DeltaGrad: Rapid retraining of machine learning models

Yinjun Wu [1] **Edgar Dobriban** [2] **Susan B. Davidson** [1]

Abstract

Machine learning models are not static and may need to be retrained on slightly changed datasets, for instance, with the addition or deletion of a set of datapoints. This has many applications, including privacy, robustness, bias reduction, and uncertainty quantification. However, it is expensive to retrain models from scratch. To address this problem, we propose the DeltaGrad algorithm for rapid retraining machine learning models based on information cached during the training phase. We provide both theoretical and empirical support for the effectiveness of DeltaGrad, and show that it compares favorably to the state of the art.

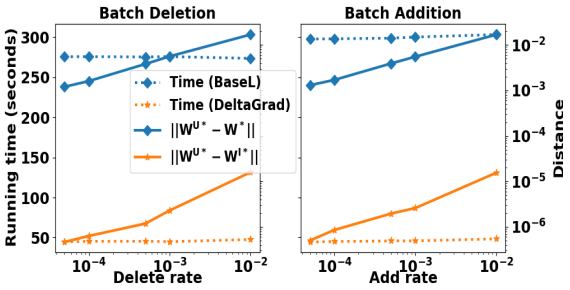

Figure 1. Running time of our DeltaGrad algorithm for retraining a logistic regression model on RCV1 as a function of the fraction of data deleted and added. Our algorithm is faster than training from scratch (Running time BaseL). Also shown is the distance of DeltaGrad and the model trained on full data from the correct values (Distance BaseL and Distance DeltaGrad, resp.), illustrating that our algorithm is accurate. See Section 4.1 for details.

1. Introduction

Machine learning models are used increasingly often, and are rarely static. Models may need to be retrained on slightly changed datasets, for instance when datapoints have been added or deleted. This has many applications, including privacy, robustness, bias reduction, and uncertainty quantification. For instance, it may be necessary to remove certain datapoints from the training data for privacy and robustness reasons. Constructing models with some datapoints removed can also be used for constructing bias-corrected models, such as in jackknife resampling (Quenouille, 1956) which requires retraining the model on all leave-one-out datasets. In addition, retraining models on subsets of data can be used for uncertainty quantification, such as constructing statistically valid prediction intervals via conformal prediction e.g., Shafer & Vovk (2008).

Unfortunately, it is expensive to retrain models from scratch. The most common training mechanisms for large-scale models are based on (stochastic) gradient descent (SGD) and its variants. Retraining the models on a slightly different dataset would involve re-computing the entire optimization path. When adding or removing a small number of data points, this can be of the same complexity as the original training process.

However, we expect models on two similar datasets to be similar. If we retrain the models on many different new datasets, it may be more efficient to cache some information about the training process on the original data, and compute the "updates". Such ideas have been used recently e.g., Ginart et al. (2019); Guo et al. (2019); Wu et al. (2020). However, the existing approaches have various limitations: They only apply to specialized problems such as k-means (Ginart et al., 2019) or logistic regression (Wu et al., 2020), or they require additional randomization leading to non-standard training algorithms (Guo et al., 2019).

To address this problem, we propose the *DeltaGrad* algorithm for rapid retraining of machine learning models when slight changes happen in the training dataset, e.g. deletion or addition of samples, based on information cached during training. DeltaGrad addresses several limitations of prior work: it is applicable to general machine learning models defined by empirical risk minimization trained using SGD, and does not require additional randomization. It is based on the idea of "*differentiating the optimization path*" with respect to the data, and is inspired by ideas from Quasi-Newton

[1]Department of Computer and Information Science, University of Pennsylvania, PA, United States [2]Department of Statistics, University of Pennsylvania, PA, United States. Correspondence to: Yinjun Wu <wuyinjun@seas.upenn.edu>, Edgar Dobriban <dobriban@wharton.upenn.edu>, Susan B. Davidson <susan@cis.upenn.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

methods.

We provide both theoretical and empirical support for the effectivenss of DeltaGrad. We prove that it approximates the true optimization path at a fast rate for strongly convex objectives. We show experimentally that it is accurate and fast on several medium-scale problems on standard datasests, including two-layer neural networks. The speed-ups can be up to 6.5x with negligible accuracy loss (see e.g., Fig. 1). This paves the way toward a large-scale, efficient, general-purpose data deletion/addition machine learning system. We also illustrate how it can be used in several applications described above.

1.1. Related work

There is a great deal of work on model retraining and updating. Recently, this has gotten attention due to worldwide efforts on human-centric AI, data confidentiality and privacy, such as the General Data Protection Regulation (GDPR) in the European Union (European Union, 2016). This mandates that users can ask for their data to be removed from analysis in current AI systems. The required guarantees are thus *stronger* than what is provided by differential privacy (which may leave a non-vanishing contribution of the datapoints in the model, Dwork et al. (2014)), and or defense against data poisoning attacks (which only requires that the performance of the models does not degrade after poisoning, Steinhardt et al. (2017)).

Efficient data deletion is also crucial for many other applications, e.g. *model interpretability* and *model debugging*. For example, repeated retraining by removing different subsets of training data each time is essential in many existing data systems (Doshi-Velez & Kim, 2017; Krishnan & Wu, 2017) to understand the effect of those removed data over the model behavior. It is also close to *deletion diagnotics*, targeting locating the most influential data point for the ML models through deletion in the training set, dating back to (Cook, 1977). Some recent work (Koh & Liang, 2017) targets general ML models, but requires explicitly maintaining Hessian matrices and can only handle the deletion of one sample, thus inapplicable for many large-scale applications.

Efficient model updating for adding and removing datapoints is possible for linear models, based on efficient rank one updates of matrix inverses (e.g., Birattari et al., 1999; Horn & Johnson, 2012; Cao & Yang, 2015, etc). The scope of linear methods is extended if one uses linear feature embeddings, either randomized or learned via pretraining. Updates have been proposed for support vector machines (Syed et al., 1999; Cauwenberghs & Poggio, 2001) and nearest neighbors (Schelter, 2019).

Ginart et al. (2019) propose a definition of data erasure completeness and a quantization-based algorithm for k-means clustering achieving this. They also propose several principles that can enable efficient model updating. Guo et al. (2019) propose a general theoretical condition that guarantees that randomized algorithms can remove data from machine learning models. Their randomized approach needs standard algorithms such as logistic regression to be changed to apply. (Bourtoule et al., 2019) propose the SISA (or Sharded, Isolated, Sliced, Aggregated) training framework for "un-learning", which relies on ideas similar to distributed training. Their approach requires dividing the training data in multiple shards such that a training point is included in a small number of shards only.

