Comparison of Modern Stochastic Optimization Algorithms

George Papamakarios

December 2014

Abstract

Gradient-based optimization methods are popular in machine learning applications. In large-scale problems, stochastic methods are preferred due to their good scaling properties. In this project, we compare the performance of four gradient-based methods; gradient descent, stochastic gradient descent, semi-stochastic gradient descent and stochastic average gradient. We consider logistic regression with synthetic data and softmax regression on the MNIST dataset of handwritten digits.

1 Introduction

In machine learning, statistics and data science, one often needs to solve an unconstrained optimization problem of the following type

$$\min_{\mathbf{w}} f(\mathbf{w}) \equiv \frac{1}{N} \sum_{n=1}^{N} f_n(\mathbf{w}) \tag{1.1}$$

where $\mathbf{w} \in \mathbb{R}^D$ and f is given by the average of N functions $f_n : \mathbb{R}^D \to \mathbb{R}$. In a machine learning context, $\mathbf{w}$ could be the parameters of a classifier we wish to learn, and f_n could be the loss on the n^{th} data point in a train set of N data points. Solving the above problem would correspond to fitting the classifier to the train set by minimizing the average loss.

In the common case where functions f_n are smooth, iterative gradient-based methods, such as gradient descent, are popular optimization tools. In principle, such methods need access to the gradient of f, given by

$$\nabla f(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \nabla f_n(\mathbf{w}). \tag{1.2}$$

Evaluating ∇f scales linearly with the number of functions N and becomes computationally expensive as N becomes large. As datasets grow bigger, such methods tend to become inefficient in practical machine learning applications.

A viable alternative to gradient-based methods are stochastic gradient-based methods, such as stochastic gradient descent. These methods do not fully compute ∇f, but rather approximate it by a stochastic estimate $\nabla \tilde{f}$ that is cheaper to compute. The estimate is typically computed by using only a subset of functions f_n. The main drawback of stochastic gradient-based methods is that, due to their stochastic nature, their rate of convergence can be significantly worse than that of their deterministic counterparts.

In the literature, there has been a significant number of approaches towards accelerating the convergence of stochastic gradient-based methods (including for instance adding momentum or averaging the gradients—see relevant discussion in [10] for more information). Recently, a family of semi-stochastic gradient-based methods, such as semi-stochastic gradient descent and stochastic average gradient, have been proposed, that combine elements from both deterministic and stochastic gradient-based methods. As a result, they inherit both the fast convergence of deterministic methods and the computational efficiency of stochastic methods.

In this report, we compare two classic gradient-based methods, a deterministic one and a stochastic one, and two modern semi-stochastic methods, based on their performance in practical

machine learning scenarios. We perform two sets of experiments; the first one involves logistic regression on a synthetically generated dataset and the second one involves classification with softmax regression of the handwritten digits in the MNIST dataset.

2 Gradient-Based Optimization Algorithms

In our comparisons, we used (a) gradient descent, a classic deterministic algorithm, (b) stochastic gradient descent, a classic stochastic algorithm, (c) semi-stochastic gradient descent and (d) stochastic average gradient, the last two being modern semi-stochastic algorithms. In this section, we give brief descriptions of the above algorithms and provide relevant references.

2.1 Gradient Descent

Gradient descent (or simply GD) is a classic iterative gradient-based algorithm. In iteration k, variable $\mathbf{w}_k$ is updated towards the opposite direction of the gradient $\nabla f(\mathbf{w}_k)$, which is (locally) the direction towards which $f(\mathbf{w}_k)$ decreases the most quickly (hence its alternative name, steepest descent). A step size parameter α determines the degree by which $\mathbf{w}_k$ is changed. Algorithm 2.1 describes the version of GD where α is the same in each iteration (this is the version we use in our experiments). Figure 2.1a shows a typical run of the algorithm for a strongly convex function f.

Algorithm 2.1: Gradient descent with constant step size

Input: step size $\alpha > 0$, $\mathbf{w}_0$
1 **for** $k \in \{0, 1, \ldots\}$ **do**
2 $\quad \mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \nabla f(\mathbf{w}_k)$
3 **end for**
Output: $\mathbf{w}_k$

This determined Gradient descent is based on all datasets to update the weights of model. But for each step to update parameters, it have to compute all datasets with their prediction and true labels, so for each iteration, model needs to compute so many losses when n grows bigger

It can be shown (Theorem 2.1.15 in [9]) that if ∇f is Lipschitz continuous and f is strongly convex, then, for an appropriate selection of α, GD has a linear convergence rate, in the sense that

$$f(\mathbf{w}_k) - f(\mathbf{w}^*) \in \mathcal{O}(c^k) \tag{2.1}$$

where $\mathbf{w}^*$ is the global minimiser of f and $c < 1$ depends on the condition number of f (the higher the condition number, the closest c is to 1 and the slowest the convergence becomes). This type of convergence is referred to as linear because the logarithm of the optimality gap $f(\mathbf{w}_k) - f(\mathbf{w}^*)$ decays linearly with k.

Despite its good convergence properties, when applied to problem (1.1), GD has to do work $\mathcal{O}(N)$ per iteration in order to compute ∇f, which can be prohibitively expensive for large N. Therefore, its applicability in large-scale machine learning is rather limited.

