Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Stochastic gradient descent methods for estimation with large data sets

Dustin Tran
Harvard University

Panos Toulis
Harvard University

Edoardo M. Airoldi
Harvard University

Abstract

We develop methods for parameter estimation in settings with large-scale data sets, where traditional methods are no longer tenable. Our methods rely on stochastic approximations, which are computationally efficient as they maintain one iterate as a parameter estimate, and successively update that iterate based on a single data point. When the update is based on a noisy gradient, the stochastic approximation is known as standard *stochastic gradient descent*, which has been fundamental in modern applications with large data sets. Additionally, our methods are numerically stable because they employ *implicit* updates of the iterates. Intuitively, an implicit update is a shrinked version of a standard one, where the shrinkage factor depends on the observed Fisher information at the corresponding data point. This shrinkage prevents numerical divergence of the iterates, which can be caused either by excess noise or outliers. Our **sgd** package in R offers the most extensive and robust implementation of stochastic gradient descent methods. We demonstrate that **sgd** dominates alternative software in runtime for several estimation problems with massive data sets. Our applications include the wide class of generalized linear models as well as M-estimation for robust regression.

Keywords: stochastic gradient descent, implicit updates, massive data, exponential family, generalized linear models, M-estimation.

1. Introduction

Massive data sets as well as streaming data, in which one observes only a group of data points at a time, are becoming increasingly common in modern statistical analysis. Under the setting of hundreds of millions of observations and hundreds or thousands of covariates (National Research Council 2013), it becomes difficult to estimate the parameters of a statistical model; the three ideal properties are computational efficiency, statistical optimality, and numerical stability, and it is challenging to address all three with a single estimation method.

More formally, suppose there exists a vector of parameters $\theta_\star \in \mathbb{R}^p$ and that we observe i.i.d. samples $\mathbf{D} = \{\mathbf{x}_n, \mathbf{y}_n\}$, for $n = 1, 2, \ldots, N$; in the n^{th} data point $(\mathbf{x}_n, \mathbf{y}_n)$, the outcome $\mathbf{y}_n \in \mathbb{R}^d$ is distributed conditional on covariates $\mathbf{x}_n \in \mathbb{R}^p$ according to a known density $f(\mathbf{y}_n; \mathbf{x}_n, \theta_\star)$, and thus the log-likelihood function for the entire data set $\mathbf{D}$ is given by $\ell(\theta; \mathbf{D}) = \sum_{n=1}^{N} \log f(\mathbf{y}_n; \mathbf{x}_n, \theta)$. The task is to estimate the true parameter value $\theta_\star$ when N is infinite (streaming setting), or to approximate some estimator of $\theta_\star$, such as the maximum-likelihood estimator $\theta^{\text{mle}} = \arg\max_{\theta \in \mathbb{R}^p} \ell(\theta; \mathbf{D})$, when N is finite.

Widely used methods for statistical estimation, such as Fisher scoring, the EM algorithm, and iteratively reweighted least squares (Fisher 1925; Dempster, Laird, and Rubin 1977; Green 1984) are not feasible in such settings; either they strictly do not apply in the streaming setting (infinite N), or they do not scale to large data (finite but large N). Fisher scoring, for example, requires at each iteration the inversion of a $p \times p$ matrix and evaluation of the log-likelihood over the full data set $\mathbf{D}$. This roughly yields $\mathcal{O}(Np^{2+\epsilon})$ running time complexity, which is prohibitive when N and p are large. In contrast, estimation with massive data sets typically requires a running time complexity that is $\mathcal{O}(Np^{1-\epsilon})$, i.e., that is linear in N but sublinear in the parameter dimension p.

Such performance is achieved in general by the stochastic gradient descent (SGD) algorithm, which was initially proposed by Sakrison (1965) as a modification of the Robbins-Monro procedure (Robbins and Monro 1951) for recursive estimation. It is defined through the iteration

$$\theta_n^{\text{sgd}} = \theta_{n-1}^{\text{sgd}} + \gamma_n C_n \nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1}^{\text{sgd}}). \tag{1}$$

We will refer to Equation (1) as SGD with explicit updates, or explicit SGD for short, because the next iterate θ_n^{sgd} can be computed immediately after the n^{th} data point $(\mathbf{x}_n, \mathbf{y}_n)$ is observed. The sequence $\gamma_n > 0$ is the *learning rate* sequence, and is typically defined such that $n\gamma_n \to \gamma > 0$ as $n \to \infty$; the hyperparameter $\gamma > 0$ is fixed and known as the *learning rate parameter*. The sequence $\{C_n\}$ is a sequence of positive-definite matrices, such that $C_n \to C$ with C known, and is used to better condition the iteration; in the simplest case $C_n = I$, i.e., we simply use the identity matrix, which results in *first-order* explicit SGD.

From a computational perspective, explicit SGD is efficient because it replaces the expensive inversion of $p \times p$ matrices, as in Fisher scoring, by a scalar sequence $\gamma_n > 0$ and a matrix C_n that is fast to manipulate numerically, by design. Furthermore, the log-likelihood is evaluated at a single observation $\mathbf{y}_n$ given $\mathbf{x}_n$, rather than the entire data set $\mathbf{D}$, which saves significant computation time. From a theoretical perspective, explicit SGD is justified because the theory of stochastic approximations (Robbins and Monro 1951, Theorem 1) implies that θ_n^{sgd} converges to a point θ_∞ such that $\mathbb{E}\left(\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_\infty)\right) = 0$. Under standard statistical theory, $\mathbb{E}\left(\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_\star)\right) = 0$, and this point is unique under typical regularity conditions (Lehmann and Casella 1998, Theorem 5.1, p.463), such as concavity of log-likelihood; this is true, for example, in the popular exponential family of statistical models (Brown 1986). Therefore, $\theta_\infty = \theta_\star$, i.e., explicit SGD converges to the true parameter value. In the finite N setting, a similar condition holds where θ_n^{sgd} approximates θ^{mle} if the n^{th} data point in Equation (1) is an unbiased sample from the total N data points; see also Toulis and Airoldi (2015b) for a review of applications of SGD on modern machine learning applications.

Despite these theoretical guarantees, explicit SGD requires careful tuning of the hyperparameter γ in the learning rate: small values of the parameter make the iteration (1) very slow

to converge in practice, whereas large values can cause numerical divergence. Moreover, it is known that explicit SGD is statistically inefficient even when γ is correctly specified (Toulis, Airoldi, and Rennie 2014). In particular, the amount of information loss from procedure (1) depends on the spectral gap of the Fisher information matrix, $\boldsymbol{\mathcal{I}}(\theta) = -\mathbb{E}\left(\nabla^2 \log f(\mathbf{y}_n; \mathbf{x}_n, \theta)\right)$, calculated at the true parameter value $\theta = \theta_\star$. A large spectral gap makes it hard, or even impossible, to make the learning rates large enough for fast convergence, and also small enough for stability (Toulis and Airoldi 2015a, Section 3.5).

Motivated by these challenges, Toulis, Tran, and Airoldi (2015) introduced averaged implicit stochastic gradient descent (AI-SGD), which is defined by the procedure

$$\theta_n^{\text{im}} = \theta_{n-1}^{\text{im}} + \gamma_n C_n \nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_n^{\text{im}}), \tag{2}$$

$$\bar{\theta}_n = (1/n) \sum_{i=1}^{n} \theta_i^{\text{im}}. \tag{3}$$

The first key component of AI-SGD is the *implicit* update (2). Note that it is implicit because the next iterate θ_n^{im} appears on both sides of the equation. This simple modification of the explicit SGD procedure offers several statistical advantages. In particular, assuming a common starting point $\theta_{n-1}^{\text{sgd}} = \theta_{n-1}^{\text{im}} \triangleq \theta_0$, one can show through a Taylor approximation of (2) around θ_0 that the implicit update satisfies

$$\Delta\theta_n^{\text{im}} = (I + \gamma_n C_n \hat{\mathcal{I}}(\theta_0; \mathbf{x}_n, \mathbf{y}_n))^{-1} \Delta\theta_n^{\text{sgd}} + \mathcal{O}(\gamma_n^2), \tag{4}$$

where $\Delta\theta_n = \theta_n - \theta_{n-1}$ for both methods, I is the identity matrix, and $\hat{\mathcal{I}}(\theta_0; \mathbf{x}_n, \mathbf{y}_n) = -\nabla^2 \ell(\theta_0; \mathbf{x}_n, \mathbf{y}_n)$ is the observed Fisher information matrix at θ_0 (equivalent to the Hessian of the negative log-likelihood at θ_0). Equation (4) implies that the implicit update (2) is a *shrinked* version of the explicit update (1). This shrinkage makes the iterations significantly more stable in small-to-moderate samples, and also robust to misspecifications of the learning rate parameter γ (Toulis *et al.* 2014). The implicit update (2) also has a Bayesian interpretation, where θ_n^{im} is the posterior mode of a model with the standard multivariate normal $\mathcal{N}(\theta_{n-1}^{\text{im}}, \gamma_n C_n)$ as the prior, and $f(\theta; \mathbf{x}_n, \mathbf{y}_n)$ as the likelihood. Thus it provides an iterative form of regularization. In optimization, update (2) is known as a *proximal update*, and corresponds to a stochastic version of the proximal point algorithm (Rockafellar 1976). Krakowski, Mahony, Williamson, and Warmuth (2007) and Nemirovski, Juditsky, Lan, and Shapiro (2009) have shown that proximal methods fit better in the geometry of the parameter space.

