf{x}_s$ are taken to refer to the sth columns of the gradient difference and position difference matrix respectively.

In addition to satisfying the secant equation, $\mathbf{B}_s$ is chosen such that the difference between it and the prior estimate $\mathbf{B}_{s-1}$ has the smallest weighted Frobenius norm[2]. This produces the update equation

$$\mathbf{B}_s = \mathbf{B}_{s-1} + \frac{\Delta f'_s \Delta f'^T_s}{\Delta f'^T_s \Delta \mathbf{x}_s} - \frac{\mathbf{B}_{s-1}\Delta \mathbf{x}_s \Delta \mathbf{x}_s^T \mathbf{B}_{s-1}}{\Delta \mathbf{x}_s^T \mathbf{B}_{s-1} \Delta \mathbf{x}_s}. \tag{6}$$

The final update is used as the approximate Hessian for subfunction j, $\mathbf{H}_j^t = \mathbf{B}_{\max(s)}$.

[2]The weighted Frobenius norm is defined as $||\mathbf{E}||_{F,\mathbf{W}} = ||\mathbf{WEW}||_F$. For BFGS, $\mathbf{W} = \mathbf{B}_s^{-\frac{1}{2}}$ (Papakonstantinou, 2009). Equivalently, in BFGS the Frobenius norm is minimized after linearly mapping the new approximate Hessian into the identity matrix.

2.3.2. The First BFGS Step

The initial approximate Hessian matrix used in BFGS is set to a scaled identity matrix, so that $\mathbf{B}_0 = \beta\mathbf{I}$. The scaling factor β is set to the smallest non-zero eigenvalue of a matrix $\mathbf{Q}$,

$$\beta = \min_{\lambda_Q > 0} \lambda_Q. \tag{7}$$

where λ_Q indicates the eigenvalues of $\mathbf{Q}$. $\mathbf{Q}$ is the symmetric matrix with the smallest Frobenius norm which is consistent with the squared secant equations for all columns in $\Delta f'$ and $\Delta\mathbf{x}$. That is,

$$\mathbf{Q} = \left[(\Delta\mathbf{x})^{+T} (\Delta f')^T \Delta f' (\Delta\mathbf{x})^{+} \right]^{\frac{1}{2}}, \tag{8}$$

where $^+$ indicates the pseudoinverse, and $\frac{1}{2}$ indicates the matrix square root. All of the eigenvalues of $\mathbf{Q}$ are non-negative. Equations 7 and 8 are computed in the subspace defined by $\Delta f'$ and $\Delta\mathbf{x}$, reducing computational cost (see Table 1).

The use of the smallest non-zero eigenvalue can be motived by observing that β sets the approximate Hessian in all unexplored directions in parameter space. Gradient descent routines tend to progress from directions with large slopes and curvatures, and correspondingly large eigenvalues, to directions with shallow slopes and curvatures, and smaller eigenvalues. The typical eigenvalue in an unexplored direction is thus expected to be smaller than in the previously explored directions. Equation 7 sets β to the smallest eigenvalue of $\mathbf{Q}$ in an explored direction, and may thus be a reasonable guess for the curvature in unexplored directions.

2.3.3. Enforcing Positive Definiteness

It is typical in quasi-Newton techniques to enforce that the Hessian approximation remain positive definite. In SFO each $\mathbf{H}_i^t$ is constrained to be positive definite by performing an eigendecomposition, and setting any eigenvalues which are too small to the median positive eigenvalue. If λ_{max} is the maximum eigenvalue of $\mathbf{H}_i^t$, then any eigenvalues smaller than $\gamma\lambda_{max}$ are set to be equal to $\text{median}_{\lambda>0}\,\lambda$. The median is used because it provides a measure of "typical" curvature, and when an eigenvalue is negative or extremely small it is an indication that it cannot be trusted as a measure of curvature. For all experiments shown here, $\gamma = 10^{-8}$.

2.3.4. Properties

If the Hessian is constant, then BFGS will eventually converge to the true Hessian (Dennis Jr & Moré, 1977). However, the updates in Equation 6 can make $\mathbf{B}_s$ inconsistent with the secant equation for earlier steps r, $r < s$. This makes BFGS particularly effective in settings where the Hessian is changing over the course of the learning trajectory, since more recent gradient evaluations tend to overwrite older evaluations.

