Adaptive Second Order Coresets for Data-efficient Machine Learning

Omead Pooladzandi [1] **David Davini** [2] **Baharan Mirzasoleiman** [2]

Abstract

Training machine learning models on massive datasets incurs substantial computational costs. To alleviate such costs, there has been a sustained effort to develop data-efficient training methods that can carefully select subsets of the training examples that generalize on par with the full training data. However, existing methods are limited in providing theoretical guarantees for the quality of the models trained on the extracted subsets, and may perform poorly in practice. We propose ADACORE, a method that leverages the geometry of the data to extract subsets of the training examples for efficient machine learning. The key idea behind our method is to dynamically approximate the curvature of the loss function via an exponentially-averaged estimate of the Hessian to select weighted subsets (coresets) that provide a close approximation of the full gradient preconditioned with the Hessian. We prove rigorous guarantees for the convergence of various first and second-order methods applied to the subsets chosen by ADACORE. Our extensive experiments show that ADACORE extracts coresets with higher quality compared to baselines and speeds up training of convex and non-convex machine learning models, such as logistic regression and neural networks, by over 2.9x over the full data and 4.5x over random subsets[1].

1. Introduction

Large datasets have been crucial for the success of modern machine learning models. Learning from massive datasets, however, incurs substantial computational costs and becomes very challenging (Asi & Duchi, 2019; Strubell et al., 2019; Schwartz et al., 2019). Crucially, not all data points are equally important for learning (Birodkar et al., 2019; Katharopoulos & Fleuret, 2018; Toneva et al., 2018). While several examples can be excluded from training without harming the accuracy of the final model (Birodkar et al., 2019; Toneva et al., 2018), other points need to be trained on many times to be learned (Birodkar et al., 2019). To improve scalability of machine learning, it is essential to theoretically understand and quantify the value of different data points on training and optimization. This allows identifying examples that contribute the most to learning and safely excluding those that are redundant or non-informative.

To find essential data points, recent empirical studies used heuristics such as the fully trained or a smaller proxy model's uncertainty (entropy of predicted class probabilities) (Coleman et al., 2020), or forgetting events (Toneva et al., 2018) to identify examples that frequently transition from being classified correctly to incorrectly. Others employ either the gradient norm (Alain et al., 2015; Katharopoulos & Fleuret, 2018) or the loss (Loshchilov & Hutter, 2015; Schaul et al., 2015) to sample important points that reduce variance of stochastic optimization methods. Such methods, however, do not provide any theoretical guarantee for the quality of the trained model on the extracted examples.

Quantifying the importance of different data points without training a model to convergence is very challenging. First, the value of each example cannot be measured without updating the model parameters and measuring the loss or accuracy. Second, as the effect of different data points changes throughout training, their value cannot be precisely measured before training converges. Third, to eliminate redundancies, one needs to look at the importance of individual data points as well as the higher-order interactions between data points. Finally, one needs to provide theoretical guarantees for the performance and convergence of the model trained on the extracted data points.

Here, we focus on finding data points that contribute the most to learning and automatically excluding redundancies while training a model. A practical and effective approach is to carefully select a small subset of training examples that closely approximate the full gradient, i.e., the sum of the gradients over all the training data points. This idea has been recently employed to find a subset of data points that guarantee convergence of first-order methods to near-optimal solution for training convex models (Mirzasoleiman et al., 2020). However, modern machine learning models are high dimensional and non-convex in nature. In such scenarios, subsets selected based on gradient information only capture gradient along the sharp dimensions, and lack

[1]Code is available at `https://github.com/opooladz/AdaCore`

diversity within groups of examples with similar training dynamics. Hence, they representative large groups of examples with a few data points with substantial weights. This introduces a large error in the gradient estimation and result in first-order coresets to perform poorly.

We propose *ADAptive second-order COREsets* (ADACORE) that incorporates the geometry of the data to iteratively select weighted subsets (coresets) of training examples that captures the gradient of the loss preconditioned with the Hessian, by maximizing a submodular function. Such subsets capture the curvature of the loss landscape along different dimensions, and provide convergence guarantees for first and second-order methods. As a naive use of Hessian at every iteration is prohibitively expensive for overparameterized models, ADACORE relies on Hessian-free methods to extract coresets that capture the full gradient preconditioned by the Hessian diagonal. Furthermore, ADACORE exponentially averages first and second-order information in order to smooth the noise in the local gradient and curvature information.

We first provide a theoretical analysis of our method and prove its convergence for convex and non-convex functions. For a β-smooth and α-strongly convex loss function and a subset S selected by ADACORE that estimates the full preconditioned gradient by an error of at most ϵ, we prove that Newton's method and AdaHessian applied to S with constant stepsize $\eta = \alpha/\beta$ converges to a $\beta\epsilon/\alpha$ neighborhood of the optimal solution, in exponential rate. For non-convex overparameterized functions such as deep networks, we prove that for a β-smooth and μ-PL* loss function satisfying $\|\nabla\mathcal{L}(w)\|^2/2 \geq \mu\mathcal{L}(w)$, (stochastic) gradient descent applied to subsets found by ADACORE has similar training dynamics to that of training on full data, and converges at a exponential rate. In both cases, ADACORE leads to a speedup by training on smaller subsets.

Next, we empirically study the examples selected by ADACORE during training. We show that as training continues, ADACORE selects more uncertain or forgettable samples. Hence, ADACORE effectively determines the value of every learning example, i.e., when and how many times a sample needs to be trained on, and automatically excludes redundant and non-informative instances. Importantly, incorporating curvature in selecting coresets allows ADACORE to quantify the value of training examples more accurately, and find fewer but more diverse samples than existing methods.

We demonstrate the effectiveness of various first and second-order methods, namely SGD with momentum, Newton's method and AdaHessian, applied to ADACORE for training models with a convex loss function (logistic regression) as well as models with a non-convex loss functions, namely ResNet-20, ResNet-18, and ResNet-50, on MNIST, CIFAR10, (Imbalanced) CIFAR100, and BDD100k (Deng, 2012; Krizhevsky et al., 2009; Yu et al., 2020). Our experiments show that ADACORE can effectively extract crucial samples for machine learning, resulting in higher accuracy while achieving over 2.9x speedup over the full data and 4.5x over random subsets, for training models with convex and non-convex loss functions.

2. Related Work

Data-efficient methods have recently gained a lot of interest. However, existing methods often require training the original (Birodkar et al., 2019; Ghorbani & Zou, 2019; Toneva et al., 2018) or a proxy model (Coleman et al., 2020) to convergence, and use features or predictions of the trained model to find subsets of examples that contribute the most to learning. While these results empirically confirm the existence of notable semantic redundancies in large datasets (Birodkar et al., 2019), such methods cannot identify the crucial subsets before fully training the original or the proxy model on the entire dataset. Most importantly, such methods do not provide any theoretical guarantees for the model's performance trained on the extracted subsets.

There have been recent efforts to take advantage of the difference in importance among various samples to reduce the variance and improve the convergence rate of stochastic optimization methods. Those that are applicable to overparameterized models employ either the gradient norm (Alain et al., 2015; Katharopoulos & Fleuret, 2018) or the loss (Loshchilov & Hutter, 2015; Schaul et al., 2015) to compute each sample's importance. However, these methods do not provide rigorous convergence guarantees and cannot provide a notable speedup. A recent study proposed a method, CRAIG, to find subsets of samples that closely approximate the full gradient, i.e., sum of the gradients over all the training samples (Mirzasoleiman et al., 2020). CRAIG finds the subsets by maximizing a submodular function, and provides convergence guarantees to a neighborhood of the optimal solution for strongly-convex models. GRADMATCH (Killamsetty et al., 2021) proposes a variation to address the same objective using orthogonal matching pursuit (OMP) (Killamsetty et al., 2021), and GLISTER Killamsetty et al. (2020) aims at finding subsets that closely approximate the gradient of a held-out validation set. However, GLISTER requires a validation set, and GRADMATCH uses OMP which may return subsets as little as 0.1% of the intended size. Such subsets are then augmented with random samples. In contrast, our method successfully finds subsets of higher quality by preconditioning the gradient by the Hessian information.

3. Background and Problem Setting

Training machine learning models often reduces to minimizing an empirical risk function. Given a not-necessarily

convex loss $\mathcal{L}$, one aims to find model parameter vector w_* in the parameter space $\mathcal{W}$ that minimizes the loss $\mathcal{L}$ over the training data:

$$w_* \in \arg\min_{w\in\mathcal{W}} \mathcal{L}(w), \qquad (1)$$
$$\mathcal{L}(w) := \sum_{i\in V} l_i(w), \quad l_i(w) = l(f(x_i, w), y_i).$$

Here, $V = \{1, \ldots, n\}$ is an index set of the training data, $w \in \mathbb{R}^d$ is the parameters of the model f being trained, and l_i is the loss function associated with training example $i \in V$ with feature vector $x_i \in \mathbb{R}^d$ and label y_i. We denote the gradient of the loss w.r.t. model parameters by $\mathbf{g} = \nabla\mathcal{L}(w) = \frac{1}{|V|}\sum_{i\in V} \frac{\partial l_i}{\partial w}$, and the corresponding second derivative (i.e., Hessian) by $\mathbf{H} = \nabla^2\mathcal{L}(w) = \frac{1}{|V|}\sum_{i\in V} \frac{\partial^2 l_i}{\partial w_j \partial w_k}$.

First order gradient methods are popular for solving Problem (1). They start from an initial point w_0 and at every iteration t, step in the negative direction of the gradient $\mathbf{g}_t$ multiplied by learning rate η_t. The most popular first-order method is Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951):

$$w_{t+1} = w_t - \eta_t \mathbf{v}_t, \qquad \mathbf{v}_t = \mathbf{g}_t, \qquad (2)$$

SGD is often used with momentum, i.e., $\mathbf{v}_t = \beta\mathbf{v}_{t-1} + (1-\beta)\mathbf{g}_t$ where $\beta \in [0, 1]$, accelerating it in dimensions whose gradients point in the same directions and dampening oscillations in dimensions whose gradients change directions (Qian, 1999). For larger datasets, mini-batched SGD is used, where $\mathbf{v}_t = \frac{1}{m}\sum_{j=1}^{m} l_{i_t^{(j)}}(w_t)$, where m is the size of the mini-batch of datapoints whose indices $\{i_t^{(1)}, \ldots, i_t^{(m)}\}$ are uniformly drawn with replacement from V, at each iteration t.

Second-order gradient methods rely on the geometry of the problem to automatically rotate and scale the gradient vectors, using the curvature of the loss landscape. In doing so, second-order methods can choose a better descent direction and automatically adjust the learning rate for each parameter. Hence, second-order methods have superior convergence properties compared to first-order methods. Newton's method (Bertsekas, 1982) is a classical second order method that preconditions the gradient vector with inverse of the local Hessian at every iteration, $\mathbf{H}_t^{-1}$:

$$w_{t+1} = w_t - \eta_t \mathbf{H}_t^{-1}\mathbf{g}_t. \qquad (3)$$

As inverting the Hessian matrix requires quadratic memory and cubic computational complexity, several methods approximate Hessian information to significantly reduce time and memory complexity (Nocedal, 1980; Schaul et al., 2013; Martens & Grosse, 2015; Xu et al., 2020). In particular, AdaHessian (Yao et al., 2020) directly approximates the diagonal of the Hessian and relies on exponential moving averaging and block diagonal averaging to smooth out and reduce the variation of the Hessian diagonal.