2.2 Stochastic Gradient Descent

Stochastic gradient descent (or simply SGD) is similar in principle to GD but instead of computing the full gradient $\nabla f(\mathbf{w}_k)$, it simply chooses one of the gradients $\nabla f_n(\mathbf{w}_k)$ uniformly at random and uses that to update $\mathbf{w}_k$. Even though this may move $\mathbf{w}_k$ away from the minimum, it works in expectation, since

$$\mathbb{E}[\nabla f_n(\mathbf{w}_k)] = \nabla f(\mathbf{w}_k). \tag{2.2}$$

SGD is described in Algorithm 2.2. Note however that in many practical implementations (including ours), n in step 3 is chosen such that the algorithm considers all gradients before choosing the same one again. Figure 2.1b shows a typical run of SGD for a strongly convex function f.

An important aspect of SGD is that $\nabla f_n(\mathbf{w}_k)$ is a noisy estimate of the true gradient. Even at the optimum where the true gradient is zero, the algorithm might continue to change $\mathbf{w}_k$.

Algorithm 2.2: Stochastic gradient descent

Input: initial step size $\alpha_0 > 0$, decay parameter $\lambda > 0$, $\mathbf{w}_0$
for $k \in \{0, 1, \ldots\}$ **do**
  $\alpha_k = \alpha_0 (1 + \alpha_0 \lambda k)^{-1}$
  choose $n \in \{1, 2, \ldots, N\}$ uniformly at random
  $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla f_n (\mathbf{w}_k)$
end for
Output: $\mathbf{w}_k$

SGD is based on random choosen data to compute loss and update the model parameters, but with iteration goes on, then we have to decrease the step for decreasing parameters(that is learning rate will be smaller with steps go on). SGD will give more noise for updating weights, even go to global minimization, it will also update parameters. So it may not converge to global min.

Therefore, in order to ensure convergence, the step size α_k needs to decrease in every iteration. According to [2], when the Hessian of f at the optimum is strictly positive definite, decreasing α_k as in Algorithm 2.2 guarantees convergence at the best possible rate, which is

$$\mathbb{E}[f(\mathbf{w}_k)] - f(\mathbf{w}^*) \in \mathcal{O}(k^{-1}) \tag{2.3}$$

We can see that SGD has a significantly slower convergence rate than GD. Notice also that the convergence is only in expectation. However, since the work in each iteration is independent of N, SGD is very attractive in large-scale machine learning applications.

2.3 Semi-Stochastic Gradient Descent

Semi-stochastic gradient descent (also denoted as S2GD) was proposed in 2013 by Konečný and Richtárik [6]. It combines elements from both GD and SGD (hence its name). S2GD starts by computing the full gradient once and then proceeds with stochastic updates by choosing one of the gradients at a time. If the full gradient was computed at $\mathbf{w}_0$, after i stochastic updates the gradient estimate is taken to be

$$\nabla \tilde{f}(\mathbf{w}_i) = \nabla f(\mathbf{w}_0) - \nabla f_n(\mathbf{w}_0) + \nabla f_n(\mathbf{w}_i) \tag{2.4}$$

for some uniformly random choice of n, and $\mathbf{w}_i$ is subsequently updated using $\nabla \tilde{f}(\mathbf{w}_i)$. As in SGD, the gradient estimate is an unbiased estimate of the true gradient, since

$$\mathbb{E}[\nabla f(\mathbf{w}_0) - \nabla f_n(\mathbf{w}_0) + \nabla f_n(\mathbf{w}_i)] = \nabla f(\mathbf{w}_0) - \nabla f(\mathbf{w}_0) + \nabla f(\mathbf{w}_i) = \nabla f(\mathbf{w}_i). \tag{2.5}$$

However, unlike SGD, in every stochastic update the true gradient at $\mathbf{w}_0$ is also used, effectively reducing the noise of the estimate and speeding up convergence. After a (stochastic) number t of stochastic updates, the algorithm computes again the full gradient at $\mathbf{w}_t$, and so on. Algorithm 2.3 describes S2GD in detail and Figure 2.1c shows an typical run.

Algorithm 2.3: Semi-stochastic gradient descent

Input: step size $\alpha > 0$, maximum # of inner iterations $T > 0$, $\lambda \geq 0$, $\mathbf{w}_0$
for $k \in \{0, 1, \ldots\}$ **do**
  $\mathbf{g} \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla f_n(\mathbf{w}_k)$
  $\mathbf{y} \leftarrow \mathbf{w}_k$
  choose $t \in \{1, 2, \ldots, T\}$ with probability $\propto (1 - \lambda\alpha)^{-t}$
  for $i \in \{1, 2, \ldots, t\}$ **do**
    choose $n \in \{1, 2, \ldots, N\}$ uniformly at random
    $\mathbf{y} \leftarrow \mathbf{y} - \alpha [\mathbf{g} - \nabla f_n(\mathbf{y}) + \nabla f_n(\mathbf{w}_k)]$
  end for
  $\mathbf{w}_{k+1} \leftarrow \mathbf{y}$
end for
Output: $\mathbf{w}_k$

Semi-SGD is based on GD and SGD, for first iteration, model will first compute Loss based on all data, then update parameters, then use SGD for n steps for updating model params, then it will go also with GD based on all data, then will again to do SGD for random choosen data to update. Semi-SGD converges faster than GD.