Our approach relies on large-scale optimization, which has an enormous literature. Stochastic gradient methods date back to Robbins & Monro (1951). More recently a lot of work (see e.g., Bottou, 1998; 2003; Zhang, 2004; Bousquet & Bottou, 2008; Bottou, 2010; Bottou et al., 2018) focuses on empirical risk minimization.

The convergence proofs for SGD are based on the contraction of the expected residuals. They are based on assumptions such as bounded variances, the strong or weak growth, smoothness, convexity (or Polyak-Lojasiewicz) on the individual and overall loss functions. See e.g., (Gladyshev, 1965; Amari, 1967; Kul'chitskiy & Mozgovoy, 1992; Bertsekas & Tsitsiklis, 1996; Moulines & Bach, 2011; Karimi et al., 2016; Bottou et al., 2018; Gorbunov et al., 2019; Gower et al., 2019), etc, and references therein. Our approach is similar, but the technical details are very different, and more closely related to Quasi-Newton methods such as L-BFGS (Zhu et al., 1997).

Contributions. Our contributions are:

1. **DeltaGrad**: We propose the DeltaGrad algorithm for fast retraining of (stochastic) gradient descent based machine learning models on small changes of the data (small number of added or deleted points).
2. **Theoretical support**: We provide theoretical results showing the accuracy of the DeltaGrad. Both for GD and SGD we show the error is of smaller order than the fraction of points removed.
3. **Empirical results**: We provide empirical results showing the speed and accuracy of DeltaGrad, for addition, removal, and continuous updates, on a number of standard datasets.
4. **Applications**: We describe the applications of DeltaGrad to several problems in machine learning, including privacy, robustness, debiasing, and statistical inference.

2. Algorithms

2.1. Setup

The training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ has n samples. The loss or objective function for a general machine learning model is defined as:

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} F_i(\mathbf{w})$$

where $\mathbf{w}$ represents a vector of the model parameters and $F_i(\mathbf{w})$ is the loss for the i-th sample. The gradient and Hessian matrix of $F(\mathbf{w})$ are

$$\nabla F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla F_i(\mathbf{w}), \quad \mathbf{H}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{H}_i(\mathbf{w})$$

Suppose the model parameter is updated through mini-batch stochastic gradient descent (SGD) for $t = 1, \ldots, T$:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{B}\sum_{i\in\mathscr{B}_t} \nabla F_i(\mathbf{w}_t)$$

where $\mathscr{B}_t$ is a randomly sampled mini-batch of size B and η_t is the learning rate at the t_{th} iteration. As a special case of SGD, the update rule of gradient descent (GD) is $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t/n\sum_{i=1}^{n}\nabla F_i(\mathbf{w}_t)$. After training on the full dataset, the training samples with indices $R = \{i_1, i_2, \ldots, i_r\}$ are removed, where $r \ll n$. Our goal is to efficiently update the model parameter to the minimizer of the new empirical loss. Our algorithm also applies when r new datapoints are *added*.

The naive solution is to apply GD directly over the remaining training samples (we use $\mathbf{w}^U$ to denote the corresponding model parameter), i.e. run:

$$\mathbf{w}^U{}_{t+1} \leftarrow \mathbf{w}^U{}_t - \frac{\eta_t}{n-r}\sum_{i\notin R} \nabla F_i\left(\mathbf{w}^U{}_t\right) \tag{1}$$

which aims to minimize $F^U(\mathbf{w}) = 1/(n-r)\sum_{i\notin R} F_i(\mathbf{w})$.

2.2. Proposed DeltaGrad Algorithm

To obtain a more efficient method, we rewrite Equation (1) via the following *"leave-r-out" gradient formula* (we use $\mathbf{w}^I$ to denote the model parameter derived by DeltaGrad):

$$\mathbf{w}^I{}_{t+1} = \mathbf{w}^I{}_t - \frac{\eta_t}{n-r}\left[n\nabla F\left(\mathbf{w}^I{}_t\right) - \sum_{i\in R}\nabla F_i\left(\mathbf{w}^I{}_t\right)\right]. \tag{2}$$

Computing the sum $\sum_{i\in R}\nabla F_i\left(\mathbf{w}^I{}_t\right)$ of a small number of terms is more efficient than computing $\sum_{i\notin R}\nabla F_i\left(\mathbf{w}^I{}_t\right)$ when $|R| = r \ll n$. For this we need to approximate $n\nabla F\left(\mathbf{w}^I{}_t\right) = \sum_{i=1}^{n}\nabla F_i\left(\mathbf{w}^I{}_t\right)$ by leveraging the historical gradient $\nabla F(\mathbf{w}_t)$ (recall that $\mathbf{w}_t$ is the model parameter before deletions), for each of the T iterations.

Algorithm 1 DeltaGrad

Input : The full training set $(\mathbf{X}, \mathbf{Y})$, model parameters cached during the training phase over the full training samples $\{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_t\}$ and corresponding gradients $\{\nabla F(\mathbf{w}_0), \nabla F(\mathbf{w}_1), \ldots, \nabla F(\mathbf{w}_t)\}$, the indices of the removed training samples R, period T_0, total iteration number T, history size m, "burn-in" iteration number j_0, learning rate η_t

Output : Updated model parameter $\mathbf{w}^I{}_t$

Initialize $\mathbf{w}^I{}_0 \leftarrow \mathbf{w}_0$
Initialize an array $\Delta G = []$
Initialize an array $\Delta W = []$
for $t = 0; t < T; t++$ **do**
  if $[((t - j_0) \mod T_0) == 0]$ *or* $t \le j_0$ **then**
    compute $\nabla F\left(\mathbf{w}^I{}_t\right)$ exactly
    compute $\nabla F\left(\mathbf{w}^I{}_t\right) - \nabla F(\mathbf{w}_t)$ based on the cached gradient $\nabla F(\mathbf{w}_t)$
    set $\Delta G[k] = \nabla F\left(\mathbf{w}^I{}_t\right) - \nabla F(\mathbf{w}_t)$
    set $\Delta W[k] = \mathbf{w}^I{}_t - \mathbf{w}_t$, based on the cached parameters $\mathbf{w}_t$
    $k \leftarrow k + 1$
    compute $\mathbf{w}^I{}_{t+1}$ by using exact GD update (equation (1))
  else
    Pass $\Delta W[-m:]$, $\Delta G[-m:]$, the last m elements in ΔW and ΔG, which are from the $j_1^{th}, j_2^{th}, \ldots, j_m^{th}$ iterations where $j_1 < j_2 < \cdots < j_m$ depend on t, $\mathbf{v} = \mathbf{w}^I{}_t - \mathbf{w}_t$, and the history size m, to the L-BFGS Algorithm (see Section A.2.1 in the Appendix) to get the approximation of $\mathbf{H}(\mathbf{w}_t)\mathbf{v}$, i.e., $\mathbf{B}_{j_m}\mathbf{v}$
    Approximate $\nabla F\left(\mathbf{w}^I{}_t\right) = \nabla F(\mathbf{w}_t) + \mathbf{B}_{j_m}\left(\mathbf{w}^I{}_t - \mathbf{w}_t\right)$
    Compute $\mathbf{w}^I{}_{t+1}$ by using the "leave-r-out" gradient formula, based on the approximated $\nabla F(\mathbf{w}^I{}_t)$
  end
end
return $\mathbf{w}^I{}_t$