4. ADACORE: Adaptive Second order Coresets

The key idea behind our proposed method is to leverage the geometry of the data, precisely the curvature of the loss landscape, to select subsets of the training examples that enable fast convergence. Here, we first discuss why coresets that only capture the full gradient perform poorly in various scenarios. Then, we show how to incorporate curvature information in subset selection for training convex and non-convex models with provable convergence guarantees—ameliorating problems of first-order coresets.

4.1. When First-order Coresets Fail

First-order coreset methods iteratively select weighted subsets of training data that closely approximate the full gradient at particular values of w_t, e.g. beginning of every epoch (Killamsetty et al., 2021; 2020; Mirzasoleiman et al., 2020):

$$S_t^* = \underset{S\subseteq V, \gamma_{t,j}\geq 0\ \forall j}{\arg\min} |S| \quad \text{s.t.} \quad \|\mathbf{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{g}_{t,j}\| \leq \epsilon, \quad (4)$$

where $\mathbf{g}_{t,j}$ and $\gamma_{t,j} > 0$ are the gradient and the weight of element j in the coreset S. Such subsets often perform poorly for high-dimensional and non-convex functions, due to the following reasons: (1) the scale of gradient $\mathbf{g} \in \mathbb{R}^d$ is often different along different dimensions. Hence, the selected subsets estimate the full gradient closely only along dimensions with a larger gradient scale. This can introduce a significant error in the optimization trajectory for both convex and non-convex loss functions; (2) the loss functions associated with different data points l_i may have similar gradients but very different curvature properties at a particular w_t. Thus, for a small $\delta > 0$, the gradients $\nabla l_i(w_t + \delta)$ at $w_t + \delta$ may be totally different than the gradients $\nabla l_i(w_t)$ at w_t. Consequently, subsets that capture the gradient well at at a particular point during training may not provide a close approximation of the full gradient after a few gradient updates, e.g., mini-batches. This often results in inferior performance, particularly when selecting larger subsets for non-convex loss functions; (3) subsets that only capture the gradient, select one representative example with a large weight from data points with similar gradients at w_t. Such subsets lack diversity and cannot distinguish different subgroups of the data. Importantly, the large weights introduce a substantial error in estimating the full gradient and result in a poor performance, as we show in Fig. 7 in the Appendix.

4.2. Adaptive Second-order Coresets

To address the above issues, our main idea is to select subsets of training examples that capture the full gradient preconditioned with the curvature of the loss landscape. In doing so, we normalize the gradient by multiplying it by the Hessian inverse, $\mathbf{H}^{-1}\mathbf{g}$, before selecting the subsets. This

allows selecting subsets that (1) can capture the full gradient in all dimensions equally well; (2) contain a more diverse set of data points with similar gradients, but different curvature properties; and (3) allow adaptive first and second-order methods trained on the coresets to obtain similar training dynamics to that of training on the full data.

Formally, our goal in ADACORE is to adaptively find the smallest subset $S \subseteq V$ and corresponding per-element weights $\gamma_j > 0$ that approximates the full gradient preconditioned with the Hessian matrix, with an error of at most $\epsilon > 0$ at every iteration t, I.e.,:

$$S_t^* = \underset{S \subseteq V, \gamma_{t,j} \geq 0 \ \forall j}{\arg\min} |S|, \quad \text{s.t.} \quad \|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j \in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}\| \leq \epsilon, \tag{5}$$

where $\mathbf{H}_t^{-1}\mathbf{g}_t$ and $\sum_{j \in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}$ are preconditioned gradients of the full data and the subset S.

4.3. Scaling up to Over-parameterized Models

Directly solving the optimization problem (5) requires explicit calculation and storage of the Hessian matrix and its inverse. This is infeasible for large models such as neural networks. In the following, we first address the issue of calculating the inverse Hessian at every iteration. Then, we discuss how to efficiently find a near-optimal subset to estimates the full preconditioned gradient by solving Eq. (5).

Approximating the Gradients For neural networks, derivative of the loss $\mathcal{L}$ w.r.t. the input to the last layer (Katharopoulos & Fleuret, 2018; Mirzasoleiman et al., 2020) or the penultimate layer (Killamsetty et al., 2021) can capture the variation of gradient norm well. We extend these results (Appendix B.2) to show that the normed difference preconditioned gradient difference between data points can be approximately efficiently bound by:

$$\|\mathbf{H}_i^{-1}\mathbf{g}_i - \mathbf{H}_j^{-1}\mathbf{g}_j\| \leq c_1 \|\Sigma'_L(z_i^{(L)})(\mathbf{H}_i^{-1}\mathbf{g}_i)^{(L)} - \Sigma'_L(z_j^{(L)})(\mathbf{H}_j^{-1}\mathbf{g}_j)^{(L)}\| + c_2, \tag{6}$$

where $\Sigma'_L(z_i^{(L)})(\mathbf{H}_i^{-1}\mathbf{g}_i)^{(L)}$ is gradient preconditioned by the inverse of the Hessian of the loss w.r.t. the input to the last layer for data point i, and c_1, c_2 are constants. Since the upper bound depends on the weight parameters, we need to update our subset S using ADACORE during the training.

Calculating the last layer gradient often requires only a forward pass, which is as expensive as calculating the loss, and does not require any extra storage. For example, having a softmax as the last layer, the gradients of the loss w.r.t. the i^{th} input to the softmax is $p_i - y_i$, where p_i is the i^{th} output the softmax and y is the one-hot encoded label with the same dimensionality as the number of classes. Using this low-dimensional approximation $\hat{\mathbf{g}}_i$ for the gradient $\mathbf{g}_i$ we can efficiently calculate the preconditioned gradient for every data point. For non-convex functions, the local gradient information can be very noisy. To smooth out the local gradient information and get a better approximation of the global gradient, we apply exponential moving average with a parameter $0 < \beta_1$ to the low-dimensional gradient approximations:

$$\overline{\mathbf{g}}_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_2^{t-i} \hat{\mathbf{g}}_\mathbf{i}}{1-\beta_2^t}. \tag{7}$$

Approximating the Hessian Preconditioner Since it is infeasible to calculate, store, and invert the full Hessian matrix every iteration, we use an inexact Newton method, where an approximate Hessian operator is used instead of the full Hessian. To efficiently calculate the Hessian diagonal, we first use the Hessian-Free method (Yao et al., 2018) to compute the multiplication between Hessian $\mathbf{H}_t$ and a random vector z with Rademacher distribution. To do so, we backpropagate on the low-dimensional gradient estimates multiplied by z to get $\mathbf{H}_t z = \partial \hat{\mathbf{g}}_t^T z / \partial w_t$. Now, we can use the Hutchinson's method of obtains a stochastic estimate of the diagonal of the Hessian matrix as follows:

$$\text{diag}(\mathbf{H}_t) = \mathbb{E}[z \odot (\mathbf{H}_t z)], \tag{8}$$

without having to form the Hessian matrix explicitly (Bekas et al., 2007). The diagonal approximation has the same convergence rate as using Hessian for strongly convex, and strictly smooth functions (Proof in Appendix A.1). Nevertheless, our method can be applied to general machine learning problems, such as deep networks and regularized classical methods (e.g., SVM, LASSO), which are strongly-convex. To smooth out the noisy local curvature and get a better approximation of the global Hessian information, we apply an exponential moving average with parameter $0 < \beta_2 < 1$ to the Hessian diagonal estimate in Eq. (8):

$$\overline{\mathbf{H}}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \text{diag}(\mathbf{H}_i)\text{diag}(\mathbf{H}_i)}{1-\beta_2^t}}. \tag{9}$$

Using exponentially averaged gradient and Hessian approximations in Eq. (7), and (9), the preconditioned gradients in Eq. (5) can be approximated as follows:

$$S_t^* = \underset{S \subseteq V, \gamma_{t,j} \geq 0 \ \forall j}{\arg\min} |S|, \quad \text{s.t.} \quad \|\overline{\mathbf{H}}_t^{-1}\overline{\mathbf{g}}_t - \sum_{j \in S} \gamma_{t,j} \overline{\mathbf{H}}_{t,j}^{-1} \overline{\mathbf{g}}_{t,j}\| \leq \epsilon. \tag{10}$$

Next, we discuss how to efficiently find near-optimal weighted subsets that closely approximate the full preconditioned gradient by solving Eq. (5).

4.4. Extracting Second-order Coresets

The subset selection problem (5) is NP-hard (Natarajan, 1995). However, it can be considered as a special case of the sparse vector approximation problem that has been studied in the literature, including convex optimization formulations—e.g. basis pursuit (Chen et al., 2001), sparse projections (Pilanci et al., 2012; Kyrillidis et al., 2013), LASSO (Tibshirani, 1996), and compressed sensing (Donoho, 2006). These methods, however, are expensive to solve and often require tuning regularization coefficients and thresholding to ensure cardinality constraints. More recently, the connection between sparse modeling and *submodular*[2] optimization have been demonstrated (Elenberg et al., 2018; Mirzasoleiman et al., 2020). The advantage of submodular optimization is that a fast and simple greedy algorithm often provides a near-optimal solution. Next, we briefly discuss how submodularity can be used to find a near-optimal solution for Eq. (5). We build on the recent result of (Mirzasoleiman et al., 2020) that showed that the error of estimating an expectation by a weighted sum of a subset of elements is upper-bounded by a submodular facility location function. In particular, via the above result, we get:

$$\min_{S\subseteq V} \|\overline{\mathbf{H}}_t^{-1}\overline{\mathbf{g}}_t - \sum_{j\in S}\gamma_{t,j}\overline{\mathbf{H}}_{t,j.}^{-1}\overline{\mathbf{g}}_{t,j}\| \leq \sum_{i\in V}\min_{j\in S}\|\overline{\mathbf{H}}_{t,i.}^{-1}\overline{\mathbf{g}}_{t,i} - \overline{\mathbf{H}}_{t,j.}^{-1}\overline{\mathbf{g}}_{t,j}\|. \tag{11}$$

Setting the upper bound in the right-hand side of Eq. (11) to be less than ϵ results in the smallest weighted subset S^* that approximates full preconditioned gradient by an error of at most ϵ, at iteration t. Formally, we wish to solve the following optimization problem:

$$S^* \in \arg\min_{S\subseteq V}|S|, \quad \text{s.t.} \quad L(S) = \sum_{i\in V}\min_{j\in S}\|\overline{\mathbf{H}}_{t,i.}^{-1}\overline{\mathbf{g}}_{t,i} - \overline{\mathbf{H}}_{t,j.}^{-1}\overline{\mathbf{g}}_{t,j}\| \leq \epsilon, \tag{12}$$

By introducing a phantom example e, we can turn the minimization problem (12) into the following submodular cover problem, with a facility location objective $F(S)$:

$$S^* \in \arg\min_{S\subseteq V}|S|, \quad \text{s.t.} \quad F(S) = C_1 - L(S\cup\{e\}) \geq C_1 - \epsilon, \tag{13}$$

where $C_1 = L(\{e\})$ is a constant upper-bounding the value of $L(S)$. The subset S^* obtained by solving the maximization problem (13) is the medoid of the preconditioned gradients, and the weights γ_j are the number of elements that are closest to the medoid $j \in S^*$, i.e. $\gamma_j = \sum_{i\in V}\mathbb{I}[j = \min_{s\in S}\|\overline{\mathbf{H}}_{t,i}^{-1}\overline{\mathbf{g}}_{t,i} - \overline{\mathbf{H}}_{t,s}^{-1}\overline{\mathbf{g}}_{t,s}\|]$. For the above submodular cover problem, the classical greedy algorithm provides a logarithmic approximation guarantee $|S| \leq \big(1+\ln(\max_e F(e|\emptyset))\big)|S^*|$ (Wolsey, 1982). The greedy algorithm starts with the empty set $S_0 = \emptyset$, and at each iteration t, it chooses an element $e \in V$ that maximizes the marginal utility $F(e|S_t) = F(S_t\cup\{e\}) - F(S_t)$. Formally, $S_t = S_{t-1}\cup\{\arg\max_{e\in V}F(e|S_{t-1})\}$. The computational complexity of the greedy algorithm is $\mathcal{O}(nk)$. However, its complexity can be reduced to $\mathcal{O}(|V|)$ using stochastic methods (Mirzasoleiman et al., 2015), and can be further improved using lazy evaluation (Minoux, 1978) and distributed implementations (Mirzasoleiman et al., 2013). The pseudocode can be found in Alg. 1 in Appendix A.3.