If all ∇f_n are Lipschitz continuous and f is strongly convex, then it can be proven [6] that, for an appropriate choice of its parameters, S2GD converges linearly—in expectation—to the optimum with the number of outermost iterations, that is

$$\mathbb{E}\left[f\left(\mathbf{w}_k\right)\right] - f\left(\mathbf{w}^*\right) \in \mathcal{O}\left(c^k\right) \tag{2.6}$$

for some $c < 1$ which depends on the condition number of f. In other words, S2GD inherits the fast convergence of GD. Unlike GD though, after computing a full gradient, S2GD also performs a number of stochastic updates, which leads to better performance for large N. Compared to SGD, its major weakness is that it still has to wait for at least a full gradient evaluation before making a single update, by which time SGD will have made N stochastic updates already.

Finally, it is worth mentioning that if the λ parameter of S2GD is set to 0, then S2GD reduces to stochastic variance reduced gradient, a method proposed also in 2013 by Johnson and Zhang [4].

2.4 Stochastic Average Gradient

Stochastic average gradient (or simply SAG) was proposed in 2012 by Roux et al. [10]. Similarly to SGD, SAG chooses in iteration k some gradient $\nabla f_n\left(\mathbf{w}_k\right)$ uniformly at random. However, it also remembers the most recently computed values for all gradients other than $\nabla f_n\left(\mathbf{w}_k\right)$, and uses the following gradient estimate to update $\mathbf{w}_k$

$$\nabla \tilde{f}\left(\mathbf{w}_k\right) = \frac{1}{N}\left[\sum_{n' \neq n} \nabla f_{n'}\left(\mathbf{w}_{k'}\right) + \nabla f_n\left(\mathbf{w}_k\right)\right] \tag{2.7}$$

where $k' < k$ is the iteration in which $\nabla f_{n'}$ was most recently evaluated. Algorithm 2.4 describes SAG in detail and Figure 2.1d shows an typical run.

Algorithm 2.4: Stochastic average gradient

Input: step size $\alpha > 0$, $\mathbf{w}_0$
initialize $\mathbf{g}_n = \mathbf{0}$ for all $n \in \{1, 2, \ldots, N\}$
for $k \in \{0, 1, \ldots\}$ **do**
  choose $n \in \{1, 2, \ldots, N\}$ uniformly at random
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \frac{\alpha}{N}\left[\sum_{n' \neq n} \mathbf{g}_{n'} + \nabla f_n\left(\mathbf{w}_k\right)\right]$
  $\mathbf{g}_n \leftarrow \nabla f_n\left(\mathbf{w}_k\right)$
end for
Output: $\mathbf{w}_k$

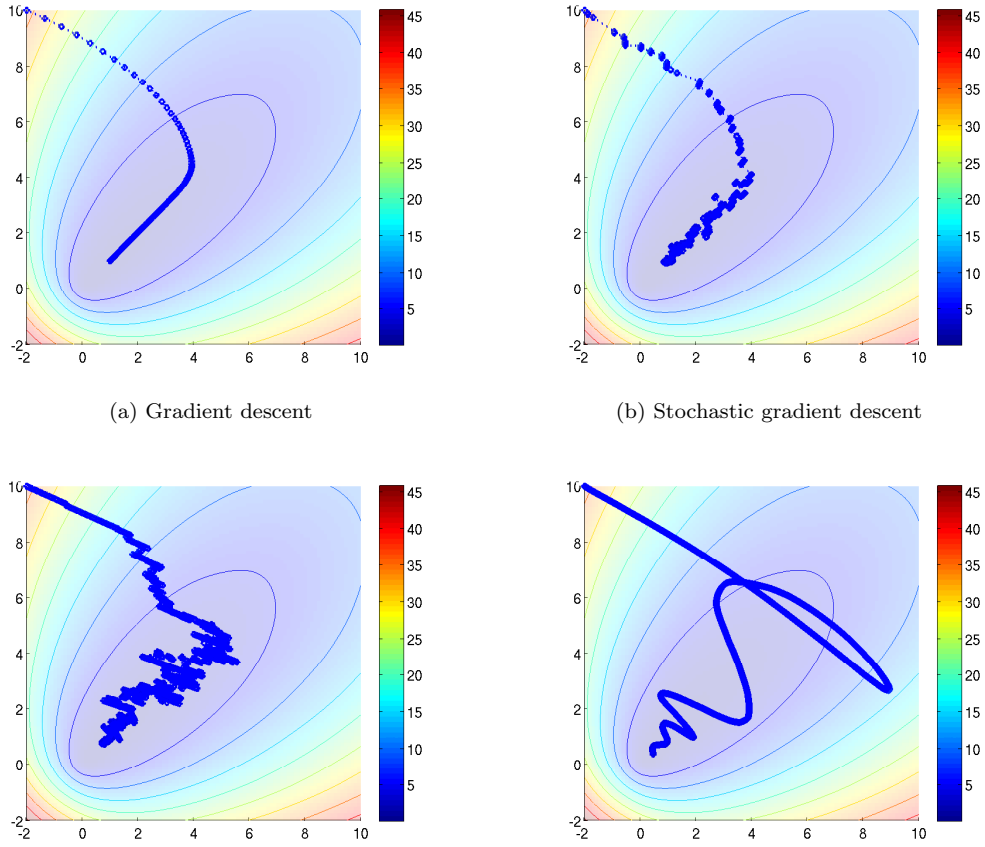
SAG is also based on SGD and GD, but not to update weights during all data, it uses SGD to get one iteration weights, but not to update weights now, it stores the weights and average the weights for n steps, then this is one iteration for model iteration.
So mean idea is store weight in memory, then average all weight with one updated weight to original weights.