2.4. A Shared, Adaptive Low-Dimensional Representation

The dimensionality M of $\mathbf{x} \in \mathcal{R}^M$ is typically large. As a result, the memory and computational cost of working directly with the matrices $\mathbf{H}_i^t \in \mathcal{R}^{M\times M}$ and the history terms for each subfunction $\Delta f'$ and $\Delta\mathbf{x}$ is typically prohibitive. To reduce the dimensionality M to a tractable value, all history is stored and all updates computed in a lower dimensional subspace, with dimensionality between K_{min} and K_{max}. The subspace is constructed such that it includes the most recent gradient and position for every subfunction. By construction, it therefore includes the steepest gradient descent direction.

For the results in this paper, $K_{min} = 2N$ and $K_{max} = 3N$. The subspace is represented by the orthonormal columns of a matrix $\mathbf{P}^t \in \mathcal{R}^{M\times K^t}$, $(\mathbf{P}^t)^T \mathbf{P}^t = \mathbf{I}$. K^t is the subspace dimensionality at optimization step t.

2.4.1. Expanding the Subspace with a New Observation

At each optimization step, an additional column is added to the subspace, expanding it to include the most recent gradient direction. This is done by first finding the component in the gradient vector which lies outside the existing subspace, and then appending that component to the current subspace,

$$\mathbf{q}_{\text{orth}} = f_j'(\mathbf{x}^t) - \mathbf{P}^{t-1}\left(\mathbf{P}^{t-1}\right)^T f_j'(\mathbf{x}^t), \tag{9}$$

$$\mathbf{P}^t = \begin{bmatrix} \mathbf{P}^{t-1} & \frac{\mathbf{q}_{\text{orth}}}{||\mathbf{q}_{\text{orth}}||} \end{bmatrix}, \tag{10}$$

where j is the subfunction updated at time t. The new position $\mathbf{x}^t$ is included automatically, since the position update was computed within the subspace $\mathbf{P}^{t-1}$. Vectors embedded in the subspace $\mathbf{P}^{t-1}$ can be updated to lie in $\mathbf{P}^t$ simply by appending a 0, since the first K^{t-1} dimensions of $\mathbf{P}^t$ consist of $\mathbf{P}^{t-1}$.

2.4.2. Restricting the Size of the Subspace

In order to prevent the dimensionality K^t of the subspace from growing too large, whenever $K^t > K_{max}$, the subspace is collapsed to only include the most recent gradient and position measurements from each subfunction. The orthonormal matrix representing this collapsed subspace is computed by a QR decomposition on the most recent gradients and positions. A new collapsed subspace is thus com-

puted as,

$$\mathbf{P}' = \text{orth}\left(\left[f_1'\left(\mathbf{x}^{\tau_1^t}\right)\cdots f_N'\left(\mathbf{x}^{\tau_N^t}\right) \quad \mathbf{x}^{\tau_1^t}\cdots\mathbf{x}^{\tau_N^t}\right]\right), \tag{11}$$

where τ_i^t indicates the learning step at which the ith subfunction was most recently evaluated, prior to the current learning step t. Vectors embedded in the prior subspace $\mathbf{P}$ are projected into the new subspace $\mathbf{P}'$ by multiplication with a projection matrix $\mathbf{T} = (\mathbf{P}')^T\mathbf{P}$. Vector components which point outside the subspace defined by the most recent positions and gradients are lost in this projection.

Note that the subspace $\mathbf{P}'$ lies within the subspace $\mathbf{P}$. The QR decomposition and the projection matrix $\mathbf{T}$ are thus both computed within $\mathbf{P}$, reducing the computational cost (see Section 3.1).

2.5. Choosing a Target Subfunction

The subfunction j to update in Equation 5 is chosen as,

$$j = \underset{i}{\text{argmax}}\left[\mathbf{x}^t - \mathbf{x}^{\tau_i}\right]^T\mathbf{H}^t\left[\mathbf{x}^t - \mathbf{x}^{\tau_i}\right], \tag{12}$$

where τ_i indicates the time at which subfunction i was last evaluated.

That is, the updated subfunction is the one which was last evaluated farthest from the current location, using the approximate Hessian as a metric. This is motivated by the observation that the approximating functions which were computed farthest from the current location tend to be the functions which are least accurate at the current location, and therefore the most useful to update. This contrasts with the cyclic choice of subfunction in (Blatt et al., 2007), and the random choice of subfunction in (Roux et al., 2012).