The second key component of AI-SGD is iterate averaging (3), which guarantees optimal statistical efficiency under fairly relaxed conditions. Ruppert (1988) and Polyak and Juditsky (1992) first proved that averaging of iterates can achieve statistical optimality in the standard context of stochastic approximation with explicit updates; Toulis *et al.* (2015) extended this result to the implicit SGD update (2). Thus, AI-SGD is effectively a recursive estimation method that is both statistically optimal and numerically stable, while remaining applicable to the setting of massive and/or streaming data.

In this paper we develop statistically efficient SGD algorithms for generalized linear models—extending Algorithm 1 of Toulis *et al.* (2014)—and also develop SGD algorithms to perform high-dimensional M-estimation. This allows for scalable estimation of such models with massive and/or streaming data. We provide a publicly available package **sgd** (Tran, Toulis, and

Airoldi 2015) written in R, which implements AI-SGD, as well as other SGD variants. In Section 2, we develop the algorithms. Section 3 contains experiments on simulated and real-world data, in which we demonstrate the advantages of the **sgd** package compared to alternative software. In Section 4, we describe the interface of **sgd** and implementation details for its use in practice.

2. Algorithms

In this section we develop algorithms which implement implicit SGD and AI-SGD for generalized linear models as well as M-estimation. We start by introducing an algorithm which efficiently computes a generalization of implicit update (2), which is useful for the aforementioned applications.

2.1. Efficient computation of implicit updates

The main difficulty in applying AI-SGD is the solution of the multidimensional fixed point equation for the implicit update (2). In the large class of models where the likelihood given covariate $\mathbf{x}$ depends on the parameter θ only through the *natural parameter* $\eta \equiv \mathbf{x}^\top\theta$, the solution of the fixed-point equation is computationally efficient. The general result is given in Theorem 2.1, whereas the assumption is made more precise below.

Assumption 2.1. *The likelihood $\ell(\theta; \mathbf{x}_n, \mathbf{y}_n) \equiv \log f(\mathbf{y}_n; \mathbf{x}_n, \theta)$ of parameter value θ given data point $(\mathbf{x}_n, \mathbf{y}_n)$ depends on θ only through the product $\mathbf{x}_n^\top\theta$, i.e.,*

$$\ell(\theta; \mathbf{x}_n, \mathbf{y}_n) \equiv \ell(\mathbf{x}_n^\top\theta; \mathbf{x}_n, \mathbf{y}_n). \tag{5}$$

A key implication of Assumption 2.1 is that the direction of the gradient of the log-likelihood does *not* depend on the parameter value since $\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta) = \ell'(\mathbf{x}_n^\top\theta; \mathbf{x}_n, \mathbf{y}_n)\mathbf{x}_n$, where the latter derivative is with respect to the natural parameter $\mathbf{x}_n^\top\theta$ and with fixed data $\mathbf{x}_n, \mathbf{y}_n$. This property is crucial because it implies that the implicit update (2) can be performed once a scalar value is found that will appropriately scale the gradient.

Theorem 2.1. *Suppose Assumption 2.1 holds. Then the gradient for the implicit iterate θ_n^{im} (2) is a scaled version of the gradient at the previous iterate, i.e.,*

$$\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_n^{\text{im}}) = s_n \nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1}^{\text{im}}). \tag{6}$$

The scalar $s_n \in \mathbb{R}$ satisfies

$$s_n\kappa_{n-1} = \ell'\left(\mathbf{x}_n^\top\theta_{n-1}^{\text{im}} + \gamma_n s_n \kappa_{n-1}\mathbf{x}_n^\top C_n \mathbf{x}_n; \mathbf{x}_n, \mathbf{y}_n\right), \tag{7}$$

where $\kappa_{n-1} = \ell'(\mathbf{x}_n^\top\theta_{n-1}^{\text{im}}; \mathbf{x}_n, \mathbf{y}_n)$.

Theorem 2.1 shows that the gradient $\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_n^{\text{im}})$ in the implicit update (2) is in fact a scaled version of the gradient $\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1}^{\text{im}})$ that would appear in update (2) if we were applying explicit updates. Therefore, computing the implicit update reduces to finding the scale factor $s_n \in \mathbb{R}$. See Toulis and Airoldi (2015a, Threorem 4.1) for a proof.

Penalized likelihood. It is possible to regularize both explicit and implicit SGD by adding a penalty to the log-likelihood. In particular, we consider the elastic net (Zou and Hastie 2005), where for some fixed $\alpha \in [0, 1]$ the penalty function is

$$P_\alpha(\theta) = (1 - \alpha)\frac{1}{2}\|\theta\|_2^2 + \alpha\|\theta\|_1. \tag{8}$$

Adding the elastic net with a regularization parameter $\lambda \in \mathbb{R}$ to explicit SGD is straightforward:

$$\theta_n^{\text{sgd}} = \theta_{n-1}^{\text{sgd}} + \gamma_n C_n(\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1}^{\text{sgd}}) - \lambda \nabla P_\alpha(\theta_{n-1}^{\text{sgd}})), \tag{9}$$

where the gradient of the elastic net penalty is given by

$$\nabla P_\alpha(\theta_{n-1}^{\text{im}}) = (1 - \alpha)\theta_{n-1}^{\text{sgd}} + \alpha \operatorname{sign}(\theta_{n-1}^{\text{sgd}}). \tag{10}$$

Here, the operation $\operatorname{sign}(\theta)$ is the element-wise sign operation, outputting 1 if $\theta_j > 0$, -1 if $\theta_j < 0$, and 0 otherwise.

For implicit SGD the update would be

$$\theta_n^{\text{im}} = \theta_{n-1}^{\text{im}} + \gamma_n C_n(\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_n^{\text{im}}) - \lambda \nabla P_\alpha(\theta_n^{\text{im}})). \tag{11}$$

However, it is not generally possible to compute update (11). For example, Assumption 2.1 does not hold because the gradient of the log-likelihood and the gradient of the penalty generally have two different directions. This breaks the argument of Theorem 2.1, where the direction of the update calculated at the next iterate θ_n^{im} is the same as the direction of the update calculated at the previous iterate θ_{n-1}^{im}.

To circumvent this problem, we simply penalize the previous iterate instead of the current, i.e., perform the update

$$\theta_n^{\text{im}} = \theta_{n-1}^{\text{im}} + \gamma_n C_n(\nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_n^{\text{im}}) - \lambda \nabla P_\alpha(\theta_{n-1}^{\text{im}})). \tag{12}$$

Then update (12) is equivalent to

$$\theta_n^{\text{im}} = \theta_{n-1}^{\text{im}} + \gamma_n C_n(s_n \nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1}^{\text{im}}) - \lambda \nabla P_\alpha(\theta_{n-1}^{\text{im}})), \tag{13}$$

where the scale factor s_n satisfies

$$s_n \kappa_{n-1} = \ell'\left(\mathbf{x}_n^\top \theta_{n-1}^{\text{im}} - \gamma_n \lambda \mathbf{x}_n^\top C_n \nabla P_\alpha(\theta_{n-1}^{\text{im}}) + \gamma_n s_n \kappa_{n-1} \mathbf{x}_n^\top C_n \mathbf{x}_n; \mathbf{x}_n, \mathbf{y}_n\right), \tag{14}$$

and where $\kappa_{n-1} = \ell'(\mathbf{x}_n^\top \theta_{n-1}^{\text{im}}; \mathbf{x}_n, \mathbf{y}_n)$. A proof for this case with penalized likelihoods is identical to the proof of Theorem 2.1.

Final algorithm for implicit updates. This analysis leads to Algorithm 1, which, for models satisfying Assumption 2.1, implements the most general update (13) of implicit SGD with conditioning matrices and penalty. This algorithm applies a root-finding procedure solving Equation (14) at every iteration, which is fast because the equation is one-dimensional and the search bounds for the solution are known, having a diminishing range $\mathcal{O}(\gamma_n)$. Indeed, the one-dimensional search is computationally negligible in practice, as we see in Section 3.

Algorithm 1 Efficient implementation of implicit update (13)

function IMPLICIT_UPDATE($\ell'(\cdot;\cdot), \gamma_n, \theta_{n-1}^{\text{im}}, \mathbf{x}_n, \mathbf{y}_n, C_n, P_\alpha$)
  # Compute search bounds B
  $r_n \leftarrow \gamma_n \ell'\left(\mathbf{x}_n^\top \theta_{n-1}^{\text{im}}; \mathbf{x}_n, \mathbf{y}_n\right)$
  $B \leftarrow [0, r_n]$
  if $r_n \leq 0$ **then**
    $B \leftarrow [r_n, 0]$
  end if
  # Solve fixed-point equation by a root-finding method
  $\xi = \gamma_n \ell'(\mathbf{x}_n^\top \theta_{n-1}^{\text{im}} - \gamma_n \lambda \mathbf{x}_n^\top C_n \nabla P_\alpha(\theta_{n-1}^{\text{im}}) + \xi \mathbf{x}_n^\top C_n \mathbf{x}_n; \mathbf{x}_n, \mathbf{y}_n)$, $\xi \in B$
  $s_n \leftarrow \xi / r_n$
  # Equivalent to implicit update (13)
  return $\theta_{n-1}^{\text{im}} + \gamma_n C_n \left(s_n \ell'\left(\mathbf{x}_n^\top \theta_{n-1}^{\text{im}}; \mathbf{x}_n, \mathbf{y}_n\right) \mathbf{x}_n - \lambda \nabla P_\alpha(\theta_{n-1}^{\text{im}})\right)$
end function

We also note that because the implicit update (17) effectively does regularization as a shrinkage estimate (see Equation (4)), the use of penalization is not as crucial in practice as it is for explicit updates. We make extensive experiments using Algorithm 2 and also examine this effect in Section 3.