If all ∇f_n are Lipschitz continuous and f is strongly convex, then it can be proven [10] that, for some appropriate choice of step size α, SAG converges linearly—in expectation—to the optimum in the sense that

$$\mathbb{E}\left[\left\|\mathbf{w}_k - \mathbf{w}^*\right\|^2\right] \in \mathcal{O}\left(c^k\right) \tag{2.8}$$

for some $c < 1$ which depends on the condition number of f and the number of functions N. It therefore achieves similar convergence rate as GD and S2GD but, unlike those algorithms, it only requires a single gradient evaluation per iteration (same as SGD). In fact, as it was pointed out in [10], SAG was the first ever gradient-based method to achieve linear convergence at an iteration cost similar to SGD. However, its major drawback is its memory cost of $\mathcal{O}(N)$, which can make it impractical for large N.

Finally, it is worth mentioning that SAG can be viewed as the randomized version of incremental aggregate gradient (or simply IAG), a method proposed in 2007 by Blatt et al. [1]. Unlike SAG which chooses a gradient uniformly at random, IAG cycles deterministically over gradients. However, according to [10] (and our experience also confirm this), SAG can be used with step sizes for which IAG may diverge. In other words, SAG can tolerate a larger step size than IAG without becoming unstable.



(a) Gradient descent

(b) Stochastic gradient descent

(c) Semi-stochastic gradient descent

(d) Stochastic average gradient

Figure 2.1: Typical convergent runs of each optimization algorithm for a strongly convex function f where $N = 1000$ and $\mathbf{w} \in \mathbb{R}^2$.

2.5 Mini-Batch Versions of Stochastic Algorithms

The stochastic algorithms, i.e. SGD, S2GD and SAG, can be easily modified to work on mini-batches instead of individual functions. In this context, a mini-batch is a subset of functions f_n of some predetermined fixed size $M < N$. In every stochastic update, instead of choosing a single gradient ∇f_n, the stochastic algorithm can be extended to randomly choose a whole mini-batch instead. Let the chosen mini-batch consist of gradients $\nabla f_{n_1}, \nabla f_{n_2}, \ldots, \nabla f_{n_M}$. Then, the following gradient is used in the update of $\mathbf{w}_k$

$$\nabla f_B(\mathbf{w}_k) = \frac{1}{M} \sum_{m=1}^{M} \nabla f_{n_m}(\mathbf{w}_k). \tag{2.9}$$

Of course, for $M = 1$ we recover the original formulation of the algorithms.

It is easy to see that for $M > 1$, ∇f_B is a more reliable estimate of the full gradient that any single gradient ∇f_n. Nevertheless, by the time a mini-batch algorithm makes a single update, the original algorithm would have made M updates that, in expectation, move towards the right direction. It is therefore not obvious whether large mini-batches are advantageous. In fact, Hinton [3] refers to the use of large mini-batches with SGD as "a serious mistake".

There is however a significant potential advantage in using mini-batches, since computing the M gradients in (2.9) can be easily vectorized or parallelized. Therefore, on a parallel architecture,

computing the mini-batch gradient ∇f_B can cost much less that M single gradient evaluations, which makes the use of mini-batches particularly attractive.

3 Experimental Evaluation

In this section, we compare the performance of the four optimization algorithms described in the previous section in practical machine learning tasks, namely (a) logistic regression with synthetic data and (b) softmax regression with the MNIST dataset of handwritten digits.

3.1 Implementation Details

Each optimization algorithm has a number of parameters to set, which affect its convergence and performance. All algorithms have a step size parameter α (α_0 for SGD). In addition, SGD has a decay parameter λ and S2GD has a parameter T controlling the maximum number of stochastic updates per epoch and a non-negative parameter λ.

For SGD, in our implementation we set λ to be equal to the regularization parameter of the problem (logistic or softmax regression), as suggested in [2]. For S2GD, we set T equal to N (such that the number of stochastic updates is of the same order as the size of the data) and we set λ to be equal to the regularization parameter of the problem. According to [6], if f is μ-strongly convex, then parameter λ of S2GD should be a lower bound on μ. By choosing λ to be the regularization parameter, this will always be the case. Finally, for each algorithm, we manually tuned the step size parameter independently so as to maximize convergence speed. The particular step sizes used will be reported in each individual experiment.

All four optimization algorithms, as well as logistic and softmax regression, were implemented in MATLAB [8].

3.2 Logistic Regression on Synthetic Data

In this experiment, we generated N data points $(\mathbf{x}_n, y_n)$ for $n \in \{1, 2, \ldots, N\}$, where $\mathbf{x}_n \in \mathbb{R}^{100}$ is an input feature vector and $y_n \in \{-1, 1\}$ is its class label. The feature vectors $\mathbf{x}_n$ were independently generated from a Gaussian distribution with zero mean and symmetric covariance, in particular

$$P(\mathbf{x}_n) = \mathcal{N}(\mathbf{x}_n | \mathbf{0}, 20\mathbf{I}). \tag{3.1}$$

For each $\mathbf{x}_n$, the class label y_n was generated by a logistic regression model, as follows

$$P(y_n | \mathbf{x}_n, \mathbf{w}) = \frac{1}{1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)} \tag{3.2}$$

where we selected $\mathbf{w} = \frac{1}{2}\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T \in \mathbb{R}^{100}$.