Suppose we can *cache* the model parameters $\mathbf{w}_0, \ldots, \mathbf{w}_t$ and the gradients $\nabla F(\mathbf{w}_0), \ldots, \nabla F(\mathbf{w}_t)$ for each iteration of training over the original dataset. Suppose that we have been able to approximate $\mathbf{w}^I{}_0, \ldots, \mathbf{w}^I{}_t$. Then at iteration $t+1$, $\nabla F\left(\mathbf{w}^I{}_t\right)$ can be approximated using the Cauchy mean-value theorem:

$$\nabla F\left(\mathbf{w}^I{}_t\right) = \nabla F(\mathbf{w}_t) + \mathbf{H}_t \cdot \left(\mathbf{w}^I{}_t - \mathbf{w}_t\right) \tag{3}$$

in which $\mathbf{H}_t$ is an integrated Hessian, $\mathbf{H}_t = \int_0^1 \mathbf{H}\left(\mathbf{w}_t + x\left(\mathbf{w}^I{}_t - \mathbf{w}_t\right)\right)dx$.

Equation (3) requires a Hessian-vector product at every iteration. We leverage the L-BFGS algorithm to approximate this, see e.g. Matthies & Strang (1979); Nocedal (1980); Byrd et al. (1994; 1995); Zhu et al. (1997); Nocedal & Wright (2006); Mokhtari & Ribeiro (2015) and references therein. The L-BFGS algorithm uses past data to approximate the projection of the Hessian matrix in the

direction of $\mathbf{w}_{t+1} - \mathbf{w}_t$. We denote the required historical observations at prior iterations j as: $\Delta w_j = \mathbf{w}^I{}_j - \mathbf{w}_j$, $\Delta g_j = \nabla F\ \ \mathbf{w}^I{}_j) - \nabla F(\mathbf{w}_j)$.

L-BFGS computes *Quasi-Hessians* $\mathbf{B}_t$ approximating the true Hessians $\mathbf{H}_t$ (we follow the notations from the classical L-BFGS papers, e.g., Byrd et al. (1994)). DeltaGrad (Algorithm 1) starts with a "burn-in" period of j_0 iterations, where it computes the full gradients $\nabla F\ \ \mathbf{w}^I{}_t)$ exactly. Afterwards, it only computes the full gradients every T_0 iterations. For other iterations t, it uses the L-BGFS algorithm, maintaining a set of updates at some prior iterations j_1, j_2, ..., j_m, i.e. $\Delta w_{j_1}, \Delta w_{j_2}$, ..., Δw_{j_m} and $\Delta g_{j_1}, \Delta g_{j_2}, \ldots, \Delta g_{j_m}$ where $j_k - j_{k-1} \leq T_0$. Then it uses an efficient L-BGFS update from Byrd et al. (1994) (see Appendix A.2.1 for the details of the L-BGFS update).

By approximating $\mathbf{H}_t$ with $\mathbf{B}_t$ in Equation (3) and plugging Equation (3) into Equation (2), the DeltaGrad update is: $\mathbf{w}^I{}_{t+1} - \mathbf{w}^I{}_t = \eta_t/(n-r)\cdot$

$$\cdot \begin{cases} \sum_{i \notin R} \nabla F(\mathbf{w}^I{}_t), \ (t - j_0) \mod T_0 = 0 \text{ or } t \leq j_0 \\ n[\mathbf{B}_{j_m}(\mathbf{w}^I{}_t - \mathbf{w}_t) + \nabla F(\mathbf{w}_t)] - \sum_{i \in R} \nabla F(\mathbf{w}^I{}_t), \text{ else} \end{cases}$$

2.3. Convergence rate for strongly convex objectives

We provide the convergence rate of DeltaGrad for strongly convex objectives in Theorem 1. We need to introduce some assumptions. The norm used throughout the rest of the paper is ℓ_2 norm.

Assumption 1 (Small number of samples removed)**.** *The number of removed samples, r, is far smaller than the total number of training samples, n. There is a small constant $\delta > 0$ such that $r/n \leq \delta$.*

Assumption 2 (Strong convexity and smoothness)**.** *Each $F_i(\boldsymbol{w})$ ($i = 1, 2, \ldots, n$) is $\mu-$strongly convex and L-smooth with $\mu > 0$, so for any $\boldsymbol{w}_1, \boldsymbol{w}_2$*

$$(\nabla F_i(\boldsymbol{w}_1) - \nabla F_i(\boldsymbol{w}_2))^T (\boldsymbol{w}_1 - \boldsymbol{w}_2) \geq \mu \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|^2,$$

$$\|\nabla F_i(\boldsymbol{w}_1) - \nabla F_i(\boldsymbol{w}_2)\| \leq L \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|.$$

Then $F(\mathbf{w})$ and $F^U(\mathbf{w})$ are L-smooth and μ-strongly convex. Typical choices of η_t are based on the smoothness and strong convexity parameters, so the same choices lead to the convergence for both $\mathbf{w}_t$ and $\mathbf{w}^U{}_t$. For instance, GD over a strongly convex objective with fixed step size $\eta_t = \eta \leq 2/[L + \mu]$ converges geometrically at rate $(L-\mu)/(L+\mu) < 1$. For simplicity, we will use a constant learning rate $\eta_t = \eta \leq 2/[L + \mu]$.

We assume bounded gradients and Lipschitz Hessians, which are standard (Boyd & Vandenberghe, 2004; Bottou et al., 2016). The proof may be relaxed to weak growth conditions, see the related works for references.

Assumption 3 (Bounded gradients)**.** *For any model parameter $\boldsymbol{w}$ in the sequence $[\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_t, \ldots]$, the norm of the gradient at every sample is bounded by a constant c_2, i.e. for all i, j:*

$$\|\nabla F_i(\boldsymbol{w}_j)\| \leq c_2.$$

Assumption 4 (Lipschitz Hessian)**.** *The Hessian $\boldsymbol{H}(\boldsymbol{w})$ is Lipschitz continuous. There exists a constant c_0 such that for all $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$,*

$$\|\boldsymbol{H}(\boldsymbol{w}_1) - \boldsymbol{H}(\boldsymbol{w}_2)\| \leq c_0 \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|.$$

An assumption specific to Quasi-Newton methods is the *strong independence* of the weight updates: the smallest singular value of the normalized weight updates is bounded away from zero (Ortega & Rheinboldt, 1970; Conn et al., 1991). This has sometimes been motivated empirically, as the iterates of certain quasi-newton iterations empirically satisfy it (Conn et al., 1988).

