# Towards Sustainable Learning: Coresets for Data-efficient Deep Learning

**Yu Yang** [1] **Hao Kang** [2] **Baharan Mirzasoleiman** [1]

## Abstract

To improve the efficiency and sustainability of learning deep models, we propose CREST, the first scalable framework with rigorous theoretical guarantees to identify the most valuable examples for training non-convex models, particularly deep networks. To guarantee convergence to a stationary point of a non-convex function, CREST models the non-convex loss as a series of quadratic functions and extracts a coreset for each quadratic sub-region. In addition, to ensure faster convergence of stochastic gradient methods such as (mini-batch) SGD, CREST iteratively extracts multiple mini-batch coresets from larger random subsets of training data, to ensure nearly-unbiased gradients with small variances. Finally, to further improve scalability and efficiency, CREST identifies and excludes the examples that are learned from the coreset selection pipeline. Our extensive experiments on several deep networks trained on vision and NLP datasets, including CIFAR-10, CIFAR-100, TinyImageNet, and SNLI, confirm that CREST speeds up training deep networks on very large datasets, by 1.7x to 2.5x with minimum loss in the performance. By analyzing the learning difficulty of the subsets selected by CREST, we show that deep models benefit the most by learning from subsets of increasing difficulty levels [1].

## 1. Introduction

Large datasets have enabled over-parameterized neural networks to achieve unprecedented success (Devlin et al., 2018; Brown et al., 2020; Zhai et al., 2022). However, training such models, with millions or billions of parameters, on large data requires expensive computational resources, which consume substantial energy, leave a massive amount of carbon footprint, and often soon become obsolete and turn into e-waste (Asi & Duchi, 2019; Schwartz et al., 2019; Strubell et al., 2019). While there has been a persistent effort to improve the performance and reliability of machine learning models (Brown et al., 2020; Xiao et al., 2015; Zhai et al., 2022), their sustainability is often neglected.

Indeed, not all examples are equally valuable or even required to guarantee a good generalization performance. To address the sustainability and efficiency of machine learning, one approach involves selecting the most relevant data for training. A recent line of work (Mirzasoleiman et al., 2020; Killamsetty et al., 2021b;a; Pooladzandi et al., 2022) showed that for strongly convex models, a weighted subset (coreset) of data that closely matches full gradient—sum of the gradient of all the training examples—provide convergence guarantees for (Incremental) Gradient Descent. Such coresets speed up learning by up to 6x. Intuitively, this is possible as for popular strongly convex models—logistic, linear regression, and regularized support vector machines—the gradient error of a coreset during the *entire training* can be upper-bounded in advanced (Mirzasoleiman et al., 2020).

Unfortunately, we cannot simply apply the same idea to non-convex models, for three reasons. First, for non-convex models, training dynamics—loss and gradient of examples—drastically change during the training and cannot be upper-bounded beforehand. As a result, the importance of examples for learning changes throughout training, and one coreset cannot guarantee convergence anymore. Second, non-convex models are learned with (mini-batch) stochastic gradient methods, such as (mini-batch) SGD, which require unbiased estimates of the full gradient with a bounded variance. Existing coresets that capture the gradient of the full data cannot provide any guarantee for stochastic gradient methods, as the gradient of mini-batches selected from the coreset may be biased and have a large variance. Finally, iteratively selecting coresets from full data becomes very expensive for large datasets, and does not yield speedup.

In this work, we address the above challenges and propose CREST, a rigorous method to find coresets for non-convex models, by making the following contributions:

**Coreset selection by modeling the non-convex loss.** To

[*]Equal contribution [1]Department of Computer Science, University of California Los Angeles, USA [2]School of Computer Science, Georgia Institute of Technology. Correspondence to: Yu Yang <yuyang@cs.ucla.edu>, Baharan Mirzasoleiman <baharan@cs.ucla.edu>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]Code can be found at https://github.com/bigml-cs-ucla/crest

ensure a small gradient error throughout training, our key idea is to divide the loss into multiple quadratic sub-regions and find a coreset for learning every quadratic sub-region. To do so, we model the loss of every example as a quadratic function based on its current gradient and curvature information at model parameters $\boldsymbol{w}_{t_l}$. Then, we find a coreset that captures the full gradient at $\boldsymbol{w}_{t_l}$, and keep training on it as long as the quadratic approximated loss of the coreset (sum of quadratic functions corresponding to its elements) closely captures the actual loss of the full data. Otherwise, we update the coreset. In doing so, we ensure a small gradient error during the entire training, which we leverage to guarantee convergence to a stationary point.

**Coresets for (mini-batch) stochastic gradient methods.** To address coreset selection for (mini-batch) stochastic gradient methods, our idea is to sample multiple subsets of training data uniformly at random, and select a mini-batch coreset from every random sample to closely capture its gradient. The gradients of larger random subsets are unbiased estimates of the full gradient, but have a considerably smaller variance. Hence, the mini-batch coreset gradients are nearly unbiased, and have a small variance. Updating the mini-batch coresets based on above piece-wise quadratic loss approximation, ensures a small bias throughout training. This allows providing superior convergence guarantee for training with stochastic gradient methods. Besides, it significantly improves the computational complexity of finding coresets and scales coreset selection to much larger datasets.

**Further improving the efficiency of coreset selection.** To further improve the efficiency and scalability of coreset selection, we make the following observation. When a group of examples are learned, their gradients become nearly zero. Hence, a few examples can well represent the gradient of the corresponding group and the entire group can be safely excluded from coreset selection afterwards. CREST iteratively excludes examples that are learned and have a very small loss during multiple consecutive training iterations, and finds mini-batch coresets from the remaining examples. This speeds up learning, and improves the efficiency and performance of coreset selection, in later stages of training.

Through extensive experiments, we demonstrate the effectiveness of CREST for training various over-parameterized models on different vision and NLP benchmark datasets, including ResNet20 on CIFAR-10 (Krizhevsky et al., 2009), ResNet18 on CIFAR-100 (Krizhevsky et al., 2009), ResNet-50 on TinyImageNet (Russakovsky et al., 2015), and RoBERTa (Liu et al., 2019) on SNLI (Bowman et al., 2015) with 570K examples. CREST is able to achieve 1.7x to 2.5x speedup over training on the full data, while introducing the smallest relative error compared to the baselines. To our knowledge, this is the first time coreset selection has been applied to such large models and datasets in vision and NLP.

Finally, we analyze the examples selected by CREST at different times during the training. We quantify the learning difficulty of every example using the forgettability score (Toneva et al., 2018), which counts the number of times an example is misclassified after being correctly classified during training. We find that early in training, the most effective subsets for learning deep models are easy-to-learn examples. As training proceeds, the model learns the most from examples with increasing levels of learning difficulty. Interestingly, the model never requires training on easiest-to-learn examples to achieve a good generalization performance.

## 2. Related Work

Several heuristics have been recently proposed for finding coresets for training machine learning models. A line of work first fully trains the original model (Birodkar et al., 2019) or a smaller proxy (Coleman et al., 2020). Then, it selects examples with the most centrally located embeddings (Birodkar et al., 2019), highest uncertainty, i.e., the entropy of predicted class probabilities (Coleman et al., 2020), largest forgetting events, i.e., the number of times an example is misclassified after being correctly classified (Toneva et al., 2018), or large expected gradient norm over multiple initializations (Paul et al., 2021). These methods do not yield any speedup or theoretical guarantees.

Another line of work selects examples during training to speed up learning. Importance sampling techniques employ the gradient norm (Alain et al., 2015; Katharopoulos & Fleuret, 2018) or the loss (Loshchilov & Hutter, 2015; Schaul et al., 2015) to reduce the variance of stochastic gradients during the training. However, importance sampling does not provide rigorous convergence guarantees and cannot provide a notable speedup for training deep models. Mindermann et al. (2022) finds examples that are non-noisy, non-redundant, task-relevant, and reduce the loss on a hold-out set the most. This method speeds up training but requires a validation set and does not guarantee convergence.

Most relevant to our work are recent theoretically rigorous techniques that select coresets by iteratively matching the (preconditioned) gradient of full training data, namely (Mirzasoleiman et al., 2020; Killamsetty et al., 2021a; Pooladzandi et al., 2022), or validation set (Killamsetty et al., 2021b). Such methods guarantee convergence to a near-optimal solution, for training (strongly) convex or nearly convex models under Polyak-Lojasiewicz (PL) condition, using Incremental Gradient (IG) methods, or Gradient Descent (GD). However, they do not guarantee convergence for training non-convex models trained with (mini-batch) stochastic gradient methods, and do not scale to very large datasets. Our work addresses the above shortcomings by developing a rigorous and scalable framework to extract coresets for data-efficient deep learning.

## 3. Problem Formulation and Background

The standard approach to training machine learning models is empirical risk minimization (ERM). Formally, given a loss function $\mathcal{L}$, we find the model parameters $\boldsymbol{w}$ that minimize the expected loss on training examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ indexed by $V=\{1,\cdots,n\}$, sampled from distribution $\mathcal{D}$:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\mathcal{W}} \mathcal{L}(\boldsymbol{w}) := \mathbb{E}_{(\boldsymbol{x}_i,y_i)\sim\mathcal{D}}[\mathcal{L}(\boldsymbol{w};(\boldsymbol{x}_i,y_i))]. \quad (1)$$

For over-parameterized models trained on large training data, GD becomes prohibitively slow. Hence, stochastic gradient methods, such as mini-batch SGD are employed in practice. Such methods select one or a mini-batch $\mathcal{M}$ of $m$ examples sampled i.i.d. from the training data, and iteratively step in the negative direction of the stochastic gradient of the sampled examples, scaled by step-size $\eta$:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\frac{1}{m}\sum_{i\in\mathcal{M}} \mathbf{g}_{t,i}, \quad (2)$$

where $\mathbf{g}_{t,i} = \nabla\mathcal{L}_i(\boldsymbol{w}_t) := \nabla\mathcal{L}(\boldsymbol{w}_t;(\boldsymbol{x}_i,y_i))$ is the gradient of example $i$ at iteration $t$. Random examples have an unbiased gradient with a bounded variance, i.e., $\mathbb{E}_{i\in V}[\|\mathbf{g}_{t,i}-\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2]\leq\sigma^2$. Hence, they guarantee convergence with an $\mathcal{O}(1/\sqrt{t})$ rate to a stationary point of a non-convex loss (Ghadimi & Lan, 2013). Importantly, random mini-batches of size $m$ have an unbiased gradient with a reduced variance of $\sigma^2/m$. As long as mini-batches are not too large, mini-batch SGD achieves a faster convergence rate of $\mathcal{O}(1/\sqrt{mt})$ (Wang & Srebro, 2019; Jin et al., 2021).

