Prioritized Training on Points that are Learnable, Worth Learning, and Not Yet Learnt

Sören Mindermann [*1] **Jan Brauner** [*1] **Muhammed Razzak** [*1] **Mrinank Sharma** [*2] **Andreas Kirsch** [1] **Winnie Xu** [3 4] **Benedikt Höltgen** [1] **Aidan N. Gomez** [3 1] **Adrien Morisot** [3] **Sebastian Farquhar** [1] **Yarin Gal** [1]

Abstract

Training on web-scale data can take months. But most computation and time is wasted on redundant and noisy points that are already learnt or not learnable. To accelerate training, we introduce *Reducible Holdout Loss Selection* (RHO-LOSS), a simple but principled technique which selects approximately those points for training that most reduce the model's generalization loss. As a result, RHO-LOSS mitigates the weaknesses of existing data selection methods: techniques from the optimization literature typically select "hard" (e.g. high loss) points, but such points are often noisy (not learnable) or less task-relevant. Conversely, curriculum learning prioritizes "easy" points, but such points need not be trained on once learnt. In contrast, RHO-LOSS selects points that are learnable, worth learning, and not yet learnt. RHO-LOSS trains in far fewer steps than prior art, improves accuracy, and speeds up training on a wide range of datasets, hyperparameters, and architectures (MLPs, CNNs, and BERT). On the large web-scraped image dataset Clothing-1M, RHO-LOSS trains in 18x fewer steps and reaches 2% higher final accuracy than uniform data shuffling.

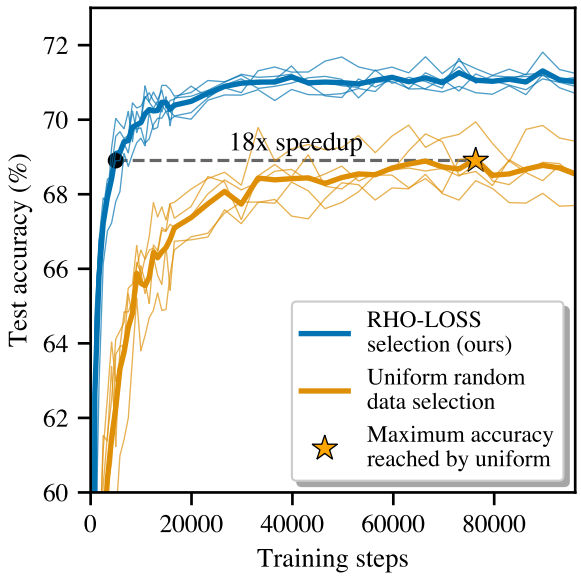

Figure 1: **Speedup on large-scale classification of web-scraped data (Clothing-1M)**. RHO-LOSS trains all architectures with fewer gradient steps than standard uniform data selection (i.e. shuffling), helping reduce training time. Thin lines: ResNet-50, MobileNet v2, DenseNet121, Inception v3, GoogleNet, mean across seeds. Bold lines: mean across all architectures.

1. Introduction

State-of-the-art models such as GPT-3 (Brown et al., 2020), CLIP (Radford et al., 2021), and ViT (Dosovitskiy et al., 2021) achieve remarkable results by training on vast amounts of web-scraped data. But despite intense parallelization, training such a model takes weeks or months (Radford et al., 2021; Chowdhery et al., 2022). Even practitioners who work with smaller models face slow development cycles, due to numerous iterations of algorithm design and hyperparameter selection. As a result, the total time required for training is a core constraint in the development of such deep learning models.

If it further sped up training, practitioners with sufficient resources would use much larger batches and distribute them across many more machines (Anil et al., 2018). However, this has rapidly diminishing returns (LeCun et al., 2012), to a point where adding machines does not reduce training time (McCandlish et al., 2018; Anil et al., 2018)—see e.g. GPT-3 and PaLM (Chowdhery et al., 2022).

Code: `https://github.com/OATML/RHO-Loss`

[*]Equal contribution [1]OATML, Department of Computer Science, University of Oxford [2]Department of Statistics, University of Oxford [3]Cohere [4]University of Toronto, performed at Cohere. Correspondence to: Sören Mindermann <soeren.mindermann@gmail.com>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Additional machines can, however, still speed up training by filtering out less useful samples (Alain et al., 2015). Many web-scraped samples are *noisy*, i.e. their label is incorrect or inherently ambiguous. For example, the text associated with a web-scraped image is rarely an accurate description of the image. Other samples are learnt quickly and are then *redundant*. Redundant samples are commonly part of object classes that are over-represented in web-scraped data (Tian et al., 2021) and they can often be left out without losing performance. Given that web-scraped data is plentiful—often enough to finish training in a single epoch (Komatsuzaki, 2019; Brown et al., 2020)—one can afford to skip less useful points.

However, there is no consensus on which datapoints are the most useful. Some works, including curriculum learning, suggest prioritizing *easy* points with low label noise before training on all points equally (Bengio et al., 2009). While this approach may improve convergence and generalization, it lacks a mechanism to skip points that are already learnt (redundant). Other works instead suggest training on points that are *hard* for the model, thereby avoiding redundant points, whose loss cannot be further reduced. Online batch selection methods (Loshchilov & Hutter, 2015; Katharopoulos & Fleuret, 2018; Jiang et al., 2019; Schaul et al., 2015) do so by selecting points with high loss or high gradient norm.

We show two failure modes of prioritising hard examples. Firstly, in real-world noisy datasets, high loss examples may be mislabelled or ambiguous. Indeed, in controlled experiments, points selected by high loss or gradient norm are overwhelmingly those with noise-corrupted labels. Our results show that this failure mode degrades performance severely. More subtly, we show that some samples are hard because they are outliers—points with unusual features that are less likely to appear at test time. For the aim of reducing test loss, such points are less *worth learning*.