[2] A set function $F: 2^V \to \mathbb{R}^+$ is submodular if $F(S\cup\{e\}) - F(S) \geq F(T\cup\{e\}) - F(T)$, for any $S \subseteq T \subseteq V$ and $e \in V\setminus T$.

One coreset for convex functions For convex functions, normed gradient differences between data points can be efficiently upper-bounded by the normed difference between feature vectors (Allen-Zhu et al., 2016; Hofmann et al., 2015; Mirzasoleiman et al., 2020). We apply a similar idea to upper-bound the normed difference between preconditioned gradients. This allows us to find one subset before the training. See proof in Appendix B.1.

4.5. Convergence Analysis

Here, we analyze the convergence rate of first and second order methods applied to the weighted subsets S found by ADACORE. By minimizing Eq. (13) at every iteration t, ADACORE finds subsets that approximate full preconditioned gradient by an error of at most ϵ, i.e. $\|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\mathbf{g}_{t,j}\| \leq \epsilon$. This allows us to effectively analyze the reduction in the value of the loss function $\mathcal{L}$ at every iteration t. Below, we discuss the convergence of a first and second-order gradient methodapplied to subsets extracted by ADACORE.

Convergence for Newton's Methods and AdaHessian We first provide the convergence analysis for the case where the function $\mathcal{L}$ in Problem (1) is strongly convex, i.e. there exist a constant $\alpha > 0$ such that $\forall w, w' \in \mathbb{R}^d$ we have $\mathcal{L}(w) \geq \mathcal{L}(w') + \langle\nabla\mathcal{L}(w'), w - w'\rangle + \frac{\alpha}{2}\|w' - w\|^2$, and each component function has a Lipschitz gradient, i.e. $\forall w \in \mathcal{W}$ we have $\|\nabla\mathcal{L}(w) - \nabla\mathcal{L}(w')\| \leq \beta\|w - w'\|$. We get the following results by applying Newton's method and AdaHessian to the weighted subsets S extracted by ADACORE.

Theorem 4.1. *Assume that $\mathcal{L}$ is α-strongly convex and β-smooth. Let S be a weighted subset obtained by* ADACORE *that estimate the preconditioned gradient by an error of at most ϵ at every iteration t, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate α/β, Newton's method with update rule of Eq.* (3) *applied to the subsets has the following convergence behavior:*

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^3}{2\beta^4}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{14}$$

In particular, the algorithm converges to a $\beta\epsilon/\alpha$-neighborhood of the optimal solution w_*.

Corollary 4.2. *For an α-strongly convex and β-smooth loss $\mathcal{L}$, AdaHessian with Hessian power k, applied to subsets found by* ADACORE *converges to a $\beta\epsilon/\alpha$-neighborhood of the optimal solution w_*, and satisfies:*

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^{k+2}}{2\beta^{k+3}}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \quad (15)$$

The proofs can be found in Appendix A.1.

Convergence for (S)GD in Over-parameterized Case Next, we discuss the convergence behavior of gradient descent applied to the subsets found by ADACORE. In particular, we build upon the recent results of (Liu et al., 2020) that guarantees convergence for first-order methods on a broad class of general over-parameterized non-linear systems, including neural networks for which the tangent kernel, defined as $\mathbf{J}^T\mathbf{J}$ are not close to constant, but satisfy the Polyak-Lojasiewicz (PL) condition. Where $\mathbf{J} = \partial f/\partial w$ is the Jacobian of the function f with respect to the parameters w. A loss function $\mathcal{L}$ is μ-PL* on a set $\mathcal{W}$, if $\frac{1}{2}\|\nabla\mathcal{L}(w)\|^2 \geq \mu\mathcal{L}(w), \forall w \in \mathcal{W}$.

Theorem 4.3. *Assume that the loss function $\mathcal{L}(w)$ is β-smooth, and μ-PL* on a set $\mathcal{W}$, and S is a weighted subset obtained by* ADACORE *that estimates the preconditioned gradient by an error of at most ϵ, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate η, gradient descent with update rule of Eq.* (2) *applied to the subsets have the following convergence behavior at iteration t:*

$$\mathcal{L}(w_t) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})^t\mathcal{L}(w_0) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}), \quad (16)$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients.

Theorem 4.4. *Under the same assumptions as in Theorem 4.3, for mini-batch SGD with mini-batch size $m \in \mathbb{N}$, the mini-batch SGD with update rule Eq.* (2)*, with learning rate $\eta = \frac{m}{\beta(m-1)}$, applied to the subsets have the following convergence behavior:*

$$\mathbb{E}[\mathcal{L}(w_t)] \leq (1 - \frac{\eta\mu\alpha^2}{2\beta})^t\mathbb{E}[\mathcal{L}(w_0)] - \frac{\alpha^2\eta}{2\beta}(\beta\epsilon^2 - 2\epsilon\nabla_{\max}) \quad (17)$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients, and the expectation is taken w.r.t. the randomness in the choice of mini-batch.

The proofs can be found in Appendix A.2.

We show an exponential convergence for GD (Theorem 4.3) and SGD (Theorem 4.4) under the μ-PL* condition, as well as for second order methods (Theorems 4.1, 4.2), under α-strongly convex and β-smooth assumptions on the loss.

5. Experiments

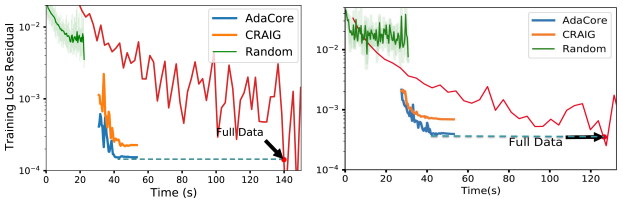
In this section, we evaluate the effectiveness of ADACORE, by answering the following questions: (1) how does the performance of various first and second-order methods compare when applied to subsets found by ADACORE vs. the full data and baselines; (2) how effective is ADACORE for extracting crucial subsets for training convex and non-convex over-parameterized models with different optimizers; and (3) how does ADACORE perform in eliminating redundancies and enhancing diversity of the selected elements.

Baselines In the convex setting, we compare the performance of ADACORE with CRAIG (Mirzasoleiman et al., 2020) that extracts subsets that approximate the full gradient, as well as Random subsets. For non-convex experiments, we additionally compare ADACORE with GRADMATCH and GLISTER (Killamsetty et al., 2021; 2020). For ADACORE and CRAIG, we use the gradient w.r.t the input to the last layer, and for GLISTER and GRADMATCH we use the gradient w.r.t the penultimate layer, as specified by the methods. In all cases, we select subsets separately from each class proportional to the class sizes, and train on the union of the subsets. We report average test accuracy across 3 trials in all experiments.

5.1. Convex Experiments

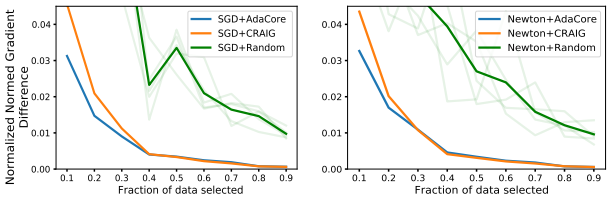
In our convex experiments, we apply ADACORE to select a coreset to classify the Ijcnn1 dataset using L2-regularized logistic regression: $f_i(x) = ln(1 - \exp(-w^T x_y y_i)) + 0.5\mu w^T w$. Ijcnn1 includes 49,990 training and 91,701 test data points of 22 dimensions, from 2 classes with 9-to-1 class imbalance ratio. In the convex setting, we only need to calculate the curvature once to find one ADACORE subset for the entire training. Hence, we utilize the complete Hessian information, computed analytically, as discussed in Appendix B.3. We apply an exponential decay learning schedule $\alpha_k = \alpha_0 b^k$ with learning rate parameters α_0 and b. For each model and method (including the random baseline) we tuned the parameters via a search and reported the best results.

ADACORE achieves smaller loss residual with a speedup Figure 1 compares the loss residual for SGD and Newton's method applied to coresets of size 10% extracted by ADACORE (blue), CRAIG (orange), and random (green) with that of full dataset (red). We see that ADACORE effectively minimizes the training loss, achieving a better loss residual than CRAIG and random sampling. In particular, ADACORE



(a) Ijcnn1 SGD (b) Ijcnn1 Newton

Figure 1: Loss residual of SGD and Newton's method for training Logistic Regression on Ijcnn1. Comparing ADACORE (blue), CRAIG (orange) and random subsets (green) of size 10% vs. entire data (red dot). ADACORE achieves 2.5x speedup for training with SGD and Newton's method.



(a) Ijcnn1 SGD (b) Ijcnn1 Newton

Figure 2: Normalized gradient difference between subsets of various sizes found by ADACORE (blue), ADACORE (orange), Random (green) vs full data, when training Logistic Regression with SGD and Newton on Ijcnn1. ADACORE has a smaller gradient error at the end of training.

matches the loss achieved on the full dataset with more than a 2.5x speedup for SGD and Newton's methods. We note that training on random 10% subsets of the data cannot effectively minimize the training loss. We show the superior performance of training with SGD on subsets of size 10% to 90% found with ADACORE vs CRAIG in Appendix Fig. 6.

ADACORE better estimates the full gradient Fig. 2 shows the normalized gradient difference between gradient of the full data vs. weighted gradient of subsets of different sizes obtained by ADACORE vs CRAIG and Random, at the end of training by each method. We see that by considering curvature information, ADACORE obtains a better gradient estimate than CRAIG and Random subsets.

5.2. Non-Convex Experiments

Datasets We use CIFAR10 (60k points from 10 classes) , class imbalanced version of CIFAR10 (32.5k points from 10 classes) and CIFAR100 (32.5k points from 100 classes) (Krizhevsky et al., 2009), BDD100k (100k points from 7 classes) (Yu et al., 2020). The results on MNIST (70k points from 10 classes) (Deng, 2012) can be found in Appendix C.6. Images are normalized to [0,1] by division with 255.

Models and Optimizers We train ResNet-20 and ResNet-18 (He et al., 2016), with convolution, average pooling and dense layers with softmax outputs and weight decay of 10^{-4}.

Table 1: Training ResNet20 using AdaHessian and SGD+momentum on coresets of size 1% selected by different methods from CIFAR10. Percent of full data selected during entire training is shown (in parentheses). Using b_H=64, ADACORE achieves up to 16.8% higher accuracy, while selecting a smaller fraction of data points. Exponential averaging of gradient and Hessian, and a smaller b_H helps.