2.2. Generalized linear models

In the family of generalized linear models (GLMs), the outcome $y_n \in \mathbb{R}$ follows an exponential family distribution conditional on $\mathbf{x}_n$,

$$y_n \mid \mathbf{x}_n \sim \exp\left\{\frac{1}{\psi}(\eta_n y_n - b(\eta_n))\right\} c(y_n, \psi), \quad \eta_n \equiv \mathbf{x}_n^\top \theta_\star, \tag{15}$$

where the scalar $\psi > 0$ is the dispersion parameter which affects the variance of the outcome, $c(\cdot,\cdot)$ is the base measure, and $b(\cdot)$ is the log normalizer which ensures that the distribution integrates to one.[1] Additionally, in a GLM it is assumed that $\mathbb{E}(y_n|\,\mathbf{x}_n) = h(\mathbf{x}_n^\top \theta_\star)$, where $h : \mathbb{R} \to \mathbb{R}$ is known as the *transfer function* (Nelder and Wedderburn 1972; Dobson and Barnett 2008). A simple property of GLMs is that the transfer function is the first derivative of the log normalizer, i.e., $h(\mathbf{x}_n^\top \theta) = b'(\mathbf{x}_n^\top \theta)$, for all $\mathbf{x}_n, \theta$.

A straightforward implementation of explicit SGD for estimation with GLMs is

$$\theta_n^{\text{sgd}} = \theta_{n-1}^{\text{sgd}} + \gamma_n C_n [y_n - h(\mathbf{x}_n^\top \theta_{n-1}^{\text{sgd}})] \mathbf{x}_n. \tag{16}$$

Similarly, the AI-SGD procedure can be written as

$$\begin{aligned} \theta_n^{\text{im}} &= \theta_{n-1}^{\text{im}} + \gamma_n C_n [y_n - h(\mathbf{x}_n^\top \theta_n^{\text{im}})] \mathbf{x}_n, \\ \bar{\theta}_n &= \frac{1}{n} \sum_{i=1}^{n} \theta_n^{\text{im}}. \end{aligned} \tag{17}$$

[1] We present one-dimensional outcomes for simplicity. However, our theory easily extends to multidimensional outcomes. Such an extension is given, for example, in Section 2.3 on M-estimation.

Algorithm 2 Estimation of generalized linear models with AI-SGD

Initialize $\theta_0^{\text{im}}, \bar{\theta}_0$
for $n = 1, 2, \ldots$ **do**
  Define $\ell'(\mathbf{x}_n^\top \theta; \mathbf{x}_n, \mathbf{y}_n) \equiv y_n - h(\mathbf{x}_n^\top \theta)$
  Calculate implicit update

$$\theta_n^{\text{im}} \leftarrow \text{IMPLICIT\_UPDATE}(\ell'(\cdot;\cdot), \gamma_n, \theta_{n-1}^{\text{im}}, \mathbf{x}_n, \mathbf{y}_n, C_n, P_\alpha)$$

  $\bar{\theta}_n \leftarrow \frac{n-1}{n}\bar{\theta}_{n-1} + \frac{1}{n}\theta_n^{\text{im}}$
end for

By assumption, $\ell(\theta; y_n, \mathbf{x}_n) \propto (\mathbf{x}_n^\top \theta_\star) y_n - b(\mathbf{x}_n^\top \theta_\star)$, and thus the log-likelihood depends on parameter value $\theta_\star$ only through its linear combination with covariate value $\mathbf{x}_n$. Additionally, $\text{Var}\,(y_n|\mathbf{x}_n) = h'(\mathbf{x}_n^\top \theta_\star)||\mathbf{x}_n||^2$, and thus $h' \geq 0$, which implies that ℓ is twice-differentiable and concave, thus fulfilling Assumption 2.1.

Penalized likelihood. As argued before, one can add the elastic penalty by applying it to the previous estimate instead of the current. That is, for fixed $\alpha \in [0, 1]$ and regularization parameter $\lambda \in \mathbb{R}$, the AI-SGD procedure for generalized linear models with elastic net is

$$\begin{aligned} \theta_n^{\text{im}} &= \theta_{n-1}^{\text{im}} + \gamma_n C_n \left([y_n - h(\mathbf{x}_n^\top \theta_n^{\text{im}})]\mathbf{x}_n - \lambda \nabla P_\alpha(\theta_{n-1}^{\text{im}}) \right), \\ \bar{\theta}_n &= \frac{1}{n} \sum_{i=1}^{n} \theta_n^{\text{im}}. \end{aligned} \tag{18}$$

Algorithm 2 implements estimation of GLMs through AI-SGD based on updates (18).

2.3. M-Estimation

Given a data set of N observations $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}$ and a convex function $\rho : \mathbb{R} \to \mathbb{R}^+$, the M-estimator is defined as

$$\hat{\theta}^m = \arg\min_{\theta \in \mathbb{R}^p} \sum_{n=1}^{N} \rho(\mathbf{y}_n - \mathbf{x}_n^\top \theta), \tag{19}$$

where it is assumed $\mathbf{y}_n = \mathbf{x}_n^\top \theta_\star + \epsilon_n$, and ϵ_n are i.i.d. zero mean-valued noise. M-estimators are especially useful in robust statistics (Huber 1964; Huber and Ronchetti 2009), as appropriate choice of ρ can reduce the influence of outliers in data. Recently, there has been increased interest in the literature for fast approximation of M-estimators due to their robustness (Donoho and Montanari 2013; Jain, Tewari, and Kar 2014).

Typically in M-estimation, ρ is twice-differentiable around zero and

$$\mathbb{E}\left(\rho'(\mathbf{y}_n - \mathbf{x}_n^\top \hat{\theta}^m)\mathbf{x}_n \right) = 0, \tag{20}$$

where the expectation is over the empirical data distribution. Therefore SGD algorithms can be applied to approximate the M-estimator $\hat{\theta}^m$. Importantly, ρ is convex, which implies that the conditions of Assumption 2.1 are met.

Algorithm 3 M-estimation with AI-SGD

1: Initialize $\theta_0^{\text{im}}, \bar{\theta}_0$
2: **for** $n = 1, 2, \ldots$ **do**
3: Define $\ell'(\mathbf{x}_n^\top \theta; \mathbf{x}_n, \mathbf{y}_n) \equiv -\rho'(\mathbf{y}_n - \mathbf{x}_n^\top \theta)$
4: Calculate implicit update

$$\theta_n^{\text{im}} \leftarrow \text{IMPLICIT\_UPDATE}(\ell'(\cdot;\cdot), \gamma_n, \theta_{n-1}^{\text{im}}, \mathbf{x}_n, \mathbf{y}_n, C_n, P_\alpha)$$

5: $\bar{\theta}_n \leftarrow \frac{n-1}{n}\bar{\theta}_{n-1} + \frac{1}{n}\theta_n^{\text{im}}$
6: **end for**

The AI-SGD procedure for approximating M-estimators is

$$\theta_n^{\text{im}} = \theta_{n-1}^{\text{im}} + \gamma_n C_n [\rho'(\mathbf{y}_n - \mathbf{x}_n^\top \theta_n^{\text{im}})]\mathbf{x}_n, \tag{21}$$

$$\bar{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} \theta_n^{\text{im}}. \tag{22}$$

An outline of the procedure is given in Algorithm 3. As before, Algorithm 3 also includes the optional use of a sequence of conditioning matrices C_n and a penalty function P_α. The use of penalization has particularly been considered as a way to merge the robustness properties given by a choice of ρ with sparsity, e.g, through lasso (Owen 2007; Lambert-Lacroix, Zwald *et al.* 2011; Li, Peng, and Zhu 2011).

It is also typical to assume that the density of ϵ_n is symmetric around zero. Therefore, it also holds $\mathbb{E}\left(\rho'(\mathbf{y}_n - \mathbf{x}_n^\top \theta_\star)\mathbf{x}_n\right) = 0$, where the expectation is over the true data distribution. Hence SGD procedures can be used to estimate $\theta_\star$ in the case of an infinite stream of observations ($N = \infty$). We write Algorithm 3 for the case of finite N, but it is trivial to adapt the procedure to infinite N.

3. Experiments

In this section, we compare the SGD methods implemented in the **sgd** package, such as explicit SGD and AI-SGD, with standard, deterministic optimization methods that are widely used in statistical practice, such as **glmnet**, **biglm**, and **speedglm**. We demonstrate in both massive and streaming data settings that standard methods are not applicable, and furthermore that SGD methods outperform such methods upon orders of magnitude in runtime and convergence.

As standard methods are not competitive, we also compare the proposed SGD methods to each other, e.g., comparing AI-SGD to explicit SGD, across a wide range of learning rate specifications, including adaptive specifications such as AdaGrad (Duchi, Hazan, and Singer 2011) and RMSProp (Tieleman and Hinton 2012); more details on the specifications which are available in **sgd** are given in Section 4.3.

All timings are carried out on a general-purpose 2.6 GHz Intel Core i5 processor, and are reported for various algorithms which reach a thresholded L_2 distance to the true parameter value.