To overcome these limitations, we introduce *reducible holdout loss selection* (RHO-LOSS). We propose a selection function grounded in probabilistic modelling that quantifies by how much each point would reduce the generalizaiton loss if we were to train on it, *without actually training on it*. We show that optimal points for reducing holdout loss are non-noisy, non-redundant, and task-relevant. To approximate optimal selection, we derive an efficient and easy to implement selection function: the reducible holdout loss.

We explore RHO-LOSS in extensive experiments on 7 datasets. We evaluate the reduction in required training steps compared to uniform sampling and state-of-the-art batch selection methods. Our evaluation includes Clothing-1M, the main large benchmark with noisy, web-scraped labels, matching our main application. RHO-LOSS reaches target accuracy in 18x fewer steps than uniform selection and achieves 2% higher final accuracy (Fig. 1). Further, RHO-LOSS consistently outperforms prior art and speeds up training across datasets, modalities, architectures, and hyperparameter choices. Explaining this, we show that methods selecting "hard" points prioritize noisy and less relevant examples. In contrast, RHO-LOSS chooses low-noise, task-relevant, non-redundant points—points that are learnable, worth learning, and not yet learnt.

2. Background: Online Batch Selection

Consider a model $\mathrm{p}(y \mid x; \theta)$ with parameters θ training on data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ using stochastic gradient descent (SGD). At each training step t, we load a batch b_t of size n_b from $\mathcal{D}$. In online batch selection (Loshchilov & Hutter, 2015), we uniformly pre-sample a larger batch B_t of size $n_B > n_b$. Then, we construct a smaller batch b_t that consists of the top-ranking n_b points in B_t ranked by a label-aware selection function $S(x_i, y_i)$. We perform a gradient step to minimize a mini-batch loss $L(y_i, \mathrm{p}(y_i \mid x_i; \theta))$ summed over $i \in b_t$. The next large batch B_{t+1} is then pre-sampled from $\mathcal{D}$ without replacement of previously sampled points (i.e. random shuffling: replacement when the next epoch starts).

3. Reducible Holdout Loss Selection

Previous online batch selection methods, such as loss or gradient norm selection, aim to select points that, if we were to train on them, would minimize the *training set* loss. (Loshchilov & Hutter, 2015; Katharopoulos & Fleuret, 2018; Kawaguchi & Lu, 2020; Alain et al., 2015). Instead, we aim to select points that minimize the loss on a *holdout set*. It would be too expensive to naively train on every candidate point and evaluate the holdout loss each time. In this section, we show how to (approximately) find the points that would most reduce the holdout loss if we were to train the current model on them, without actually training on them.

For simplicity, we first assume only one point $(x, y) \in B_t$ is selected for training at each time step t (we discuss selection of multiple points below). $\mathrm{p}(y' \mid x'; \mathcal{D}_t)$ is the predictive distribution of the current model, where $\mathcal{D}_t$ is the sequence of data the model was trained on before training step t. $\mathcal{D}_{\text{ho}} = \{(x_i^{\text{ho}}, y_i^{\text{ho}})\}_{i=1}^{n^{\text{ho}}}$, written as $\mathbf{x}^{\text{ho}}$ and $\mathbf{y}^{\text{ho}}$ for brevity, is a holdout set drawn from the same data-generating distribution $\mathrm{p}_{\text{true}}(x', y')$ as the training set $\mathcal{D}$. We aim to acquire the point $(x, y) \in B_t$ that, if we were to train on it, would minimize the negative log-likelihood/cross-entropy loss on the holdout set:

$$\underset{(x,y)\in B_t}{\arg\min} - \log \mathrm{p}(\mathbf{y}^{\text{ho}} \mid \mathbf{x}^{\text{ho}}; \mathcal{D}_t \cup (x, y)). \tag{1}$$

For a model using a point estimate of θ (such as an MLE or MAP), rather than a distribution over θ, the

holdout loss factorises and (up to a constant factor) forms a Monte Carlo approximation of the expected loss under p_{true}: $\mathbb{E}_{p_{true}(x',y')}[L[y' \mid x'; \mathcal{D}_t \cup (x,y)]] \approx \frac{1}{|\mathcal{D}_{ho}|} \sum_{(x_i^{ho}, y_i^{ho}) \in \mathcal{D}_{ho}} L[y_i^{ho} \mid x_i^{ho}; \mathcal{D}_t \cup (x,y)]$, where $L[\cdot]$ denotes the cross-entropy loss: $L[y \mid x] := -\log p(y \mid x)$.

Deriving a tractable selection function. We now derive a tractable expression for the term in Eq. (1) that does not require us to train on each candidate point $(x,y) \in B_t$ and then evaluate the loss on $\mathcal{D}_{ho}$. To make our claims precise and our assumptions transparent, we use the language of Bayesian probability theory. We treat model parameters as a random variable with prior $p(\theta)$ and infer a posterior $p(\theta|\mathcal{D}_t)$ using the already-seen training data $\mathcal{D}_t$. The model has a predictive distribution $p(y|x,\mathcal{D}_t) = \int_\theta p(y|x,\theta)\, p(\theta|\mathcal{D}_t) d\theta$. When using a point estimate of θ, the predictive distribution can be written as an integral with respect to a Dirac delta.

Using Bayes rule and the conditional independence $p(y_i \mid x_i, x_j; \mathcal{D}_t) = p(y_i \mid x_i; \mathcal{D}_t)$, we can derive a tractable selection function from Eq. (1). For readability, we switch the sign of the selection function, later changing the minimization to a maximization.

$$
\begin{aligned}
&\log p(\mathbf{y}^{ho} \mid \mathbf{x}^{ho}; \mathcal{D}_t \cup (x,y)) \\
&= \log \frac{p(y \mid x; \mathbf{x}^{ho}, \mathbf{y}^{ho}, \mathcal{D}_t)\, p(\mathbf{y}^{ho} \mid \mathbf{x}^{ho}, x; \mathcal{D}_t)}{p(y \mid x, \mathbf{x}^{ho}; \mathcal{D}_t)} && \text{Bayes rule} \\
&= \log \frac{p(y \mid x; \mathbf{y}^{ho}, \mathbf{x}^{ho}, \mathcal{D}_t)\, p(\mathbf{y}^{ho} \mid \mathbf{x}^{ho}; \mathcal{D}_t)}{p(y \mid x; \mathcal{D}_t)} && \text{conditional independence} \\
&\propto L[y \mid x; \mathcal{D}_t] - L[y \mid x; \mathcal{D}_{ho}, \mathcal{D}_t], && (2)
\end{aligned}
$$

where in the final line, we dropped terms independent of (x,y), rearranged, and applied the definition of $L[\cdot]$.