Assumption 5 (Strong independence)**.** *For any sequence, $[\Delta w_{j_1}, \Delta w_{j_2}, \ldots, \Delta w_{j_m}]$, the matrix of normalized vectors*

$$\Delta W_{j_1, j_2, \ldots, j_m} = [\Delta w_{j_1}, \Delta w_{j_2}, \ldots, \Delta w_{j_m}]/s_{j_1, j_m}$$

where $s_{j_1,j_m} = \max(\|\Delta w_{j_1}\|, \|\Delta w_{j_2}\|, \ldots, \|\Delta w_{j_m}\|)$, has its minimum singular value $\sigma_{\min}$ bounded away from zero. We have $\sigma_{\min}(\Delta W_{j_1,j_2,\ldots,j_m}) \geq c_1$ where c_1 is independent of $(j_1, j_2, \ldots, j_m)$.

Empirically, we find c_1 around 0.2 for the MNIST dataset using our default hyperparameters.

2.3.1. Results

Then our first main result is the convergence rate of the DeltaGrad algorithm.

Theorem 1 (Bound between true and incrementally updated iterates)**.** *For a large enough iteration counter t, the result $\mathbf{w}^I{}_t$ of DeltaGrad (Algorithm 1) approximates the correct iteration values $\mathbf{w}^U{}_t$ at the rate*

$$\|\mathbf{w}^U{}_t - \mathbf{w}^I{}_t\| = o\left(\frac{r}{n}\right).$$

So $\|\mathbf{w}^U{}_t - \mathbf{w}^I{}_t\|$ is of a lower order than r/n.

The baseline error rate between the full model parameters $\mathbf{w}_t$ and $\mathbf{w}^I{}_t$ is expected to be of the order r/n, as can be seen from the example of the sample mean. This shows that DeltaGrad has a better convergence rate for approximating $\mathbf{w}^I{}_t$. The proof is quite involved. It relies on a delicate analysis of the difference between the approximate Hessians $\mathbf{B}_t$ and the true Hessians $\mathbf{H}_t$ (see the Appendix, and specifically A.2).

2.4. Complexity analysis

We will do our complexity analysis assuming that the model is given by a computation graph. Suppose the number of model parameters is p and the time complexity for forward propagation is $f(p)$. Then according to the Baur-Strassen theorem (Griewank & Walther, 2008), the time complexity of backpropagation in one step will be at most $5f(p)$ and thus the total complexity to compute the derivatives for each training sample is $6f(p)$. Plus, the overhead of computing the product of $\mathbf{B}_{j_m}(\mathbf{w}^I{}_t - \mathbf{w}_t)$ is $O(m^3) + 6mp + p$ according to (Byrd et al., 1994), which means that the total time complexity at the step where the gradients are approximated is $6rf(p) + O(m^3) + 6mp + p$ (the gradients of r removed/added samples are explicitly evaluated), which is more efficient than explicit computation of the gradients over the full batch (a time complexity of $6(n-r)f(p)$) when $r \ll n$.

Suppose there are T iterations in the training process. Then the running time of BaseL will be $6(n-r)f(p)T$. DeltaGrad evaluates the gradients for the first j_0 iterations and once every T_0 iterations. So its total running time is $6(n-r)f(p) \times \frac{T-j_0}{T_0} + (6rf(p) + O(m^3) + 6mp + p) \times (1 - \frac{1}{T_0})(T - j_0)$, which is close to $6nf(p) \times \frac{T-j_0}{T_0} + (O(m^3) + 6mp + p) \times (1 - \frac{1}{T_0})(T - j_0)$ since r is small. Also, when n is large, the overhead of approximate computation, i.e. $(O(m^3) + 6mp + p)$ should be much smaller than that of explicit computation. Thus *speed-ups of a factor T_0 are expected* when j_0 is far smaller than T.

3. Extension to SGD

Consider now mini-batch stochastic gradient descent:

$$\mathbf{w}^S{}_{t+1} = \mathbf{w}^S{}_t - \frac{\eta}{B} \sum_{i \in \mathcal{B}_t} \nabla F_i(\mathbf{w}^S{}_t).$$

The naive solution for retraining the model is:

$$\mathbf{w}^{U,S}{}_{t+1} = \mathbf{w}^{U,S}{}_t - \frac{\eta}{B - \Delta B_t} \sum_{i \in \mathcal{B}_t, i \notin R} \nabla F_i(\mathbf{w}^{U,S}{}_t).$$

Here ΔB_t is the size of the subset removed from the t-th minibatch. If $B - \Delta B_t = 0$, then we do not change the parameters at that iteration. DeltaGrad can be naturally extended to this case: $\mathbf{w}^{I,S}{}_{t+1} - \mathbf{w}^{I,S}{}_t = \eta_t/(B - \Delta B_t) \cdot$

$$\cdot \begin{cases} \sum_{i \notin R} \nabla F(\mathbf{w}^I{}_t), \ t \mod T_0 = 0 \ or \ t \le j_0 \\ [B(\mathbf{B}_{j_m}(\mathbf{w}^I{}_t - \mathbf{w}_t)) - \sum_{i \in R} \nabla F(\mathbf{w}^I{}_t)], \ else \end{cases}$$

which relies on a series of historical observations: $\Delta w^S{}_j = \mathbf{w}^{I,S}{}_j - \mathbf{w}^S{}_j$, $\Delta g^S{}_j = B^{-1} \sum_{i \in \mathcal{B}_j} \nabla F_i(\mathbf{w}^{I,S}{}_j) - B^{-1} \sum_{i \in \mathcal{B}_j} \nabla F_i(\mathbf{w}^S{}_j)$.

3.1. Convergence rate for strongly convex objectives

Recall B is the mini-batch size, p is the total number of model parameters and T is the number of iterations in SGD. Our main result for SGD is the following.

Theorem 2 (SGD bound for DeltaGrad)**.** *With probability at least*

$$1 - T \cdot [2p \exp\left(- \frac{\log(2p)\sqrt{B}}{4 + \frac{2}{3}\left(\frac{\log^2(2p)}{B}\right)^{1/4}} \right) + (p+1)\exp(-\frac{\log(p+1)\sqrt{B}}{4 + \frac{2}{3}\left(\frac{(\log(p+1))^2}{B}\right)^{1/4}}) + 2\exp(-2\sqrt{B})],$$

the result $\mathbf{w}^{I,S}{}_t$ of Algorithm 1 approximates the correct iteration values $\mathbf{w}^{U,S}{}_t$ at the rate

$$\|\mathbf{w}^{U,S}{}_t - \mathbf{w}^{I,S}{}_t\| = o\left(\frac{r}{n} + \frac{1}{B^{\frac{1}{4}}}\right).$$

Thus, when B is large, and when r/n is small, our algorithm accurately approximates the correct iteration values.