3.1. Linear regression with the lasso

We follow an experiment used in benchmarking the **glmnet** package (Friedman, Hastie, and Tibshirani 2010, Section 5.1), which fits GLMs with the elastic net penalty over a regularization path. As **glmnet** was shown to outperform related software such as **elasticnet** (Zou and Hastie 2012) and **lars** (Hastie and Efron 2013), we compare **sgd** strictly to **glmnet**. The design matrix $\mathbf{X}$ with N observations and p predictors is generated from a normal distribution such that each pair of predictors $\mathbf{X}_j, \mathbf{X}_{j'}$ has the same correlation ρ. Each of the N outcomes $\mathbf{y}_n$, $n = 1, 2, \ldots N$, is defined as

$$\mathbf{y}_n = \mathbf{x}_n^\top \theta_\star + k\epsilon_n, \tag{23}$$

where $\theta_{\star j} = (-1)^j \exp(-2(j-1)/20)$ so that the elements of the true parameter value $\theta_\star$ have alternating signs and are exponentially decreasing. The noise ϵ_n is distributed as a standard normal, $\epsilon \sim \mathcal{N}(0, 1)$, and k is chosen so that the signal-to-noise ratio is equal to 3.0. We run **glmnet** with "covariance updates", which takes advantage of sparse updates in the parameter space to reduce the complexity of $\mathcal{O}(Np)$ calculations per iteration. It performs better in our experiments than the "naive update" also considered in Friedman *et al.* (2010).

Table 1 outlines results for a combination of triplets (N, p, ρ), ranging from $N = 1,000$ observations and scaling up to $N = 10$ million. **glmnet** is seen to be competitive with SGD procedures under the setting of $N = 1,000$ observations, and in fact **glmnet** slightly outperforms SGD algorithms for lower dimensions of N and p. It is in any higher dimensional setting where **sgd** strictly dominates **glmnet**, as seen in the table where for example, with $N = 50,000$ and $p = 10,000$, **sgd** is orders of magnitude faster.

Furthermore, **glmnet** is restricted by the memory limitations of computer hardware. For example, simulations with $100,000$ observations and $10,000$ features require 8 GB in memory for simply storing the data, and more is required for parameter storage and computational overhead. For the **sgd** package, we simply stream the data points using **bigmemory**, which requires less than 500 MB of RAM for all our experiments, a 16-fold decrease in memory requirements. This is not possible for **glmnet** in either the case of real streaming data, or simply as a way to remove memory bottlenecks. In principle, gradient descent algorithms such as **glmnet** can read and destroy data memory from disk as it loops over the full data set; however, this is impractical as it requires such an expensive memory access at each iteration.

We now compare the SGD algorithms. For small dimensional problems, explicit SGD achieves faster runtime than AI-SGD as it does not require a one-dimensional search following Algorithm 1. However, in high dimensions and high correlations, it becomes extremely difficult for explicit SGD to even converge for this toy linear model. It is sensitive to the learning rate, and any misspecification can cause it to diverge numerically. Thus, we were not able to obtain a proper timing for explicit SGD in settings of either high correlation ($\rho > 0.9$) or high dimension with medium correlation ($\rho > 0.5$). In practice one must tune the hyperparameter for explicit SGD—thus requiring significant computational overhead and user input—while also closely monitoring the stochastic gradients for consideration of other numerical issues. AI-SGD on the other hand uses additional computation per iteration, which in high dimensions is negligible compared to the cost of a stochastic gradient update. This additional computation leads to significantly more robust updates and faster convergence.

	Correlation					
	0	0.1	0.2	0.5	0.9	0.95
	$N = 1,000$ $p = 100$ (sec)					
`sgd(method="ai-sgd")`	0.03	0.03	0.03	0.03	0.04	0.34
`sgd(method="sgd")`	0.02	0.02	0.02	0.02	0.03	0.03
`glmnet`	0.02	0.02	0.02	0.02	0.02	0.03
	$N = 10,000$ $p = 1,000$ (sec)					
`sgd(method="ai-sgd")`	1.81	1.65	1.78	1.50	1.85	1.83
`sgd(method="sgd")`	2.78	2.90	2.93	2.81	–	–
`glmnet`	6.60	7.76	8.00	7.83	6.50	6.70
	$N = 50,000$ $p = 10,000$ (min)					
`sgd(method="ai-sgd")`	3.12	3.51	3.43	3.26	3.40	3.38
`sgd(method="sgd")`	4.83	4.86	5.23	–	–	–
`glmnet`	14.58	15.28	16.29	15.58	16.54	16.41
	$N = 1,000,000$ $p = 50,000$ (min)					
`sgd(method="ai-sgd")`	22.23	21.10	19.88	21.52	18.53	20.53
`sgd(method="sgd")`	27.80	34.08	–	–	–	–
`glmnet`	–	–	–	–	–	–
	$N = 10,000,000$ $p = 100,000$ (hr)					
`sgd(method="ai-sgd")`	9.38	10.20	9.58	8.54	10.11	10.74
`sgd(method="sgd")`	13.50	–	–	–	–	–
`glmnet`	–	–	–	–	–	–

Table 1: Linear regression with the lasso. Timing (in various units) is displayed for 100 λ values, averaged over 10 runs. The first line is **sgd** using AI-SGD and the second line is **sgd** using explicit SGD. Omitted entries indicate failure of the algorithm; for explicit SGD it numerically diverges, and for **glmnet** it could not run due to memory limitations.

3.2. Logistic regression with ridge penalty

Following benchmarks that are popular in the machine learning and optimization literature (Xu 2011; Shamir and Zhang 2012; Bach and Moulines 2013; Schmidt, Le Roux, and Bach 2013), we perform large-scale logistic regression on four data sets:

- `rcv1` (Lewis, Yang, Rose, and Li 2004): text data set in which the task is to classify documents belonging to class `ccat`, where we apply preprocessing provided by Bottou (2012).

- `covtype` (Blackard 1998): data set consisting of forest cover types in which the task is to classify for one specific class among 7 forest cover types.

	description	type	covariates	training set	test set	λ
covtype	forest cover type	sparse	54	464,809	116,203	10^{-6}
delta	synthetic data	dense	500	450,000	50,000	10^{-2}
rcv1	text data	sparse	47,152	781,265	23,149	10^{-5}
mnist	digit image features	dense	784	60,000	10,000	10^{-3}

Table 2: Summary of data sets and the L_2 regularization parameter λ used.

- `delta` (Sonnenburg, Franc, Yom-Tov, and Sebag 2008): synthetic data offered in the PASCAL Large Scale Challenge. We apply the default processing offered by the challenge organizers.
- `mnist` (Le Cun, Bottou, Bengio, and Haffner 1998): images of handwritten digits, where the task is to classify digit 9 against all others.

A summary of the data sets is available in Table 3, where the number of observations are typically on the order of several hundred thousand, and the covariates range from a few dozen to tens of thousands. The regularization parameter λ for the ridge penalty are set according to those used in Xu (2011).

We compare to the following three packages: **biglm** (Lumley 2013) and **speedglm** (Enea, Meiri, and Kalimi 2015), both of which perform approximate updates using iteratively reweighted least squares, and **LiblineaR** (Helleputte 2015), which is a simple wrapper to a C++ library for regularized linear classification. We use the stochastic dual coordinate ascent algorithm (Shalev-Shwartz and Zhang 2013) in **LiblineaR**. In addition, we consider the **mnlogit** package (Hasan, Zhiyu, and Mahani 2015), which implements multinomial logistic regression using the classical technique of Newton-Raphson, and exploits iterations over intermediate data structures for fast Hessian calculations. For modest-sized problems, **mnlogit** is shown to be 10-50 times faster than **mlogit** (Croissant 2013), **VGAM** (Yee 2010), and the `multinom` function in **nnet** (Venables and Ripley 2002). Finally, we also run the default function `glm.fit` as a baseline. We note that **mnlogit** and `glm.fit` can be only employed for standard (unregularized) multinomial regression, so we run them without the ridge penalty.

Table 3 outlines the runtimes for the considered packages. The two SGD algorithms are orders of magnitude faster than its competitors on all data sets. Interestingly, **biglm** and **speedglm** failed to run on the three real data sets when attempting to invert subsets of the data, and only succeeded for the one synthetic data set `delta`. We also note that the largest data set—`rcv1`—failed for the majority of algorithms: only the packages **sgd** and **LiblineaR** were able to converge, both of which natively use stochastic gradients for computationally efficient updates. However, **sgd** is significantly faster because less overhead seems to be involved in passing data structures to perform computation in native C++.

Moreover, **sgd** requires $\mathcal{O}(p)$ memory, which is optimal in the sense that $\mathcal{O}(p)$ is the minimum required for simply storing the n^{th} iterate θ_n^{im}. Both **biglm** and **speedglm** require $\mathcal{O}(p^2)$ for the inversion of a $p \times p$ matrix, as do **mnlogit** and `glm.fit`. The **mnlogit** package also requires data in the `long` format, which leads to a duplication of rows, as many entries display redundant information. Moreover, while exploitation of the Hessian structure can help in practice (as it outperforms `glm.fit`), we observe that the traditional technique of Newton-Raphson remains

data set	**sgd** (AI-SGD)	**sgd** (SGD)	**biglm**	**speedglm**	**LiblineaR**	**mnlogit**	`glm.fit`
covtype	5.21	7.58	–	–	1444.78	16.04	40.11
delta	10.10	10.23	736.13	30.50	2167.14	445.73	498.97
rcv1	14.15	15.42	–	–	133.10	–	–
mnist	3.50	3.37	–	–	208.55	232.53	890.76

Table 3: Large-scale logistic regression on four data sets. Timing (in seconds) is displayed, averaged over 10 runs. Omitted entries indicate failure of the algorithm; for **biglm** and **speedglm**, it could not run due to inversions of singular matrices; for **mnlogit** it could not run due to memory limitations.