As exact Bayesian inference (conditioning on $\mathcal{D}_t$ or $\mathcal{D}_{ho}$) is intractable in neural networks (Blundell et al., 2015), we fit the models with SGD instead (**Approximation 1**). We study the impact of this approximation in Section 4.1. The first term, $L[y \mid x; \mathcal{D}_t]$, is then the *training loss* on the point (x,y) using the current model trained on $\mathcal{D}_t$. The second term, $L[y \mid x; \mathcal{D}_{ho}, \mathcal{D}_t]$, is the loss of a model trained on $\mathcal{D}_t$ and $\mathcal{D}_{ho}$.

Although the selection function in Eq. (2) is tractable, it is still somewhat expensive to compute, as both terms must be updated after each acquisition of a new point. However, we can approximate the second term with a model trained only on the holdout dataset, $L[y \mid x; \mathcal{D}_{ho}, \mathcal{D}_t] \approx L[y \mid x; \mathcal{D}_{ho}]$ (**Approximation 2**). This approximation saves a lot of compute: it is now sufficient to compute the term once before the first epoch of training. Later on, we show that this approximation empirically does not hurt performance on any tested dataset and even has some desired properties (Section 4.1 and Appendix D). We term $L[y \mid x; \mathcal{D}_{ho}]$ the *irreducible holdout loss* (IL) since it is the remaining loss on point $(x,y) \in \mathcal{D}$ after training on the holdout set $\mathcal{D}_{ho}$; in the limit of $\mathcal{D}_{ho}$ being large, it would be the lowest loss that the model can achieve without training on (x,y). Accordingly, we name our approximation of Eq. (2) the *reducible holdout loss*—the difference between the training loss and the irreducible holdout loss (IL).

Our method still requires us to train a model on a holdout set, but a final approximation greatly reduces that cost. We can efficiently compute the IL with an "irreducible loss model" (IL model) that is smaller than the target model and has low accuracy (**Approximation 3**). We show this and explain it in Sections 4.1, 4.2, and 4.3. Counterintuitively, the reducible holdout loss can therefore be negative. Additionally, one IL model can be reused for many target model runs, amortizing its cost (Section 4.2). For example, we trained all 40 seeds of 5 target architectures in Fig. 1 using a single ResNet18 IL model. Further, this model trained for 37x fewer steps than each target model (reaching only 62% accuracy). Section 5 details further possible efficiency improvements.

In summary, selecting a point that minimizes the holdout loss in Eq. (1), for a model trained on $\mathcal{D}_t$, can be approximated with the following easy-to-compute objective:

Reducible holdout loss selection (RHO-LOSS)

$$
\underset{(x,y) \in B_t}{\arg\max} \; \overbrace{\underbrace{L[y \mid x; \mathcal{D}_t]}_{\text{training loss}} - \underbrace{L[y \mid x; \mathcal{D}_{ho}]}_{\text{irreducible holdout loss (IL)}}}^{\text{reducible holdout loss}} \qquad (3)
$$

Although we required additional data $\mathcal{D}_{ho}$, this is not essential for large (Section 4.0) nor small (Section 4.2) datasets.

Understanding reducible loss. We now provide intuition on why reducible holdout loss selection (RHO-LOSS) avoids redundant, noisy, and less relevant points. **i) Redundant points.** We call a training point redundant when the model has already learnt it, i.e. its training loss cannot be further reduced. Since redundant points have low training loss, and the reducible loss is always less than the training loss (Eq. (3)), such points have low reducible loss and are not selected. And if the model forgets them, they are revisited in the next epoch. **ii) Noisy points.** While prior methods select based on high training loss (or gradient norm), not all points with high loss are informative—some may have an ambiguous or incorrect (i.e. noisy) label. The labels of such points cannot be predicted using the holdout set (Chen et al., 2019). Such points have high IL and, consequently, low reducible loss. These noisy points are less likely to be selected compared to equivalent points with

Algorithm 1 Reducible holdout loss selection (RHO-LOSS)

Input: Small model $\mathrm{p}(y \mid x; \mathcal{D}_{\text{ho}})$ trained on a holdout set $\mathcal{D}_{\text{ho}}$, batch size n_b, large batch size $n_B > n_b$, learning rate η.
for (x_i, y_i) in `training set` **do**
  `IrreducibleLoss[i]` $\leftarrow L[y_i \mid x_i; \mathcal{D}_{\text{ho}}]$
Initialize parameters θ^0 and $t = 0$
for $t = 0, 1, \ldots$ **do**
  Randomly select a large batch B_t of size n_B.
  $\forall i \in B_t$, compute `Loss[i]`, the train loss of point i given parameters θ^t
  $\forall i \in B_t$, compute `RHOLOSS[i]`$\leftarrow$ `Loss[i]`$-$ `IrreducibleLoss[i]`
  $b_t \leftarrow$ top-n_b samples in B_t in terms of `RHOLOSS`.
  $g_t \leftarrow$ mini-batch gradient on b_t using parameters θ^t
  $\theta^{t+1} \leftarrow \theta^t - \eta g_t$