Its proof is in the Appendix (Section A.3).

4. Experiments

4.1. Experimental setup

Datasets. We used four datasets for evaluation: MNIST (LeCun et al., 1998), covtype (Blackard & Dean, 1999), HIGGS (Baldi et al., 2014) and RCV1 (Lewis et al., 2004) [1] . MNIST contains 60,000 images as the training dataset and 10,000 images as the test dataset; each image has 28×28 features (pixels), containing one digit from 0 to 9. The covtype dataset consists of 581,012 samples with 54 features, each of which may come from one of the seven forest cover types; as a test dataset, we randomly picked 10% of the data. HIGGS is a dataset produced by Monte Carlo simulations for binary classification, containing 21 features with 11,000,000 samples in total; 500,000 samples are used as the test dataset. RCV1 is a corpus dataset; we use its binary version which consists of 679,641 samples and 47,236 features, of which the first 20,242 samples are used for training.

Machine configuration. All experiments are run over a GPU machine with one Intel(R) Core(TM) i9-9920X CPU with 128 GB DRAM and 4 GeForce 2080 Titan RTX GPUs (each GPU has 10 GB DRAM). We implemented DeltaGrad with PyTorch 1.3 and used one GPU for accelerating the tensor computations.

Deletion/Addition benchmark. We run regularized logistic regression over the four datasets with L2 norm coefficient 0.005, fixed learning rate 0.1. The mini-batch sizes for RCV1 and other three datasets are 16384 and 10200 respectively (Recall that RCV1 only has around 20k training

[1] We used its binary version from LIBSVM: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#rcv1.binary`

samples). We also evaluated our approach over a two-layer neural network with 300 hidden ReLU neurons over MNIST. There L2 regularization with rate 0.001 is added along with a decaying learning rate (first 10 iterations with learning rate 0.2 and the rest with learning rate 0.1) and with deterministic GD. There are no strong convexity or smoothness guarantees for DNNs. Therefore, we adjusted Algorithm 1 to fit general DNN models (see Algorithm 3 in the Appendix C.3). In Algorithm 1, we assume that the convexity holds locally where we use the L-BFGS algorithm to estimate the gradients. For all the other regions, we explicitly evaluate the gradients. The details on how to check which regions satisfy the convexity for DNN models can be found in Algorithm 3. We also explore the use of DeltaGrad for more complicated neural network models such as ResNet by reusing and fixing the pre-trained parameters in all but the last layer during the training phase, presented in detail in Appendix D.4.

We evaluate two cases of addition/deletion: *batch* and *online*. Multiple samples are grouped together for addition and deletion in the former, while samples are removed one after another in the latter. Algorithm 1 is slightly modified to fit the online deletion/addition cases (see Algorithm 2 in Appendix C.2). In what follows, unless explicitly specified, Algorithm 1 and Algorithm 2 are used for experiments in the *batch* addition/deletion case and *online* addition/deletion case respectively.

To simulate deleting training samples, $\mathbf{w}^*$ is evaluated over the full training dataset of n samples, which is followed by the random removal of r samples and evaluation over the remaining $n-r$ samples using BaseL or DeltaGrad. To simulate adding training samples, r samples are deleted first. After $\mathbf{w}^*$ is evaluated over the remaining $n-r$ samples, the r samples are added back to the training set for updating the model. The ratio of r to the total number of training samples n is called the *Delete rate* and *Add rate* for the two scenarios, respectively.

Throughout the experiments, the running time of BaseL and DeltaGrad to update the model parameters is recorded. To show the difference between $\mathbf{w}^{U*}$ (the output of BaseL, and the correct model parameters after deletion or addition) and $\mathbf{w}^{I*}$ (the output of DeltaGrad), we compute the ℓ_2-norm or distance $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$. For comparison and justifying the theory in Section 2.3, $\|\mathbf{w}^* - \mathbf{w}^{U*}\|$ is also recorded ($\mathbf{w}^*$ are the parameters trained over the full training data). Given the same set of added or deleted samples, the experiments are repeated 10 times, with different minibatch randomness each time. After the model updates, $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ are evaluated over the test dataset and their prediction performance is reported.

Hyperparameter setup. We set T_0 (the period of explicit gradient updates) and j_0 (the length of the inital "burn-in") as follows. For regularized logistic regression, we set $T_0 = 10, j_0 = 10$ for RCV1, $T_0 = 5, j_0 = 10$ for MNIST and covtype, and $T_0 = 3, j_0 = 300$ for HIGGS. For the 2-layer DNN, $T_0 = 2$ is even smaller and the first quarter of the iterations are used as "burn-in". The history size m is 2 for all experiments. The effect of hyperparameters and suggestions on how to choose them is discussed in the Appendix D.2.

4.2. Experimental results

4.2.1. Batch addition/deletion.

To test the robustness and efficiency of DeltaGrad in batch deletion, we vary the *Delete* and *Add rate* from 0 to 0.01. The first three sub-figures in Figures 2 and 3 along with Figure 1 show the running time of BaseL and DeltaGrad (blue and red dotted lines, resp.) and the two distances, $\|\mathbf{w}^{U*} - \mathbf{w}^*\|$ and $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ (blue and red solid lines, resp.) over the four datasets using regularized logistic regression. The results on the use of 2-layer DNN over MNIST are presented in the last sub-figures in Figures 2 and 3, which are denoted by MNIST^n.

The running time of BaseL and DeltaGrad is almost constant regardless of the delete or add rate, confirming the time complexity analysis of DeltaGrad in Section 2.4. The theoretical running time is free of the number of removed samples r, when r is small. For any given delete/add rate, DeltaGrad achieves significant speed-ups (up to 2.6x for MNIST, 2x for covtype, 1.6x for HIGGS, 6.5x for RCV1) compared to BaseL. On the other hand, the distance between $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ is quite small; it is less than 0.0001 even when up to 1% of samples are removed or added. When the delete or add rate is close to 0, $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ is of magnitude 10^{-6} (10^{-8} for RCV1), indicating that the approximation brought by $\mathbf{w}^{I*}$ is negligible. Also, $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ is at least one order of magnitude smaller than $\|\mathbf{w}^{U*} - \mathbf{w}^*\|$, confirming our theoretical analysis comparing the bound of $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ to that of $\|\mathbf{w}^{U*} - \mathbf{w}^*\|$.