Existing coreset methods, such as CRAIG (Mirzasoleiman et al., 2020), GRADMATCH (Killamsetty et al., 2021a), and ADACORE (Pooladzandi et al., 2022) find weighted subsets of examples that match the full training gradient (preconditioned on Hessian). Formally, the goal is to find the smallest subset $S\subseteq V$ and corresponding per-element step-sizes (weights) $\gamma_j>0$ that approximate the full gradient with an error at most $\epsilon>0$ for all the possible values of $\boldsymbol{w}_t\in\mathcal{W}$:

$$S^* = \arg\min_{S\subseteq V,\gamma_j\geq 0\ \forall j\in S} |S|, \text{ s.t. } \max_{\boldsymbol{w}_t\in\mathcal{W}} \|\sum_{i\in V}\mathbf{g}_{t,i} - \sum_{j\in S}\gamma_j\mathbf{g}_{t,j}\|\leq\epsilon. \quad (3)$$

Problem (3) requires calculating the maximum gradient error between full and coreset gradient for all $\boldsymbol{w}_t\in\mathcal{W}$, which cannot be computed. To address this, Mirzasoleiman et al. (2020) showed that for several classes of (strongly) convex problems, including regularized linear and ridge regression, and support vector machines (SVMs), the normed gradient difference between data points *during the entire training* can be efficiently upper-bounded by the difference between feature vectors. This allows turning Problem (3) into the following submodular[2] cover problem:

$$S^* = \arg\min |S| \text{ s.t. } C - \sum_{i\in V}\min_{j\in S}\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq C-\epsilon, \quad (4)$$

where $C$ is a big constant. A near-optimal coreset of size $k$ can be found from a ground-set of $n$ elements, using the greedy algorithm with complexity of $\mathcal{O}(n\cdot k)$ as a preprocessing step before training. The weights $\gamma_j$ are calculated as the number of examples $i\in V$ for which $j\in S$ minimizes $\|\boldsymbol{x}_i-\boldsymbol{x}_j\|$. This approach has been adopted by (Killamsetty et al., 2021b; Pooladzandi et al., 2022). Killamsetty et al. (2021a) used orthogonal matching pursuit (OMP) to directly find a weighted coreset, by minimizing the regularized objective in RHS of Problem (3). However, OMP provides weaker guarantees than greedy, and does not always find a large enough subset. Hence, the coreset needs to be augmented with random examples.

For neural networks, finding coresets based on their very high-dimensional gradients is slow and does not yield high-quality coresets. Instead, one can use the gradient of the loss w.r.t the input to the last layer that is shown to capture the variation of the gradient norm well (Katharopoulos & Fleuret, 2018). Such lower-dimensional gradients $\mathbf{g}_{t,i}^L$ can be quickly obtained with a forward pass and can be used instead of the full gradient to find coresets during the training (Mirzasoleiman et al., 2013; Pooladzandi et al., 2022; Killamsetty et al., 2021b). Moreover, with a fixed training budget one can find a subset of size $k$ by solving the following submodular maximization problems, which is the dual of the submodular cover Problem (4):

$$S_t^* = \arg\max_{S\subseteq V} C - \sum_{i\in V}\min_{j\in S}\|\mathbf{g}_{t,i}^L - \mathbf{g}_{t,j}^L\|, \text{ s.t. } |S|\leq k. \quad (5)$$

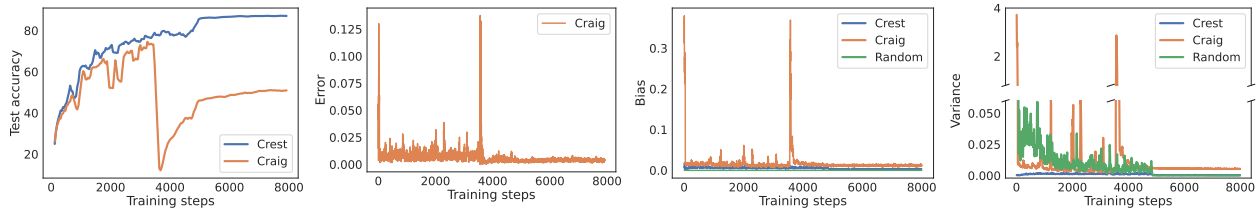
However, it is not clear when the coresets should be updated to guarantee convergence for training non-convex models. Besides, finding coresets from the full data does not scale to training on large datasets. Importantly, the above method only guarantees convergence for (Incremental) GD and cannot guarantee convergence for stochastic gradient methods used for training neural networks, as we will discuss next.

## 4. Coresets for Training Non-convex Models

In this section, we will first discuss the challenges of extracting coresets for deep models, and then introduce our proposed method, CREST, to overcome the above challenges and make coreset selection applicable to neural networks.

**Challenges.** The non-convex loss landscape of the non-convex models makes coreset selection very challenging.

[2]A set function $F: 2^V\rightarrow\mathbb{R}^+$ is submodular if $F(S\cup\{e\})-F(S)\geq F(T\cup\{e\})-F(T)$, for any $S\subseteq T\subseteq V$ and $e\in V\setminus T$. $F$ is *monotone* if $F(e|S)\geq 0$ for any $e\in V\setminus S$ and $S\subseteq V$.




(a) Test Accuracy (b) Error of coreset gradients (c) Bias of mini-batch grads. (d) Variance of mini-batch grads.

Figure 1: Training ResNet20 on CIFAR-10. (a) 10% CRAIG coresets selected at the beginning of every epoch from full data may perform very poorly. This is because, **(C1)**: (b) Coresets may have a large error: $\|\mathbf{g}_{t,S} - \nabla\mathcal{L}(\boldsymbol{w}_t)\|$, after a few training iterations; and **(C2)**: Gradient of weighted mini-batches selected from the coresets may have a (c) large bias $\|\mathbb{E}_i[\mathbf{g}_{t,\mathcal{M}_i}] - \nabla\mathcal{L}(\boldsymbol{w}_t)\|$ and (d) large variance $\mathbb{E}_i[\|\mathbf{g}_{t,\mathcal{M}_i} - \nabla\mathcal{L}(\boldsymbol{w}_t)\|^2]$, where $\mathcal{M}_i \in S$ is a mini-batch and $\mathbf{g}_{t,\mathcal{M}_i} = \mathbb{E}_{j\in\mathcal{M}_i}[\gamma_j \mathbf{g}_{t,j}]$. In contrast, our CREST coresets are nearly unbiased, and have a smaller variance than random mini-batches of same size.

Fig. 1a shows that existing coreset selection methods such as CRAIG (Mirzasoleiman et al., 2020) that find coresets by iteratively solving Eq. (5) at every epoch may perform very poorly for training deep networks, for the following reasons:

**(C1)** For deep networks, the loss functions associated with different data points $\mathcal{L}_i$ change very rapidly (Defazio & Bottou, 2019). Therefore, in contrast to (strongly) convex functions, for which the gradient error of a coreset throughout training can be effectively upper-bounded in advance, e.g., using their feature vectors, such upper bounds cannot be computed for neural networks. That is, even within a relatively small neighborhood $\mathcal{N}$ around $\boldsymbol{w}_t$, the gradient $\nabla\mathcal{L}_i(\boldsymbol{w}_t)$ may be drastically different than $\nabla\mathcal{L}_i(\boldsymbol{w}_t+\boldsymbol{\delta})$ for $\boldsymbol{w}_t+\boldsymbol{\delta}\in\mathcal{N}$. Figure 1b shows that the gradient error of coresets found by CRAIG can be very large after a few training iterations. Here, the challenge is to compute the size of the neighborhood in which a coreset closely captures the full gradient, and update the coreset otherwise.

**(C2)** Coresets found from the full training data guarantee convergence for (Incremental) GD, but cannot provide any guarantee for *stochastic* gradient methods, such as (mini-batch) SGD, that are applied to train neural networks. This is because stochastic methods require unbiased estimates of the full gradient with a bounded variance. However, as the error of the coresets found from the full data may increase during the training, the gradient of mini-batches selected from the coresets may have a large bias. Besides, as some examples may have a very large weight, the variance of weighted mini-batch gradients are much larger than the variance of random (unweighted) mini-batch gradients used when training on the full data. Figure 1c, 1d show that gradient of mini-batches selected from the coreset can have a very large bias and variance. Here, the challenge is to find coresets that are nearly unbiased and have a small variance.

**(C3)** For deep networks, the importance of examples for learning changes over time and hence the coresets should be updated frequently. The greedy algorithm has a complexity of $\mathcal{O}(n \cdot k)$ to find $k$ out of $n$ examples. For large datasets, this prevents the coreset selection methods to achieve a significant speedup. Hence, the challenge is to improve the efficiency of coreset selection for training deep networks.

Next, we discuss how we overcome the above challenges.

### 4.1. Modeling the Non-convex Loss Function

To address the challenge **(C1)** of finding the size of the neighborhood in which a coreset closely captures the full gradient, we model the non-convex loss as a piece-wise quadratic function. In doing so, we reduce the problem of finding coresets for a non-convex objective to finding coresets for a series of quadratic problems. Formally, at every coreset selection step $l$, we find a coreset $S_l$ that captures the full gradient at $\boldsymbol{w}_{t_l}$. Then, we make a quadratic loss approximation $\mathcal{F}^l$ based on the gradient and curvature of the coreset at $\boldsymbol{w}_{t_l}$. We keep training on the coreset $S_l$ within the neighborhood $\mathcal{N}_l$ in which the quadratic approximation closely follows the actual training loss, i.e., we have that $\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}) = \mathcal{F}^l(\boldsymbol{\delta})\ \forall\ \boldsymbol{w}_{t_l}+\boldsymbol{\delta}\in\mathcal{N}_l$. Otherwise, we update the coreset and make a new quadratic approximation. This ensures a small gradient error within $\mathcal{N}_l$, and similarly through the entire training. Hence, convergence can be guaranteed.

In this work, we extract the coresets by greedily solving the submodular Problem (5). But, our piece-wise quadratic approximation can be generally applied to any coreset selection method to check the validity of the coresets, and updating them to guarantee convergence for deep learning.

In the rest of this section, we first discuss how to efficiently estimate the coreset loss as a quadratic function $\mathcal{F}^l$, based on its gradient and curvature at $\boldsymbol{w}_{t_l}$. Then, we discuss finding the size of the neighborhood $\mathcal{N}_l$ in which the quadratic function $\mathcal{F}^l$ closely captures the loss $\mathcal{L}$ of full training data.

**Approximating coreset losses by quadratic functions.** We model the coreset loss within the neighborhood $\mathcal{N}_l$ by a quadratic approximation $\mathcal{F}^l$, using the 2$^{\text{nd}}$-order Taylor series of expansion of $\mathcal{L}(\boldsymbol{w}_{t_l})$ at $\boldsymbol{w}_{t_l}$ where the coreset is extracted:

$$\mathcal{F}^l(\boldsymbol{\delta}) = \frac{1}{2}\boldsymbol{\delta}^T\mathbf{H}_{t_l,S_l}\boldsymbol{\delta} + \mathbf{g}_{t_l,S_l}\boldsymbol{\delta} + \mathcal{L}(\boldsymbol{w}_{t_l}), \qquad (6)$$

where $\mathbf{H}_{t_l,S_l} = \frac{1}{|S_l|}\sum_{j\in S_l}\gamma_j\mathbf{H}_{t_l,j}$ and $\mathbf{g}_{t_l,S_l} = \frac{1}{|S_l|}\sum_{j\in S_l}\gamma_j\mathbf{g}_{t_l,j}$ are the weighted mean of the Hessian and gradient of the examples in the coreset $S_l$. Such modeling is the main idea behind the popular convexification technique in mathematical optimization, which powers Levenberg-Marquardt (Marquardt, 1963) and K-FAC (Martens & Grosse, 2015) optimization methods, among others (Bertsekas, 1979; Carmon et al., 2018; Wang & Srebro, 2019).