less noise. **iii) Less relevant points.** Loss-based selection has an additional pitfall. The training loss is likely higher for outliers in input space—values of x far from most of the training data, in regions with low input density under $\mathrm{p}_{\text{true}}(x)$. Points with low $\mathrm{p}_{\text{true}}(x)$ should not be prioritized, all else equal. Consider an 'outlier' (x, y) and a non-outlier (x', y'), with $\mathrm{p}_{\text{true}}(x) < \mathrm{p}_{\text{true}}(x')$ but *equal* training loss $L[y|x; \mathcal{D}_{\text{t}}] = L[y'|x'; \mathcal{D}_{\text{t}}]$. As the holdout set $\mathcal{D}_{\text{ho}}$ is also drawn from p_{true}, $\mathcal{D}_{\text{ho}}$ will contain fewer points from the region around x in input space compared to the region around x'. Thus, training on (x, y) is likely to reduce the holdout loss (Eq. (1)) less, and so we prefer to train on the non-outlier (x', y'). In the specific sense described, (x, y) is thus less *relevant* to the holdout set. As desired, RHO-LOSS deprioritizes (x, y): since $\mathcal{D}_{\text{ho}}$ contains few points from the region around x, the IL of (x, y) will be large.

In short, RHO-LOSS *deprioritizes* points that are redundant (low training loss), noisy (high IL), or less relevant to the holdout set (high IL). That is, RHO-LOSS *prioritizes* points that are not yet learnt, learnable, and worth learning. We provide empirical evidence for these claims in Section 4.3. See Algorithm 1 for the implementation of RHO-LOSS.

Selecting multiple points concurrently. We showed which point is optimal when selecting a single point (x, y). When selecting an entire batch b_t, we select the points with the top-n_b scores from the randomly pre-sampled set B_t. This is nearly optimal when assuming that each point has little effect on the score of other points, which is often used as a simplifying assumption in active learning (Kirsch et al., 2019). This assumption is much more reasonable in our case than in active learning because model predictions are not changed much by a single gradient step on one mini-batch.

Simple parallelized selection. For large-scale neural network training, practitioners with sufficient resources would use many more machines if it further sped up training (Anil et al., 2018). However, as more workers are added in synchronous or asynchronous gradient descent, the returns diminish to a point where adding more workers does not further improve wall clock time (Anil et al., 2018; McCandlish et al., 2018). For example, there are rapidly diminishing returns for using larger batch sizes or distributing a given batch across more workers, for multiple reasons (McCandlish et al., 2018; Keskar et al., 2016). The same holds for distributing the model across more workers along its width or depth dimension (Rasley et al., 2020; Shoeybi et al., 2019; Huang et al., 2019). However, we can circumvent these diminishing returns by adding a new dimension of parallelization, namely, for data selection.

Since parallel *forward* passes do not suffer from such diminishing returns, one can use extra workers to evaluate training losses in parallel (Alain et al., 2015). The theoretical runtime speedup can be understood as follows. The cost per training step of computing the selection function on B_t is $\frac{n_B}{3n_b}$ times as much as the cost of the forward-backward pass needed to train on b_t since a forward pass requires at least 3x less computation than a forward-backward pass (Jouppi et al., 2017). One can reduce the time for the selection phase almost arbitrarily by adding more workers that compute training losses using a copy of the model being trained. The limit is reached when the time for selection is dominated by the communication of parameter updates to workers. More sophisticated parallelization strategies allow reducing the time overhead even further (Section 5). To avoid assumptions about the particular strategy used, we report experiment results in terms of the required number of training epochs.

4. Experiments

We evaluate our selection method on several datasets (both in controlled environments and real-world conditions) and show significant speedups compared to prior art, in the process shedding light on the properties of different selection functions.

Recall that our setting assumes training time is a bottleneck but data is abundant—more than we can train on (see Bottou & LeCun (2004)). This is common e.g. for web-scraped data where state-of-the-art performance is often reached in less than half of one epoch (Komatsuzaki, 2019; Brown et al., 2020). As data is abundant, we can set aside a holdout set for training the IL model with little to no downside. For the large Clothing-1M dataset, we implement RHO-LOSS by training the IL model on 10% of the training data, while all baselines are trained on the full 100% of the training data. For the smaller datasets, we simulate abundance of data by

reserving a holdout set and training *all* methods only on the remaining data. However, RHO-LOSS also works on small datasets without additional data by double-using the training set (Section 4.2).

Datasets. We evaluate on 7 datasets: 1) QMNIST (Yadav & Bottou, 2019) extends MNIST (LeCun et al., 1998) with 50k extra images which we use as the holdout set. 2) On CIFAR-10 (Krizhevsky & Hinton, 2009) we train on half of the training set and use the other half as a holdout to train the irreducible loss (IL) model. 3) CIFAR-100: same as CIFAR-10. 4) CINIC-10 (Darlow et al., 2018) has 4.5x more images than CIFAR-100 and includes a holdout set and a test set with 90k images each. 5) Clothing-1M (Xiao et al., 2015), which contains over 1 million 256x256-resolution clothing images from 14 classes. The dataset is fully web-scraped—a key application area of our work—and is the most widely accepted benchmark for image recognition with noisy labels (Algan & Ulusoy, 2021). We use the whole training set for training and reuse 10% of it to train the IL model. We further evaluate on two NLP datasets from GLUE (Wang et al., 2018): 6) CoLA (grammatical acceptability) and 7) SST-2 (sentiment). We split their training sets as for CIFAR.

Baselines. Aside from uniform sampling (without replacement, i.e. random shuffling), we also compare to selection functions that have achieved competitive performance in online batch selection recently: the (training) loss, as implemented by Kawaguchi & Lu (2020), gradient norm, and gradient norm with importance sampling (called *gradient norm IS* in our figures), as implemented by Katharopoulos & Fleuret (2018). We also compare to the core-set method Selection-via-Proxy (SVP) that selects data offline before training (Coleman et al., 2020). We report results using maximum entropy SVP and select with the best-performing model, ResNet18. We further compare to four baselines from active learning, shown in Appendix G as they assume labels are unobserved. Finally, we include selection using the negative IL (see Eq. 3) to test if it is sufficient to only skip noisy and less relevant but not redundant points.