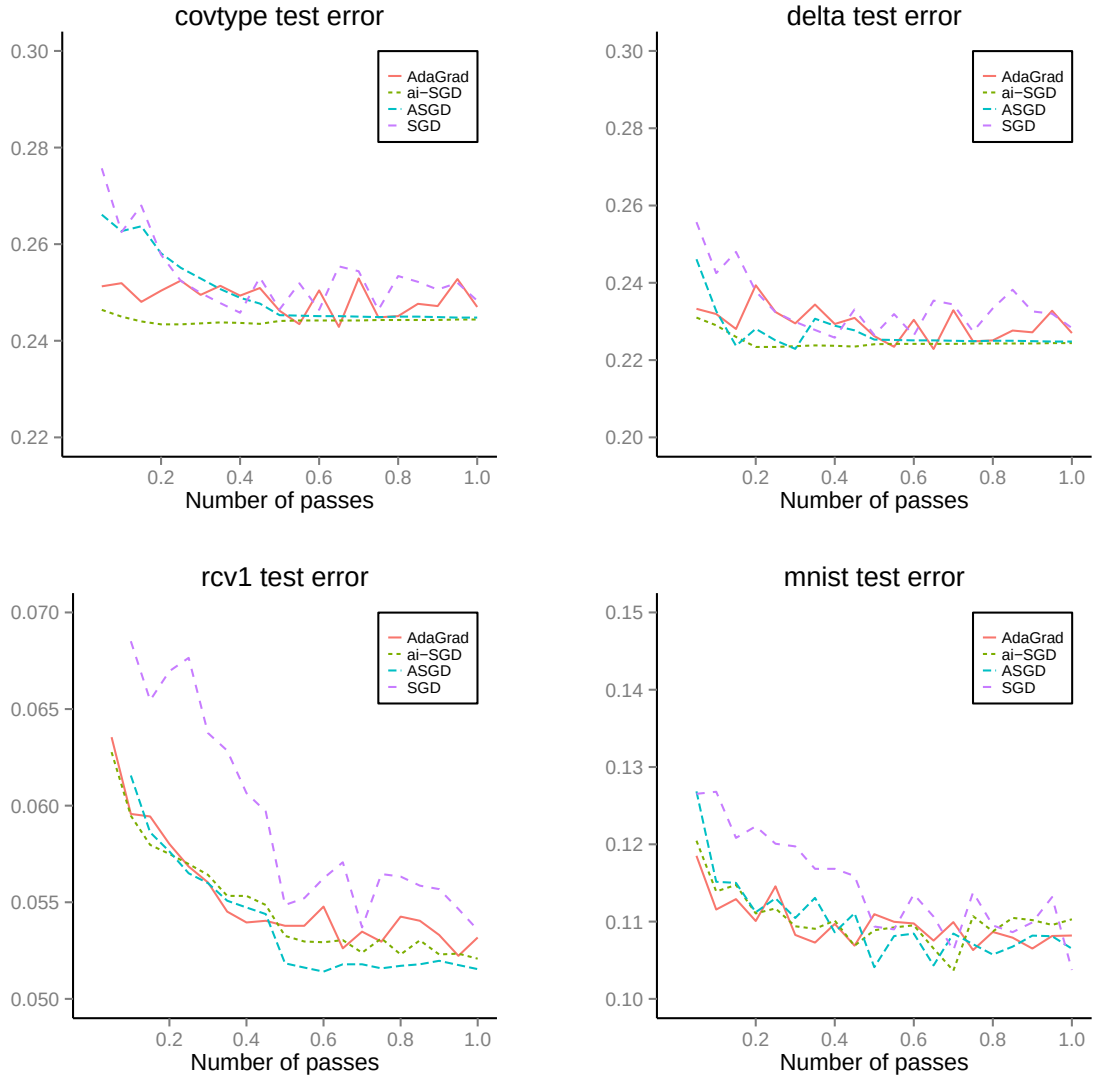

Figure 1: Large scale logistic regression on four data sets. Each plot indicates the classification error on the test set for explicit SGD with AdaGrad, AI-SGD, averaged SGD, and explicit SGD over a pass of the data.

N	p	**sgd** (AI-SGD)	**sgd** (SGD)	**hqreg**	units
1,000	100	0.05	0.04	0.03	(sec)
10,000	500	0.55	0.46	0.40	(sec)
10,000	1,000	1.30	2.22	6.34	(sec)
50,000	10,000	3.12	3.86	15.57	(min)
100,000	50,000	8.13	15.20	–	(min)
1,000,000	100,000	35.88	51.93	–	(min)
10,000,000	100,000	8.64	9.55	–	(hr)
100,000	1,000,000	18.80	26.43	–	(hr)

Table 4: High-dimensional M-estimation with the Huber loss. Timing (in units given by the last column) is displayed for 100 λ values, averaged over 10 runs. Omitted entries indicate failure of the algorithm; for **hqreg**, it could not run due to memory limitations.

untenable because it still requires $\mathcal{O}(Np^2)$ complexity per iteration in the worst case.

For demonstration, Figure 1 shows the progress of multiple SGD algorithms available in **sgd** (see Section 4.2) over a pass of the data. We note that AI-SGD achieves the fastest or competitive convergence rates, without requiring significant tuning of parameters as the other algorithms do; this includes popular adaptive learning rate specifications, such as explicit SGD with AdaGrad.

3.3. M-estimation with the Huber loss

We follow an example for high-dimensional M-estimation in Donoho and Montanari (2013, Section 2.4). Define the convex function $\rho : \mathbb{R} \to \mathbb{R}^+$ to be the *Huber loss*,

$$\rho(z; \lambda) = \begin{cases} z^2/2, & \text{if } |z| \leq \lambda, \\ \lambda|z| - \lambda^2/2, & \text{otherwise.} \end{cases}$$

Fix the thresholding parameter $\lambda = 3$, and generate the $N \times p$ design matrix with i.i.d. entries $\mathbf{X}_{i,j} \sim \mathcal{N}(0, \frac{1}{N})$. We fix the true set of parameters $\theta_\star$ to be a vector randomly drawn with fixed norm $\|\theta_\star\|_2 = 6\sqrt{p}$, and then generate outcome $\mathbf{y}_n$, $n = 1, 2, \ldots, N$, as

$$\mathbf{y}_n = \mathbf{x}_n^\top \theta_\star + \epsilon_n. \tag{24}$$

For the distribution of errors ϵ_n, we use Huber's contaminated normal distribution $\mathsf{CN}(0.05, 10)$, i.e., $\epsilon_n \sim 0.95z + 0.05h_{10}$, i.i.d., where z is standard normal and h_x is a point mass at x.

Few alternative packages to **sgd** exist for high-dimensional robust estimation. We compare to **hqreg** (Yi 2015), which fits regularization paths for Huber loss regression with the elastic net penalty. Note that **hqreg** is specialized to the Huber loss and cannot perform estimation for the general setting of M-estimation problems considered here.

Table 4 outlines results for a combination of pairs (N, p), ranging from small problems of $N = 1,000$ observations to massive data settings of $N = 10$ million. We apply the elastic net penalty with $\alpha = 0.5$, which puts even weight on both the lasso and ridge components,

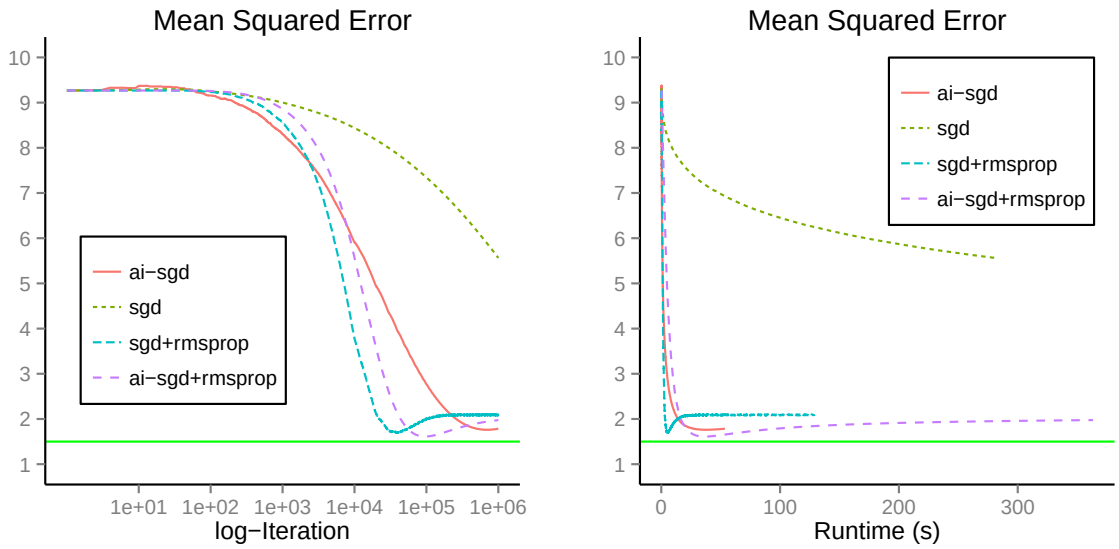

Figure 2: High dimensional M-estimation with the Huber loss, for $N = 100,000$ observations, $p = 10,000$ covariates, and a fixed regularization parameter λ. The plots indicate the mean-squared error across iterations (left) and time (right) for SGD algorithms. The horizontal line displays the mean-squared error for the exact M-estimator $\widehat{\theta}^m$.

and then compute a regularization path for both packages. We also include an example of $N = 100,000$ observations and $p = 1,000,000$ covariates, where there exist far more covariates than data points; this occurs often in applications, e.g., in text analysis, bioinformatics, and signal processing (Lustig, Donoho, Santos, and Pauly 2008; Blei 2012).