To obtain an efficient estimate of the Hessian of the coreset, we use an approximate Hessian operator instead of the full Hessian. Specifically, we employ the Hutchinson's trace estimator method (Hutchinson, 1989) to obtain a stochastic estimate of the coreset Hessian diagonal (Bollapragada et al., 2019; Dembo et al., 1982; Xu et al., 2020; Yao et al., 2018), without having to form the Hessian matrix explicitly:

$$\text{diag}(\mathbf{H}_{t_l,S_l}) = \mathbb{E}[\boldsymbol{z}\odot(\mathbf{H}_{t_l,S_l}\boldsymbol{z})]. \tag{7}$$

This method approximates the Hessian diagonal as the expectation of Hessian $\mathbf{H}_{t_l,S_l}$ multiplied by a random vector $\boldsymbol{z}$ with Rademacher distribution. The multiplication $\mathbf{H}_{t_l,S_l}\boldsymbol{z}$ can be efficiently calculated via backprop on gradients of the coreset multiplied by $\boldsymbol{z}$. i.e., $\mathbf{H}_{t_l,S_l}\boldsymbol{z} = \partial\mathbf{g}_{t_l,S_l}^T\boldsymbol{z}/\partial\boldsymbol{w}_{t_l}$.

As the local gradient and curvature information can be very noisy for neural networks (Yao et al., 2020), to better approximate the global gradient and Hessian information, we smooth them out by applying exponential averaging with parameters $0<\beta_1$, $0<\beta_2<1$:

$$\overline{\mathbf{g}}_{t_l,j} = \frac{(1-\beta_1)\sum_{t=1}^{t_l}\beta_1^{t_l-t}\mathbf{g}_{t,j}}{1-\beta_1^{t_l}}, \tag{8}$$

$$\overline{\mathbf{H}}_{j,t_l} = \sqrt{\frac{(1-\beta_2)\sum_{t=1}^{t_l}\beta_2^{t_l-t}\text{diag}(\mathbf{H}_{t,j})\text{diag}(\mathbf{H}_{t,j})}{1-\beta_2^{t_l}}}. \tag{9}$$

For very large networks, gradient and Hessian diagonal w.r.t. the input to the penultimate layer can be used in Eq. (7-9).

**Estimating the size of the quadratic neighborhoods.** To check the validity of the coreset, we iteratively compare the value of the quadratic loss $\mathcal{F}^l(\boldsymbol{\delta})$ with the value of the actual training loss. For efficiency, we obtain an unbiased estimate of the actual loss on a small random sample of training examples $V_r\subseteq V$, i.e., $\mathcal{L}^r$. We update the coreset $S_l$ and the quadratic approximation $\mathcal{F}^l(\boldsymbol{\delta})$, when the quadratic coreset loss does not closely follow the actual loss estimate $\mathcal{L}^r(\boldsymbol{\delta}+\boldsymbol{w}_{t_l})$. More precisely, every $T_1$ iterations, we compute the ratio of the absolute loss difference to the actual loss, i.e.,

$$\rho_{t_l} = \frac{|\mathcal{F}^l(\boldsymbol{\delta}) - \mathcal{L}^r(\boldsymbol{\delta}+\boldsymbol{w}_{t_l})|}{\mathcal{L}^r(\boldsymbol{\delta}+\boldsymbol{w}_{t_l})}. \tag{10}$$

We consider the quadratic approximation of the coreset loss to be sufficiently accurate if $\rho_{t_l}$ is smaller than a threshold $\tau$. If $\rho_{t_l}\leq\tau$, we keep using the coreset for the $T_1$ subsequent iterations. Otherwise, we find a new coreset and update the quadratic approximation, accordingly. Computing $\rho_{t_l}$ can be done quite efficiently. $\mathcal{F}^l(\boldsymbol{\delta})$ can be efficiently calculated based on the gradient and Hessian of the coreset using Eq. (7-9). $\boldsymbol{\delta}$ is the total amount of updates calculated by the optimization algorithm in $T_1$ training iterations. Calculating $\mathcal{L}^r(\boldsymbol{\delta}+\boldsymbol{w}_{t_l})$ requires an additional forward pass on a subset $V_r$ of data, which we only need once every $T_1$ iterations.

**Remark.** In the initial phase of training, gradients evolve very rapidly. Hence, early in training, the quadratic approximations are accurate in a small neighborhood $\mathcal{N}_l$. Therefore, it is crucial to update the coresets frequently to be able to closely capture the full gradient. In contrast, in the final stage of training, the loss becomes well approximated as a convex quadratic within a sufficiently large neighborhood of the local optimum (Martens & Grosse, 2015). Hence, the same subset can be used for several training iterations. We show in our experiments that for a fixed $\tau$, CREST updates the coresets much less frequently as training proceeds. In practice, $T_1$ can grow proportional to the inverse of the norm of the Hessian diagonal, as we confirm experimentally.

### 4.2. Coresets for (Mini-batch) Stochastic GD

Next, we address the challenge **(C2)** of finding coresets for (mini-batch) stochastic gradient methods, that are used for training deep networks. To address this problem, our main idea is to sample multiple subsets of training data $\{V_1,\cdots,V_P\}$ uniformly at random, and directly select a smaller coreset $S_l^p, p\in[P]$ of the mini-batch size $m$ from each random subset $V_p$. Effectively, instead of selecting a subset to capture the full gradient at $\boldsymbol{w}_{t_l}$, we select multiple *mini-batch coresets* $\{S_l^1,\cdots,S_l^P\}$ at $\boldsymbol{w}_{t_l}$, where each coreset $S_l^p$ is of mini-batch size $m$, and captures the full gradient of a random subset $V_p$ of training data at $\boldsymbol{w}_{t_l}$.

Formally, at every coreset selection iteration $l$, we solve $P$ smaller submodular maximization problems. I.e. for $p\in[P]$:

$$S_l^{p*} = \arg\max_{S\subseteq V_p} C - \sum_{i\in V_p}\min_{j\in S}\|\mathbf{g}_{t_l,i}^L - \mathbf{g}_{t_l,j}^L\|, \text{ s.t. } |S|\leq m, \tag{11}$$

where $\mathbf{g}_{t_l,i}^L$ is the gradient of the loss w.r.t. the input to the last layer of the network at $\boldsymbol{w}_{t_l}$.

Then, we make a quadratic loss approximation of the form Eq. (6) to the *union* of mini-batch coresets $S_l = \bigcup_{p\in[P]} S_l^p$. Each random subset $V_p$ provides an unbiased estimate of the full gradient, and since each mini-batch coreset $S_l^p$ closely captures the gradient of $V_p$, it provides a nearly unbiased estimate of the full gradient. Therefore, the union of the mini-batch coresets $S_l$ also captures the full gradient. However, $S_l$ has a smaller error in capturing the full gradient compared to each of the mini-batch coresets, as small errors of mini-batch coresets cancel each other out (*c.f.* Figure 6a

**Algorithm 1** CoREsets for STochastic GD (CREST)

**Require:** Model parameter $\boldsymbol{w}_0$, mini-batch size $m$, random partition size $r$, learning rate $\eta$, total training iterations $N$, checking interval $T_2$, multipliers $b, h$, thresholds $\alpha, \tau$.
$t \leftarrow 0, T_1 \leftarrow 1$, update $\leftarrow 1$
**while** $t < N$ **do**
  **if** update $== 1$ **then**
    **for** $p = 1$ to $P$ **do**
      Select a random subset $V_p \subseteq V$ s.t. $|V_p| = r$
      $S_l^p \in \arg\max_{\substack{S \subseteq V_p \\ |S| \leq m}} C - \sum_{i \in V_p} \min_{j \in S} \|\mathbf{g}_{t_l,i}^L - \mathbf{g}_{t_l,j}^L\|$
    $S_l = \bigcup_{p \in [P]} S_l^p$
    Calculate $\mathcal{F}^l$ with $\overline{\mathbf{H}}_{t,S_l}, \overline{\mathbf{g}}_{t,S_l}$
  **for** $j = 1$ to $T_1$ **do**
    $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \mathbf{g}_{S_l,t}$
    $t \leftarrow t + 1$
    **if** $t \mod T_2 == 0$ **then**
      $V = \{j \in V | \mathcal{L}_j(\boldsymbol{w}_i) > \alpha, \forall i \in [t - T_2, t]\}$.
  $\boldsymbol{\delta} \leftarrow \boldsymbol{w}_t - \boldsymbol{w}_{t-T_1}$
  Calculate $\rho_t$ from Equation (10).
  **if** $\rho_t > \tau$ **then**
    update $\leftarrow 1$,
    $T_1 \leftarrow h \times \|\overline{\mathbf{H}}_0\| / \|\overline{\mathbf{H}}_t\|$, $P \leftarrow b \times T_1$
  **else**
    update $\leftarrow 0$

in Appendix A.2). Hence, the union of mini-batch coresets makes a more accurate approximation to the full loss.

As long as the quadratic approximation is valid, we can train on *any* of the mini-batch coresets found at $w_{t_l}$. Thus, we keep selecting mini-batch coresets at random from $\{S_l^1, \cdots, S_l^P\}$, and training on them, as long as the quadratic approximation on the union of selected mini-batch coresets accurately captures the full loss according to Eq. (10).

Notably, mini-batch coresets selected from random subsets are nearly unbiased and have a very small variance (*c.f.* Figure 1c, 1d). This is because random subsets $V_p$ of size $r$ are unbiased and have a $r/m$ times smaller variance than that of random mini-batches of size $m$. As long as random subsets are not too large, mini-batch coresets capture the gradient of random subsets very closely. This ensures that the gradients of mini-batch coresets are nearly unbiased and have a nearly $r/m$ times smaller variance than random mini-batches of same size (*c.f.* Figure 9 in Appendix). Note that there is a trade-off. For a fixed mini-batch size, selecting mini-batch coresets from larger random subsets results in a smaller variance but may introduce a larger bias. The very small bias of CREST mini-batch coresets allows guaranteed convergence to a stationary point. At the same time, their smaller variance ensures superior convergence rate compared to training on full data, as we will show in Theorem 4.1. This cannot be achieved by coresets capturing the full gradient.

Note that selecting mini-batches from smaller random partitions speeds up the coreset selection, by breaking one large problem into smaller ones. For example, using the greedy algorithm to solve the submodular maximization Problem (5) has a complexity of $\mathcal{O}(n.k)$ to select $k$ examples from a ground-set of $n$ examples. But, solving Eq. (11) to select $P$ mini-batches of size $k/P$ from random subsets of size $r$ has a total complexity of $\mathcal{O}(P \times r \cdot \frac{k}{P}) = \mathcal{O}(r \cdot k)$.