To investigate whether the tiny difference between $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ will lead to any difference in prediction behavior, the prediction accuracy using $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ is presented in Table 1. Due to space limitations, only results on a very small (0.005%) and the largest (1%) add/delete rates are presented. Due to the randomness in SGD, the standard deviation for the prediction accuracy is also presented. In most cases, the models produced by BaseL and DeltaGrad end up with effectively the same prediction power. There are a few cases where the prediction results of $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ are not exactly the same (e.g. Add (1%) over MNIST), their confidence intervals overlap, so that statistically $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ provide the same prediction results.

For the 2-layer net model where strong convexity does not

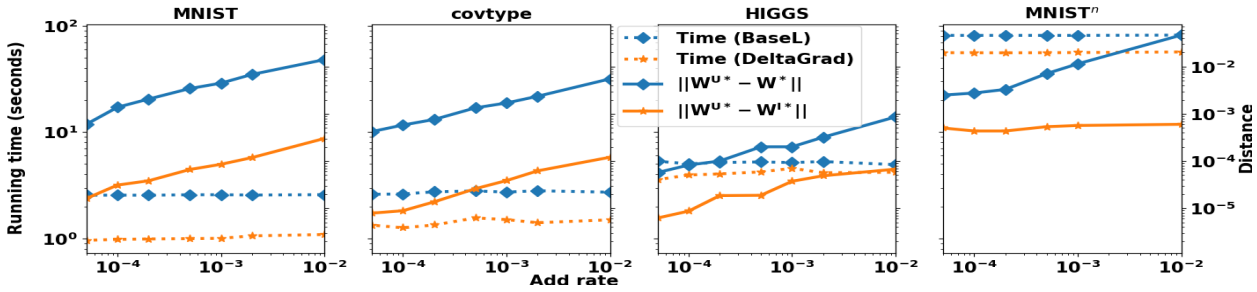

Figure 2. Running time and distance with varied add rate

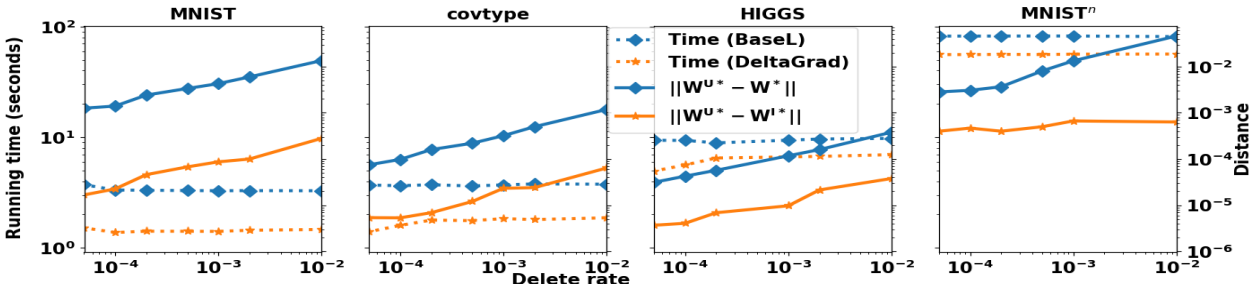

Figure 3. Running time and distance with varied delete rate

Table 1. Prediction accuracy of BaseL and DeltaGrad with batch addition/deletion. MNIST^n refers to MNIST with a neural net.

Dataset		BaseL(%)	DeltaGrad(%)
Add (0.005%)	MNIST	87.530 ± 0.0025	87.530 ± 0.0025
	MNIST^n	92.340 ± 0.002	92.340 ± 0.002
	covtype	62.991 ± 0.0027	62.991 ± 0.0027
	HIGGS	55.372 ± 0.0002	55.372 ± 0.0002
	RCV1	92.222 ± 0.00004	92.222 ± 0.00004
Add (1%)	MNIST	87.540 ± 0.0011	87.542 ± 0.0011
	MNIST^n	92.397 ± 0.001	92.397 ± 0.001
	covtype	63.022 ± 0.0008	63.022 ± 0.0008
	HIGGS	55.381 ± 0.0007	55.380 ± 0.0007
	RCV1	92.233 ± 0.00010	92.233 ± 0.00010
Delete (0.005%)	MNIST	86.272 ± 0.0035	86.272 ± 0.0035
	MNIST^n	92.203 ± 0.004	92.203 ± 0.004
	covtype	62.966 ± 0.0017	62.966 ± 0.0017
	HIGGS	52.950 ± 0.0001	52.950 ± 0.0001
	RCV1	92.241 ± 0.00004	92.241 ± 0.00004
Delete (1%)	MNIST	86.082 ± 0.0046	86.074 ± 0.0048
	MNIST^n	92.373 ± 0.003	92.370 ± 0.003
	covtype	62.943 ± 0.0007	62.943 ± 0.0007
	HIGGS	52.975 ± 0.0002	52.975 ± 0.0002
	RCV1	92.203 ± 0.00007	92.203 ± 0.00007

hold, we use the variant of DeltaGrad$\tilde{m}$entioned above, i.e. Algorithm 3. See the last sub-figures in Figure 2 and 3. The figures show that DeltaGrad achieves about 1.4x speedup compared to BaseL while maintaining a relatively small difference between $\mathbf{w}^{I*}$ and $\mathbf{w}^{U*}$. This suggests that it may be possible to extend our analysis for DeltaGrad beyond strong convexity; this is left for future work.

4.2.2. Online addition/deletion.

To simulate deletion and addition requests over the training data continuously in an on-line setting, 100 random selected samples are added or deleted sequentially. Each triggers model updates by either BaseL or DeltaGrad. The running time comparison between the two approaches in this experiment is presented in Figure 4, which shows that DeltaGrad is about 2.5x, 2x, 1.8x and 6.5x faster than BaseL on MNIST, covtype, HIGGS and RCV1 respectively. The accuracy comparison is shown in Table 2. There is essentially no prediction performance difference between $\mathbf{w}^{U*}$ and $\mathbf{w}^*$.