Models and hyperparameters. To show our method needs no tuning, we use the PyTorch default hyperparameters (with the AdamW optimizer (Loshchilov & Hutter, 2017)) and $\frac{n_b}{n_B} = 0.1$. We test many additional hyperparameter settings in Figs. 2 (row 5) and 8. We test various architectures in Figs. 1 and 2 (row 4). In all other figures, we use a 3 layer MLP for experiments on QMNIST, a ResNet-18 adapted for small images for CIFAR-10/CIFAR-100/CINIC-10, and a ResNet-50 for Clothing-1M. All models for Clothing-1M are pre-trained on ImageNet (standard for this dataset (Algan & Ulusoy, 2021)) and the IL model is *always* a ResNet-18. For the NLP datasets, we use a pre-trained ALBERT v2 (Lan et al., 2019). We always use the IL model checkpoint with lowest validation loss (not highest accuracy); this performs best. Details in Appendix B.

Table 1: Spearman's rank correlation of rankings of data points by selection functions that are increasingly less faithful approximations of Eq. (2), compared to the most faithful approximation. Approximations added from left to right. Mean across 3 seeds.

	Non-Bayesian	Not converged	Not updating IL model	Small IL model
Rank correlation	0.75	0.76	0.63	0.51

Evaluation. We measure speedup in terms of the number of epochs needed to reach a given test accuracy. We measure epochs needed, rather than wall clock time, as our focus is on evaluating a new selection function, not an entire training pipeline. Wall clock time depends primarily on the hardware used and implementation details that are beyond our scope. Most importantly, data selection is amenable to parallelization beyond standard data parallelism as discussed in Section 3.

4.1. Impact of Approximations

In Section 3, we introduced a function for selecting exactly the points that most reduce the model's loss on a holdout set. To make this selection function efficient for deep neural networks, we made several approximations. Here, we study how these approximations affect the points selected, by successively introducing one approximation after the other.

Because the exact selection function (Eq. (2)) is intractable, we start with a close (and expensive) approximation as the gold standard (Approximation 0). To make Approximation 0 feasible, the experiments are conducted on an easy dataset—QMNIST (with 10% uniform label noise and data duplication to mimic the properties of web-scraped data). We then successively introduce the Approximations 1, 2, and 3 described in Section 3. To assess the impact of each approximation, we train a model without and with the approximations, and then compute the rank correlation (Spearman's correlation coefficient) of the selection function evaluated on each batch B_t. Across the first epoch, we present the mean of the rank correlations. Since each approximation selects different data, the corresponding models become more different over time; this divergence causes some of the observed difference in the points they select. See Appendix E for details.

Approximation 0. To get as close as possible to the Bayesian inference/conditioning used in Eq. (2), we use a deep ensemble of 5 neural networks and train them *to convergence* after every time step t on the acquired dataset $b_t \cup \mathcal{D}_t$ (Wilson & Izmailov, 2020).

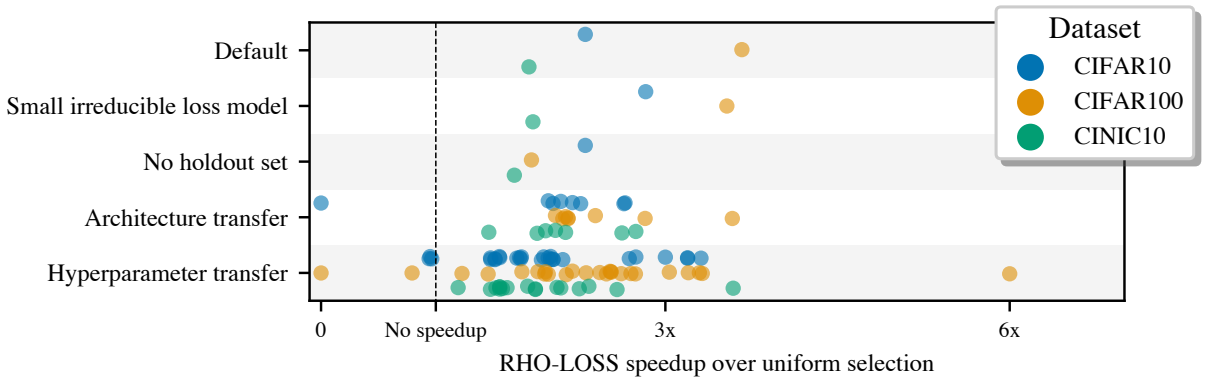

Figure 2: **The irreducible loss model can be small, trained with no holdout data, and reused across target architectures and hyperparameters.** Here, we use clean datasets, where speedups are smallest. The x-axis shows speedup, i.e. after how many fewer epochs RHO-LOSS exceeds the highest accuracy uniform selection achieves within 100 epochs. Row 1 uses a ResNet18 as irreducible loss model. All other rows instead use a small, cheap CNN. Each dot shows an experiment with a given combination of irreducible loss model and target model (mean across 2-3 seeds for all but the last row).

Approximation 1: SGD instead of Bayesian inference/conditioning. Approximation 0 is a close approximation of Eq. (2), but training an ensemble to convergence at every step t is far too expensive in practice. Starting from this gold-standard, we introduce two stronger approximations (1a and 1b) to move to standard neural network fitting with AdamW. 1a) First, we replace the ensemble with a single model, while still training to convergence at each time step. The Spearman's coefficient between this approximation and Approximation 0 is 0.75, suggesting similar points are selected ("Non-Bayesian" in Table 1). 1b) Next, we only take one gradient step on each new batch b_t. The Spearman's coefficient, when comparing this to Approximation 0, is 0.76 ("Not Converged" in Table 1).

Approximation 2. Not updating the IL model on the acquired data $\mathcal{D}_t$. Second, we save compute by approximating $L[y \mid x; \mathcal{D}_t, \mathcal{D}_{ho}]$ with $L[y \mid x; \mathcal{D}_{ho}]$. The points selected are still similar to Approximation 0 (Spearman's coefficient 0.63, "Not updating IL model" in Table 1). This approximation also performs well on other datasets (Appendix D).