**Remark.** Early in training, quadratic approximations are accurate in a small neighborhood $\mathcal{N}_l$. Hence, a smaller number of mini-batches can be extracted simultaneously. In the final stage of training, the loss can be well captured by a quadratic function (Martens & Grosse, 2015). Hence, a larger number of mini-batches can be selected simultaneously later in training. In practice, simply increasing $P$ proportional to the inverse of the norm of the Hessian diagonal works well, as we confirm by our experiments.

### 4.3. Further Improving Efficiency of Coreset Selection

To address the challenge **(C3)** of further improving the efficiency and performance of selecting coresets, we make the following observation. Examples are gradually learned during the training. When an example is learned, its gradient and loss become nearly zero. Hence, such examples do not affect training and can be dropped from the coreset selection pipeline to improve efficiently. However, the gradient or loss of an example at a single point during training can be very noisy. To quickly identify such examples, we monitor the loss of examples within non-overlapping intervals of length $T_2$ during the training, and exclude those that consistently have a loss smaller than a threshold $\alpha$. This shrinks the size of the selection problem over time, and allows CREST to focus more on examples that are not learned. Hence, it further improves the efficiency and speedup of the algorithm.

To efficiently exclude the learned examples, we only rely on the loss values calculated for random subsets used for selecting the coresets, and drop examples for which the calculated loss values are smaller than $\alpha$ in an interval of length $T_2$.

Effectively, dropping the learned examples speeds up training by increasing the learning rate. Specifically, dropping $s$ examples with nearly-zero gradients from a ground-set of $n$ examples increases the full (average) gradient by nearly $n/(n-s)$, which has a similar effect to that of increasing the learning rate by a factor of nearly $n/(n-s)$.

The pseudocode of CREST is illustrated in Alg. 1.

The following Theorem shows that training with stochastic gradient descent on mini-batch coresets found by CREST converges to a stationary point of the non-convex loss.

**Theorem 4.1.** *For any $\delta, \lambda > 0$, assume that the function $\mathcal{L}$ is $L$-gradient Lipschitz, and stochastic gradients $\mathbf{g}_{t,i}$ have a bounded variance, i.e., $\mathbb{E}_{i \in V}[\|\mathbf{g}_{t,i} - \nabla \mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \sigma^2$.*

***Case 1 (CREST: Nearly-unbiased).*** *Let step size be* $\eta = \min\{\frac{1}{L}, \frac{\tilde{D}\sqrt{r}}{\sigma\sqrt{N}}\}$*, for some* $\tilde{D} > 0$ *and* $N$ *be the number of training iterations. If the gradient bias of mini-batch coresets* $\mathbb{E}[\|\boldsymbol{\xi}_{t_l}\|] \leq \epsilon\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l})\|$ *and* $\tau \leq \min_l(\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)\| - 3\epsilon\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l})\|)\|\boldsymbol{\delta}_l\|/2\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)$*, for* $0 \leq \epsilon \leq \min\{1, \|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)\|/3\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l})\|\}$*, then with probability at least* $1-\lambda$*,* CREST *will visit a* $\nu$*-stationary point at least once in the following number of iterations:*

$$\tilde{\mathcal{O}}\left(\frac{L(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}{\nu^2}(1 + \frac{\sigma^2}{r\nu^2})\right). \tag{12}$$

***Case 2 (Biased).*** *If the bias of mini-batches* $\mathbb{E}[\|\boldsymbol{\xi}_t\|] \leq \epsilon$*, but* $\epsilon$ *is larger than the full gradient norm anytime during the training, then the number of iterations is:*

$$\tilde{\mathcal{O}}\left(\frac{L(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}{\nu^2 - \epsilon}(1 + \frac{\sigma^2 + r\epsilon^2}{r(\nu^2 - \epsilon)})\right). \tag{13}$$

*In particular, if* $\epsilon \geq \nu^2$*, convergence is not guaranteed.*

The proof can be found in Appendix A.1. At a high level, Theorem 4.1 shows that if mini-batch coresets closely capture gradient of random subsets $V_p$, CREST with a small enough $\tau$, converges to a $\nu$-stationary point of the non-convex loss, but $r/m$ times faster than mini-batch SGD with mini-batch size $m$ on full data, as discussed next.

**Case 1.** As CREST mini-batch coresets capture the gradient of random subsets closely, the bias of mini-batch coresets is a small fraction, $\epsilon \in [0, 1]$, of the full gradient norm at selection time. If $\epsilon \leq \min\{1, \|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)\|/3\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l})\|\}$, a small enough $\tau$ ensures that the bias stays smaller than the full gradient norm within the neighborhood $\mathcal{N}_l$ (*c.f.* Figure 6b in Appendix A.2). Importantly, as the gradient norm shrinks as we get close to a stationary point, a small $\epsilon$ implies that the bias in the entire neighborhood vanishes close to convergence. This guarantees convergence of CREST to a $\nu$-stationary point. Notably, as long as $r \leq \sigma^2/\nu^2$, training with CREST linearly speeds up training by a factor of $r$. In particular, compared to SGD with mini-batch size $m$, CREST speeds up training by a factor of $r/m$.

**Case 2.**If the bias of the mini-batch gradients $\epsilon$ is is larger than the full gradient norm or larger than $\nu$, (mini-batch) SGD does not converge to a $\nu$-stationary point. This explains why larger bias of mini-batches selected from coresets extracted from the full data results in a poor performance (*c.f.* Figure 1a). Besides, the $\epsilon$ bias slows down the training by a factor of $\nu^2 - \epsilon$. Note that such mini-batches also have a larger variance than mini-batch coresets found by CREST, which should be replaced by $1/r$ in Eq. (13).

## 5. Experiments

In this section, we evaluate the performance of our coreset selection, CREST. First, we compare CREST to the state-of-the-art coreset selection algorithms, namely CRAIG (Mirzasoleiman et al., 2020), GLISTER (Killamsetty et al., 2021b), and GRADMATCH (Killamsetty et al., 2021a), as well as the Random baseline. Second, we evaluate the effectiveness of our quadratic approximations in determining the time that the coresets needs to be updated. In addition, we compare the speedup of training with CREST to other baselines. Then, we conduct an ablation study to investigate the necessity of the quadratic vs. linear approximation, smoothing gradient and curvature, and dropping the learned examples from the selection pipeline. Finally, we study the learning difficulty of subsets that are selected by CREST during the course of training.

**Datasets and Models.** To demonstrate the effectiveness of CREST across different datasets and architectures, we apply CREST to several image and language benchmarks, including training ResNet-20 on CIFAR10, ResNet-18 on CIFAR-100 (Krizhevsky et al., 2009), ResNet-50 on TinyImageNet (Russakovsky et al., 2015), and fine-tuning RoBERTa on Stanford Natural Language Inference (SNLI) (Bowman et al., 2015). Table 4 summarizes the datasets and models.

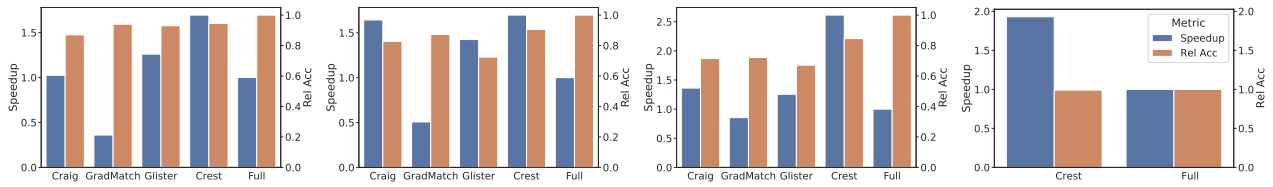
**Training Setup.** For all datasets except SNLI, we consider a standard deep learning training pipeline that runs for 200 epochs with a SGD optimizer with a momentum of 0.9, and decays the learning rate by a factor of 0.1 after 60% and 85% of training, and use mini-batch size 128. We warm-start the learning rate to 0.1 in the first 10% of training, which is essential for stability of all the methods, except CREST and Random. However, for fair comparison, we compare all the methods using learning rate warm-start. For fine-tuning RoBERTa on SNLI we used an AdamW optimizer and a learning rate of 1e-5 for 8 epochs, with mini-batch size 32. We ran all experiments with a single NVIDIA RTX A6000 GPU.

**Evaluation.** We evaluate all the methods under 10% budget for training. That is, for CRAIG, GRADMATCH, and GLISTER, we find a new coreset of size 10% of the full data at the beginning of *every epoch*. On the other hand, Random iteratively selects random mini-batches, and CREST finds mini-batch coresets and automatically finds the time to update them. We stop all methods after the same number of training iterations as that of 10% training on the full data. Note that under the above 'training setup', the Random baseline achieves a much higher accuracy than that of epoch 20 of a standard 200 epoch training pipeline (see SGD† in Table 1). This is because the learning rate drops twice during training on Random (and coresets) under 10% budget.

**CREST Setup.** In our experiments, we used $b = 5$, and $T_2 = 20$ for all the datasets, and tuned $\tau, \alpha$ and $h$, as discussed in Appendix A.2. Nevertheless, our method is not very sensitive to the choice of $\alpha$. We used $|V_p| = |V_r| = r = 0.005 \times n$ for SNLI and $0.01 \times n$ for the rest of the

Table 1: Relative error (%) of different methods over training on the full data. All the baselines select subsets of size 10% of full data at the beginning of every epoch. On the other hand, CREST selects mini-batches and decides when to update the mini-batches based on its quadratic loss approximation. (*) GLISTER uses the validation set, and (‡) GRADMATCH uses higher dimensional gradients to find coresets. SGD† shows accuracy of a standard mini-batch SGD pipeline at 10% training.

| DATASET - MODEL | BACKPROP | SGD† | RANDOM | CRAIG | GRAD-MATCH‡ | GLISTER* | **CREST** (OURS) |
|---|---|---|---|---|---|---|---|
| CIFAR-10 - RESNET-20 | 10% | $21.3_{\pm 8.0}$ | $7.2_{\pm 1.4}$ | $13.0_{\pm 5.1}$ | $6.0_{\pm 0.1}$ | $7.0_{\pm 0.1}$ | $\mathbf{5.5}_{\pm \mathbf{0.2}}$ |
| CIFAR-100 - RESNET-18 | 10% | $36.5_{\pm 2.9}$ | $11.7_{\pm 0.4}$ | $17.2_{\pm 4.5}$ | $12.7_{\pm 0.9}$ | $27.6_{\pm 4.0}$ | $\mathbf{9.4}_{\pm \mathbf{0.3}}$ |
| TINYIMAGENET - RESNET-50 | 10% | $32.8_{\pm 2.1}$ | $16.0_{\pm 0.5}$ | $28.5_{\pm 0.6}$ | $27.7_{\pm 0.2}$ | $32.8_{\pm 2.1}$ | $\mathbf{15.4}_{\pm \mathbf{0.6}}$ |
| SNLI - ROBERTA (FINETUNE) | 10% | $1.2_{\pm 0.3}$ | $1.2_{\pm 0.3}$ | - | - | - | $\mathbf{0.8}_{\pm \mathbf{0.2}}$ |




(a) CIFAR-10 (b) CIFAR-100 (c) TinyImageNet (d) SNLI

Figure 2: Normalized run-time and test accuracy of CREST by that of full data, when training ResNet-20 on CIFAR10, ResNet-18 on CIFAR100, ResNet-50 on TinyImagenet, and fine-tuning RoBERTa on SNLI.

datasets, without further tuning. For RoBERTa we used last layer gradient and Hessian diagonal, and for other networks we used full gradient and Hessian diagonal in Eq. (6).