The SGD algorithms begin to outperform **hqreg** on the order of tens of thousands of observations, and significantly so for larger data settings. Similar to the memory limitations of **glmnet**, **hqreg** requires access to the full data set per iteration of its algorithm, which is infeasible when the data cannot be held in memory. Thus we were unable to obtain proper timings for data sets of size greater than $50,000$ observations and $10,000$ covariates.

Figure 2 displays the progress of the SGD algorithms for the setting of $N = 100,000$ observations and $p = 10,000$ covariates, for a fixed regularization parameter λ. For demonstration, we run the algorithms over 10 passes of the data and thus over a total of 1 million iterations. AI-SGD is seen to achieve a significantly faster convergence rate than explicit SGD. We also consider the use of adaptive schedules, here with RMSProp, as it performs the fastest among other available learning rates (see Section 4.3). With RMSProp, the difference between the two methods—SGD and AI-SGD—is noticeably smaller, and in fact SGD seems to converge slightly faster. We note however that the use of SGD algorithms with RMSProp breaks statistical efficiency, and indeed we see this effect as the mean-squared error oscillates around a value higher than the MSE of the exact M-estimator (green line). Therefore we advocate the use of AI-SGD with a one-dimensional learning rate, which still converges quite quickly.

4. Interface and implementation

We now discuss the interface of **sgd** and various technical details that are important for its use in practice.

4.1. Interface

The **sgd** package provides an intuitive and accessible set of methods for performing estimation with large-scale data sets. At the core of the package is the function

```
sgd(formula, data, model, model.control, sgd.control)
```

The user provides a `formula` on the data frame `data`—similar to function primitives, such as `lm`—and then specifies the `model`. The model parameters are estimated using SGD methods, which defaults to AI-SGD. The optional arguments `model.control` and `sgd.control` specify attributes one can tweak about the model and the stochastic gradient method, respectively. For example, given a data frame `dat` with response vector stored as the column `y`,

```
sgd(y ~ ., data=dat, model="lm")
```

fits a linear model with the default specifications, e.g., AI-SGD with a one-dimensional learning rate. Similarly,

```
sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"))
```

fits logistic regression with the default specifications. Numerous examples are available in the package by running `demo(package="sgd")`.

The `sgd` function also interfaces with data sets that are too large to fit into memory or are streaming (more details in Section 4.4), and can be run with a custom loss function if desired.

The output of the `sgd` function is a `sgd` object, which is a light wrapper on a `list` which collects quantities, such as the final parameter estimates and convergence diagnostics. Custom generic methods are also available for the `sgd` class, such as `print`, `predict`, and `plot`.

4.2. Stochastic gradient methods

While we describe the explicit SGD and AI-SGD algorithms in Section 2, the following stochastic gradient methods are also implemented in **sgd**:

- implicit SGD: Proposed by Toulis *et al.* (2014) in the context of generalized linear models, this algorithm uses the implicit update (2) and does not do any iterate averaging.
- averaged SGD: Proposed by Ruppert (1988) and Bather (1989) independently, this algorithm uses the explicit update (1) followed by iterate averaging (3).
- classical momentum (CM): Proposed by Polyak (1964), this algorithm uses the update

$$v_n = \mu v_{n-1} + a_n \nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1}), \tag{25}$$
$$\theta_n = \theta_{n-1} + v_n, \tag{26}$$

where $\mu \in [0, 1]$ is a fixed momentum coefficient. CM accelerates gradient descent with a velocity vector which accumulates directions of large increase in the log-likelihood.

- Nesterov's accelerated gradient (NAG): Proposed by Nesterov (1983), this algorithm uses the update

$$v_n = \mu v_{n-1} + a_n \nabla \log f(\mathbf{y}_n; \mathbf{x}_n, \theta_{n-1} + \mu v_{n-1}), \tag{27}$$
$$\theta_n = \theta_{n-1} + v_n, \tag{28}$$

where $\mu \in [0, 1]$ is a fixed momentum coefficient. NAG is similar to CM but accumulates velocity at a "look-ahead" point $\theta_{n-1} + \mu v_{n-1}$. This makes a partial update closer to θ_n, allowing NAG to change its velocity more quickly and responsively.

While all these methods are available, we recommend and apply AI-SGD as the default. It can be seen as an effective combination of the advantages from both implicit SGD and averaged SGD (Toulis *et al.* 2015). The momentum-based methods CM and NAG enjoy faster convergence rates than the original explicit SGD, but offer no theoretical benefits against AI-SGD. Without averaging techniques they also are statistically inefficient, whereas iterate averaging can be interpreted as an acceleration technique because larger learning rates are used. The velocity update in NAG is also a proxy for the implicit update, as its benefit mostly relies on making updates close to where the new estimate would lie.

4.3. Learning rates

We describe the available learning rates in more detail because they are critical for convergence of SGD methods, in practice. It is well-known (Sakrison 1965; Amari 1998; Toulis *et al.* 2014) that explicit SGD (1) and implicit SGD (2) have optimal statistical efficiency if the learning rate sequence γ_n together, with the conditioning matrices C_n, approximate the inverse Fisher information matrix $\boldsymbol{\mathcal{I}}(\theta_\star) = -\mathbb{E}\left(\nabla^2 \ell(\theta_\star; \mathbf{x}_n, \mathbf{y}_n)\right)$, i.e., $\gamma_n C_n \to \boldsymbol{\mathcal{I}}(\theta_\star)^{-1}$, in the limit. Therefore in first-order methods where $C_n = I$, the learning rate sequence acts as a scalar-valued approximation to the optimal rescaling as it is used in Fisher scoring (Fisher 1925). Based on this theory, the following learning rates are implemented in **sgd**:

- One-dimensional (Xu 2011): The learning rate is of the form

$$\gamma_n = \gamma_0 (1 + a\gamma_0 n)^{-c},$$

where $\gamma_0, a, c \in \mathbb{R}$ are fixed constants. For SGD algorithms without iterate averaging and SGD algorithms with iterate averaging, Xu (2011) proved that setting $c = 1$ and $c = 2/3$, respectively, leads to optimal statistical efficiency; a similar result holds for AI-SGD (Toulis *et al.* 2015).

- AdaGrad (Duchi *et al.* 2011): Rather than specify a one-dimensional learning rate $\gamma_n \in \mathbb{R}$, Duchi *et al.* (2011) propose a diagonal conditioning matrix $C_n \in \mathbb{R}^{p \times p}$ given by

$$\mathcal{I}_n = \mathcal{I}_{n-1} + \operatorname{diag}(\nabla \ell(\theta_{n-1}; \mathbf{x}_n, \mathbf{y}_n) \nabla \ell(\theta_{n-1}; \mathbf{x}_n, \mathbf{y}_n)^\top),$$
$$C_n = \eta (\mathcal{I}_n + \epsilon I)^{-1/2},$$

where $\operatorname{diag}(\cdot)$ extracts the diagonal entries of its matrix argument, $\eta \in \mathbb{R}$ is a constant, I is the identity matrix, and ϵ is a fixed value, typically 10^{-6}, to prevent division by zero.

In the limit, I_n is an unbiased estimate of the diagonal entries of the Fisher information, and the proposed diagonal matrix C_n, which accumulates such curvature information, is proven to be optimal for minimization of the regret bound.

- RMSProp (Tieleman and Hinton 2012): A learning rate which is popular in the deep learning literature (Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov 2014; Ranganath, Tang, Charlin, and Blei 2015; Rezende and Mohamed 2015), Tieleman and Hinton (2012) propose the diagonal conditioning matrix $C_n \in \mathbb{R}^{p \times p}$ given by

$$\mathcal{I}_n = \beta \mathcal{I}_{n-1} + (1 - \beta) \operatorname{diag}(\nabla \ell(\theta_{n-1}; \mathbf{x}_n, \mathbf{y}_n) \nabla \ell(\theta_{n-1}; \mathbf{x}_n, \mathbf{y}_n)^\top),$$
$$C_n = \eta (\mathcal{I}_n + \epsilon I)^{-1/2},$$

  where $\beta \in [0, 1]$ is the *discount factor*, $\eta \in \mathbb{R}$ is a constant, I is the identity matrix, and ϵ is a fixed value to prevent division by zero, as in AdaGrad. RMSProp uses a decay in the estimate for the Fisher information by taking a weighted average, and thus it gives more weight onto newer than older information. RMSProp aims to offset one problem AdaGrad often encounters in practice, where very large values occur for initial estimates of I_n (e.g., due to poor initialization), thus slowing down the AdaGrad procedure as it tries to accumulate enough curvature information to compensate for such an error (Schaul, Antonoglou, and Silver 2014). RMSProp balances this by taking a weighted average of previous and new information, and sees much empirical success. One problem, however, is that RMSProp is no longer decaying sufficiently quickly (Robbins and Monro 1951; Duchi *et al.* 2011), and thus it has no guarantees on convergence. Moreover, assuming convergence, the limit of the learning rate sequence is a constant, which makes the iterates jitter around the true parameter value, ad infinitum.