In exploratory experiments we found that choosing the function to evaluate based on distance led to better optimization. For instance, for the protein logistic regression objective in Section 4, the objective value after 25 effective passes through the data is 1.045 for SFO using the distance metric. Using the random function ordering however it is only 1.066, and using cyclic function ordering it is 1.191.

2.6. Growing the Batch Size

For many problems of the form in Equation 1, the gradient information is nearly identical between the different subfunctions early in learning. In order to achieve faster initial convergence we begin with a small number of active subfunctions. We then increment the number of subfunctions every time the average gradient shrinks to within a factor α of the standard error in the average gradient. This comparison is performed using the inverse approximate Hessian as the metric. That is, we increment the batch size by one whenever

$$\left(\bar{f}'^t\right)^T\mathbf{H}^{t-1}\bar{f}'^t < \alpha\frac{\sum_i\left(f_i'^t\right)^T\mathbf{H}^{t-1}f_i'^t}{\left(N^t-1\right)N^t}, \tag{13}$$

where N^t is the active batch size at time t, $\mathbf{H}^t$ is the full Hessian, and $\bar{f}'^t$ is the average gradient,

$$\bar{f}'^t = \frac{1}{N^t}\sum_i f_i'\left(\mathbf{x}_i^t\right). \tag{14}$$

For all the experiments shown here, $\alpha = 1$, and the initial batch size is $N^1 = 2$. The active batch size is also increased by 1 when a bad update is detected, as described in Section 2.7.

2.7. Detecting bad updates

A heuristic detects extremely bad updates, and resets $\mathbf{x}^t$ to its previous value $\mathbf{x}^{t-1}$ when they are detected. This is triggered whenever the value of a subfunction has increased since its previous evaluation, and also exceeds its predicted value by more than the reduction in the summed approximating function (ie $f_j\left(\mathbf{x}^t\right) - g^{t-1}\left(\mathbf{x}^t\right) > G^{t-1}\left(\mathbf{x}^{t-1}\right) - G^{t-1}\left(\mathbf{x}^t\right)$).

2.8. Initialization

An approximate Hessian can only be computed as described in Section 2.3 after multiple gradient evaluations. If a subfunction j only has one gradient evaluation, then its approximate Hessian $\mathbf{H}_j^t$ is set to the identity times the median eigenvalue of the average Hessian of the other active subfunctions. If j is the very first subfunction to be evaluated, $\mathbf{H}_j^t$ is initialized as the identity matrix times a large positive constant (10^6).

3. Properties

3.1. Computational Cost

The computational cost per full pass through the data for each portion of the algorithm is given in Table 1. Typically the largest terms are the $\mathcal{O}(QN)$ term from evaluating the objective and gradient for all N subfunctions, and the $\mathcal{O}\left(MN^2\right)$ term resulting from projecting positions and gradients into and out of the low dimensional subspace. This algorithm is thus suited to the case that the $\mathbf{O}(Q)$ cost of a single subfunction evaluation is larger than the $\mathbf{O}(MN)$ cost of projecting an M dimensional vector into an $\mathbf{O}(N)$ dimensional subspace. Note that N can be reduced, and the algorithmic overhead reduced, by merging subfunctions or choosing larger minibatches. Without the use of the low dimensional subspace, the leading term in the computational cost of SFO would be the far larger $\mathbf{O}\left(M^2N\right)$ per pass.

Operation	*One time cost*	*Repeats per pass*	*Cost per pass*
Function and gradient computation	$\mathcal{O}(Q)$	$\mathcal{O}(N)$	$\mathcal{O}(QN)$
Minimize $G^t(\mathbf{x})$	$\mathcal{O}\left(N^2\right)$	$\mathcal{O}(N)$	$\mathcal{O}\left(N^3\right)$
BFGS	$\mathcal{O}\left(NL^{1.4}+L^{2.4}\right)$	$\mathcal{O}(N)$	$\mathcal{O}\left(N^2L^{1.4}+NL^{2.4}\right)$
Subspace projection	$\mathcal{O}(MN)$	$\mathcal{O}(N)$	$\mathcal{O}\left(MN^2\right)$
Subspace collapse	$\mathcal{O}\left(MN^{1.4}\right)$	$\mathcal{O}(1)$	$\mathcal{O}\left(MN^{1.4}\right)$
Total			$\mathcal{O}\left(QN+MN^2+N^3+N^2L^{1.4}+NL^{2.4}\right)$

Table 1. Computational cost for components of the algorithm. Q is the cost of evaluating the objective function and gradient for a single subfunction, M is the number of data dimensions, N is the number of subfunctions, L is the number of history terms kept per subfunction. Typically, $M >> N >> L$, and the number of subfunctions can be chosen such that $Q \approx MN$.