### 5.1. Evaluating Accuracy and Speedup

**Accuracy.** Table 1 shows the relative error, i.e., $\frac{|acc_{coreset} - acc_{full}|}{acc_{full}}$ of models trained with each coreset selection algorithm. We see that while the baselines yield a very high relative error in particular for larger models and more difficult tasks, e.g. CIFAR100 and TinyImageNet, CREST can successfully outperform all the baselines and obtain up to 18.2% better relative error compared to baseline coreset selection methods, and up to 2.3% better relative error compared to Random baseline. Note that as the size of the data increases, existing methods that select coresets from the full data become prohibitively expensive. Notably, CREST is the only coreset selection method that is applicable to SNLI with 570k examples. Other coreset baselines that find subsets from the full data cannot scale to such a large data. Table 1 confirms that CREST can successfully finds mini-batch coresets with small bias and variance and identify when they need to be updated during the training.

**Speedup.** Figure 2 compares the accuracy and wall-clock run time of CREST vs baselines, and training on full data. We see that CREST is able to achieve up to 2.5x speeds up over training on full data, while introducing the smallest relative error compared to the baselines, when training ResNet-20 on CIFAR-10, ResNet-18 on CIFAR-100, ResNet-50 on TinyImageNet, and fine-tuning RoBERTa on SNLI. Table 2 further lists the average wall-clock time for selecting every mini-batch coreset of size 128, calculating the quadratic loss approximation based on Eq. (6), and checking the validity of the approximation on a random subset of data according to Eq. (10), when selecting coresets with CREST to train ResNet-18 on CIFAR100. Note that selecting a mini-batch from a larger random subset is much faster than selecting a subset of size 10% from the full data, done by the baselines.

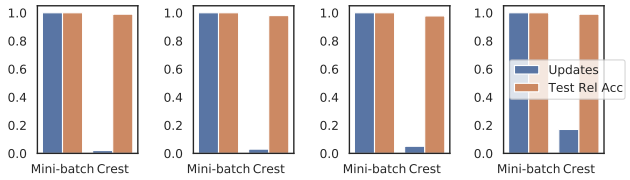
Table 2: Average time for different components of CREST for training ResNet-18 on CIFAR-100 with batch size 128.

| STEP | TIME (SECONDS) |
|---|---|
| SELECTION (CREST) | 0.006 |
| SELECTION (CRAIG) | 0.089 |
| LOSS APPROXIMATION | 0.115 |
| CHECKING THRESHOLD | 0.796 |

### 5.2. Ablation Study

**Modeling the loss.** Next, we evaluate the effectiveness of CREST in approximating the loss as piece-wise quadratic regions and identifying the time that the coresets need to be updated. To do so, we compare CREST with greedy mini-batch selection, which selects every mini-batch by applying the greedy algorithm to solve Eq. (5) on one random subset, and trains on it before selecting the next mini-batch. Figure 3 compares the relative error and the number of times CREST updates the coresets to greedy mini-batch selection. We see that CREST can effectively reduce the number of updates to 2% and 3% of the total update time of greedy mini-batch selection while preserving 98% and 99% of its performance, when training ResNet18 on CIFAR-100 and ResNet20 on CIFAR-10. For training ResNet50 on TinyImagenet and fine-tuning RoBERTa on SNLI, CREST reduces the number of updates to 19% and 26% respectively, while preserving 95% and 99% of the performance.

**Quadratic approximation.** As discussed in Sec. 4.1, in later stages of training, the loss can be better approximated




(a) CIFAR-10 (b) CIFAR-100 (c) TinyImageNet (d) SNLI

Figure 3: Normalized test accuracy and number of coreset updates for CREST over greedily selecting every mini-batch from a larger random subset by solving Eq. (5).

Table 3:Effect of CREST components (ResNet20/CIFAR10).

| ALGORITHM | REL. ERROR | # UPDATES |
|---|---|---|
| CREST-FIRST | 7.45 | 343 |
| CREST W/O SMOOTH | 7.44 | 369 |
| CREST W/O EXCLUDING | 4.61 | 346 |
| CREST | 4.33 | 185 |

as a convex quadratic function within larger neighborhoods. Figure 4 (left) shows that as training proceeds, CREST can successfully increase the size of the neighborhoods in which the quadratic approximation is valid, and reduce the number of updates over time. Moreover, Figure 4 (right) shows that using a first-order approximation instead of our quadratic approximation, or not smoothing the gradient and curvature in calculating the quadratic approximation, leads to higher number of coreset updates, and harms the accuracy. Table 3 further compares the number of updates and the relative error at the end of training. We see that excluding the learned examples further improves the performance of CREST.

Table 3 and Figure 4 show that it is crucial *when* the coresets are updated. Figure 4 shows that updating the coreset more frequently in the beginning is the key (notice that the green line is slightly higher than blue and orange in the first 1000 iterations). This slight difference results in a much better final accuracy. However, updating the coreset frequently later in training does not improve the accuracy (it does not hurt but does not help). Hence, blue and orange lines achieve a lower accuracy than Crest with more updates. CREST can accurately find when is best to update the coresets based on its quadratic loss approximation, and achieve a better accuracy while minimizing the number of updates.

**Importance of Examples during the Training.** Figure 5 shows the average forgetting score for the selected examples during the training. Forgetting score counts the number of times examples are misclassified after being correctly classified during the training, and quantifies the difficulty of learning an example (Toneva et al., 2018). We see that CREST selects examples of increasing difficulty during the training, and excluding the learned examples allows further focusing on the difficult-to-learn examples. In contrast, random subsets have a constantly lower forgetting score during the training. Figure 7b in Appendix A.2 shows that while

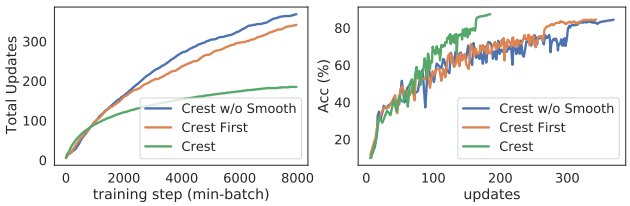



Figure 4: Training ResNet-20 on CIFAR-10 with CREST under 10% training budget. (Left) Number of coreset updates vs. training iterations. (Right) test accuracy vs. the total number of coreset updates.

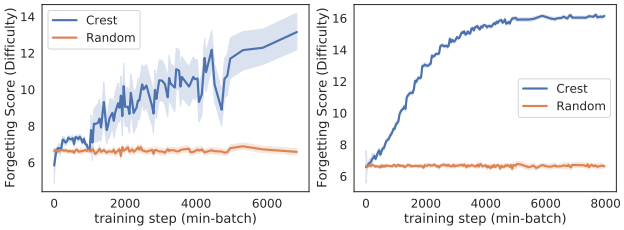



Figure 5: Average forgettability score of CREST coresets during training, when learned examples are not discarded (Left), and are discarded (Right). Learning difficulty of examples selected by CREST increases during the training.

CREST trains on a diverse set of examples, the distribution of the number of times different examples are selected by CREST is very long-tailed. This shows not all examples contribute equally to training, and CREST can successfully find examples that are important for learning at different times.

**Limitations.** In general, coreset methods are most beneficial under a limited training budget. While CREST can still achieve a superior accuracy under a larger budget (Table 5 in Appendix A.2), it achieves a smaller accuracy gap compared to the Random baseline. Besides, more efficient data loading can significantly speed up coreset selection.

## 6. Conclusion

We proposed the first scalable framework with rigorous theoretical guarantees to identify the most valuable examples for training non-convex models, particularly deep networks. Our approach models the non-convex loss as a series of quadratic functions and extracts a coreset for each quadratic sub-region. In addition, to ensure convergence of stochastic gradient methods such as (mini-batch) SGD, it iteratively extracts multiple coresets from smaller random subsets of training data, to ensure nearly-unbiased gradient estimates with small variance. In doing so, it provides rigorous theoretical guarantee for convergence of the extracted coresets to stationary point of a non-convex function. With extensive experiments, we confirmed the effectiveness of our method on various vision and NLP deep learning tasks.

**Acknowledgment.** This research was supported by the National Science Foundation CAREER Award 2146492.

## 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.

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

Bertsekas, D. P. Convexification procedures and decomposition methods for nonconvex optimization problems. *Journal of Optimization Theory and Applications*, 29(2): 169–197, 1979.

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.

Bollapragada, R., Byrd, R. H., and Nocedal, J. Exact and inexact subsampled newton methods for optimization. *IMA Journal of Numerical Analysis*, 39(2):545–578, 2019.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL `https://aclanthology.org/D15-1075`.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018.

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.

Defazio, A. and Bottou, L. On the ineffectiveness of variance reduced optimization for deep learning. *Advances in Neural Information Processing Systems*, 32:1755–1765, 2019.

Dembo, R. S., Eisenstat, S. C., and Steihaug, T. Inexact newton methods. *SIAM Journal on Numerical analysis*, 19(2):400–408, 1982.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Ghadimi, S. and Lan, G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Jin, C., Netrapalli, P., Ge, R., Kakade, S. M., and Jordan, M. I. On nonconvex optimization for machine learning: Gradients, stochasticity, and saddle points. *Journal of the ACM (JACM)*, 68(2):1–29, 2021.

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., Durga, S., Ramakrishnan, G., De, A., and Iyer, R. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pp. 5464–5474. PMLR, 2021a.

Killamsetty, K., Sivasubramanian, D., Ramakrishnan, G., and Iyer, R. Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8110–8118, 2021b.

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

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

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

Marquardt, D. W. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

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

Mindermann, S., Brauner, J. M., Razzak, M. T., Sharma, M., Kirsch, A., Xu, W., Höltgen, B., Gomez, A. N., Morisot, A., Farquhar, S., et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pp. 15630–15649. PMLR, 2022.

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., 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.

Paul, M., Ganguli, S., and Dziugaite, G. K. Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34: 20596–20607, 2021.

Pooladzandi, O., Davini, D., and Mirzasoleiman, B. Adaptive second order coresets for data-efficient machine learning. In *International Conference on Machine Learning*, pp. 17848–17869. PMLR, 2022.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

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.

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.

Wang, W. and Srebro, N. Stochastic nonconvex optimization with large minibatches. In *Algorithmic Learning Theory*, pp. 857–882. PMLR, 2019.

Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2691–2699, 2015.

Xu, P., Roosta, F., and Mahoney, M. W. Newton-type methods for non-convex optimization under inexact hessian information. *Mathematical Programming*, 184(1):35–70, 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.

Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022.