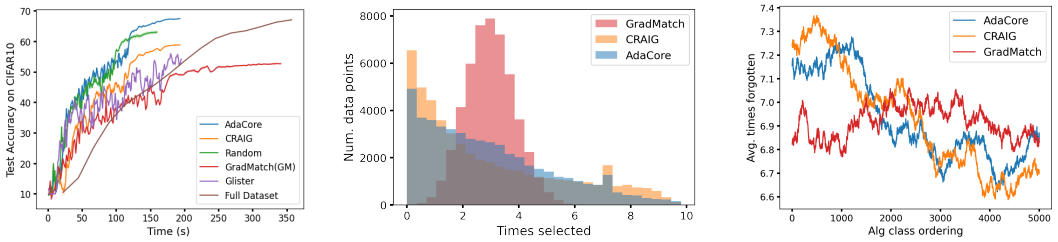
	AdaHessian	SGD+Momentum
Random	59.1%± 2.8(87%)	45.9%± 2.5(87%)
CRAIG	59.5%± 2.8(74%)	43.6%± 1.6(75%)
GRADMATCH	57.5%± 1.3(74%)	49.4%± 1.6(74%)
GLISTER	37.5%± 1.3(74%)	38.6% ± 1.6(74%)
ADACORE(no avg)	58.4%± 0.2(73%)	51.5%± 1.1(74%)
ADACORE (avg g)	59.8%± 0.5(73%)	53.2% ± 1.1(74%)
ADACORE(avg H)	60.2%± 0.5(73%)	54.4%± 1.1(74%)
ADACORE	**60.2%** ± 0.5(**73%**)	**55.4%** ± 1.1(**74%**)
ADACORE b_h=512	57.2% ± 0.5(73%)	52.4% ± 1.1(74%)

We use a batch size of 256 in all experiments (except Table 3, Fig. 4a), and train using SGD with momentum of 0.9 (default), or AdaHessian. For training, we use standard learning rate scheduler for ResNet starting with 0.1 and exponentially decaying by factor 0.1 at epochs 100 and 150. We used linear learning rate warm-up for the first 20 epochs to prevent weights from diverging when training with subsets. All experiments were ran on a 2.4GHz CPU and RTX 2080 Ti GPU.

Calculating the Curvature To calculate the Hessian diagonal using Eq. (8), we use a batch size of $b_H = 64$ to calculate the expected Hessian diagonal over the training data. We observed that a smaller batch size provides a higher quality Hessian compared to larger batch sizes, as shown in Table 1.

Baseline Comparison and Ablation Study Table 1 shows the accuracy of training ResNet-20, using SGD with momentum of 0.9 and AdaHessian, for 200 epochs on S=1% subsets of CIFAR-10 chosen every R=1 epoch by different methods. For SGD+momentum, ADACORE outperforms CRAIG by 12%, Random by 10%, GRADMATCH by 6%, and GLISTER by 16.8%. Note that in total, ADACORE selects 74% of the dataset during the entire training process, whereas Random visits 87%. Thus, ADACORE effectively selects subsets contributing the most to generalization. We see that the accuracy gap between the baselines and ADACORE shrinks when applying more powerful optimizers such as AdaHessian. Table 1 also shows the effect of exponential averaging of gradients and Hessian diagonal, and larger batch sizes for calculating the Hessian diagonal b_H. We see that exponential averaging help ADACORE achieving better performance, and smaller b_H provides better results.

Fig. 3a compares the performance of ResNet-18 on 1% subsets selected from CIFAR-10 with different methods. We compare the performance of training on ADACORE, CRAIG, GRADMATCH, GLISTER, and Random subsets for 200 epochs, with training on full data for 15 epochs. This is the number of iterations required for training on



(a) Test Accuracy (b) Distribution of selected points (c) Forgetting vs class ranking

Figure 3: Training ResNet-18 on subsets of size S=1% selected every R=1 epoch, with ADACORE, CRAIG, GLISTER, GRADMATCH and Random for 200 epochs vs. full for 15 epochs. (a) ADACORE outperforms baselines by providing 2x speedup over full, and more than 4.5x speedup over Random. (b) Histograms of the number of times a point is selected by ADACORE, CRAIG and GRADMATCH. ADACORE selects a more diverse set of examples compare to CRAIG, and GRADMATCH augments several randomly selected examples. (c) Forgetting scores for examples of a class sorted by ADACORE at the end of training. ADACORE priorities less forgettable examples compared to CRAIG.

the full data to achieve a comparable performance to that of ADACORE subsets. We see that training on ADACORE coresets achieves a better accuracy 2.5x faster than training on the full dataset, and more than 4.5x faster than the next best subset selection algorithm for this setting (*c.f.* Fig. 8b in Appendix for complete results).

Frequency and size of subsets selection Table 2, 4 shows the performance of different methods for selecting subsets of size $S\%$ of the data every R epochs, from CIFAR-10 and imbalanced CIFAR-10. Table 2 shows that selecting subsets of size 1% every $R = 5, 10, 20$ epochs with ADACORE achieves a superior performance compared to the baselines. Table 4 shows that ADACORE can successfully select larger subsets of size $S = 10\%, 30\%$ and outperform CRAIG (Std is reported in Appendix, Table 5).

ADACORE speeds up training Fig 8 compares the speedup of various methods during training ResNet18 on 10% subsets selected every $R = 20$ epochs from BDD 100k and CIFAR-100. All the methods are trained to achieve a test accuracy between 72% and 74% on BDD 100k, and between 57% and 50% on CIFAR-100. On BDD 100k, ADACORE achieves 74% accuracy in 100 epochs and training on full data achieves a similar performance in 45 epochs. For CIFAR-100, ADACORE achieves 59% accuracy in 200 epochs and training on full data achieves a similar performance in 40 epochs. Complete results on speedup and test accuracy of each method can be found in Appendix C.3, C.4. We see that ADACORE achieves 2.5x speedup over training on full data and 1.7x over that of training on random subsets on BDD 100k. For CIFAR-100, ADACORE achieves 4.2x speedup over training on random subsets and 2.9x over training on full data. Compared to the baselines, ADACORE can achieve achieve the desired accuracy much faster.

Table 2: Test accuracy and percent of full data selected (in parentheses), when selecting S=1% coresets every R epochs from Imbalanced CIFAR-10 to train ResNet18.

	S=1%, R=20	S=1%, R=10	S=1%, R=5
AdaCore	**57.3%(5%)**	**57.12(9.5%)**	**60.2%(14.5%)**
CRAIG	48.6%(8%)	55(16%)	53.05%(27.5%)
Random	54.7%(8%)	54.6(18%)	54.6%(33.2%)
GRADM	29.9%(8.2%)	29.1%(14.7%)	32.75%(23.2%)
GLISTER	21.1%(8.6%)	17.2%(16%)	14.4%(22.2%)

Effect of batch size Table 3 compares the performance of training with different batch sizes on subsets found by various methods. We see that training with larger batch size on subsets selected by ADACORE can achieve a superior accuracy. As ADACORE selects more diverse subsets with smaller weights, one can train with larger mini-batches on the subsets without increasing the gradient estimate error. In contrast, CRAIG subsets have elements with larger weights and hence training with fewer larger mini-batches has larger gradient error and does not improve the performance.

In summary, see that ADACORE consistently outperforms the baselines over various architectures, optimizers, subset sizes, selection frequency, and batch sizes.

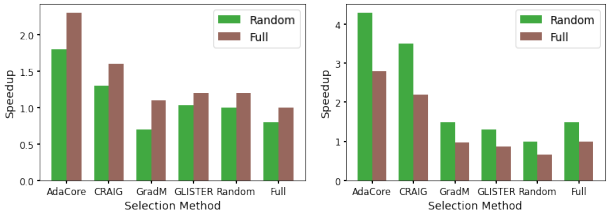
ADACORE selects more diverse subsets Fig. 3b shows the number of times different methods selected a particular elements during the entire training. We see that ADACORE successfully selects a more diverse set of examples compared to CRAIG. We note that GRADMATCH may not be able to select subsets with the desired size, and instead augments the selected subset with randomly selected

Table 3: Training ResNet18 with S=1% subsets every R=1 epoch from CIFAR10 using batch size b= 512, 256, 128. ADACORE can leverage larger mini-bath size and obtain a larger accuracy gap to CRAIG and Random. For b=512, we have 1 mini-batch (GD). Std is reported in Appendix Table 9.

	ADAC.	CRAIG	Rand	Gap/ CRAIG	Gap/ Rand
GD b=512	**58.32%**	56.32%	49.14%	1.69%	**8.91%**
SGD b=256	**68.23%**	58.3%	60.7%	**9.93%**	8.16%
SGD b=128	**66.89%**	58.17%	65.46%	**8.81%**	1.52%

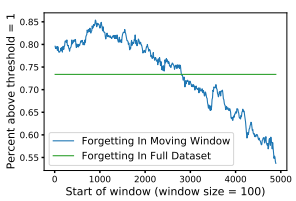
Table 4: Test accuracy and percent of full data selected (in parentheses), when selecting $S\%$ coresets every R epochs from CIFAR-10 and Imbalanced CIFAR-10 to train ResNet18.

	ResNet20, CIFAR10 $S=30\%$, $R=20$	ResNet20, CIFAR10 $S=10\%$, $R=20$	ResNet18, CIFAR10-IMB $S=30\%$, $R=20$	ResNet18, CIFAR10-IMB $S=10\%$, $R=20$
ADACORE	**80.57%** $\pm$ 0.11 (**74.6%**)	**70.6%** $\pm$ 0.33 (**44.8%**)	**85.7%** $\pm$ 0.1 (**74%**)	**76%** $\pm$ 0.3 (**43.8%**)
CRAIG	65.8% $\pm$ 0.41 (90.9%)	58.5% $\pm$ 1.27 (60.75%)	79.3% $\pm$ 1.6 (84.5%)	71.6% $\pm$ 0.15 (56.4%)

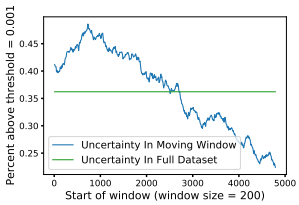


(a) BDD100k, ResNet50 (b) CIFAR100, ResNet18

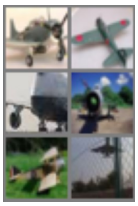
Figure 4: Speedup of various methods over training on random subsets and full data, for training ResNet18 on CIFAR100 and ResNet50 on BDD100k with batch size=128.



(a) Forgetting vs class ranking



(b)Uncertaintyvs class ranking



(c) Most selected

(d) Not selected

Figure 5: Training ResNet20 on S=1% subsets of CIFAR-10 selected by ADACORE. (a) Forgetting scores, and (b) uncertainty of examples in a class sorted by ADACORE at the end of training. ADACORE prioritize selecting more forgettable and uncertain examples. (c) Six images selected by ADACORE most frequently (25 times) from the airplane class. (d) Subset of images never selected by ADACORE.

examples. Hence, it has a normal-shaped distribution. Fig. 3c shows mean forgetting score for all examples within a class ranked by ADACORE at the end of training, over sliding window of size 100. We see that ADACORE prioritizes selecting less forgettable examples. This shows that indeed ADACORE is able to distinguish different groups of easier examples better, and hence can prevent catastrophic forgetting by including their representatives in the coresets.

ADACORE vs Forgettability and Uncertainty Fig. 5a, 5b show mean forgettability and uncertainty in sliding windows of size 100, 200 over examples sorted by ADACORE at the end of training. We see that ADACORE heavily biases its selections towards forgettable and uncertain points, as training proceeds. Interestingly, 5a reveals that ADACORE avoids the most forgettable samples in favor of slightly more memorable ones, suggesting that ADACORE can better distinguish easier groups of examples. Figure 5b shows similar bias towards uncertain samples. Fig. 5c, 5d show the most and least selected images by ADACORE, respectively. We see the redundancies in the never selected images, whereas images frequented by ADACORE are quite diverse in color, angles, occluded subjects, and airplane models. This confirms the effectiveness of ADACORE in extracting the most crucial subsets for learning and eliminating redundancies.