For many problems the cost of a single subfunction evaluation is proportional to the minibatch size, $\mathbf{O}(Q) = \mathbf{O}\left(M\frac{D}{N}\right)$, where D is the size of the full data batch. In this case, the ideal minibatch size to minimize total computational cost per iteration is $N \sim \sqrt{D}$.

3.2. Convergence

Concurrent work by (Mairal, 2013) considers a similar algorithm to that described in 2.2, but with $\mathbf{H}_i^t$ a scalar constant rather than a matrix. Proposition 6.1 in (Mairal, 2013) shows that in the case that each g_i majorizes its respective f_i, and subject to some additional smoothness constraints, $G^t(\mathbf{x})$ monotonically decreases, and $\mathbf{x}^*$ is an asymptotic stationary point. Proposition 6.2 in (Mairal, 2013) further shows that for strongly convex f_i, the algorithm exhibits a linear convergence rate to $\mathbf{x}^*$.

The same convergence results hold for SFO with near-identical proofs, but requiring some modifications to the algorithm. For the proofs to hold: The eigenvalues of $\mathbf{H}_i^t$ must be bounded from above by some constant. It must be possible for g_i to majorize f_i, and $\mathbf{H}_i^t$ must be chosen so as to guarantee this majorization (eg, by addition of diagonal regularizer). The subfunction update order (Section 2.5) must be made random, rather than the current choice of the most distant subfunction in Section 2.5.

We conjecture that the convergence rate for SFO is superlinear. This is because in addition to matching function value and gradient at the current location (Equation 5), we additionally converge on the true Hessian in our approximating functions.

4. Experimental Results

We compared our optimization technique to several competing optimization techniques for several objective functions. The results are illustrated in Figure 2, and the objectives are described below. For all problems our method outperformed all other techniques in the comparison. Code generating the plots in Figure 2 is included in the Supplementary Material[3]. For all experiments we chose a number of subfunctions $N = 100$.

SFO refers to Sum of Functions Optimizer, and is the new algorithm presented in this paper. *SAG* refers to Stochastic Average Gradient method, with the trailing number providing the Lipschitz constant. *SGD* refers to Stochastic Gradient Descent, with the trailing number indicating the step size. *ADAGrad* indicates the AdaGrad algorithm, with the trailing number indicating the initial step size. *LBFGS* refers to the limited memory BFGS algorithm. *LBFGS minibatch* repeatedly chooses one tenth of the subfunctions, and runs LBFGS for ten iterations on them.

For SAG, SGD, and ADAGrad the hyperparameter was chosen by a grid search. The winning hyperparameter value, and the hyperparameter values immediately larger and smaller in the grid search, are shown in the plots and legends for each model in Figure 2. In SGD+momentum the two hyperparameters for both step size and momentum coefficient were chosen by a grid search, but only the winning parameter values are shown. The grid-searched momenta were 0.5, 0.9, 0.95, and 0.99, and the grid-searched step lengths were integer powers of ten between 10^{-3} and 10^3.

4.1. Logistic Regression

We chose the logistic regression objective, L2 regularization penalty, and training dataset to be identical to the protein homology test case in the recent Stochastic Average Gradient paper (Roux et al., 2012), to allow for direct comparison of techniques. The one difference is that our total objective function is divided by the number of samples per minibatch, but unlike in (Roux et al., 2012) is not also divided by the number of minibatches. This different scaling places the hyperparameters for logistic regression in the

[3]All figures in the paper can be reproduced simply by downloading code and training data, typing "python figures.py", and then waiting a week for all optimizers to run on all objective functions for all hyperparameters.