Pretending not to know the value of $\mathbf{w}$ that generated the data, our objective is to fit a regularized logistic regression model to the generated data, i.e. to minimize the following objective function

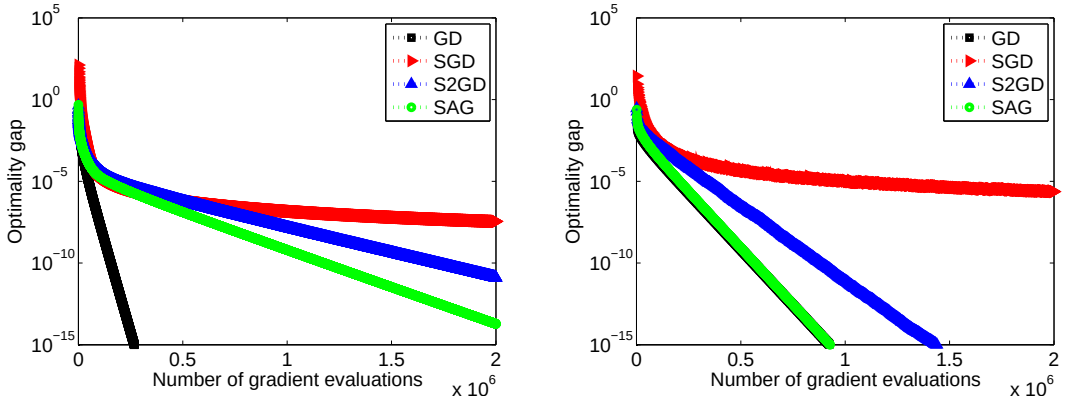
$$f(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \log\left[1 + \exp\left(-y_n \mathbf{w}^T \mathbf{x}_n\right)\right] + \frac{\lambda}{2} \|\mathbf{w}\|^2. \tag{3.3}$$

For our experiments we selected $\lambda = 0.1$. Minimizing f can be formulated as in problem (1.1) by identifying

$$f_n(\mathbf{w}) = \log\left[1 + \exp\left(-y_n \mathbf{w}^T \mathbf{x}_n\right)\right] + \frac{\lambda}{2} \|\mathbf{w}\|^2. \tag{3.4}$$

It can easily be shown that all ∇f_n are Lipschitz continuous and that f is λ-strongly convex, therefore it has a unique global minimum. However, no closed-from solution for finding the global minimum exists, therefore one needs to resort to numeric optimization instead.

Figure 3.1 shows the convergence behaviour of each optimization algorithm for N ranging from 10^2 to 10^5. As expected, the convergence of GD, S2GD and SAG is linear, whereas the



(a) $N = 10^2$ (b) $N = 10^3$

(c) $N = 10^4$ (d) $N = 10^5$

Figure 3.1: Convergence behaviour of each optimization algorithm in fitting logistic regression for various numbers of data points. The horizontal axis corresponds to the number of times a gradient ∇f_n needed to be evaluated. The vertical axis shows the optimality gap $f(\mathbf{w}_k) - f(\mathbf{w}_*)$. The optimality gap was measured after every epoch, i.e. a full pass over the dataset. All algorithms were initialised at $\mathbf{w}_0 = \mathbf{0}$. The step sizes used were $\alpha = 0.05$ for GD, $\alpha_0 = 0.1$ for SGD, $\alpha = 10^{-4}$ for S2GD and $\alpha = 5 \times 10^{-5}$ for SAG.

Figure 3.2: Examples of images of handwritten digits from the MNIST dataset. White corresponds to a pixel value of 1 and black corresponds to a pixel value of 0.

convergence of SGD is sublinear. For small N (i.e. 10^2) GD is by far the fastest, however, as N becomes larger, GD becomes increasingly inefficient, since it needs a full pass over the dataset for each update of $\mathbf{w}$. We can easily see that stochastic methods have an advantage over GD for large N. Unlike SGD though, whose convergence is rather slow, S2GD and SAG are capable of reaching very high precision (e.g. 10^{-15}) despite their stochastic nature. Between S2GD and SAG, the latter appears to have an advantage in this experiment.

3.3 Softmax Regression on the MNIST Dataset of Handwritten Digits

The MNIST dataset [7] consists of greyscale images of handwritten digits. The resolution of the images is 28×28, i.e. a total of 784 pixels. The dataset is divided into a train set of 60000 images and a test set of 10000 images. Figure 3.2 shows examples of images from the dataset.

In this experiment, we use the train set of $N = 60000$ images to fit a regularized softmax regression model for digit classification. Let $\mathbf{x}_n \in \mathbb{R}^{784}$ for $n \in \{1, 2, \ldots, N\}$ be an image in the train set and let $y_n \in \{1, 2, \ldots, 10\}$ be the digit that it represents. Given a set of weights $\mathbf{w} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{10}\}$, the softmax regression model predicts that the probability of y_n being digit ℓ is

$$P(y_n = \ell | \mathbf{x}_n, \mathbf{w}) = \frac{\exp\left(\mathbf{w}_\ell^T \mathbf{x}_n\right)}{\sum_{\ell'=1}^{10} \exp\left(\mathbf{w}_{\ell'}^T \mathbf{x}_n\right)} \tag{3.5}$$

In order to fit the regularized softmax regression model to the train set, one needs to minimize the following function with respect to $\mathbf{w}$