# A. Appendix

## A.1. Proofs

We assume that the stochastic gradients are unbiased and have a bounded variance, i.e.,

$$\mathbb{E}_{i\in V}[\|\mathbf{g}_{t,i}-\mathbf{g}_{t,V}\|]=\mathbb{E}[\|\boldsymbol{\zeta}_t\|]=0, \qquad \mathbb{E}_{i\in V}[\|\mathbf{g}_{t,i}-\mathbf{g}_{t,V}\|^2]=\mathbb{E}[\|\boldsymbol{\zeta}_t\|^2]\leq\sigma^2. \tag{14}$$

Also assume that the function $\mathcal{L}$ is $L$-gradient Lipschitz, i.e.,

$$\|\nabla\mathcal{L}(\boldsymbol{w}_1)-\nabla\mathcal{L}(\boldsymbol{w}_2)\|\leq L\|\boldsymbol{w}_1-\boldsymbol{w}_2\|, \quad \forall\boldsymbol{w}_1,\boldsymbol{w}_2\in\mathcal{W}. \tag{15}$$

Then, we have that:

$$|\mathcal{L}(\boldsymbol{w}_1)-\mathcal{L}(\boldsymbol{w}_2)-\langle\nabla\mathcal{L}(\boldsymbol{w}_2),\boldsymbol{w}_1-\boldsymbol{w}_2\rangle|\leq\frac{L}{2}\|\boldsymbol{w}_1-\boldsymbol{w}_2\|^2, \quad \forall\boldsymbol{w}_1,\boldsymbol{w}_2\in\mathcal{W}. \tag{16}$$

We can write the gradient descent updates when training on mini-batch coresets found by CREST, as follows:

$$\boldsymbol{w}_{t+1}\leftarrow\boldsymbol{w}_t-\eta_t(\nabla\mathcal{L}(\boldsymbol{w}_t)+\tilde{\boldsymbol{\zeta}}_t), \qquad s.t. \qquad \tilde{\boldsymbol{\zeta}}_t=\boldsymbol{\zeta}_t+\boldsymbol{\xi}_t \tag{17}$$

where $\boldsymbol{\zeta}_t$ is the error of random subset $V_p$ in capturing the full gradient, and $\boldsymbol{\xi}_t$ is the error of mini-batch coreset $S_l^p$ in capturing the gradient of $V_p$.

We build on the analysis of (Ghadimi & Lan, 2013) and characterize the effect of the coreset gradient error on the convergence. From Eq. (16), (17) we have:

$$\mathcal{L}(\boldsymbol{w}_{t+1})\leq\mathcal{L}(\boldsymbol{w}_t)+\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{w}_{t+1}-\boldsymbol{w}_t\rangle+\frac{L}{2}\eta_t^2\|\nabla\mathcal{L}(\boldsymbol{w}_t)+\tilde{\boldsymbol{\zeta}}_t\|^2 \tag{18}$$

$$\leq\mathcal{L}(\boldsymbol{w}_t)-\eta_t\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\nabla\mathcal{L}(\boldsymbol{w}_t)+\tilde{\boldsymbol{\zeta}}_t\right\rangle+\frac{L}{2}\eta_t^2\|\nabla\mathcal{L}(\boldsymbol{w}_t)+\tilde{\boldsymbol{\zeta}}_t\|^2 \tag{19}$$

$$=\mathcal{L}(\boldsymbol{w}_t)-\eta_t\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2-\eta_t\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\tilde{\boldsymbol{\zeta}}_t\right\rangle+\frac{L}{2}\eta_t^2\Big[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2+2\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\tilde{\boldsymbol{\zeta}}_t\right\rangle+\|\tilde{\boldsymbol{\zeta}}_t\|^2\Big] \tag{20}$$

$$=\mathcal{L}(\boldsymbol{w}_t)-(\eta_t-\frac{L}{2}\eta_t^2)\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2-(\eta_t-L\eta_t^2)\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\tilde{\boldsymbol{\zeta}}_t\right\rangle+\frac{L}{2}\eta_t^2\|\tilde{\boldsymbol{\zeta}}_t\|^2 \tag{21}$$

For $\eta_t<2/L$, we have $\eta_t-L\eta_t^2/2>0$. Summing up the above inequalities and re-arranging the terms, we obtain:

$$\sum_{t=1}^{N}(\eta_t-\frac{L}{2}\eta_t^2)\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2\leq\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}(\boldsymbol{w}_{N+1})-\sum_{t=1}^{N}(\eta_t-L\eta_t^2)\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\tilde{\boldsymbol{\zeta}}_t\right\rangle+\frac{L}{2}\sum_{t=1}^{N}\eta_t^2\|\tilde{\boldsymbol{\zeta}}_t\|^2 \tag{22}$$

$$\leq\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*-\sum_{t=1}^{N}(\eta_t-L\eta_t^2)\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\tilde{\boldsymbol{\zeta}}_t\right\rangle+\frac{L}{2}\sum_{t=1}^{N}\eta_t^2\|\tilde{\boldsymbol{\zeta}}_t\|^2, \tag{23}$$

$$=\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*-\sum_{t=1}^{N}(\eta_t-L\eta_t^2)\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{\zeta}_t+\boldsymbol{\xi}_t\right\rangle+\frac{L}{2}\sum_{t=1}^{N}\eta_t^2\|\boldsymbol{\zeta}_t+\boldsymbol{\xi}_t\|^2, \tag{24}$$

$$=\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*-\sum_{t=1}^{N}(\eta_t-L\eta_t^2)\left\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{\zeta}_t+\boldsymbol{\xi}_t\right\rangle+\frac{L}{2}\sum_{t=1}^{N}\eta_t^2(\|\boldsymbol{\zeta}_t^2\|+\|\boldsymbol{\xi}_t^2\|+2\left\langle\boldsymbol{\zeta}_t,\boldsymbol{\xi}_t\right\rangle), \tag{25}$$

where $\mathcal{L}^*$ is the optimal solution and Eq. (23) follows from the fact that $\mathcal{L}(\boldsymbol{w}_{N+1})\geq\mathcal{L}^*$. Taking expectations (with respect to the history $\Psi_N$ of the generated random process) on both sides of Eq. (25) and noting that $\mathbb{E}[\|\boldsymbol{\zeta}_t\|]=0$, and $\mathbb{E}[\|\boldsymbol{\zeta}_t\|^2]\leq\sigma^2$, and $\mathbb{E}[\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{\zeta}_t\rangle\,|\Psi_{t-1}]=0$, and $\mathbb{E}[\langle\boldsymbol{\zeta}_t,\boldsymbol{\xi}_t\rangle\,|\Psi_{t-1}]=0$ (since $\nabla\mathcal{L}(\boldsymbol{w}_t)$ and $\boldsymbol{\xi}_t$ and $\boldsymbol{\zeta}_t$ are independent), we obtain:

$$\sum_{t=1}^{N}(\eta_t-\frac{L}{2}\eta_t^2)\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2]\leq\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*-\sum_{t=1}^{N}(\eta_t-L\eta_t^2)\mathbb{E}_{\Psi_N}[\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{\xi}_t\rangle]+\sum_{t=1}^{N}\frac{L}{2}\eta_t^2(\frac{\sigma^2}{r}+\mathbb{E}_{\Psi_N}[\|\boldsymbol{\xi}_t\|^2]). \tag{26}$$

Next, we analyze convergence under two cases: (1) where $\mathbb{E}[\|\boldsymbol{\xi}_t\|]\leq\epsilon\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|$, and (2) where $\mathbb{E}[\|\boldsymbol{\xi}_t\|]\leq\epsilon$.

**Case 1. Assuming $\mathbb{E}[\|\boldsymbol{\xi}_t\|] \leq \epsilon\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|$ for $0 \leq \epsilon < 1$.** With $1/L \leq \eta_t < 2/L$, we have $\eta_t - L\eta_t^2 \leq 0$. Hence,

$$\sum_{t=1}^{N}(\eta_t - \frac{L}{2}\eta_t^2)\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^* - \sum_{t=1}^{N}(\eta_t - L\eta_t^2)\epsilon\mathbb{E}_{\Psi}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] + \sum_{t=1}^{N}\frac{L}{2}\eta_t^2(\frac{\sigma^2}{r} + \epsilon^2\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2]). \tag{27}$$

Hence,

$$\sum_{t=1}^{N}\left(\eta_t - \frac{L}{2}\eta_t^2 + \epsilon(\eta_t - L\eta_t^2) - \frac{L}{2}\eta_t^2\epsilon^2\right)\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^* + \frac{L\sigma^2}{2r}\sum_{t=1}^{N}\eta_t^2, \tag{28}$$

$$\sum_{t=1}^{N}\left((1+\epsilon)\eta_t - \frac{L}{2}(1+\epsilon)^2\eta_t^2\right)\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^* + \frac{L\sigma^2}{2r}\sum_{t=1}^{N}\eta_t^2, \tag{29}$$

and we get:

$$\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \frac{1}{\sum_{t=1}^{N}(1+\epsilon)\eta_t - \frac{L}{2}(1+\epsilon)^2\eta_t^2}\left[\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^* + \frac{\sigma^2 L}{2r}\sum_{t=1}^{N}\eta_t^2\right] \tag{30}$$

If $\eta_t < (1+\epsilon)/(\frac{L}{2}(1+\epsilon)^2) = 2/L(1+\epsilon)$, then $(1+\epsilon) - \frac{L}{2}(1+\epsilon)^2\eta_t > 0$ and we have:

$$\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \frac{1}{\sum_{t=1}^{N}\eta_t}\left[2(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*) + \frac{\sigma^2 L}{r}\sum_{t=1}^{N}\eta_t^2\right]. \tag{31}$$

For a random iterate $R$ of a run of the algorithm that is selected with probability $(2(1+\epsilon)\eta_t - L(1+\epsilon)^2\eta_t^2)/\sum_{t=1}^{N}(2(1+\epsilon)\eta_t - L(1+\epsilon)^2\eta_t^2)$, we have that

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] = \mathbb{E}_{R,\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] = \frac{\sum_{t=1}^{N}(2(1+\epsilon)\eta_t - L(1+\epsilon)^2\eta_t^2)\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2]}{\sum_{t=1}^{N}(2(1+\epsilon)\eta_t - L(1+\epsilon)^2\eta_t^2)} = \mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \tag{32}$$

Hence, for $\eta_t = \eta$ we get:

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] \leq \frac{1}{N\eta}\left[2(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*) + \frac{\sigma^2 L}{r}N\eta^2\right]. \tag{33}$$

For $\eta = \min\{\frac{1}{L}, \frac{\tilde{D}\sqrt{r}}{\sigma\sqrt{N}}\}$, and $\tilde{D} > 0$, we get

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] \leq \frac{1}{N\eta}\left[2(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)\right] + \frac{\sigma^2 L}{r}\eta \tag{34}$$

$$\leq \frac{2(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}{N}\max\{L, \frac{\sigma\sqrt{N}}{\tilde{D}\sqrt{r}}\} + \frac{\sigma^2 L}{r}\frac{\tilde{D}\sqrt{r}}{\sigma\sqrt{N}} \tag{35}$$

$$\leq \frac{2L(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}{N} + \left(L\tilde{D} + \frac{2(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}{\tilde{D}}\right)\frac{\sigma}{\sqrt{rN}} \tag{36}$$

Replacing the optimal value $\tilde{D} = \sqrt{2(\mathcal{L}(\boldsymbol{w}_1) - \mathcal{L}^*)/L}$, we get

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] \leq \mathcal{B}_N := \frac{2L(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}{N} + \frac{2\sigma\sqrt{2L(\mathcal{L}(\boldsymbol{w}_0) - \mathcal{L}^*)}}{\sqrt{rN}} \tag{37}$$

Hence, training with CREST exhibits an $\mathcal{O}(1/\sqrt{rN})$ rate of convergence, compared to $\mathcal{O}(1/\sqrt{mN})$ for mini-batch SGD with mini-batch size $m < r$.