6. Conclusion

We proposed ADACORE, a method that leverages the topology of the dataset to extract salient subsets of large datasets for efficient machine learning. The key idea behind ADACORE is to dynamically incorporate the curvature and gradient of the loss function via an adaptive estimate of the Hessian to select weighted subsets (coresets) which closely approximate the preconditioned gradient of the full dataset. We proved exponential convergence rate for first and second-order optimization methods applied to ADACORE coresets, under certain assumptions. Our extensive experiments, using various optimizers e.g., SGD, AdaHessian, and Newton's method, show that ADACORE can extract higher quality coresets compared to baselines, rejecting potentially redundant data points. This speeds up the training of various machine learning models, such as logistic regression and neural networks, by over 4.5x while selecting fewer but more diverse data points for training.

Acknowledgements

This research was supported in part by UCLA-Amazon Science Hub for Humanity and Artificial Intelligence.

References

Alain, G., Lamb, A., Sankar, C., Courville, A., and Bengio, Y. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

Allen-Zhu, Z., Yuan, Y., and Sridharan, K. Exploiting the structure: Stochastic gradient methods using raw clusters. In *Advances in Neural Information Processing Systems*,

pp. 1642–1650, 2016.

Asi, H. and Duchi, J. C. The importance of better models in stochastic optimization. *arXiv preprint arXiv:1903.08619*, 2019.

Bekas, C., Kokiopoulou, E., and Saad, Y. An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 57(11):1214–1229, 2007. ISSN 0168-9274. doi: https://doi.org/10.1016/j.apnum.2007.01.003. URL https://www.sciencedirect.com/science/article/pii/S0168927407000244. Numerical Algorithms, Parallelism and Applications (2).

Bertsekas, D. P. Projected newton methods for optimization problems with simple constraints. *SIAM Journal on control and Optimization*, 20(2):221–246, 1982.

Birodkar, V., Mobahi, H., and Bengio, S. Semantic redundancies in image-classification datasets: The 10% you don't need. *arXiv preprint arXiv:1901.11409*, 2019.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, USA, 2004. ISBN 0521833787.

Chen, S. S., Donoho, D. L., and Saunders, M. A. Atomic decomposition by basis pursuit. *SIAM review*, 43(1): 129–159, 2001.

Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., and Zaharia, M. Selection via proxy: Efficient data selection for deep learning. In *International Conference on Learning Representations (ICLR)*, 2020.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Donoho, D. L. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

Elenberg, E. R., Khanna, R., Dimakis, A. G., Negahban, S., et al. Restricted strong convexity implies weak submodularity. *Annals of Statistics*, 46(6B):3539–3568, 2018.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hofmann, T., Lucchi, A., Lacoste-Julien, S., and McWilliams, B. Variance reduced stochastic gradient descent with neighbors. In *Advances in Neural Information Processing Systems*, pp. 2305–2313, 2015.

Katharopoulos, A. and Fleuret, F. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., and Iyer, R. Glister: Generalization based data subset selection for efficient and robust learning. *arXiv preprint arXiv:2012.10630*, 2020.

Killamsetty, K., Sivasubramanian, D., Mirzasoleiman, B., Ramakrishnan, G., De, A., and Iyer, R. Grad-match: A gradient matching based data subset selection for efficient learning. *arXiv preprint arXiv:2103.00123*, 2021.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Kyrillidis, A., Becker, S., Cevher, V., and Koch, C. Sparse projections onto the simplex. In *International Conference on Machine Learning*, pp. 235–243. PMLR, 2013.

Liu, C., Zhu, L., and Belkin, M. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307*, 2020.

Loshchilov, I. and Hutter, F. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Minoux, M. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*, pp. 234–243. Springer, 1978.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pp. 2049–2057, 2013.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. Lazier than lazy greedy. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Mirzasoleiman, B., Bilmes, J., and Leskovec, J. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Natarajan, B. K. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.

Nocedal, J. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2006193.

Pilanci, M., El Ghaoui, L., and Chandrasekaran, V. Recovery of sparse probability measures via convex programming. 2012.

Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

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

Schaul, T., Zhang, S., and LeCun, Y. No more pesky learning rates. In *International Conference on Machine Learning*, pp. 343–351. PMLR, 2013.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green ai. *arXiv preprint arXiv:1907.10597*, 2019.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Toneva, M., Sordoni, A., des Combes, R. T., Trischler, A., Bengio, Y., and Gordon, G. J. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2018.

Wolsey, L. A. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4): 385–393, 1982.

Xu, P., Roosta, F., and Mahoney, M. W. Second-order optimization for non-convex machine learning: An empirical study. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 199–207. SIAM, 2020.

Yao, Z., Xu, P., Roosta-Khorasani, F., and Mahoney, M. W. Inexact non-convex newton-type methods. *arXiv preprint arXiv:1802.06925*, 2018.

Yao, Z., Gholami, A., Shen, S., Keutzer, K., and Mahoney, M. W. Adahessian: An adaptive second order optimizer for machine learning. *arXiv preprint arXiv:2006.00719*, 2020.

Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., and Darrell, T. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

A. Proofs of Theorems

A.1. Proof of Theorem 4.1

Theorem 4.1. *Assume that $\mathcal{L}$ is α-strongly convex and β-smooth. Let S be a weighted subset obtained by* ADACORE *that estimate the preconditioned gradient by an error of at most ϵ at every iteration t, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate α/β, Newton's method with update rule of Eq.* (3) *applied to the subsets has the following convergence behavior:*

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^3}{2\beta^4}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{14}$$

In particular, the algorithm converges to a $\beta\epsilon/\alpha$-neighborhood of the optimal solution w_*.

Proof. We prove Theorem 4.1 (similarly to the proof of Newton's method in (Boyd & Vandenberghe, 2004)) for the following general update rule for $0 \leq k \leq 1$:

$$\Delta w_t = \mathbf{H}_t^{-k}\mathbf{g}_t \tag{18}$$

$$w_{t+1} = w_t - \eta\Delta w_t \tag{19}$$

For $k = 1$, this corresponds to the update rule of the Newton's method. Define $\lambda(w_t) = (\mathbf{g}_t^T\mathbf{H}_t^{-k}\mathbf{g}_t)^{1/2}$. Since $\mathcal{L}(w)$ is β-smooth, we have

$$\mathcal{L}(w_{t+1}) \leq \mathcal{L}(w_t) - \eta\mathbf{g}_t^T\Delta w_t + \frac{\eta^2\beta\|\Delta w_t\|^2}{2} \tag{20}$$

$$\leq \mathcal{L}(w_t) - \eta\lambda(w_t)^2 + \frac{\beta}{2\alpha^k}\eta^2\lambda(w_t)^2, \tag{21}$$

where in the last equality, we used

$$\lambda(w_t) = \Delta w_t\mathbf{H}_t^k\Delta w_t^T. \tag{22}$$

Therefore, using step size $\hat{\eta} = \frac{\alpha^k}{\beta}$ we have $w_{t+1} = w_t - \hat{\eta}\Delta w_t$

$$\mathcal{L}(w_{t+1}) \leq \mathcal{L}(w_t) - \frac{1}{2}\hat{\eta}\lambda(w_t)^2 \tag{23}$$

Since $\alpha I \preceq \mathbf{H}_t \preceq \beta I$, we have

$$\lambda(w_t)^2 = \mathbf{g}_t^T\mathbf{H}_t^{-k}\mathbf{g}_t \geq \frac{1}{\beta^k}\|\mathbf{g}_t\|^2, \tag{24}$$

and therefore $\mathcal{L}$ decreases as follows,

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{1}{2\beta^k}\hat{\eta}\|\mathbf{g}_t\|^2 = -\frac{\alpha^k}{2\beta^{k+1}}\|\mathbf{g}_t\|^2. \tag{25}$$

Now for the subset, from Eq. (5) we have that $\|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\mathbf{g}_{t,j}\| \leq \epsilon$. Hence, via reverse triangle inequality $\|\mathbf{H}_t^{-1}\mathbf{g}_t\| \leq \|\sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\mathbf{g}_{t,j}\| + \epsilon$, and we get

$$\frac{\|\mathbf{g}_t\|}{\beta} \leq \|(\mathbf{H}_t)^{-1}\mathbf{g}_t\| \leq \|(\mathbf{H}_t^S)^{-1}\mathbf{g}_t^S\boldsymbol{\gamma}\| + \epsilon \leq \frac{\|\mathbf{g}_t^S\|}{\alpha} + \epsilon, \tag{26}$$

where $\mathbf{g}_t^S = \sum_{j\in S}\mathbf{g}_{t,j}$ and $\mathbf{H}_t^S = \sum_{j\in S}\mathbf{H}_{t,j.}$ are the gradient and Hessian of the subset respectively. In Eq. (26) the RHS follows from operator norms and the LHS follows from the following lower bound on the norm of the product of two

matrices:

$$\begin{aligned}\|AB\| &= \max_{\|x\|=1} \|x^T AB\| \\ &= \max_{\|x\|=1} \|x^T A\| \left\| \frac{x^T A}{\|x^T A\|} B \right\| \\ &\geq \max_{\|x\|=1} \sigma_{\min(A)} \left\| \frac{x^T A}{\|x^T A\|} B \right\| \\ &= \max_{\|y\|=1} \sigma_{\min(A)} \left\| y^T B \right\| \\ &= \sigma_{\min(A)} \|B\|,\end{aligned} \tag{27}$$

Hence,

$$\|\mathbf{g}_t^S\| \geq \frac{\alpha}{\beta}(\|\mathbf{g}_t\| - \beta\epsilon) \tag{28}$$

Therefore, on the subset we have

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^k}{2\beta^{k+1}} \|\mathbf{g}_t^S\|^2 \tag{29}$$

$$\leq -\frac{\alpha^k}{2\beta^{k+1}} (\frac{\alpha}{\beta})^2 (\|\mathbf{g}_t\| - \beta\epsilon)^2 \tag{30}$$

$$= -\frac{\alpha^{k+2}}{2\beta^{k+3}} (\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{31}$$

The algorithm stops descending when $\|\mathbf{g}_t\| = \beta\epsilon$. From strong convexity we know that

$$\|\mathbf{g}_t\| = \beta\epsilon \geq \alpha \|w - w_*\| \tag{32}$$

Hence, we get

$$\|w - w_*\| \leq \beta\epsilon/\alpha. \tag{33}$$

As such we have Corollary 4.2. and when we set $k = 1$ we have proof of Theorem 4.1. □

Descent property for Equation 5 For a strongly convex function, $\mathcal{L}$, we have that the diagonal elements of the Hessian are positive (Yao et al., 2020). This allows the diagonal to approximate the full Hessian which has good convergence properties.

Given a function $\mathcal{L}(w)$ which is strongly convex and strictly smooth, we have that $\nabla^2\mathcal{L}(w)$ is upper and lower bounded by two constants β and α, so that $\alpha I \leq \nabla^2\mathcal{L}(w) \leq \beta I$, for all w. For a strongly convex function the diagonal elements in diag($\mathbf{H}_t$) are all positive, and we have:

$$\alpha \leq e_i^T \mathbf{H}_t e_i = e_i^T \text{diag}(\mathbf{H}_t) e_i = \text{diag}(\mathbf{H}_t)_{i,j} \leq \beta \tag{34}$$

where e_j represents the natural basis vectors. Therefore, the diagonal entries of diag($\mathbf{H}_t$) are in the range $[\alpha, \beta]$. Therefore, the average of a subset of the numbers are still in the range $[\alpha, \beta]$. As such, we can prove that Eq. (10) has the same convergence rate as its full matrix counterpart, by following the same proof as Theorem 4.1.