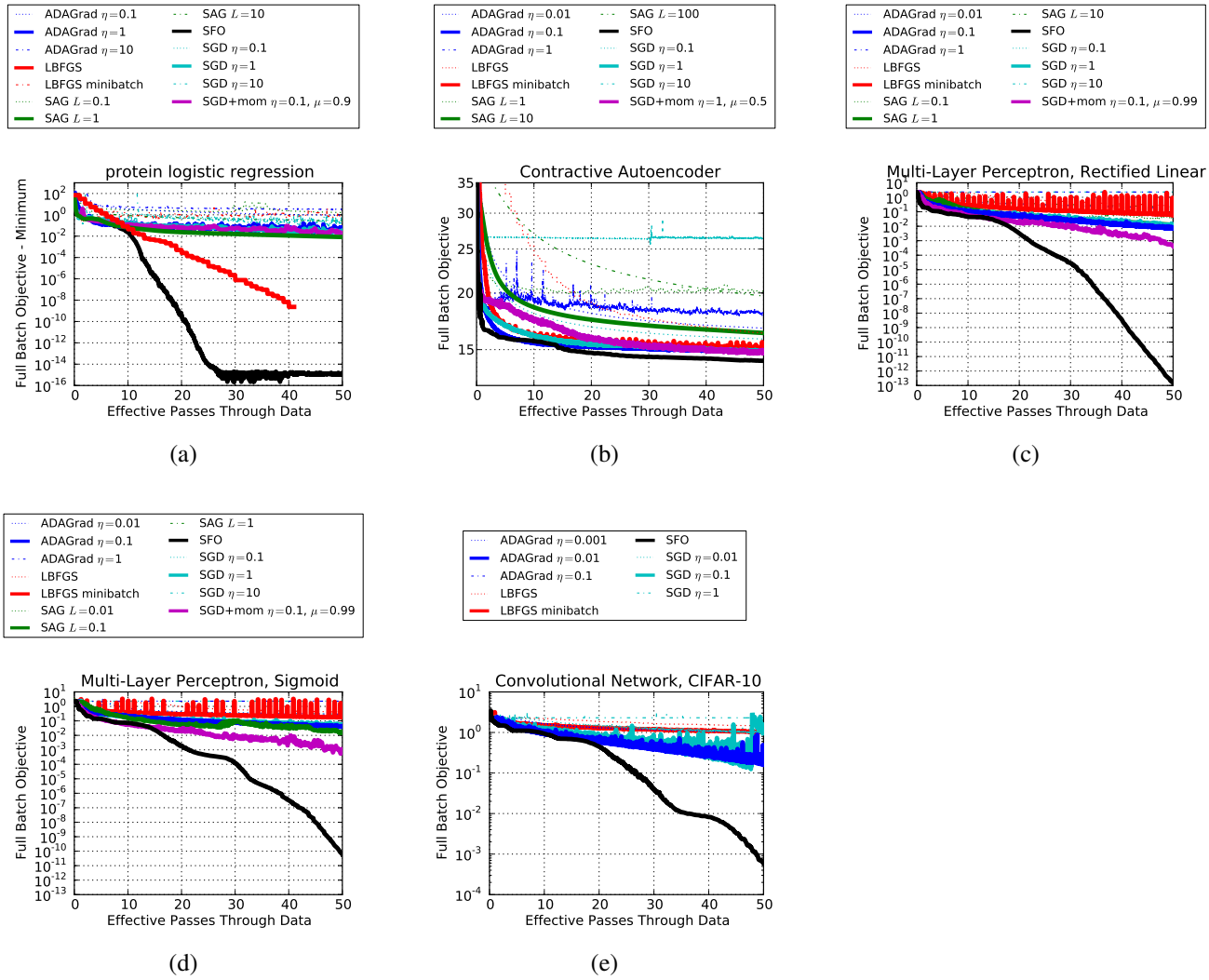

Figure 2. A comparison of the proposed Sum of Functions Optimizer (SFO) to competing techniques for several objective functions. The competing optimization techniques are described in Section 4. The bold lines indicate the best performing optimizer within each class. The objective functions shown are *(a)* a logistic regression problem, *(b)* a contractive autoencoder trained on MNIST digits, *(c)* a multi-layer perceptron with rectified linear units trained on MNIST digits, *(d)* a multi-layer perceptron with sigmoidal units trained on MNIST digits, and *(e)* a multilayer convolutional network trained on CIFAR-10.

same range as for our other experiments.