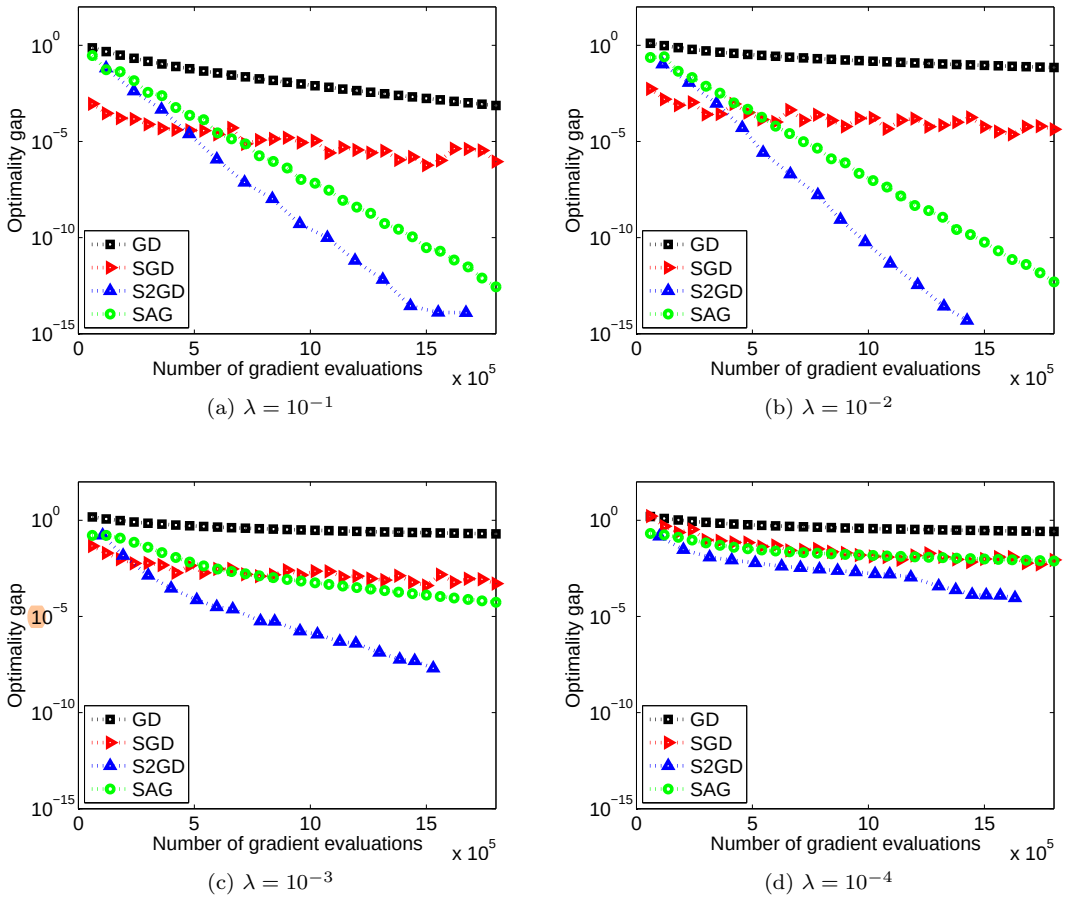
$$f(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{\ell=1}^{10} I(y_n = \ell) \log P(y_n = \ell | \mathbf{x}_n, \mathbf{w}) + \frac{\lambda}{2} \sum_{\ell=1}^{10} \|\mathbf{w}_\ell\|^2 \tag{3.6}$$

where $I(\cdot)$ is the indicator function, i.e. 1 if its argument is true, 0 otherwise. The above can be reformulated as in equation (1.1) by identifying

$$f_n(\mathbf{w}) = -\sum_{\ell=1}^{10} I(y_n = \ell) \log P(y_n = \ell | \mathbf{x}_n, \mathbf{w}) + \frac{\lambda}{2} \sum_{\ell=1}^{10} \|\mathbf{w}_\ell\|^2. \tag{3.7}$$

It is not difficult to show that all ∇f_n are Lipschitz continuous and f is λ-strongly convex, therefore there exists a unique global minimizer. However, as it is the case with logistic regression, there is no closed-form solution for optimizing f, therefore numeric optimization needs to be used instead.

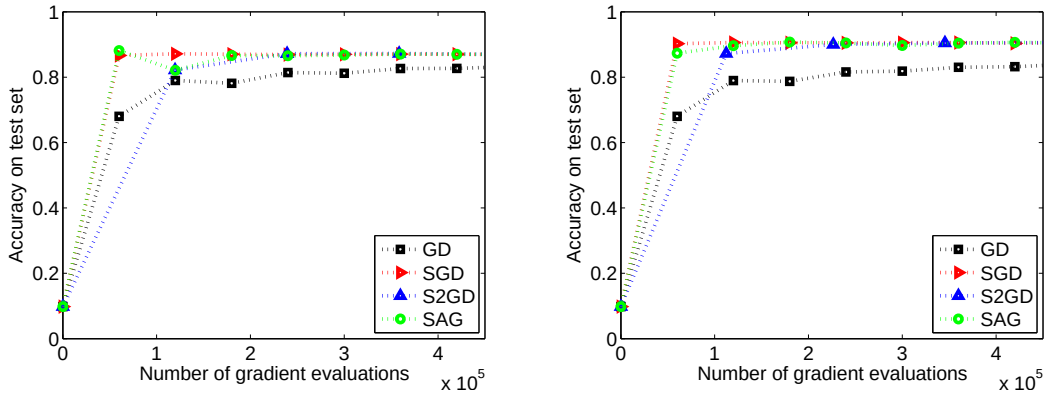
Figure 3.3 shows the convergence behaviour of each algorithm for different values of the regularization parameter λ ranging from 10^{-1} to 10^{-4}. Note that as λ decreases, the condition number of the problem increases, since λ is a lower bound on the smallest eigenvalue of the Hessian of f. As a result, this slows down the convergence of all algorithms, since they are all gradient-based methods. Other than that, the convergence pattern is similar to that of the logistic regression experiment. GD is the least efficient, since N is quite large. SGD has the