To derive large-deviation properties for a single run of this method, we are interested in the number of iterations required to find a point satisfying $\mathbb{P}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2 \leq \nu^2] \geq 1-\frac{1}{\lambda}$. We use Markov's inequality to calculate the probability $\mathbb{P}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2 \geq \lambda\mathcal{B}_N] \leq \frac{1}{\lambda}$. We get that with probability at least $1-\lambda$, at least one iteration of a single run of the algorithm visits a $\nu$-stationary point in the following number of iterations:

$$\tilde{\mathcal{O}}\left(\frac{L(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}{\nu^2}(1+\frac{\sigma^2}{r\nu^2})\right). \tag{38}$$

As long as $r \leq \sigma^2/\nu^2$ increasing the size of the random subsets $r$ used by CREST will reduce the number of iterations linearly, while not increasing the total number of stochastic gradient queries.

**Case 2. Assuming $\mathbb{E}[\|\boldsymbol{\xi}_t\|] \leq \epsilon < \nu^2$.** For $1/L \leq \eta_t < 2/L$ we have $\eta_t - L\eta_t^2 < 0$. Hence:

$$\sum_{t=1}^{N}(\eta_t - \frac{L}{2}\eta_t^2)\mathbb{E}_\Psi[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \leq \mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^* - \sum_{t=1}^{N}(\eta_t - L\eta_t^2)\mathbb{E}_\Psi[\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{\xi}_t\rangle] + \sum_{t=1}^{N}\frac{L}{2}\eta_t^2(\frac{\sigma^2}{r}+\epsilon^2), \tag{39}$$

$$\leq \mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^* - \sum_{t=1}^{N}(\eta_t - L\eta_t^2)\mathbb{E}_\Psi[|\langle\nabla\mathcal{L}(\boldsymbol{w}_t),\boldsymbol{\xi}_t\rangle|] + \sum_{t=1}^{N}\frac{L}{2}\eta_t^2(\frac{\sigma^2}{r}+\epsilon^2), \tag{40}$$

$$\leq \mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^* - \sum_{t=1}^{N}(\eta_t - L\eta_t^2)\epsilon\mathbb{E}_\Psi[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|] + \sum_{t=1}^{N}\frac{L}{2}\eta_t^2(\frac{\sigma^2}{r}+\epsilon^2) \tag{41}$$

$$\leq \mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^* - \sum_{t=1}^{N}(\eta_t - L\eta_t^2)\epsilon\nabla_{\max} + \sum_{t=1}^{N}\frac{L}{2}\eta_t^2(\frac{\sigma^2}{r}+\epsilon^2), \tag{42}$$

where $\nabla_{\max} = \max\{0, \max_{i\in V,\boldsymbol{w}_t\in\mathcal{W}}\|\mathbf{g}_{t,i}\|\}$. For a random iterate $R$ of a run of the algorithm that is selected with probability $(2\eta_t - L\eta_t^2)/\sum_{t=1}^{N}(2\eta_t - L\eta_t^2)$, we have that

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] = \mathbb{E}_{R,\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] = \frac{\sum_{t=1}^{N}(2\eta_t - L\eta_t^2)\mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2]}{\sum_{t=1}^{N}(2\eta_t - L\eta_t^2)} = \mathbb{E}_{\Psi_N}[\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|^2] \tag{43}$$

If $\eta = \eta_t$ and $\eta \leq 1/L + 1/2L\nabla_{\max}$, we have $-2(1-L\eta) \leq 1/\nabla_{\max}$ and we get

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] \leq \frac{1}{N\eta(1-\frac{L}{2}\eta)}\left[\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^* - N(\eta - L\eta^2)\epsilon\nabla_{\max} + (\frac{\sigma^2}{r}+\epsilon^2)\frac{L}{2}N\eta^2\right] \tag{44}$$

$$\leq \frac{2}{N\eta}\left[\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^* - N(\eta - L\eta^2)\epsilon\nabla_{\max} + (\frac{\sigma^2}{r}+\epsilon^2)\frac{L}{2}N\eta^2\right] \tag{45}$$

$$= \frac{2}{N\eta}[\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*] - 2(1-L\eta)\epsilon\nabla_{\max} + (\frac{\sigma^2}{r}+\epsilon^2)L\eta \tag{46}$$

$$\leq \frac{2}{N\eta}[\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*] + \epsilon + (\frac{\sigma^2}{r}+\epsilon^2)L\eta \tag{47}$$

For $\eta = \min\{\frac{1}{L}, \frac{\tilde{D}}{\sqrt{N(\sigma^2/r+\epsilon^2)}}\}$, and $\tilde{D} > 0$, we get

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] \leq \frac{1}{N\eta}[2(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)] + (\frac{\sigma^2}{r}+\epsilon^2)L\eta + \epsilon \tag{48}$$

$$\leq \frac{2(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}{N}\max\{L, \frac{\sqrt{N(\sigma^2/r+\epsilon^2)}}{\tilde{D}}\} + (\frac{\sigma^2}{r}+\epsilon^2)\frac{L\tilde{D}}{\sqrt{N(\sigma^2/r+\epsilon^2)}} + \epsilon \tag{49}$$

$$\leq \frac{2L(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}{N} + \left(L\tilde{D} + \frac{2(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}{\tilde{D}}\right)\frac{\sqrt{\sigma^2/r+\epsilon^2)}}{\sqrt{N}} + \epsilon \tag{50}$$

For the optimal value of $\tilde{D} = \sqrt{2(\mathcal{L}(\boldsymbol{w}_1)-\mathcal{L}^*)/L}$, we get

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{w}_R)\|^2] \leq \mathcal{B}_N := \frac{2L(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}{N} + \frac{2\sqrt{\sigma^2+r\epsilon^2}\sqrt{2L(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}}{\sqrt{rN}} + \epsilon \tag{51}$$

Hence, the number of iterations becomes:

$$\tilde{\mathcal{O}}\left(\frac{L(\mathcal{L}(\boldsymbol{w}_0)-\mathcal{L}^*)}{\nu^2-\epsilon}(1+\frac{\sigma^2+r\epsilon^2}{r(\nu^2-\epsilon)})\right) \tag{52}$$

Hence, more number of iterations is required. Besides, if $\epsilon \geq \nu^2$, convergence is not guaranteed.

**Incorporating $\tau$.** Assume $c_2$ is the error of the coreset in capturing the full gradient at the beginning of the neighborhood. From Eq. (10) we know $|\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta})-\mathcal{F}^l(\boldsymbol{\delta})| = \rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)$. Using the quadratic approximation in Eq. (6), i.e., $\mathcal{F}^l(\boldsymbol{\delta}) = \frac{1}{2}\boldsymbol{\delta}^T\mathbf{H}_{t_l,S_l}\boldsymbol{\delta}+\mathbf{g}_{t_l,S_l}\boldsymbol{\delta}+\mathcal{L}(\boldsymbol{w}_{t_l})$, and noting that $\mathcal{L}$ can also be modeled by a similar quadratic function for small $\tau$, we get:

$$\begin{aligned}
\rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l) = |\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)-\mathcal{F}^l(\boldsymbol{\delta}_l)| &= |\frac{1}{2}\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})\boldsymbol{\delta}_l+(\mathbf{g}_{t_l,V}-\mathbf{g}_{t_l,S_l})\boldsymbol{\delta}_l| && (53)\\
&\geq \Big|\frac{1}{2}|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})\boldsymbol{\delta}_l| - \|\mathbf{g}_{t_l,V}-\mathbf{g}_{t_l,S_l}\|\cdot\|\boldsymbol{\delta}_l\|\Big| && (54)\\
&\geq \Big|\frac{1}{2}|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})\boldsymbol{\delta}_l| - c_2\cdot\|\boldsymbol{\delta}_l\|\Big| && (55)\\
&\geq \frac{1}{2}|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})\boldsymbol{\delta}_l| - c_2\cdot\|\boldsymbol{\delta}_l\| && (56)
\end{aligned}$$

As long as $\rho_{t_l}$ is small, we can assume that the loss can be well modeled by a quadratic using the Hessian diagonal. Using the Hessian diagonal for both $\mathcal{L}$ and $\mathcal{F}^l$, we have $|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,S_l}-\mathbf{H}_{t_l,V})\boldsymbol{\delta}_l| = \|\boldsymbol{\delta}_l^T(\text{diag}(\mathbf{H}_{t_l,S_l})-\text{diag}(\mathbf{H}_{t_l,V}))\|\cdot\|\boldsymbol{\delta}_l\|$. Hence,

$$\frac{1}{2}|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,S_l}-\mathbf{H}_{t_l,V})\boldsymbol{\delta}_l| = \frac{1}{2}\|\boldsymbol{\delta}^T(\text{diag}(\mathbf{H}_{t_l,S_l})-\text{diag}(\mathbf{H}_{t_l,V}))\|\cdot\|\boldsymbol{\delta}_l\| \leq \rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)+c_2\|\boldsymbol{\delta}_l\|, \tag{57}$$

$$\|\boldsymbol{\delta}_l^T(\text{diag}(\mathbf{H}_{t_l,S_l})-\text{diag}(\mathbf{H}_{t_l,V}))\| \leq 2\rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)/\|\boldsymbol{\delta}_l\|+2c_2. \tag{58}$$

On the other hand, we have that $\nabla\mathcal{F}^l(\boldsymbol{\delta}) = \boldsymbol{\delta}^T\mathbf{H}_{t_l,S_l}+\mathbf{g}_{t_l,S_l}$ and $\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l) = \boldsymbol{\delta}_l^T\mathbf{H}_{t_l,V}+\mathbf{g}_{t_l,V}$. Hence, we have:

$$\begin{aligned}
\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)-\nabla\mathcal{F}^l(\boldsymbol{\delta}_l)\| &= \|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})+(\mathbf{g}_{t_l,V}-\mathbf{g}_{t_l,S_l})\| && (59)\\
&\leq \|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})\|+\|\mathbf{g}_{t_l,V}-\mathbf{g}_{t_l,S_l}\| && (60)\\
&\leq \|\boldsymbol{\delta}_l^T(\mathbf{H}_{t_l,V}-\mathbf{H}_{t_l,S_l})\|+c_2. && (61)
\end{aligned}$$

Therefore, using Hessian diagonal and from Eq. (58) and (61) we get:

$$\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)-\nabla\mathcal{F}^l(\boldsymbol{\delta}_l)\| \leq \|\boldsymbol{\delta}_l^T(\text{diag}(\mathbf{H}_{t_l,V})-\text{diag}(\mathbf{H}_{t_l,S_l}))\|+c_2 \leq 2\rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)/\|\boldsymbol{\delta}_l\|+3c_2 \tag{62}$$

Eq. (62) shows that for a fixed $\rho_{t_l}$ and loss, if the convex approximation $\mathcal{F}^l$ is valid in a larger neighborhood $\boldsymbol{\delta}_l$, then the error of the Hessian diagonal at the beginning of the neighborhood was smaller and hence the gradient error at the end of the neighborhood is smaller.