A.2. Proof of Theorem 4.3 and 4.4

A loss function $\mathcal{L}(w)$ is considered μ-PL on a set $\mathcal{S}$, if the following holds:

$$\frac{1}{2}\|\mathbf{g}\|^2 \geq \mu \left(\mathcal{L}(w) - \mathcal{L}\left(w_*\right)\right), \forall w \in \mathcal{S} \tag{35}$$

where w_* is a global minimizer. When additionally $\mathcal{L}\left(w_*\right) = 0$, the μ-PL condition is equivalent to the μ-PL* condition

$$\frac{1}{2}\|\mathbf{g}\|^2 \geq \mu\mathcal{L}(w), \forall w \in \mathcal{S}. \tag{36}$$

Theorem 4.3. *Assume that the loss function $\mathcal{L}(w)$ is β-smooth, and μ-PL* on a set $\mathcal{W}$, and S is a weighted subset obtained by* ADACORE *that estimates the preconditioned gradient by an error of at most ϵ, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate η, gradient descent with update rule of Eq.* (2) *applied to the subsets have the following convergence behavior at iteration t:*

$$\mathcal{L}(w_t) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})^t\mathcal{L}(w_0) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}), \tag{16}$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients.

For Lipschitz continuous $\mathbf{g}$ and μ-PL* condition, gradient descent on the entire dataset yields

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t\|^2 \leq -\eta\mu\mathcal{L}(w_t), \tag{37}$$

and,

$$\mathcal{L}(w_t) \leq (1 - \eta\mu)^t\mathcal{L}(w_0), \tag{38}$$

which was shown in (Liu et al., 2020). We build upon this result to an ADACORE subset.

Proof. From Eq. (26) we have that

$$\frac{\|\mathbf{g}_t\|}{\beta} \leq \|(\mathbf{H}_t)^{-1}\mathbf{g}_t\| \leq \|(\mathbf{H}_t^S)^{-1}\mathbf{g}_t^S\boldsymbol{\gamma}\| + \epsilon \leq \frac{\|\mathbf{g}_t^S\|}{\alpha} + \epsilon \tag{39}$$

Hence solving for $\|\mathbf{g}_t^S\|$ we have,

$$\|\mathbf{g}_t^S\| \geq \frac{\alpha}{\beta}(\|\mathbf{g}_t\| - \beta\epsilon). \tag{40}$$

For the subset we have

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t^S\|^2 \tag{41}$$

By substituting Eq. (51) we have.

$$\leq -\frac{\eta\alpha^2}{2\beta^2}(\|\mathbf{g}_t\| - \beta\epsilon)^2 \tag{42}$$

$$= -\frac{\eta\alpha^2}{2\beta^2}(\|\mathbf{g}_t\|^2 + \beta^2\epsilon^2 - 2\beta\epsilon\|\mathbf{g}_t\|) \tag{43}$$

$$\leq -\frac{\eta\alpha^2}{2\beta^2}(\|\mathbf{g}_t\|^2 + \beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{44}$$

$$\leq -\frac{\eta\alpha^2}{2\beta^2}(2\mu\mathcal{L}(w_t) + \beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{45}$$

Where we can upper bound the norm of $\mathbf{g}_t$ in Eq. (43) by a constant ∇_{max}. And Eq. (55) follows from the μ-PL condition from Eq. (35).

Hence,

$$\mathcal{L}(w_{t+1}) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})\mathcal{L}(w_t) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{46}$$

Since, $\sum_{j=0}^{k}(1 - \frac{\eta\mu\alpha^2}{\beta^2})^j \leq \frac{\beta^2}{\eta\mu\alpha^2}$, for a constant learning rate η we get

$$\mathcal{L}(w_{t+1}) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})^{t+1}\mathcal{L}(w_0) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{47}$$

□

Theorem 4.4. *Under the same assumptions as in Theorem 4.3, for mini-batch SGD with mini-batch size $m \in \mathbb{N}$, the mini-batch SGD with update rule Eq.* (2)*, with learning rate $\eta = \frac{m}{\beta(m-1)}$, applied to the subsets have the following convergence behavior:*

$$\mathbb{E}[\mathcal{L}(w_t)] \leq (1 - \frac{\eta\mu\alpha^2}{2\beta})^t \mathbb{E}[\mathcal{L}(w_0)] - \frac{\alpha^2\eta}{2\beta}(\beta\epsilon^2 - 2\epsilon\nabla_{\max}) \tag{17}$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients, and the expectation is taken w.r.t. the randomness in the choice of mini-batch.

For Lipschitz continuous $\mathbf{g}$ and μ-PL* condition, gradient descent on the entire dataset yields

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t\|^2 \leq -\eta\mu\mathcal{L}(w_t), \tag{48}$$

and,

$$\mathcal{L}(w_t) \leq (1 - \eta\mu)^t \mathcal{L}(w_0), \tag{49}$$

which was shown in (Liu et al., 2020). We build upon this result to an AdaCore subset.

Proof. From Eq. (26) we have that

$$\frac{\|\mathbf{g}_t\|}{\beta} \leq \|(\mathbf{H}_t)^{-1}\mathbf{g}_t\| \leq \|(\mathbf{H}_t^S)^{-1}\mathbf{g}_t^S\boldsymbol{\gamma}\| + \epsilon \leq \frac{\|\mathbf{g}_t^S\|}{\alpha} + \epsilon \tag{50}$$

Hence solving for $\|\mathbf{g}_t^S\|$ we have,

$$\|\mathbf{g}_t^S\| \geq \frac{\alpha}{\beta}(\|\mathbf{g}_t\| - \beta\epsilon). \tag{51}$$

For the subset we have

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t^S\|^2 \tag{52}$$

Fixing w_t and taking expectation with respect to the randomness in the choice of the batch $i_t^{(1)} \dots i_t^{(m)}$ (noting that those indices are i.i.d.), we have

$$\mathbb{E}_{i_t^{(1)}\dots i_t^{(m)}}[\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t)] \leq -\eta(\alpha - \eta\frac{\beta}{m}(\alpha\frac{m-1}{2} + \beta)\mathcal{L}(w_t) \tag{53}$$

$$\leq -\underbrace{\eta(1 - \frac{\eta\beta(m-1)}{2m})}_{c_1}\|\mathbf{g}_t\|^2 + \underbrace{\frac{\eta^2\beta\lambda}{m}}_{c_2}\mathcal{L}(w_t) \tag{54}$$

$$\leq -c_1\frac{\alpha^2}{\beta^2}\|\mathbf{g}_t + \beta\epsilon\|)^2 + c_2\mathcal{L}(w_t) \tag{55}$$

We can upper bound the norm of $\mathbf{g}_t$ in Eq. (55) by a constant ∇_{max}. And Eq. (55) follows from the μ-PL condition from Eq. (35) and assuming $\eta \leq \frac{2}{\beta}$.

$$\leq -c_1\frac{\alpha^2}{\beta^2}(\mu\mathcal{L}(w_t) - 2\nabla_{max}\beta\epsilon + \beta^2\epsilon^2) + c_2\mathcal{L}(w_t) \tag{56}$$

$$\leq -\eta(1 - \frac{\eta\beta(m-1)}{2m})\frac{\alpha^2}{\beta^2}(\mu\mathcal{L}(w_t) - 2\nabla_{max}\beta\epsilon + \beta^2\epsilon^2) + \frac{\eta^2\beta\lambda}{m}\mathcal{L}(w_t) \tag{57}$$

$$= \eta(\mu\frac{\alpha^2}{\beta^2} - \eta\frac{\beta}{m}(\mu\frac{\alpha^2(m-1)}{\beta^2 2} + \lambda))\mathcal{L}(w_t) + \frac{\eta^2\beta\lambda}{m}\mathcal{L}(w_t) + c_1\frac{\alpha^2}{\beta^2}(2\nabla_{max}\beta\epsilon - \beta^2\epsilon^2) \tag{58}$$

$$= \eta\mu\frac{\alpha^2}{\beta^2}(1 - \eta\beta\frac{m-1}{2m})\mathbb{E}[\mathcal{L}(w_t)] + \eta\frac{\alpha^2}{\beta^2}(1 - \eta\beta\frac{m-1}{2m})(2\nabla_{max}\beta\epsilon - \beta^2\epsilon^2) \tag{59}$$

By optimizing the quadratic term in the upper bound with respect to η we get $\eta = \frac{m}{\beta(m-1)}$.

$$\mathbb{E}[\mathcal{L}(w_{t+1})] \leq (1 - \frac{\mu\alpha^2 m}{2\beta^2(m-1)})\mathbb{E}[\mathcal{L}(w_t)] + \frac{\alpha^2 m}{\beta^2}\frac{2\nabla_{max}\beta\epsilon - \beta^2\epsilon^2}{2\beta(m-1)} \tag{60}$$

Hence,

$$\mathbb{E}[\mathcal{L}(w_{t+1})] \leq \left(1 - \frac{\eta^*(m)\mu\alpha^2}{2\beta}\right)\mathbb{E}[\mathcal{L}(w_t)] + \frac{\alpha^2\eta^*(m)}{\beta}(\nabla_{max}\epsilon - \beta\epsilon^2/2) \tag{61}$$

□

A.3. Discussion on Greedy to Extract Near-optimal Coresets

As discussed in Section 4.4, a greedy algorithm can be applied to find near-optimal coresets that estimate the general descent direction with an error of at most ϵ by solving the submodular cover problem Eq. (13). For completeness, we include the pseudocode of the greedy algorithm in Algorithm 1. The ADACORE algorithm is run per class.

Algorithm 1 ADACORE (Adaptive Coresets for Accelerating first and second order optimization methods)

Require: Set of component functions f_i for $i \in V = [n]\}$.
Ensure: Subset $S \subseteq V$ with corresponding per-element stepsizes $\{\gamma\}_{j\in S}$.
1: $S_0 \leftarrow \emptyset, s_0 = 0, i = 0$.
2: **while** $F(S) < C_1 - \epsilon$ **do**
3: $\quad j \in \arg\max_{e\in V\setminus S_{i-1}} F(e|S_{i-1})$
4: $\quad S_i = S_{i-1} \cup \{j\}$
5: $\quad i = i + 1$
6: **end while**
7: **for** $j = 1$ to $|S|$ **do**
8: $\quad \gamma_j = \sum_{i\in V} \mathbb{I}\big[j = \arg\min_{s\in S}\max_{w\in\mathcal{W}} \|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\mathbf{g}_{t,j}\|\big]$
9: **end for**

B. Bounding the Norm of Difference Between Preconditioned Gradients

B.1. Convex Loss Functions

We show the normed difference for ridge regression. Similar results can be deduced for other loss functions such as square loss (Allen-Zhu et al., 2016), logistic loss, smoothed hinge losses, etc.