(a) $\lambda = 10^{-1}$ (b) $\lambda = 10^{-2}$

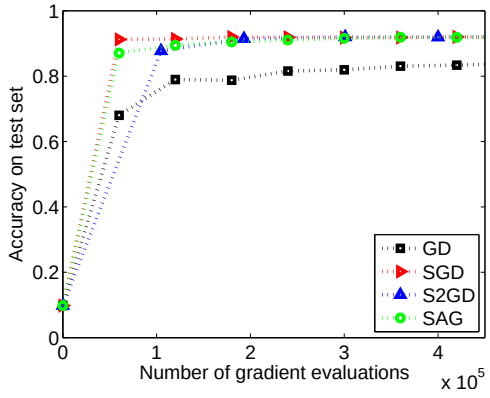
(c) $\lambda = 10^{-3}$ (d) $\lambda = 10^{-4}$

Figure 3.3: Convergence behaviour of each optimization algorithm in fitting regularized softmax regression for various regularization parameters λ. The horizontal axis corresponds to the number of times a gradient ∇f_n needed to be evaluated. The vertical axis shows the optimality gap $f(\mathbf{w}_k) - f(\mathbf{w}_*)$. The optimality gap was measured after every epoch, i.e. a full pass over the dataset. All algorithms were initialized at $\mathbf{w}_0 = \mathbf{0}$. The step sizes used were $\alpha = 0.5$ for GD, $\alpha_0 = 1$ for SGD, $\alpha = 0.01$ for S2GD and $\alpha = 0.001$ for SAG.