From Eq. (62) we know that $\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)-\nabla\mathcal{F}^l(\boldsymbol{\delta}_l)\| \leq 2\rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)/\|\boldsymbol{\delta}_l\|+3c_2$. Let $c_1$ be the desired upper-bound on the gradient error at $\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l$. Hence, we wish

$$\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)-\nabla\mathcal{F}^l(\boldsymbol{\delta}_l)\| \leq 2\rho_{t_l}\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)/\|\boldsymbol{\delta}_l\|+3c_2 \leq c_1. \tag{63}$$

Hence, for $c_2 \leq c_1/3$ we get:

$$\rho_{t_l} \leq \frac{(c_1-3c_2)\|\boldsymbol{\delta}_l\|}{2\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)}, \qquad \tau \leq \min_{t_l}\rho_{t_l}. \tag{64}$$

For $\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)-\nabla\mathcal{F}^l(\boldsymbol{\delta}_l)\| \leq c_1 = c\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)\|$, and with $c_2 \leq c\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)\|/3$, we have:

$$\rho_{t_l} \leq \frac{(c\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)\|-3c_2)\|\boldsymbol{\delta}_l\|}{2\mathcal{L}(\boldsymbol{w}_{t_l}+\boldsymbol{\delta}_l)}, \tag{65}$$

For $c=1$ and $c_2 = \epsilon\|\nabla\mathcal{L}(\boldsymbol{w}_t)\|$, we get Case 1 in the Theorem. We see that when gradient norm is smaller, we should have a smaller error $c_2$ in capturing the random subset gradients.

Table 4: Experiment setups.

| DATASET | CLASSES | TRAIN | NETWORK | PARAMETERS | FUL ACC |
|---|---|---|---|---|---|
| CIFAR-10 | 10 | 50K | RESNET-20 | 0.27M | $92.1_{\pm 0.1}$ |
| CIFAR-100 | 100 | 50K | RESNET-18 | 11M | $75.6_{\pm 0.3}$ |
| TINYIMAGENET | 200 | 100K | RESNET-50 | 23M | $66.9_{\pm 0.1}$ |
| SNLI | 3 | 570K | ROBERTA | 123M | $92.9_{\pm 0.2}$ |

Table 5: Relative error (%) with 20% of the full training budget (backprop) can reach a very close accuracy to that of full training (with only 2-3% difference) on all datasets, namely, CIFAR-10, CIFAR-100, and TinyImageNet.

| | CREST | RANDOM | SGD† |
|---|---|---|---|
| CIFAR-10 - RESNET-20 | 2.32 | 2.87 | 16.47 |
| CIFAR-100 - RESNET-18 | 3.37 | 3.66 | 32.68 |
| TINYIMAGENET - RESNET-50 | 3.05 | 3.51 | 47.43 |

### A.2. Experimental details

**Tuning Hyperparameters.** We tuned the hyperparameters $\tau \in \{0.1, 0.05, 0.01, 0.005, 0.001\}$, $h \in \{1, 2, 4, 8, 10\}$ and used $\tau = 0.05, 0.01, 0.005, 0.05,\ \ h = 1, 10, 1, 4$ on CIFAR-10, CIFAR-100, TinyImagenet, and SNLI, respectively, as listed in Table 6. To determine $\tau$, we calculated the average loss approximation error divided by the training loss, i.e. $\rho_{t_l}$ in Eq. (10), after some coresets updates during training. Across all datasets, we found that $\alpha = 0.1$ yielded satisfactory results.

**Convergence of CREST vs CRAIG.** Figure 6b shows training ResNet-20 on CIFAR-10 with CREST vs CRAIG. We see that the normalized bias of CREST mini-batch coresets over full gradient norm, i.e., $\epsilon = \mathbb{E}[\|\boldsymbol{\xi}_{t_l}\|]/\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l})\|$ is consistently small ($< 1$) during the training. As the gradient norm becomes smaller closer to a stationary point, small $\epsilon$ implies that the bias of the CREST mini-batch coresets $\mathbb{E}[\|\boldsymbol{\xi}_{t_l}\|]$ diminishes as we get closer to a stationary point. Hence, convergence of CREST can be guaranteed (Case 1 in Theorem 4.1). On the other hand, the normalized error for CRAIG coresets can be large during the training. Hence, convergence is not guaranteed (Case 2 in Theorem 4.1).

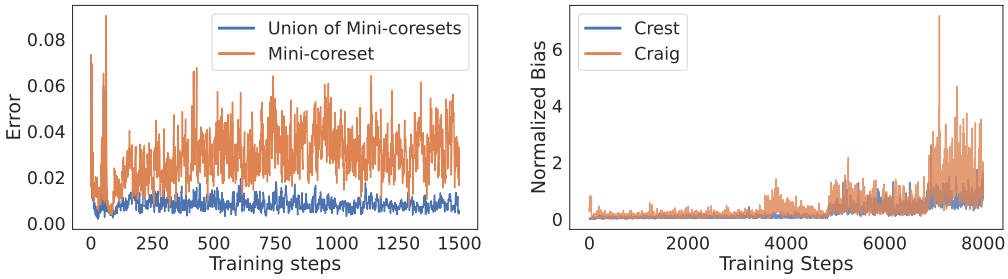
**CREST has a Similar Performance to Training with Large Mini-batches.** Figure 9 shows the variance of gradient of CREST mini-batch coresets of size $m = 128$ selected from random subsets $V_p$ of size $r = 500$. We see that the variance of CREST mini-batch coresets is very close to the variance of $V_r$. In contrast, random subsets of size $m = 128$ have a considerably larger variance. Figure 8 further compares the relative error of CREST with mini-batch coresets of size $m = 128$ selected from random subsets of size $r = 500$. We see that training on CREST mini-batch coresets has a smaller relative error than training on random mini-batches of size $m = 128$. In particular, relative error of CREST with $m = 128$ is close to that of training on random mini-batches of size $m = 500$. This is due to the smaller gradient variance of CREST mini-batch coresets, as is shown in Figure 9.

**Effect of Dropping the Learned Examples.** By tracking the prediction accuracy of the dropped training examples (Figure 7a), we found that even though some of the dropped examples could be forgotten after being dropped (the accuracy of the dropped examples is 92% earlier in training), they can be learned again when training on the coresets selected from the remaining training examples (the accuracy of dropped examples always increases to above 99% even though we never train on them again). This confirms that dropping the learned examples does not harm the performance.

**CREST with Larger Training Budget.** In general, coreset methods are most beneficial under a limited training budget. Table 5 compares the relative error of training ResNet-18 on CIFAR-10, ResNet-20 on CIFAR-100 and ResNet-50 on TinyImagenet with CREST vs. Random, under 20% training budget. Note that under the standard learning rate schedule used for training on the above datasets for 200 epochs, there is a large gap up to 44.38% between SGD† (i.e., training for $20\% \times 200 = 40$ epochs on full data with mini-batch SGD) and CREST. But, the gap reduces when learning rate drops at 60% and 85% of training (Random vs. CREST).

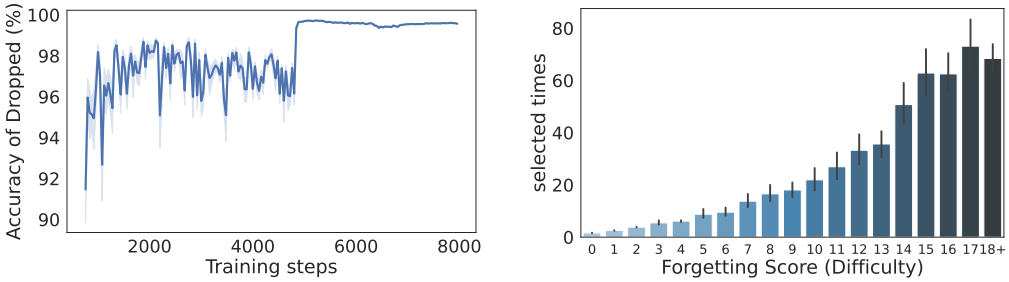
Table 6: Hyperparameters used for different datasets.

| DATASET | $\tau$ | $h$ |
|---|---|---|
| CIFAR-10 | 0.05 | 1 |
| CIFAR-100 | 0.01 | 10 |
| TINYIMAGENET | 0.005 | 1 |
| SNLI | 0.05 | 4 |




(a) (b)

Figure 6: Training ResNet-20 on CIFAR-10. (a) Union of mini-batch coresets has a smaller error in capturing the full gradient, compared to the bias of the individual mini-batch coresets. (b) Normalized bias of coresets by the full gradient norm, i.e., $\epsilon = \mathbb{E}[\|\boldsymbol{\xi}_{t_l}\|]/\|\nabla\mathcal{L}(\boldsymbol{w}_{t_l})\|$ in Theorem 4.1. CREST coresets have a consistently small $\epsilon < 1$. As the gradient norm becomes smaller closer to the stationary points, small $\epsilon$ implies that the bias of the CREST mini-batch coresets $\mathbb{E}[\|\boldsymbol{\xi}_{t_l}\|]$ diminishes closer to the stationary points. Hence, convergence of CREST can be guaranteed (Case 1 in Theorem 4.1). On the other hand, $\epsilon$ can be large for CRAIG coresets. Hence, convergence is not guaranteed (Case 2 in Theorem 4.1).




(a) (b)

Figure 7: Training ResNet-20 on CIFAR-10 with CREST. (a) Dropped examples are learned later in training, by training on CREST subsets. (b) Distribution of forgetting scores for the examples selected by CREST during the training. The distribution is long-tailed, confirming that not all examples contribute equally to training.

Figure 8: Relative error (%) with 10% training budget. Training on CREST mini-batch coresets of size $m = 128$ selected from random subsets $V_p$ of size $r = 500$ has a smaller relative error than training on random mini-batches of size $m = 128$. In particular, relative error of CREST with $m = 128$ is close to that of training on random mini-batches of size $m = 500$.

| METHOD | RELATIVE ERROR |
|---|---|
| CREST M=128 | 5.2 |
| RANDOM M=128 | 7.1 |
| RANDOM M=512 | 4.0 |

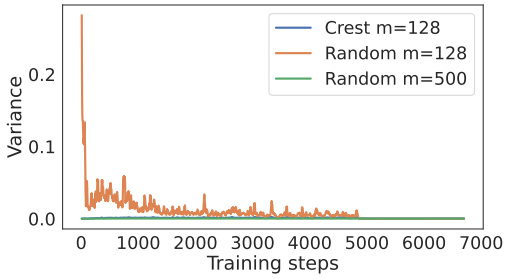



Figure 9: Variance of gradients of CREST mini-batches of size $m = 128$ selected from random subsets $V_p$ of size $r = 500$ is very closer to the variance of $V_r$. In contrast, random subsets of size $m = 128$ have a considerably larger variance.