- Fisher: Following results on statistical efficiency and Fisher scoring, we propose a learning rate using a diagonal conditioning matrix $C_n \in \mathbb{R}^{p \times p}$ given by

$$\mathcal{I}_n = (1 - \gamma_n) \mathcal{I}_{n-1} + \gamma_n \operatorname{diag}(\nabla \ell(\theta_{n-1}; \mathbf{x}_n, \mathbf{y}_n) \nabla \ell(\theta_{n-1}; \mathbf{x}_n, \mathbf{y}_n)^\top),$$
$$C_n = (\mathcal{I}_n + \epsilon I)^{-1},$$

  where $\gamma_n \propto 1/n$, and ϵ is a small fixed value to prevent division by zero, as in AdaGrad. As before, I_n in the limit is an unbiased estimate of the diagonal Fisher information, and C_n is adaptive to curvature information.

One critical but often unnoticed issue with AdaGrad, RMSProp, and similar adaptive schedules is that they are statistically inefficient: the specification of the learning rates leads to biased estimation of the inverse Fisher information matrix $\boldsymbol{\mathcal{I}}(\theta_\star)^{-1}$ that, as mentioned earlier, is necessary for optimal statistical efficiency (an important exception is iterate averaging). This leads to a suboptimal asymptotic variance for the SGD procedure. Thus we recommend and apply the last proposed learning rate ("Fisher") by default: it takes advantage of the curvature information such methods benefit from, while still preserving as much statistical efficiency as possible in diagonal conditioning matrices.

4.4. Software integration

For data sets that cannot be loaded into memory, we access subsets of the data using **bigmemory** (Kane, Emerson, and Weston 2013). This allows one to perform stochastic gradient

descent by passing over the data loaded into RAM, and then to reload a new data set. This naturally applies to both large data sets, e.g., on the order of dozens of gigabytes, and streaming settings, in which one has access only to a subset of the (potentially infinite) data at a time.

In principle, with **bigmemory** the memory requirement for these stochastic gradient methods is only a single data point and the current parameter estimate, which is the minimum $\mathcal{O}(p)$ complexity for simply storing the estimate. In our implementation we use these savings to try to load as much data into RAM as possible. This speeds up convergence in practice, as it reduces the amount of I/O overhead; this especially becomes a significant bottleneck when reading many objects from disk.

For fast implementations we use **Rcpp** (Eddelbuettel and François 2011), where all algorithms are written in C++ and only interface-level code is written in R. Aside from the major computational gains, this also provides the opportunity to extend the library to other programming languages. **RcppArmadillo** (Eddelbuettel and Sanderson 2014) is applied for access to pre-optimized linear algebra routines, and **BH** (Eddelbuettel, Emerson, and Kane 2015) for access to the Boost libraries. We apply template meta-programming and reusable classes in an object-oriented framework, including concepts such as stochastic gradient methods, models, and learning rates. Such concepts make it easy for other users to develop new algorithms and prototype them in their own research or practices.

The plotting routines adopt many features from **ggplot2** (Wickham 2009), and are effectively templated `ggplot` objects. Our software is also robust through unit testing which follows the paradigm from **testthat** (Wickham 2011).

5. Discussion

As explicit SGD has been used extensively in practice, particularly in the deep learning community, many heuristics have been proposed to solve issues that often occur. We describe several of these issues and their proposed solutions in the literature, and compare to how our **sgd** package handles them.

Overfitting. As SGD algorithms simply minimize a loss function evaluted over the training data, overfitting is a prevalent problem as it is for all estimation methods. This is particularly an issue in complex likelihood functions such as neural networks (see, e.g., Giles (2001); Bakker and Heskes (2003)). Even with penalization terms that try to offset the fit of the parameters, it is still difficult for explicit SGD to find the right set of hyperparameters for such regularization without a computationally intensive search.

As a solution many practictioners adopt *early stopping*, which simply halts the optimization routine before it converges. However, there is little theory on the estimates obtained from early stopping. Most practically, it is difficult to know when to stop the algorithm and how to use it in combination with other regularization techniques, such as penalization.

Fortunately, one of the advantages of AI-SGD is that it requires less such tweaking: the implicit update effectively performs a regularization as seen from the Bayesian perspective, c.f., Section 1. We've also seen in practice that penalization terms do not affect the final estimates from AI-SGD, which makes it less reliant on heuristics, such as early stopping.

Vanishing or exploding gradients. The numerical instability of explicit SGD is a widespread issue in practice (Bengio, Simard, and Frasconi 1994; Hochreiter 1998; Hochreiter, Bengio, Frasconi, and Schmidhuber 2001; Toulis *et al.* 2014). The stochastic gradients can easily be too large leading to divergence, and when chained through compositions of functions can either vanish to zero, or even explode to numerically infinite values; for example, Toulis *et al.* (2014) demonstrate the instability of explicit SGD in a simple bivariate Poisson model, where slight misspecification of learning rate parameters lead to divergence.

Pascanu, Mikolov, and Bengio (2012) propose *gradient clipping*, which simply thresholds the stochastic gradient if it is outside a bounded interval. Unfortunately, while it can work in practice, it is a heuristic that breaks the key assumptions for convergence rate guarantees on SGD algorithms. Similarly, there is no principled way to set the bounds. For AI-SGD algorithms applied to the settings we consider in Section 2, such an issue never arises. Theoretical results establish stability regardless of the specification of the learning rate (Toulis and Airoldi 2015a, Section 3), and perform well in practice, as seen in Section 3.

6. Concluding remarks

The **sgd** package is the most extensive implementation in R of stochastic gradient methods for estimation with massive and/or streaming data sets. Thus, **sgd** broadens the capabilities of R for estimation with modern large data sets—on the orders of hundreds of millions of observations and hundreds of thousands of covariates—while retaining desirable statistical properties. The software is based on solid theory of stochastic approximations, which help guide the optimal selection of parameters, e.g., learning rates, in the underlying optimization routines. In this paper, we show how **sgd** can be applied for estimation of generalized linear models and M-estimation, which comprise a sizeable portion of estimation problems encountered in statistical practice.

There are many software extensions that are currently in development. We are working to interface with other high-performance computing packages, namely **sqldf** for faster I/O applications with streaming data, **doParallel** (Analytics and Weston 2014) and **Rmpi** (Yu 2002) for parallel updates across environments, and **gputools** (Buckner, Seligman, and Wilson 2011) for efficient computing with GPUs. The algorithms described here directly appeal to asynchronous implementations, following `Hogwild!` (Nui, Recht, Re, and Wright 2011), which allows for lock-free allocation of CPU cores. Sparse data structures would allow for fast structured matrix and vector products, which occur, for example, when looping over the covariates of a data point, and would significantly speed up computation on sparse data sets.

Finally, there has been little attention on, and in fact a pressing need for, model selection and hypothesis testing in SGD procedures. We are pursuing this in light of the new statistical challenges presented to us while developing the **sgd** package.

References

Amari SI (1998). "Natural Gradient Works Efficiently in Learning." *Neural computation*, **10**(2), 251–276.

Analytics R, Weston S (2014). ***doParallel**: Foreach Parallel Adaptor for the **Parallel** Package*. R package version 1.0.8, URL http://CRAN.R-project.org/package=doParallel.

Bach F, Moulines E (2013). "Non-strongly-convex Smooth Stochastic Approximation with Convergence Rate O(1/n)." In *Advances in Neural Information Processing Systems*, pp. 773–781.

Bakker B, Heskes T (2003). "Task Clustering and Gating for Bayesian Multitask Learning." *The Journal of Machine Learning Research*, **4**, 83–99.

Bather J (1989). *Stochastic Approximation: A Generalisation of the Robbins-Monro Procedure*, volume 89. Mathematical Sciences Institute, Cornell University.

Bengio Y, Simard P, Frasconi P (1994). "Learning Long-term Dependencies with Gradient Descent is Difficult." *Neural Networks, IEEE Transactions on*, **5**(2), 157–166.

Blackard J (1998). *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. Ph.D. thesis, Department of Forest Sciences, Colorado State University.

Blei DM (2012). "Probabilistic Topic Models." *Communications of the ACM*, **55**(4), 77–84.

Bottou L (2012). "Stochastic Gradient Descent Tricks." In *Neural Networks: Tricks of the Trade*, volume 1, pp. 421–436.

Brown LD (1986). "Fundamentals of Statistical Exponential Families with Applications in Statistical Decision Theory." *Lecture Notes-monograph series*, pp. i–279.

Buckner J, Seligman M, Wilson J (2011). ***gputools**: A Few GPU Enabled Functions*. R package version 0.26, URL http://CRAN.R-project.org/package=gputools.

Croissant Y (2013). ***mlogit**: Multinomial Logit Model*. R package version 0.2-4, URL http://CRAN.R-project.org/package=mlogit.

Dempster A, Laird N, Rubin D (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society, Series B*, **39**, 1–38.

Dobson A, Barnett A (2008). *An Introduction to Generalized Linear Models*. Texts in Statistical Science Series. CRC Press, London. ISBN 9781584889502.

Donoho D, Montanari A (2013). "High Dimensional Robust M-Estimation: Asymptotic Variance via Approximate Message Passing." *arXiv preprint arXiv:1310.7320v3*.