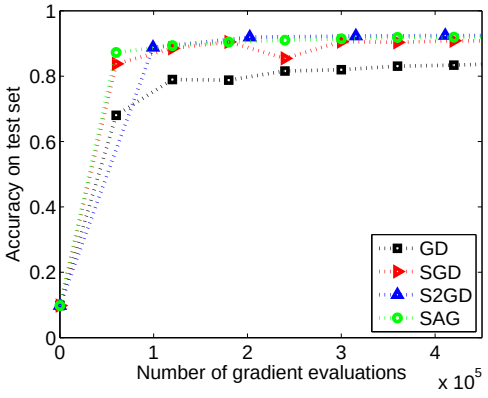


(a) $\lambda = 10^{-1}$

(b) $\lambda = 10^{-2}$

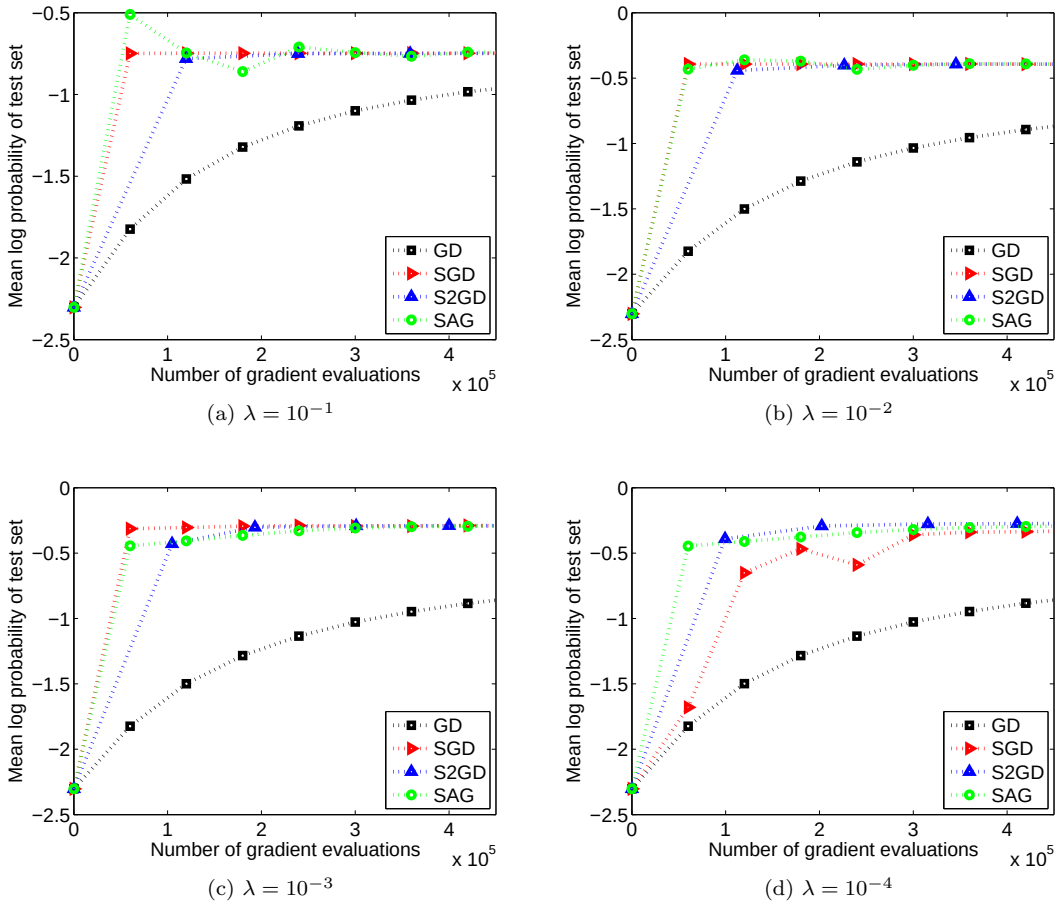


(c) $\lambda = 10^{-3}$



(d) $\lambda = 10^{-4}$

Figure 3.4: Prediction accuracy of regularized softmax regression for various regularization parameters λ, as it varies along the optimization process in the first few epochs. The predicted label is the one given the highest probability under softmax regression. The prediction accuracy is the proportion of correctly predicted test examples in the test set. For the setting of the parameters of the algorithms, see Figure 3.3.



(a) $\lambda = 10^{-1}$ (b) $\lambda = 10^{-2}$

(c) $\lambda = 10^{-3}$ (d) $\lambda = 10^{-4}$

Figure 3.5: Mean log probability of the test set, under softmax regression for various regularization parameters λ, as it varies along the optimization process in the first few epochs. Mean log probability is calculated as $\frac{1}{N}\sum_n \log P(y_n|\mathbf{x}_n, \mathbf{w})$, where $(\mathbf{x}_n, y_n)$ are examples from the test set. For the setting of the parameters of the algorithms, see Figure 3.3.

slowest asymptotic convergence. S2GD and SAG manage to achieve the best precision, with the former being faster. It is an interesting observation though that in the first few epochs (i.e. passes over the dataset) SGD appears to achieve better performance than S2GD and SAG. Indeed, S2GD and SAG have a large overhead; S2GD needs a full pass over the dataset to make the first update, and the first iterations of SAG are dominated by zero-initialized gradients. Eventually though, due to SGD's slow convergence, S2GD and SAG overtake SGD.

Even though we have established that S2GD and SAG can achieve higher precision in fitting the objective than SGD or GD after the same amount of work, it is natural to wonder whether this extra precision makes any difference within a machine learning context. In other words, we can see that SGD can also achieve a reasonable accuracy fairly quickly, would that be good enough? To attempt an answer to this question, we have plotted in Figure 3.4 the predictive accuracy on the test set and in Figure 3.5 the mean log probability of the test set, under the learned softmax regression model after the first few epochs. We can see that after only one epoch, i.e. a single pass over the data, SGD is often very close already to the asymptotic performance, which means that any further improvement makes little to no difference to the predictive performance of the model. This observation is consistent with the view expressed in [2] that, unless it is accompanied by a reduction in the generalization error of the model, further reducing the optimality gap on the train set is "wasted computational effort".

4 Conclusions and Future Work

The conclusions of this project can be summarized as follows.

(i) Gradient descent, although efficient at a small scale, becomes impractical in large-scale problems. In such a case, stochastic alternatives are necessary.

(ii) Stochastic gradient descent is capable of achieving good solutions fairly quickly, although it is impractical if high precision is required. Fortunately, this might not be necessary in a machine learning context.

(iii) Semi-stochastic gradient descent and stochastic average gradient combine the best of both worlds by being capable of achieving very accurate solutions in a feasible amount of time.

In practice, perhaps the best strategy would be to run SGD for a single or a couple of epochs, get a fairly good solution and, if further precision is required, use S2GD or SAG initialized with this solution. The idea of initializing S2GD or SAG by the output of SGD was also suggested in their original papers, [6] and [10] respectively.

We would like to emphasize however that the conclusions of our experiments are relevant to fitting convex objectives. It would be interesting to see how the algorithms behave with non-convex objectives, such as when training a neural network. Also, this project has focused on optimization algorithms that update all parameters at once; choosing only a subset of parameters to update in each iteration gives rise to methods under the name coordinate descent (for example [5]), which would be worth including in future comparisons.

For now, the paper is based on using convex function as objective function, to compare with GD and SGD, but for neural networks' objective function is non-convex, so here for now we often use mini-batch gradient descent is best!

Acknowledgements

I would like to thank Dr Peter Richtárik for his guidance, especially regarding S2GD, and his useful feedback.

References

[1] D. Blatt, A. Hero, and H. Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[2] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks, Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 430–445. Springer, 2012.

[3] G. E. Hinton. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012.

[4] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, pages 315–323. 2013.

[5] J. Konečnỳ, Z. Qu, and P. Richtárik. S2CD: Semi-stochastic coordinate descent. 2014.

[6] J. Konečný and P. Richtárik. Semi-Stochastic Gradient Descent Methods. *ArXiv e-prints*, Dec. 2013.

[7] Y. LeCun, C. Cortes, and C. J. C. Burges. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`. Accessed on 4 Dec 2014.

[8] MATLAB. *version 8.2.0 (R2013b)*. The MathWorks Inc., Natick, Massachusetts, 2013.

[9] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer, 2004.

[10] N. L. Roux, M. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25*, pages 2663–2671. 2012.