For ridge regression $f_i(w) = \frac{1}{2}(\langle x_i, w\rangle - y_i)^2 + \frac{\lambda}{2}\|w\|^2$, we have $\nabla f_i(w) = x_i(\langle x_i, w\rangle - y_i) + \lambda w$. Furthermore for invertable Hessian H, let $H_i^{-1} = A$ and $H_j^{-1} = B$. Therefore,

$$\|A\nabla f_i(w) - B\nabla f_j(w)\| = \|A(x_i\langle x_i, w\rangle - x_i y_i + \lambda w) - B(x_j\langle x_j, w\rangle - x_j y_j + \lambda w)\| \tag{62}$$

$$= \|Ax_i\langle x_i, w\rangle - Bx_j\langle x_j, w\rangle + Bx_j y_j - Ax_i y_i + \lambda(A-B)w\| \tag{63}$$

$$= \|Ax_i\langle x_i, w\rangle + Bx_j\langle x_i, w\rangle - Bx_j\langle x_i, w\rangle - Bx_j\langle x_j, w\rangle + Bx_j y_j - Ax_i y_i + Bx_j y_i - Bx_j y_i + \lambda(A+B)w\| \tag{64}$$

$$= \|\langle x_i, w\rangle(Ax_i - Bx_j) + \langle x_i - x_j, w\rangle Bx_j + (y_j - y_i)Bx_j + y_i(Bx_j - Ax_i) + \lambda(A-B)w\| \tag{65}$$

$$= \|(\langle x_i, w\rangle - y_i)(Ax_i - Bx_j) + (\langle x_i - x_j, w\rangle + y_j - y_i)Bx_j + \lambda(A+B)w\| \tag{66}$$

$$\leq |\langle x_i, w\rangle - y_i|\|Ax_i - Bx_j\| + |\langle x_i - x_j, w\rangle + y_j - y_i|\|Bx_j\| + \lambda\|(A-B)w\| \tag{67}$$

$$\leq |\langle x_i, w\rangle - y_i|(\|A-B\|\|x_i\| + \|B\|\|x_i - x_j\|) + |\langle x_i - x_j, w\rangle + y_j - y_i|\|Bx_j\| + \lambda\|(A+B)w\| \tag{68}$$

$$\leq O(\|w\|)(\|A-B\| + \|B\|\|x_i - x_j\|) + O(\|w\|)\|B\|\|x_j\|\|x_i - x_j\| \tag{69}$$

$$\leq O(\|w\|)(\|A\| + \|B\| + \|B\|\|x_i - x_j\|) + O(\|w\|)\|B\|\|x_j\|\|x_i - x_j\| \tag{70}$$

In Eq. (70) we have the norm of the inverse of the Hessian matrix. Since H is invertible we have $\min_i \sigma_i > 0$,

$$\min_i \sigma_i = \inf_{x \neq 0} \frac{\|Hx\|_2}{\|x\|_2} \iff \frac{1}{\min_i \sigma_i} = \sup_{x \neq 0} \frac{\|x\|_2}{\|Hx\|_2} \tag{71}$$

$$\frac{1}{\min_i \sigma_i} = \sup_{x \neq 0} \frac{\|x\|_2}{\|Hx\|_2} = \sup_{H^{-1}z \neq 0} \frac{\left\|H^{-1}z\right\|_2}{\|z\|_2} = \sup_{z \neq 0} \frac{\left\|H^{-1}z\right\|_2}{\|z\|_2} = \left\|H^{-1}\right\|_2, \tag{72}$$

where the substitution $Hx = z$ was made, and utilized that $H^{-1}z = 0 \iff z = 0$ since H is invertible. Hence,

$$\leq O(\|w\|)\|B\|\|x_i - x_j\| \tag{73}$$

$$\leq O(\|w\|)\|x_i - x_j\| \tag{74}$$

For $\|x_i\| \leq 1$, and $|y_i - y_j| \approx 0$.

Assuming that $\|w\|$ is bounded for all $w \in \mathcal{W}$, an upper bound on the euclidean distance between preconditioned gradients can be precomputed.

B.2. Neural Networks

We closely follow proofs from (Katharopoulos & Fleuret, 2018) and (Mirzasoleiman et al., 2020) to show that we can bound the difference between the Hessian inverse preconditioned gradients of an entire NN up to a constant of the difference between the Hessian inverse preconditioned gradients of the last layer of the NN, between arbitrary datapoints i and j.

Consider an L-layer perceptron, where $w^{(l)} \in \mathbb{R}^{M_l x M_{l-1}}$ is the weight matrix for the l^{th} layer with M_l hidden units. Furthermore assume $\sigma^{(l)}(.)$ is a Lipschitz continuous activation function. Then we let,

$$x_i^{(0)} = x_i, \tag{75}$$

$$z_i^{(l)} = w^{(l)} x_i^{(l-1)}, \tag{76}$$

$$x_i^{(l)} = \sigma^{(l)}\left(z_i^{(l)}\right). \tag{77}$$

With,

$$\Sigma_l'\left(z_i^{(l)}\right) = \text{diag}\left(\sigma'^{(l)}\left(z_{i,1}^{(l)}\right), \cdots \sigma'^{(l)}\left(z_{i,M_l}^{(l)}\right)\right) \tag{78}$$

$$\Delta_i^{(l)} = \Sigma_l'\left(z_i^{(l)}\right) w_{l+1}^T \cdots \Sigma_l'\left(z_i^{(L-1)}\right) w_L^T. \tag{79}$$

We have,

$$\|\mathbf{H}_i^{-1}\mathbf{g}_i - \mathbf{H}_j^{-1}\mathbf{g}_j\| \tag{80}$$

$$= \left\| \left(\Delta_i^{(l)} \Sigma_L'(z_i^{(L)})(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right)(x_i^{(l-1)})^T - \left(\Delta_j^{(l)} \Sigma_L'(z_j^{(L)})(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right)(x_j^{(l-1)})^T \right\| \tag{81}$$

$$\leq \left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| \cdot \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| \tag{82}$$

$$+ \left\|\Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\| \cdot \left\|\Delta_i^{(l)}\left(x_i^{(l-1)}\right)^T - \Delta_j^{(l)}\left(x_j^{(l-1)}\right)^T\right\|$$

$$\leq \left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| \cdot \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| \tag{83}$$

$$+ \left\|\Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\| \cdot \left(\left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| + \left\|\Delta_j^{(l)}\right\| \cdot \left\|x_j^{(l-1)}\right\|\right)$$

$$\leq \underbrace{\max_l \left(\left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\|\right)}_{c_{l,i}} \cdot \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| \tag{84}$$

$$+ \underbrace{\left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\| \cdot \max_{l,i,j} \left(\left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| + \left\|\Delta_j^{(l)}\right\| \cdot \left\|x_j^{(l-1)}\right\|\right)}_{c_2}$$

From (Katharopoulos & Fleuret, 2018), (Mirzasoleiman et al., 2020), we have that the variation of the gradient norm is mostly captured by the gradient of the loss function with respect to the pre-activation outputs of the last layer of our neural network. Here we have a similar result, where, the variation of the gradient preconditioned on the inverse of the Hessian norm is mostly captured by the gradient preconditioned on the inverse of the Hessian of the loss function with respect to the pre-activation outputs of the last layer of our neural network. Assuming $\left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\|$ is bounded, we get

$$\|\mathbf{H}_i^{-1}\mathbf{g}_i - \mathbf{H}_j^{-1}\mathbf{g}_j\| \leq c_1 \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| + c_2 \tag{85}$$

where c_1, c_2 are constants. The above holds for an affine operation followed by a slope-bounded non-linearity ($|\sigma'(w)| \leq K$).

B.3. Analytic Hessian for Logistic Regression

Here we provide the analytical formulation of the Hessian of the binary cross entropy loss per data point n with respect to weights $\mathbf{w}$ for Logistic Regression.

For Binary Logistic Regression we have a loss function $\mathcal{L}(\mathbf{w})$ defined as:

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{N} l_i(\mathbf{w}) = -\sum_{i=1}^{N} y_i ln(\hat{\sigma}) + (1-y_i) ln(1-\hat{\sigma}), \text{ where } \hat{\sigma}_i = \sigma(\mathbf{w^T x_i} + b) \tag{86}$$

and σ is the sigmoid function.

We form a Hessian matrix for each data point i based on loss function $\ell_i(\mathbf{w})$ as follows:

$$H_n = \left(\begin{array}{c|c} \frac{\partial}{\partial \mathbf{w^2}} l_i(\mathbf{w}) & \frac{\partial}{\partial \mathbf{w} \partial b} l_i(\mathbf{w}) \\ \hline \frac{\partial}{\partial b \partial \mathbf{w}} l_i(\mathbf{w}) & \frac{\partial}{\partial b \partial b} l_i(\mathbf{w}) \end{array}\right) = \left(\begin{array}{c|c} \hat{\sigma}_i(1-\hat{\sigma}_i)\mathbf{x_i}\mathbf{x_i^T} & \hat{\sigma}_i(1-\hat{\sigma}_i)\mathbf{x_i} \\ \hline [\hat{\sigma}_i(1-\hat{\sigma}_i)\mathbf{x_i}]^T & \hat{\sigma}_i(1-\hat{\sigma}_i) \end{array}\right)$$

This allows us to analytically form the Hessian information per point which is needed to precompute a single coreset which will be used throughout training of the convex regularized logistic regression problem.

C. Further Empirical Evidence

C.1. AdaCore estimates full gradient closely, reaching smaller loss

AdaCore obtains a better estimate of the preconditioned gradient by considering curvature and gradient information compared to the state-of-the-art algorithm Craig and random subsets. This is quantified by calculating the difference

between weighted gradients of coresets and the gradient of the complete dataset. Fig 6, shows the difference in loss reached by ADACORE vs CRAIG over different subset sizes. This shows that corsets selected using ADACORE to classify the Ijcnn1 dataset using logistic regression can reach a lower loss over varying subset sizes than CRAIG.

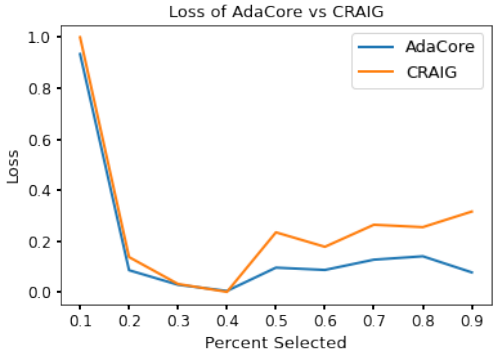


Figure 6: Normalized loss for Logistic Regression over different subset sizes on Ijcnn1 dataset using SGD. ADACORE corsets, considering curvature information, to classify Ijcnn1 dataset using logistic regression consistently reaches a lower loss compared to CRAIG, which only considers the gradient information.

C.2. Class imbalance CIFAR-10

To provide further empirical evidence, we include results using a class-imbalanced version of the CIFAR-10 dataset for ResNet18. We skewed the class distribution linearly, keeping 90% of class 9, 80% of class $8 \ldots 10\%$ of class 1, and 0% of class 0, and trained for 200 epochs. Selecting a coreset for every epoch can be computationally expensive; instead, one can compute a coreset once every R epochs. Here we investigate ADACORE's performance on various R values. As Table 5 shows, ADACORE can withstand class imbalance much better than CRAIG and randomly selected subsets. When $R = 20$, ADACORE achieves 57.3% final test accuracy, $+8.7\%$ above CRAIG, $+2.6\%$ above Random, 27.4% above GRADMATCH and 36.2% above GLISTER.

Table 5: CIFAR-10 Class Imbalance, ResNet18. Final test accuracy and percent of full data selected (in parentheses). Trained with SGD + Momentum, selecting a coreset every R epochs that is S percent of the full dataset. Note ADACORE has greater accuracy while seeing fewer data points.

Accuracy	$S = 1\% R = 20$	$S = 1\% R = 10$	$S = 1\% R = 5$
ADACORE	**57.3%** ± 0.5 (**5%**)	**57.12** ± 0.96 (**9.5%**)	**60.2%** ± 0.36 (**14.5%**)
CRAIG	$48.6\% \pm 0.8$ (8%)	55 ± 1 (16%)	$53.05\% \pm 0.24$ (27.5%)
Random	$54.7\% \pm 0.3$ (8%)	54.6 ± 0.76 (18%)	$54.6\% \pm 0.74$ (33.2%)
GRADMATCH	$29.9\% \pm 0.4$ (8.2%)	$29.1\% \pm 0.8$ (14.7%)	$32.75\% \pm 0.83$ (23.2%)
GLISTER	$21.1\% \pm 0.42$ (8.6%)	$17.2\% \pm 0.75$ (16%)	$14.4\% \pm 0.83$ (22.2%)

Not only is ADACORE able to outperform CRAIG, random, GRADMATCH, and GLISTER, but it can do so while selecting a smaller fraction of the data points during training, as shown under all settings in Table 5.