Duchi J, Hazan E, Singer Y (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *The Journal of Machine Learning Research*, **999999**, 2121–2159.

Eddelbuettel D, Emerson JW, Kane MJ (2015). ***BH**: Boost C++ Header Files*. R package version 1.58.0-1, URL http://CRAN.R-project.org/package=BH.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with hHgh-performance C++ Linear Algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Enea M, Meiri R, Kalimi T (2015). ***speedglm**: Fitting Linear and Generalized Linear Models to Large Data Sets*. R package version 0.3, URL http://CRAN.R-project.org/package=speedglm.

Fisher RA (1925). *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **27**(6), 957–968.

Giles RCSLL (2001). "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping." In *Advances in Neural Information Processing Systems*.

Green PJ (1984). "Iteratively Reweighted Least Squares for Maximum Likelihood Estimation, and Some Robust and Resistant Alternatives." *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 149–192.

Hasan A, Zhiyu W, Mahani AS (2015). ***mnlogit**: Multinomial Logit Model*. R package version 1.2.2, URL http://CRAN.R-project.org/package=mnlogit.

Hastie T, Efron B (2013). ***lars**: Least Angle Regression, Lasso and Forward Stagewise*. R package version 1.2, URL http://CRAN.R-project.org/package=lars.

Helleputte T (2015). ***LiblineaR**: Linear Predictive Models Based on the LIBLINEAR C/C++ Library*. R package version 1.94-2.

Hochreiter S (1998). "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, **6**(02), 107–116.

Hochreiter S, Bengio Y, Frasconi P, Schmidhuber J (2001). "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-term Dependencies."

Huber P (1964). "Robust Estimation of a Location Parameter." *The Annals of Mathematical Statistics*, **35**(1), 73–101.

Huber P, Ronchetti E (2009). *Robust Statistics*. Wiley Series in Probability and Statistics.

Jain P, Tewari A, Kar P (2014). "On Iterative Hard Thresholding Methods for High-dimensional M-Estimation." In *Advances in Neural Information Processing Systems*, pp. 685–693.

Kane MJ, Emerson J, Weston S (2013). "Scalable Strategies for Computing with Massive Data." *Journal of Statistical Software*, **55**(14), 1–19. URL http://www.jstatsoft.org/v55/i14/.

Krakowski KA, Mahony RE, Williamson RC, Warmuth MK (2007). "A Geometric View of Non-Linear On-Line Stochastic Gradient Descent." *Author website*.

Lambert-Lacroix S, Zwald L, *et al.* (2011). "Robust Regression Through the HuberâĂŹs Criterion and Adaptive Lasso Penalty." *Electronic Journal of Statistics*, **5**, 1015–1053.

Le Cun Y, Bottou L, Bengio Y, Haffner P (1998). "Gradient-based Learning Applied to Document Recognition." *Proceedings of IEEE*, **86**(11), 2278–2324.

Lehmann EL, Casella G (1998). *Theory of Point Estimation*, volume 31. Springer Science & Business Media.

Lewis D, Yang Y, Rose T, Li F (2004). "RCV1: A New Benchmark Collection for Text Categorization Research." *The Journal of Machine Learning Research*, **5**, 361–397.

Li G, Peng H, Zhu L (2011). "Nonconcave Penalized M-estimation with a Diverging Number of Parameters." *Statistica Sinica*, **21**(1), 391.

Lumley T (2013). ***biglm**: Bounded Memory Linear and Generalized Linear Models*. R package version 0.9-1, URL http://CRAN.R-project.org/package=biglm.

Lustig M, Donoho DL, Santos JM, Pauly JM (2008). "Compressed Sensing MRI." *Signal Processing Magazine, IEEE*, **25**(2), 72–82.

National Research Council (2013). *Frontiers in Massive Data Analysis*. The National Academies Press, Washington, DC.

Nelder J, Wedderburn R (1972). "Generalized Linear Models." *Journal of the Royal Statistical Society. Series A (General)*, pp. 370–384.

Nemirovski A, Juditsky A, Lan G, Shapiro A (2009). "Robust Stochastic Approximation Approach to Stochastic Programming." *SIAM Journal on Optimization*, **19**(4), 1574–1609.

Nesterov Y (1983). "A Method of Solving a Convex Programming Problem with Convergence Rate O(1/k2)." In *Soviet Mathematics Doklady*, volume 27, pp. 372–376.

Nui F, Recht B, Re C, Wright SJ (2011). "Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent." In *Advances in Neural Information Processing Systems*.

Owen AB (2007). "A Robust Hybrid of Lasso and Ridge Regression." *Contemporary Mathematics*, **443**, 59–72.

Pascanu R, Mikolov T, Bengio Y (2012). "On the Difficulty of Training Recurrent Neural Networks." *arXiv preprint arXiv:1211.5063*.

Polyak BT (1964). "Some Methods of Speeding Up the Convergence of Iteration Methods." *USSR Computational Mathematics and Mathematical Physics*, **4**(5), 1–17.

Polyak BT, Juditsky AB (1992). "Acceleration of Stochastic Approximation by Averaging." *SIAM Journal on Control and Optimization*, **30**(4), 838–855.

Ranganath R, Tang L, Charlin L, Blei DM (2015). "Deep Exponential Families." In *Artificial Intelligence and Statistics*.

Rezende DJ, Mohamed S (2015). "Variational Inference with Normalizing Flows." In *International Conference on Machine Learning*.

Robbins H, Monro S (1951). "A Stochastic Approximation Method." *The Annals of Mathematical Statistics*, pp. 400–407.

Rockafellar RT (1976). "Monotone Operators and the Proximal Point Algorithm." *SIAM journal on control and optimization*, **14**(5), 877–898.

Ruppert D (1988). "Efficient Estimations from a Slowly Convergent Robbins-Monro Process." *Technical report*, Cornell University Operations Research and Industrial Engineering.

Sakrison DJ (1965). "Efficient Recursive Estimation; Application to Estimating the Parameters of a Covariance Function." *International Journal of Engineering Science*, **3**(4), 461–483.

Schaul T, Antonoglou I, Silver D (2014). "Unit Tests for Stochastic Optimization."

Schmidt M, Le Roux N, Bach F (2013). "Minimizing Finite Sums with the Stochastic Average Gradient." *Technical report*, HAL 00860051.

Shalev-Shwartz S, Zhang T (2013). "Stochastic Dual Coordinate Ascent Methods for Regularized Loss." *The Journal of Machine Learning Research*, **14**(1), 567–599.

Shamir O, Zhang T (2012). "Stochastic Gradient Descent for Non-smooth Optimization: Convergence Results and Optimal Averaging Schemes." *arXiv preprint arXiv:1212.1824*.

Sonnenburg S, Franc V, Yom-Tov E, Sebag M (2008). "Pascal Large Scale Learning Challenge." URL http://largescale.first.fraunhofer.de.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *The Journal of Machine Learning Research*, **15**(1), 1929–1958.

Tieleman T, Hinton G (2012). "Lecture 6.5—RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude." COURSERA: Neural Networks for Machine Learning.

Toulis P, Airoldi E, Rennie J (2014). "Statistical Analysis of Stochastic Gradient Methods for Generalized Linear Models." In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 667–675.

Toulis P, Airoldi EM (2015a). "Implicit Stochastic Gradient Descent." *arXiv preprint arXiv:1408.2923*.

Toulis P, Airoldi EM (2015b). "Scalable Estimation Strategies Based on Stochastic Approximations: Classical Results and New Insights." *Statistics and computing*, **25**(4), 781–795.

Toulis P, Tran D, Airoldi EM (2015). "Stability and Optimality in Stochastic Gradient Descent." *arXiv preprint arXiv:1505.02417v1*.

Tran D, Toulis P, Airoldi EM (2015). ***sgd**: Stochastic Gradient Descent for Scalable Estimation*. R package version 1.0, URL https://github.com/airoldilab/sgd.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer New York. URL http://had.co.nz/ggplot2/book.

Wickham H (2011). "**testthat**: Get Started with Testing." *The R Journal*, **3**, 5–10. URL http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

Xu W (2011). "Towards Optimal One Pass Large Scale Learning with Averaged Stochastic Gradient Descent." *arXiv preprint arXiv:1107.2490*.

Yee TW (2010). "The **VGAM** Package for Categorical Data Analysis." *Journal of Statistical Software*, **32**(10), 1–34.

Yi C (2015). ***hqreg**: Regularization Paths for Huber Loss Regression and Quantile Regression Penalized by Lasso or Elastic-Net*. R package version 1.0, URL http://CRAN.R-project.org/package=hqreg.

Yu H (2002). "**Rmpi**: Parallel Statistical Computing in R." *R News*, **2**(2), 10–14. URL http://cran.r-project.org/doc/Rnews/Rnews_2002-2.pdf.

Zou H, Hastie T (2005). "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **67**(2), 301–320.

Zou H, Hastie T (2012). ***elasticnet**: Elastic-Net for Sparse Estimation and Sparse PCA*. R package version 1.1, URL http://CRAN.R-project.org/package=elasticnet.

Affiliation:

Dustin Tran, Panos Toulis, Edoardo M. Airoldi
Department of Statistics
Harvard University
1 Oxford Street, Cambridge, MA 02138, USA
E-mail: dtran@g.harvard.edu, ptoulis@fas.harvard.edu, airoldi@fas.harvard.edu