Approximation 3: Small IL model. Lastly, we use a model with 256 hidden units instead of 512 (4x fewer parameters) as the IL model and see again that similar points are selected (Spearman's coefficient 0.51). We study cheaper IL models in other forms and datasets in the next section.

4.2. Cheap Irreducible Loss Models & Robustness

RHO-LOSS requires training an IL model on a holdout set, which poses additional costs. Here, we show how to minimize these costs and amortize them across many training runs of target models. The same experiments also show the robustness of RHO-LOSS across architectures and hyperparameter settings. To fit our computational budget, we perform these experiments on moderate-sized clean benchmark datasets although RHO-LOSS produces greater speedups on noisy or redundant web-scraped data (see Section 4.4).

Irreducible loss models can be small and cheap. In our default setting (Fig. 2, row 1), both the target model and IL model have the same architecture (ResNet-18). In rows 2 and below, we instead used a small CNN similar to LeNet as the IL model (LeCun et al., 1989). It has 21x fewer parameters and requires 29x fewer FLOP per forward pass than the ResNet-18. **The smaller IL model accelerates training as much or more than the larger model**, even though its final accuracy is far lower than the target ResNet-18 (11.5% lower on CIFAR-10, 7% on CIFAR-100, and 8.1% on CINIC-10). We examine in Section 4.3 why this useful result holds.

Irreducible loss models without holdout data. Web-scraped datasets are often so large that even a small fraction of the overall data can be sufficient to train the IL model. E.g., in our experiments on Clothing-1M (Fig. 1), the holdout set is only 10% as large as the main train set. Additionally, we can train the IL model without any holdout data (Fig. 2, row 3). We split the training set $\mathcal{D}$ into two halves and train an IL model on each half (still using small IL models). Each model computes the IL for the half of $\mathcal{D}$ that it was not trained on. Training two IL models costs no additional compute since each model is trained on half as much data compared to the default settings.

Irreducible loss models can be reused to train different target architectures. We find that a single small

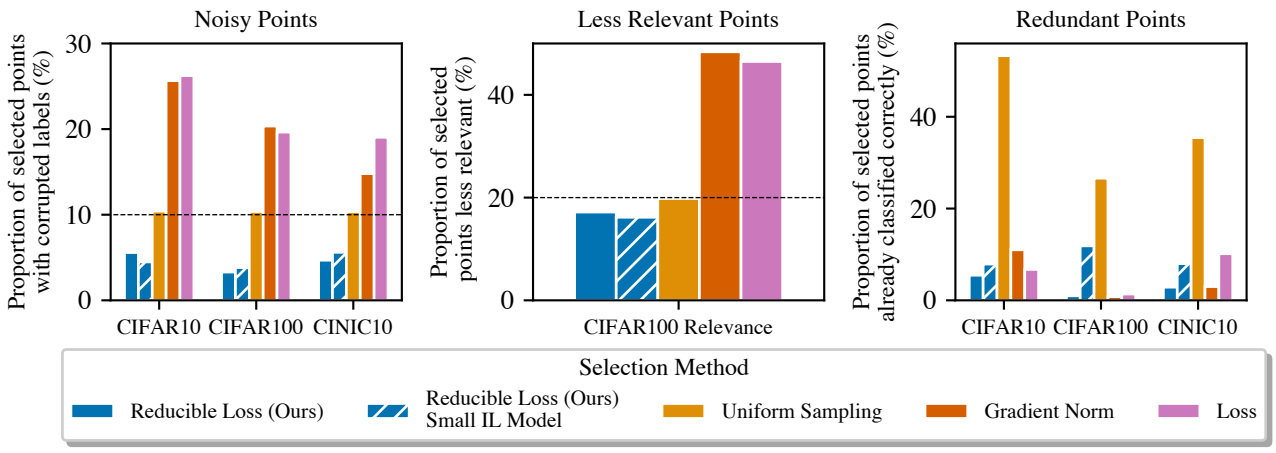

Figure 3: Properties of RHO-LOSS and other methods. RHO-LOSS prioritizes points that are non-noisy, task-relevant, and non-redundant—even when the irreducible loss (IL) model is a small CNN. In contrast, loss and gradient norm prioritize noisy and less relevant points (while also avoiding redundant points). **Left.** Proportion of selected points with corrupted labels. We added 10% uniform label noise, i.e., we randomly switched each point's label with 10% probability. **Middle.** Proportion of selected points from low relevance classes on CIFAR100 Relevance dataset. **Right.** Proportion of selected points that are already classified correctly, which is a proxy for redundancy. Mean over 150 epochs of training and 2-3 seeds.

CNN IL model accelerates the training of 7 target architectures (Fig. 2, row 4): VGG11 (with batchnorm), GoogleNet, Resnet34, Resnet50, Densenet121, MobileNet-v2, Inception-v3. RHO-LOSS does not accelerate training on CIFAR-10 for VGG11, which is also the architecture on which uniform training performs the worst; i.e. RHO-LOSS empirically does not "miss" a good architecture. Not only is RHO-LOSS robust to architectures choice, a single IL model can also be *reused by many practitioners* who use different architectures (as we did in Fig. 1).

Irreducible loss models can be reused to train many targets in a hyperparameter sweep. We find that a single small CNN accelerates the training of ResNet-18 target models across a hyperparameter grid search (Fig. 2, last row). We vary the batch size (160, 320, 960), learning rate (0.0001, 0.001, 0.01), and weight decay coefficient (0.001, 0.01, 0.1). RHO-LOSS speeds up training compared to uniform on nearly all target hyperparameters. The few settings in which it doesn't speed up training are also settings in which uniform training performs very poorly ($< 30\%$ accuracy on CIFAR-100, $< 80\%$ on CIFAR-10).

4.3. Properties of RHO-LOSS & Other Selection Functions

We established that RHO-LOSS can accelerate the training of various target architectures with a single IL model, even if the IL model is smaller and has considerably lower accuracy than the target models (Section 4.2). This suggests robustness to target-IL architecture mismatches.