4.2. Autoencoder

We trained a contractive autoencoder, which penalizes the Frobenius norm of the Jacobian of the encoder function, on MNIST digits. Autoencoders of this form have been successfully used for learning deep representations in neural networks (Rifai et al., 2011). Sigmoid nonlinearities were used for both encoder and decoder. The regularization penalty was set to 1, and did not depend on the number of hidden units. The reconstruction error was divided by the number of training examples. There were 784 visible units, and 256 hidden units.

4.3. Multilayer Perceptron

We trained a deep neural network to classify digits on the MNIST digit recognition benchmark. We used a similar architecture as (Hinton & Srivastava, 2012), but with a smaller number of units to allow all competing optimizers time to run. Our network consisted of: 784 input units, one hidden layer of 120 units, one hidden layer of 12 units, and 10 output units. We ran the experiment using both rectified linear and sigmoidal units. The objective used was the standard softmax regression on the output units.

4.4. Deep Convolutional Network

We trained a deep convolutional network on CIFAR-10 using max pooling and rectified linear units. The architecture we experimented with contains two convolutional layers containing 48 and 128 units respectively, followed by one fully connected layer of 240 units. (this plot will be updated to include the $SGD + momentum$ optimizer when the corresponding grid search over hyperparameters completes)

5. Future Directions

We perform optimization in an $\mathcal{O}(N)$ dimensional subspace. It may be possible, however, to drastically reduce the dimensionality of the active subspace without significantly reducing optimization performance. For instance, the subspace could be determined by accumulating, in an online fashion, the leading eigenvectors of the covariance matrix of the gradients of the subfunctions, as well as the leading eigenvectors of the covariance matrix of update steps. This would allow the algorithm to run more quickly even for large numbers of subfunctions, and also reduce memory requirements.

Most portions of the presented algorithm are naively parallelizable. The $g_i^t(\mathbf{x})$ functions can be updated asynchronously, and can even be updated using old position information. Therefore, developing a parallelized version of this algorithm could make it a useful tool for massive scale optimization problems. Similarly, it may be possible to adapt this algorithm to an online / infinite data context by cycling through a finite set of active subfunctions.

Quadratic functions are often a poor match to the geometry of the objective function (Pascanu et al., 2012). Neither the dynamically updated subspace nor the use of independent approximating subfunctions $g_i^t(\mathbf{x})$ which are fit to the true subfunctions $f_i(\mathbf{x})$ depend on the functional form of $g_i^t(\mathbf{x})$. Exploring non-quadratic approximating subfunctions has the potential to greatly improve performance.

Section 2.3.2 initializes the approximate Hessian using a diagonal matrix. Instead, it might be effective to initialize the approximate Hessian for each subfunction using the average approximate Hessian from all other subfunctions. Where the measurements from a single subfunction disagreed with this initialization they would overwrite it. This would take advantage of the fact that the Hessians for different subfunctions are very similar for many objective functions.

Finally, the natural gradient (Amari, 1998) can greatly accelerate optimization by removing the effect of dependencies and relative scalings between parameters. The natural gradient can be simply combined with other optimization methods by performing a change of variables, such that in the new parameter space the natural gradient and the ordinary gradient are identical (Sohl-Dickstein, 2012). It should be straightforward to incorporate this change-of-variables technique into SFO.

References

Amari, Shun-Ichi. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998. ISSN 08997667. doi: 10.1162/089976698300017746. URL http://www.maths.tcd.ie/~mnl/store/Amari1998a.pdfhttp://www.mitpressjournals.org/doi/abs/10.1162/089976698300017746.

Blatt, Doron, Hero, Alfred O, and Gauchman, Hillel. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007. URL http://epubs.siam.org/doi/abs/10.1137/040615961.

Bordes, Antoine, Bottou, Léon, and Gallinari, Patrick. SGD-QN: Careful quasi-Newton stochastic gradient descent. *The Journal of Machine Learning Research*, 10:1737–1754, 2009. URL http://dl.acm.org/citation.cfm?id=1755842.

Bottou, L. Stochastic gradient learning in neu-

ral networks. *leon.bottou.org*, 1991. URL http://leon.bottou.org/publications/pdf/nimes-1991.pdf.