Discussion. By comparing the speed-ups brought by DeltaGrad and the choice of T_0, we found that the theoretical speed-ups are not fully achieved. One reason is that in the approximate L-BFGS computation, a series of small matrix multiplications are involved. Their computation on GPU vs CPU cannot bring about very significant speed-ups compared to the larger matrix operations[2], which indicates that the overhead of L-BFGS is non-negligible compared to gradient computation. Besides, although r is far smaller than n, to compute the gradients over the r samples, other overhead becomes more significant: copying data from CPU DRAM

[2]See the matrix computation benchmark on GPU with varied matrix sizes: https://developer.nvidia.com/cublas

Table 2. Distance and prediction performance of BaseL and DeltaGrad in online deletion/addition

Dataset	Distance		Prediction accuracy (%)	
	$\|\mathbf{w}^{U*} - \mathbf{w}^*\|$	$\|\mathbf{w}^{I*} - \mathbf{w}^{U*}\|$	BaseL	DeltaGrad
MNIST (Addition)	5.7×10^{-3}	2×10^{-4}	87.548 ± 0.0002	87.548 ± 0.0002
MNIST (Deletion)	5.0×10^{-3}	1.4×10^{-4}	87.465 ± 0.002	87.465 ± 0.002
covtype (Addition)	8.0×10^{-3}	2.0×10^{-5}	63.054 ± 0.0007	63.054 ± 0.0007
covtype (Deletion)	7.0×10^{-3}	2.0×10^{-5}	62.836 ± 0.0002	62.836 ± 0.0002
HIGGS (Addition)	2.1×10^{-5}	1.4×10^{-6}	55.303 ± 0.0003	55.303 ± 0.0003
HIGGS (Deletion)	2.5×10^{-5}	1.7×10^{-6}	55.333 ± 0.0008	55.333 ± 0.0008
RCV1 (Addition)	0.0122	3.6×10^{-6}	92.255 ± 0.0003	92.255 ± 0.0003
RCV1 (Deletion)	0.0119	3.5×10^{-6}	92.229 ± 0.0006	92.229 ± 0.0006

to GPU DRAM, the time to launch the kernel on GPU, etc. This leads to non-negligible explicit gradient computation cost over the r samples. It would be interesting to explore how to adjust DeltaGrad to fully utilize the computation power of GPU in the future.

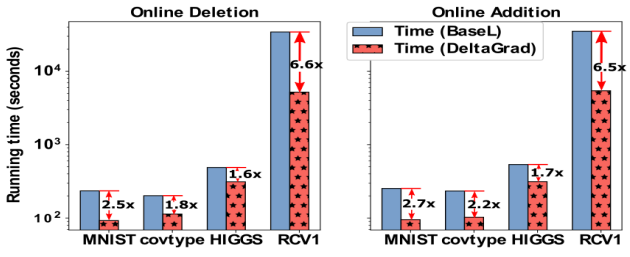

Figure 4. Running time comparison of BaseL and DeltaGrad with 100 continuous deletions/addition

Other experiments with DeltaGrad are in the Appendix (Section D): evaluations with larger delete rate (i.e. when $r \ll n$ may not hold), comparisons with state-of-the-art work and studies on the effect of mini-batch sizes and hyper-parameters etc.

5. Applications

Our algorithm has many applications, including privacy related data deletion, continuous model updating, robustness, bias reduction, and uncertainty quantification (predictive inference). Some of these applications are quite direct, and so for space limitations we only briefly describe them. Some initial experimental results on how our method can accelerate some of those applications such as robust learning are included in Appendix D.5.

5.1. Privacy related data deletion

By adding a bit of noise one can often guarantee differential privacy, the impossibility to distinguish the presence or absence of a datapoint from the output of an algorithm (Dwork et al., 2014). We leverage and slightly extend a closely related notion, approximate data deletion, (Ginart et al., 2019) to guarantee private deletion.

We will consider learning algorithms A that take as input a dataset D, and output a model $A(D)$ in the hypothesis space $\mathcal{H}$. With the i-th sample removed, the resulting model is thus $A(D_{-i})$. A data deletion operation R_A maps D, $A(D)$ and the index of the removed sample i to the model $R_A(D, A(D), i)$. We call R_A an ϵ-approximate deletion if for all D and measurable subsets $S \subset \mathcal{H}$:

$$|\log \frac{P(A(D_{-i}) \in S|D_{-i})}{P(R_A(D, A(D), i) \in S|D_{-i})}| \leq \epsilon$$

Here if either of the two probabilities is zero, the other must be zero too. Using the standard Laplace mechanism (Dwork et al., 2014), we can make the output of our algorithm an ϵ-approximate deletion. We add independent $Laplace\ (\delta/\epsilon)$ noise to each coordinate of $\mathbf{w}^*$, $\mathbf{w}^{U^*}$ and $\mathbf{w}^{I^*}$, where

$$\delta = \frac{\sqrt{p} A M_1^2 r^2}{\eta(\frac{1}{2}\mu - \frac{r}{n-r}\mu - \frac{c_0 M_1 r}{2n})^2 (n-r)(n/2-r)}$$

is an upper bound on $p^{1/2}\|\mathbf{w}^{U^*} - \mathbf{w}^{I^*}\|$. See the appendix for details.

5.2. Continous model updating

Continous model updating is a direct application. In many cases, machine learning models run in production need to be retrained on newly acquired data. DeltaGrad can be used to update the models. Similarly, if there are changes in the data, then we can run DeltaGrad twice: first to remove the original data, then to add the changed data.

5.3. Robustness

Our method has applications to robust statistical learning. The basic idea is that we can identify outliers by fitting a preliminary model. Then we can prune them and re-fit the model. Methods based on this idea are some of the most statistically efficient ones for certain problems, see e.g., the review Yu & Yao (2017).

5.4. Data valuation

Our method can be also used to evaluate the importance of training samples (see Cook (1977) and the follow-up works such as Ghorbani & Zou (2019)). One common method to do this is the *leave-one-out* test, i.e. comparing the difference of the model parameters before and after the deletion of one single training sample of interest. Our method is thus useful to speed up evaluating the model parameters after the deletion operations.

5.5. Bias reduction

Our algorithms can be used directly to speed up existing techniques for bias correction. There are many different techniques based on *subsampling* (Politis et al., 1999). A basic one is the *jackknife* (Quenouille, 1956). Suppose we have an estimator $\hat{f}_n$ computed based on n training datapoints, and defined for both n and $n-1$. The jackknife bias-correction is $\hat{f}_{jack} = \hat{f}_n - \hat{b}(\hat{f}_n)$ where $\hat{b}(\hat{f}_n)$ is the jackknife estimator of the bias $b(\hat{f}_n)$ of the estimator $\hat{f}_n$. This is constructed as $\hat{b}(\hat{f}_n) = (n-1)\left(n^{-1}\sum_{i=1}^{n}\hat{f}_{-i} - \hat{f}_n\right)$ where $\hat{f}_{-i}$ is the estimator $\hat{f}_{n-1}$ computed on the training data removing the i-th data point. Our algorithm can be used to recompute the estimator on all subsets of size $n-1$ of the training data. To validate that this works, a good example may be logistic regression with n not much larger than p, which will have bias (Sur & Candès, 2018).