To understand this robustness, we investigate the properties of points selected by RHO-LOSS, when the target and IL model architectures are identical, and when the IL model is smaller. Explaining the robustness, we find that, in both cases, RHO-LOSS prioritizes points that are non-noisy, task-relevant, and not redundant. We also investigate the properties of points selected by prior art.

Noisy points. We investigate how often different methods select noisy points by uniformly corrupting the labels for 10% of points and tracking what proportion of selected points are corrupted. RHO-LOSS deprioritizes noisy points for both IL models (Fig. 3). We observe a failure mode of the widely-used loss and gradient norm selection functions: they select far more noisy points than uniform. These methods also severely drop in accuracy when the noise follows the class confusion matrix (Rolnick et al., 2017) and when we add ambiguous images (Mukhoti et al., 2021) (Appendix C).

Together, this suggests that noisy points have high loss (and gradient norm), but also high IL and thus low reducible loss. Their IL is high even when the IL model is small as noisy labels cannot be predicted well using the holdout set.

Relevant points. We study how often less relevant points are selected by creating the CIFAR100 Relevance dataset, in which 80% of the data comes from 20% of the classes. This mimics natural distributions of NLP and vision data where most data comes from few object classes, topics, or words (Baayen, 2001; Tian et al., 2021). Concretely, we retain all examples from 20 randomly chosen "high relevance" classes

Table 2: Epochs required to reach a given target test accuracy (final accuracy in parentheses). Figs. 4 and 5 (Appendix) show all training curves. Some datasets have 10% uniform label noise added. Results averaged across 2-4 seeds. Best performance in **bold**. RHO-LOSS performs best in both epochs required and final accuracy. *NR* indicates that the target accuracy was not reached. *On CIFAR10/100, CoLA, and SST-2, only half of the data is used for training (Section 4.0).

Dataset	Target Acc	Number of epochs method needs to reach target accuracy ↓ (Final accuracy in parentheses)						
		Train Loss	Grad Norm	Grad Norm IS	SVP	Irred Loss	Uniform	RHO-LOSS
Clothing-1M	60.0%	8	13	2	*NR*	*NR*	2	**1**
	69.0%	*NR* (65%)	*NR* (64%)	9 (70%)	*NR* (55%)	*NR* (48%)	30 (70%)	**2 (72%)**
CIFAR10*	80.0%	81	*NR*	57	*NR*	*NR*	79	**39**
	87.5%	129 (90%)	*NR* (61%)	139 (89%)	*NR* (55%)	*NR* (60%)	*NR* (87%)	**65 (91%)**
CIFAR10*	75.0%	*NR*	*NR*	57	*NR*	*NR*	62	**27**
(Label Noise)	85.0%	*NR* (28%)	*NR* (23%)	*NR* (84%)	*NR* (48%)	*NR* (62%)	*NR* (85%)	**49 (91%)**
CIFAR100*	40.0%	138	139	71	*NR*	93	65	**48**
	52.5%	*NR* (42%)	*NR* (42%)	132 (55%)	*NR* (18%)	*NR* (43%)	133 (54%)	**77 (61%)**
CIFAR100*	40.0%	*NR*	*NR*	94	*NR*	89	79	**49**
(Label Noise)	47.5%	*NR* (4%)	*NR* (4%)	142 (48%)	*NR* (14%)	*NR* (43%)	116 (50%)	**65 (60%)**
CINIC10	70.0%	*NR*	*NR*	34	*NR*	*NR*	38	**27**
	77.5%	*NR* (36%)	*NR* (50%)	64 (82%)	*NR* (39%)	*NR* (60%)	97 (80%)	**38 (83%)**
CINIC10	60.0%	*NR*	*NR*	22	*NR*	30	24	**13**
(Label Noise)	67.5%	*NR* (16%)	*NR* (16%)	35 (79%)	*NR* (39%)	*NR* (64%)	38 (78%)	**17 (82%)**
SST2*	82.5%	8	2	3	*NR*	7	**1**	**1**
	90.0%	*NR* (87%)	4 (91%)	*NR* (89.7%)	*NR* (66%)	*NR* (83%)	6 (90%)	**3 (92%)**
CoLA*	75.0%	8	6	16	*NR*	*NR*	34	**3**
	80.0%	*NR* (78%)	*NR* (79%)	*NR* (78%)	*NR* (62%)	*NR* (69%)	*NR* (76%)	**39 (80%)**

but only 6% of the examples from other, "low relevance" classes. Intuitively, since the high relevance classes have higher $\mathrm{p}_{\text{true}}(x)$ and are 17x more likely to appear at test time, improving their accuracy improves the test accuracy much more than improving the accuracy of less relevant classes.

The loss and gradient norm methods select more points than uniform selection from the low relevance classes (Fig. 3). In contrast, RHO-LOSS selects somewhat fewer low relevance points, suggesting these classes have high IL. Since the less relevant classes are less abundant in the holdout set, both the small and large IL models have higher loss on them.

Redundant points. To investigate whether methods select redundant points, we track the percentage of selected points that are already classified correctly. This is only a proxy for redundancy; points that are classified correctly but with low confidence are not fully redundant, since their loss can be further reduced. We control for the different accuracy reached by each method by averaging only over epochs in which test accuracy is lower than the final accuracy reached by the weakest performing method. Fig. 3 shows that all methods select fewer redundant points than uniform sampling.

4.4. Speedup

Finally, we evaluate how much different selection methods speed up training. Recall that the main application area for our work is large web-scraped datasets, known for high levels of noise and redundancy. Clothing-1M is such a dataset (Section 4.0). We also include smaller, clean benchmarks from vision (CIFAR-10, CIFAR-100, CINIC-10) and NLP (CoLA, SST-2). Finally, we study if selection functions are robust to the controlled addition of label noise.