Boyd, S P and Vandenberghe, L. *Convex optimization*. Cambridge Univ Press, 2004. ISBN 0521833787.

Byrd, RH Richard H, Chin, GM Gillian M, Neveitt, Will, and Nocedal, Jorge. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011. URL http://epubs.siam.org/doi/abs/10.1137/10079923X.

Dennis Jr, John E and Moré, Jorge J. Quasi-Newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977. URL http://epubs.siam.org/doi/abs/10.1137/1019005.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.pdf.

Hinton, GE and Srivastava, N. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv: ...*, 2012. URL http://arxiv.org/abs/1207.0580.

Huber, PJ. Robust statistics. *Wiley, New York*, 1981. URL http://scholar.google.com/scholar?hl=en&q=P.J.+Huber%2C+Robust+Statistics%2C+Wiley%2C+New+York%2C+1981.&btnG=&as_sdt=1%2C5&as_sdtp=#0.

Lin, Chih-Jen, Weng, Ruby C, and Keerthi, S Sathiya. Trust region newton method for logistic regression. *The Journal of Machine Learning Research*, 9:627–650, 2008. URL http://dl.acm.org/citation.cfm?id=1390703.

Liu, Dong C DC and Nocedal, Jorge. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. URL http://link.springer.com/article/10.1007/BF01589116.

Mairal, J. Optimization with First-Order Surrogate Functions. *arXiv preprint arXiv:1305.3120*, 2013. URL http://arxiv.org/abs/1305.3120.

Martens, James. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, volume 951, pp. 2010, 2010. URL http://www.cs.toronto.edu/~asamir/cifar/HFO_James.pdf.

Ngiam, J and Coates, A. On optimization methods for deep learning. *... of the 28th ...*, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Le_210.pdf.

Papakonstantinou, JM. *Historical Development of the BFGS Secant Method and Its Characterization Properties*. 2009. URL http://books.google.com/books?hl=en&lr=&id=wcdyxoPy2A0C&oi=fnd&pg=PR2&dq=+Historical+Development+of+the+BFGS+Secant+Method+and+Its+Characterization+Properties&ots=kFzPiQq_d9&sig=4l5TyaeBdBmfsIhdcWrHtbTfIdw.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training Recurrent Neural Networks. November 2012. URL http://arxiv.org/abs/1211.5063.

Rifai, Salah, Vincent, Pascal, Muller, Xavier, Glorot, Xavier, and Bengio, Yoshua. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 833–840, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Rifai_455.pdf.

Robbins, Herbert and Monro, Sutton. A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407, 1951. URL http://www.jstor.org/stable/10.2307/2236626.

Roux, N Le, Schmidt, M, and Bach, F. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. 2012. URL http://hal.inria.fr/view_by_stamp.php?label=TESTINRIA&langue=fr&action_todo=view&id=hal-00674995.

Schraudolph, Nicol, Yu, Jin, and Günter, Simon. A stochastic quasi-Newton method for online convex optimization. 2007. URL http://eprints.pascal-network.org/archive/00003992/.

Schraudolph, Nicol N. Local gain adaptation in stochastic gradient descent. In *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, volume 2, pp. 569–574. IET, 1999. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=817990.

Sohl-Dickstein, Jascha. The Natural Gradient by Analogy to Signal Whitening, and Recipes and Tricks for its Use. *arXiv:1205.1828v1*, May 2012. URL http://arxiv.org/abs/1205.1828.

Sohl-Dickstein, Jascha, Battaglino, Peter B., and DeWeese, Michael R. Minimum Probability Flow Learning. *International Conference on Machine Learning*, 107(22):11–14, November 2011. ISSN 0031-9007. doi: 10.1103/PhysRevLett.107.220601. URL http://link.aps.org/doi/10.1103/PhysRevLett.107.220601http://www.icml-2011.org/papers/480_icmlpaper.pdf.

Sunehag, Peter, Trumpf, Jochen, Vishwanathan, S V N, and Schraudolph, Nicol. Variable metric stochastic approximation theory. *arXiv preprint arXiv:0908.3529*, August 2009. URL http://arxiv.org/abs/0908.3529.

Vinyals, Oriol and Povey, Daniel. Krylov subspace descent for deep learning. *arXiv preprint arXiv:1111.4259*, 2011. URL http://arxiv.org/abs/1111.4259.