5.6. Uncertainty quantification / Predictive inference

Our algorithm has applications to uncertainty quantification and predictive inference. These are fundamental problems of wide applicability. Techniques based on conformal prediction (e.g., Shafer & Vovk, 2008) rely on retraining models on subsets of the data. As an example, in cross-conformal prediction (Vovk, 2015) we have a predictive model $\hat{f}$ that can be trained on any subset of the data. We can split the data into K subsets of roughly equal size. We can train $\hat{f}_{-S_k}$ on the data excluding S_k, and compute the cross-validation residuals $R_i = |\mathbf{y}_i - \hat{f}_{-S_k}(\mathbf{x}_i)|$ for $i \in S_i$. Then for a test datapoint $\mathbf{x}_{n+1}$, we form a prediction set $C(\mathbf{x}_{n+1})$ with all $\mathbf{y}$ overlapping $n - (1-\alpha)(n+1)$ of the intervals $\hat{f}_{-S_k}(\mathbf{x}_{n+1}) \pm R_i$. This forms a valid $\beta = 1 - 2\alpha - 2K/n$ level prediction set in the sense that $P(\mathbf{y}_{n+1} \in C(\mathbf{x}_{n+1})) \geq \beta$ over the randomness in all samples. The "best" (shortest) intervals arise for large K, which means a small number of samples is removed to find f_{-S_k}. Thus our algorithm is applicable.

6. Conclusion

In this work, we developed the efficient DeltaGrad retraining algorithm after slight changes (deletions/additions) of the training dataset by differentiating the optimization path with Quasi-Newton method. This is provably more accurate than the baseline of retraining from scratch. Its performance advantage has been empirically demonstrated with some medium-scale public datasets, revealing its great potential in constructing data deletion/addition machine learning systems for various applications. The code for replicating our experiments is available on Github: `https://github.com/thuwuyinjun/DeltaGrad`. Adjusting DeltaGrad to handle smaller mini-batch sizes in SGD and more complicated ML models without strong convexity and smoothness guarantees is important future work.

Acknowledgements

This material is based upon work that is in part supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0047. Partial support was provided by NSF Awards 1547360 and 1733794.

References

Amari, S. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, (3):299–307, 1967.

Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

Bertsekas, D. P. and Tsitsiklis, J. N. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

Birattari, M., Bontempi, G., and Bersini, H. Lazy learning meets the recursive least squares algorithm. In *Advances in neural information processing systems*, pp. 375–381, 1999.

Blackard, J. A. and Dean, D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

Bottou, L. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

Bottou, L. Stochastic learning. In *Summer School on Machine Learning*, pp. 146–168. Springer, 2003.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. Machine unlearning. *arXiv preprint arXiv:1912.03817*, 2019.

Bousquet, O. and Bottou, L. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pp. 161–168, 2008.

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

Byrd, R. H., Nocedal, J., and Schnabel, R. B. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1-3): 129–156, 1994.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

Cao, Y. and Yang, J. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pp. 463–480. IEEE, 2015.

Cauwenberghs, G. and Poggio, T. Incremental and decremental support vector machine learning. In *Advances in neural information processing systems*, pp. 409–415, 2001.

Conn, A. R., Gould, N. I., and Toint, P. L. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of computation*, 50 (182):399–430, 1988.

Conn, A. R., Gould, N. I., and Toint, P. L. Convergence of quasi-newton matrices generated by the symmetric rank one update. *Mathematical programming*, 50(1-3): 177–195, 1991.

Cook, R. D. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977.

Doshi-Velez, F. and Kim, B. A roadmap for a rigorous science of interpretability. *arXiv preprint arXiv:1702.08608*, 150, 2017.

Dwork, C., Roth, A., et al. The algorithmic foundations of differential privacy. *Foundations and Trends R in Theoretical Computer Science*, 9(3–4):211–407, 2014.

European Union, C. o. Council regulation (eu) no 2016/679. 2016. URL `https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:02016R0679-20160504`.

Ghorbani, A. and Zou, J. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pp. 2242–2251, 2019.

Ginart, A., Guan, M., Valiant, G., and Zou, J. Y. Making ai forget you: Data deletion in machine learning. In *Advances in Neural Information Processing Systems*, pp. 3513–3526, 2019.

Gladyshev, E. On stochastic approximation. *Theory of Probability & Its Applications*, 10(2):275–278, 1965.

Gorbunov, E., Hanzely, F., and Richtárik, P. A unified theory of sgd: Variance reduction, sampling, quantization and coordinate descent. *arXiv preprint arXiv:1905.11261*, 2019.

Gower, R. M., Loizou, N., Qian, X., Sailanbayev, A., Shulgin, E., and Richtárik, P. Sgd: General analysis and improved rates. *arXiv preprint arXiv:1901.09401*, 2019.

Griewank, A. and Walther, A. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

Guo, C., Goldstein, T., Hannun, A., and van der Maaten, L. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.

Horn, R. A. and Johnson, C. R. *Matrix analysis*. Cambridge university press, 2012.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 795–811. Springer, 2016.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1885–1894. JMLR. org, 2017.

Krishnan, S. and Wu, E. Palm: Machine learning explanations for iterative debugging. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, pp. 4. ACM, 2017.

Kul'chitskiy, O. Y. and Mozgovoy, A. Estimation of convergence rate for robust identification algorithms. *International journal of adaptive control and signal processing*, 6(3):247–251, 1992.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

Matthies, H. and Strang, G. The solution of nonlinear finite element equations. *International journal for numerical methods in engineering*, 14(11):1613–1626, 1979.

Mokhtari, A. and Ribeiro, A. Global convergence of online limited memory bfgs. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

Moulines, E. and Bach, F. R. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pp. 451–459, 2011.

Nocedal, J. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

Nocedal, J. and Wright, S. *Numerical optimization*. Springer Science & Business Media, 2006.

Ortega, J. M. and Rheinboldt, W. C. *Iterative solution of nonlinear equations in several variables*, volume 30. Siam, 1970.

Politis, D. N., Romano, J. P., and Wolf, M. *Subsampling*. Springer Science & Business Media, 1999.

Quenouille, M. H. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Schelter, S. amnesia–towards machine learning models that can forget user data very fast. In *1st International Workshop on Applied AI for Database Systems and Applications (AIDB19)*, 2019.

Shafer, G. and Vovk, V. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

Steinhardt, J., Koh, P. W. W., and Liang, P. S. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*, pp. 3517–3529, 2017.

Sur, P. and Candès, E. J. A modern maximum-likelihood theory for high-dimensional logistic regression. *arXiv preprint arXiv:1803.06964*, 2018.

Syed, N. A., Huan, S., Kah, L., and Sung, K. Incremental learning with support vector machines. 1999.

Vovk, V. Cross-conformal predictors. *Annals of Mathematics and Artificial Intelligence*, 74(1-2):9–28, 2015.

Wu, Y., Tannen, V., and Davidson, S. B. Priu: A provenance-based approach for incrementally updating regression models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 447–462, 2020.

Yu, C. and Yao, W. Robust linear regression: A review and comparison. *Communications in Statistics-Simulation and Computation*, 46(8):6261–6282, 2017.

Zhang, T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 116. ACM, 2004.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.