C.3. Class imbalance BDD100k

Additionally, we compared ADACORE to CRAIG and random selection for the BDD100k dataset, which has seven inherently imbalanced classes and 100k data points. We train ResNet50 with SGD + momentum for 100 epochs choosing subset size (s = 10%) every (R = 20) epochs on the weather prediction task. We see that ADACORE can outperform CRAIG by 2% and random by 8.8% seen in Table 6.

Additionally, Table 7 shows that ADACORE outperforms baseline methods on BDD100k providing 2.3x speedup vs. training on the entire dataset and a 1.8x speedup vs. random. We see that CRAIG, GRADMATCH & GLISTER do not reach the accuracy of ADACORE even given more time and epochs. The epoch value is seen in parenthesis by accuracy. These experiments were run with SGD+momentum.

Table 6: ADACORE outperforms other baseline subset selection algorithms as well as training on the full dataset, reaching a better accuracy in less time. This provides up to a 2.3x speedup compared to to the state of the art.

SGD + Momentum Accuracy	**S = 10% R = 20**
ADACORE	74.3%
CRAIG	72.3%
Random	65.5%

Table 7: ADACORE outperforms other baseline subset selection algorithms as well as training on the full dataset, reaching a better accuracy in less time. This provides up to a 2.3x speedup compared to to the state of the art.

	BDD100k		Speedup over	
$S = 10\%$ R = 20	Accuracy (epoch)	Time (s)	Rand	Full
ADACORE	$74.3\%(100)$	7331	**1.8**	**2.3**
CRAIG	$73.1\%(150)$	10996	1.3	1.6
Random	$73.3\%(180)$	13050	1	1.2
GRADMATCH	$72\%(200)$	14040	.7	1.1
GLISTER	$73\%(200)$	12665	1.03	1.2
Full Dataset	$74.3\%(45)$	16093	0.8	1

C.4. CIFAR-100

Table 8 shows that ADACORE outperforms baseline methods on CIFAR100, providing 4x speedup vs. training on the entire dataset and a 3.8x speedup vs. Random. We see that CRAIG, GRADMATCH and GLISTER do not reach the accuracy of ADACORE even given more time and epochs. The epoch value is seen in parenthesis by accuracy. These experiments were run with SGD+momentum.

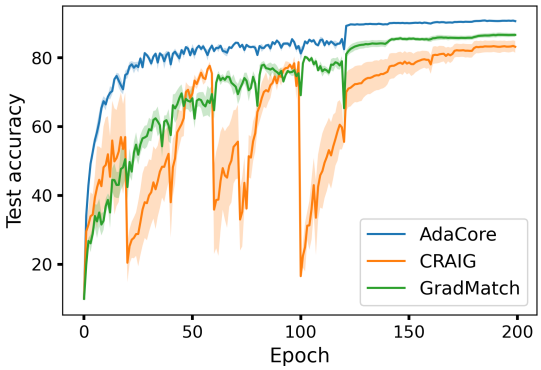
Table 8: ADACORE outperforms other baseline subset selection algorithms as well as training on the full dataset, reaching a better accuracy in less time. This provides up to a 4.3x speedup compared to to the state of the art.

	CIFAR100		Speedup over	
S = 10% R = 20	Accuracy (epoch)	Time (s)	Rand	Full
ADACORE	58.8%(200)	341	**4.3**	**2.8**
CRAIG	57.3%(250)	426	3.5	2.2
Random	58.1%(864)	1470	1	0.65
GRADMATCH	57%(200)	980	1.5	0.97
GLISTER	56%(300)	1110	1.3	0.86
Full Dataset	59% (40)	960	1.5	1

C.5. When first order coresets fail, continued

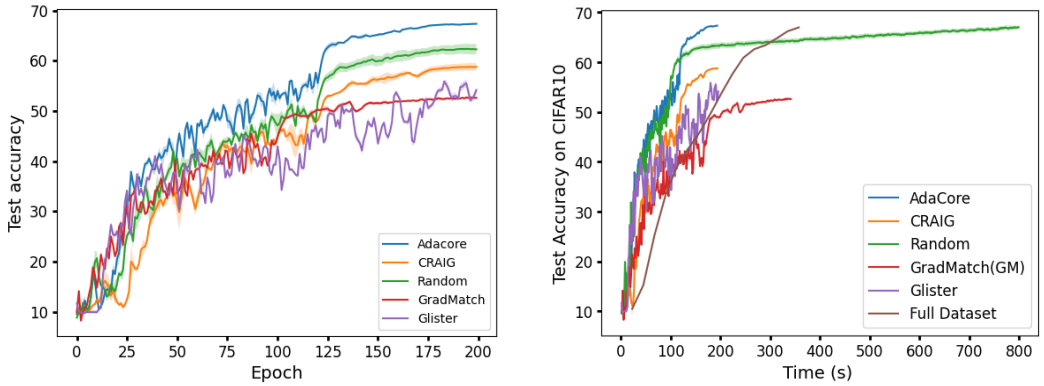
By preconditioning with curvature information, ADACORE is able to magnify smaller gradient dimensions that would otherwise be ignored during coreset selection. Moreover, it allows ADACORE to include points with similar gradients but different curvature properties. Hence, ADACORE can select more diverse subsets compared to CRAIG as well as GRADMATCH. This allows ADACORE to outperform first order coreset methods in many regimes, such as when subset size is large (e.g. $\geq$10%) and for larger batch size (e.g. $\geq$ 128).

In addition to the results shown in Figure 3a, (reproduced here as Fig 8a) where $R = 1$, ADACORE outperforms CRAIG as well as GRADMATCH when we increase the coreset selection period R. Fig 7 shows that for larger R, first-order methods succumb to catastrophic forgetting each time a new subset is chosen, whereas ADACORE achieves a smooth rise in classification accuracy. This increased stability between coresets is another benefit of ADACORE's greater selection diversity.



(a) ADACORE with gradient w.r.t the penultimate layer, training with SGD + Momentum

Figure 7: Classification accuracy of ResNet20 across training on the CIFAR10 dataset, selecting coresets with ADACORE, CRAIG and GRADMATCH. Here, all coreset selection methods used the gradients of the model's last layer (dimension 64). The algorithms were calculated every $R = 20$ epochs with coreset size $S = 10\%$. Note that CRAIG and GRADMATCH are vulnerable to catastrophic forgetting, but not ADACORE.



(a) Accuracy vs. Epoch (b) Accuracy vs. Time

Figure 8: (a) Test accuracy of ADACORE, CRAIG, Random, GradMatch and GLISTER with ResNet-18 selecting subsets of size 1% each epoch, batch size 256. (b) Training ResNet-18 on subsets of size S=1% selected every R=1 epoch, with ADACORE, CRAIG, GLISTER and GRADMATCH for 200 epochs vs. Random for 1000 epochs and full for 15 epochs. ADACORE outperforms baselines by providing 2x speedup over full, and more than 4.5x speedup over Random.

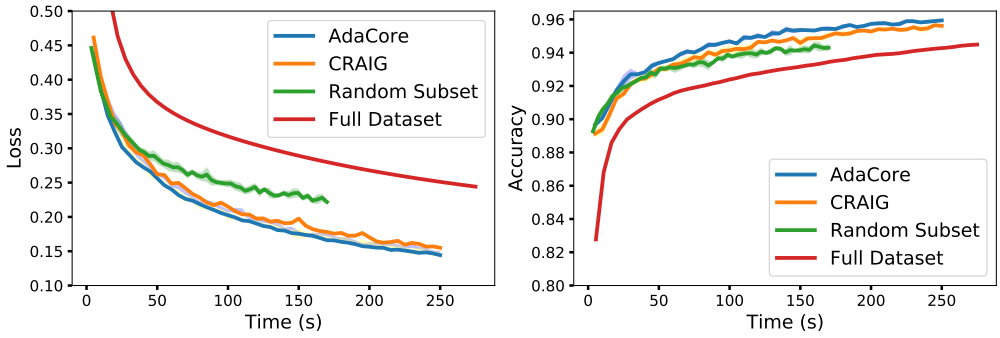
Interestingly, ADACORE achieves higher final test accuracy while selecting a smaller fraction of data points to train on during the training than CRAIG. Note that since ADACORE takes curvature into account while selecting the coresets, it can successfully select data points with a similar gradient but different curvature properties and extract a more diverse set of data points than CRAIG. However, as the coresets found by ADACORE provide a close estimation of the full preconditioned gradients for several epochs during training, the number of distinct data points selected by ADACORE is smaller than CRAIG.

For completeness we provide Fig 8b, in which we allow training random subset selection 1000 epochs. We see that it takes over 4.5x longer for Random to near the accuracy of ResNet18 trained with ADACORE and Full. We use the same experimental setup as seen in Fig 8a.

C.6. MNIST

For our MNIST classifier, we use a fully-connected hidden layer of 100 nodes and ten softmax output nodes; sigmoid activation and L2 regularization with $\mu = 10^{-4}$ and mini-batch size of 32 on the MNIST dataset of handwritten digits containing 60,000 training and 10,000 test images all normalized to [0,1] by division with 255. We apply SGD with a momentum of 0.9 to subsets of size 40% of the dataset chosen at the beginning of each epoch found by ADACORE, CRAIG, and random. Fig 9 compares the training loss and test accuracy of the network trained on coresets chosen by ADACORE, CRAIG, and random, with that of the entire dataset. We see that ADACORE can benefit from the second-order information

and effectively finds subsets that achieve superior performance to that of baselines and the entire dataset. At the same time, it achieves a 2.5x speedup over training on the entire dataset.



(a) MNIST (b) MNIST

Figure 9: Test accuracy and training loss of SGD with momentum applied to subsets found by ADACORE vs. CRAIG, and random subsets on MNIST with a 2-layer neural network. ADACORE achieves 2.5x speedup and better test accuracy, compared to training on full dataset.

C.7. How batch size affects coreset performance

We see in Table 9 that training with larger batch size on subsets selected by ADACORE can achieve a superior accuracy. We reproduce Table 3 here with standard deviation values.

Table 9: Training ResNet18 with S=1% subsets every R=1 epoch from CIFAR10 using batch size b= 512, 256, 128. ADACORE can leverage larger mini-bath size and obtain a larger accuracy gap to CRAIG and Random. For b=512, we have 1 mini-batch (GD).

	ADACORE	CRAIG	Rand	Gap/ CRAIG	Gap/ Rand
GD b=512	$58.32\% \pm 0.45$	$56.32\% \pm 0.32$	$49.14\% \pm 1.19$	1.69%	8.91%
SGD b=256	$68.23\% \pm 0.2$	$58.3\% \pm 1.38$	$60.7\% \pm 1.04$	9.93%	8.16%
SGD b=128	$66.89\% \pm 0.73$	$58.17\% \pm 1.34$	$65.46\% \pm 0.93$	8.81%	1.52%

C.8. Potential Social Impacts

Regarding social impact, our coreset method can outperform other methods in accuracy while selecting fewer data points over training and providing over 2.5x speedup. This will allow for a more efficient learning pipeline resulting in a lesser environmental impact. Our method can significantly decrease the financial and environmental costs of learning from big data. The financial costs are due to expensive computational resources, and environmental costs are due to the substantial energy consumption and the produced carbon footprint.