Speedup on clean data. RHO-LOSS reaches target accuracies in fewer epochs than uniform selection on all datasets (Table 2). It also outperforms state-of-the-art methods by a clear margin in terms of speed and final accuracy. On the challenging CoLA language understanding dataset, the speedup over uniform selection exceeds 10x. In Table 3 (Appendix A), we find similar speedups when using no holdout data.

Speedup on noisy data. When adding 10% label noise, batch selection with RHO-LOSS achieves greater speedups while, as hypothesized, prior art degrades (Table 2). Notably, on noisier data, the speedup over uniform selection grows.

Speedup on large web-scraped data. On Clothing-1M, loss-based and gradient norm-based selection fail to match uniform selection, suggesting they are not robust to noise. In contrast, RHO-LOSS reaches the highest accuracy that uniform selection achieves during 50 epochs in just 2 epochs and improves final accuracy (72% vs 70%). Notably, this was possible even though the IL model we used has low accuracy (62.2%) and was trained on ca. 10x less data. RHO-LOSS also used 2.7x fewer FLOPs to reach the peak

accuracy of uniform selection, including the cost of training the IL model (which could be amortized) and despite our implementation being designed to save time, not compute. While Table 2 shows results for a Resnet-50, Fig. 1 includes additional architectures, with an average speedup of 18x.

5. Related Work

Time-efficient data selection. Forward passes for selection can be accelerated using low-precision hardware or parallelization. While backward passes typically require high precision, forward passes can tolerate lower precision (Jouppi et al., 2017; Jiang et al., 2019), especially as we only need the loss (not the activations which would be needed for backprop). A forward pass by default requires roughly 3x less time than a forward-backward pass but this speedup can be increased to a factor around 10x when using the low-precision cores available in modern GPUs and TPUs (Jouppi et al., 2017; Jiang et al., 2019). Further, prior work uses a set of workers that perform forward passes on B_t or on the entire dataset asynchronously while the master process trains on recently selected data (Alain et al., 2015).

Compute-efficient data selection. While we limit our scope to comparing selection functions and we compute them naively, this choice is inefficient in practice. Selection can be made cheaper by reusing losses computed in previous epochs (Loshchilov & Hutter, 2015; Jiang et al., 2019) or training a small model to predict them (Katharopoulos & Fleuret, 2017; Zhang et al., 2019; Coleman et al., 2020). Alternatively, core set methods perform selection once before training (Mirzasoleiman et al., 2020; Borsos et al., 2020), although typically with more expensive selection functions.

Data selection functions. RHO-LOSS is best understood as an alternative to existing selection functions, which can be categorized by the properties of points they select and whether they use information about labels. "Hard" points are selected both by high loss (Loshchilov & Hutter, 2015; Kawaguchi & Lu, 2020; Jiang et al., 2019) and high prediction uncertainty (Settles, 2009; Li & Sethi, 2006; Coleman et al., 2020). However, prediction uncertainty does not require labels and can thus be used for active learning. Despite this, they both suffer from the same problem: high loss and high uncertainty can be caused by noisy (in particular, ambiguous) labels. This also applies to selection of points whose labels are easily forgotten during training (Toneva et al., 2018). Noisy points are avoided by our negative IL baseline and comparable offline selection methods (Pleiss et al., 2020; Chen et al., 2019; Paul et al., 2021). Points that most reduce (expected) holdout loss are also selected for other purposes (Kirsch et al., 2021; Killamsetty et al., 2020; Ren et al., 2018), although using much more computation.

Variance reduction methods. Online batch selection is also used to reduce the variance of the gradient estimator computed by SGD (Katharopoulos & Fleuret, 2018; 2017; Johnson & Guestrin, 2018; Alain et al., 2015), which is widely used in reinforcement learning (Schaul et al., 2015). Such methods typically use importance sampling—points with high (approximate) gradient norm are sampled with high probability but then down-weighted in the gradient calculation to de-bias the gradient estimate. Without de-biasing, methods like RHO-LOSS also create selection bias. However, bias can improve test performance, both in theory and practice (Farquhar et al., 2021; Kawaguchi & Lu, 2020).

6. Conclusion

To reduce excessive training times, we introduce a theoretically grounded selection function that enables substantial speedups on clean data and even larger speedups on noisy and web-scraped data. By illuminating three properties of optimal selection, we hope to motivate new directions in batch selection. However, our selection function should be combined with methods in Section 5 for cheap and fast selection with maximal speedups.

Ethics Statement

It will be important to understand how subset selection might affect performance on data about minority groups. The selection may prioritize rare groups since majority groups are learnt more quickly, or deprioritizes rare groups since they affect the loss on holdout data less.

Since such biases can also stem from the dataset itself (Mehrabi et al., 2021), it should be investigated if our method can remove data biases through the use of an unbiased holdout set. By training the irreducible loss model on unbiased data, we can implicitly specify that the model should perform well on unbiased data, even when the training set contains bias. This may be useful for specifying that all groups are equally important to learn.

Acknowledgements

For useful feedback we thank Pascal Notin and Kelsey Doerksen.

Author Contributions

Sören Mindermann, Jan Brauner, Mrinank Sharma and Muhammed Razzak designed and analysed the experiments shown in the paper.

Jan Brauner implemented experiments on ALBERT, CIFAR-10, experiments in Figure 2 and 7, Table 3 and 4, among others. Muhammed Razzak implemented experiments on

Clothing-1M, CIFAR-100, and MNIST, among others. Mrinank Sharma implemented experiments on CINIC-10, experiments in Figures 3 and 9, among others.

Sören Mindermann and Muhammed Razzak implemented pilot experiments.

Winnie Xu and Adrien Morisot implemented early experiments on language models, advised by Aidan Gomez.

Sören Mindermann, Jan Brauner, Mrinank Sharma, Muhammed Razzak, Benedikt Höltgen and Andreas Kirsch wrote the paper.

Sören Mindermann conceived of the algorithm.

Sören Mindermann, Jan Brauner, Andreas Kirsch and Yarin Gal developed the theory.

Sören Mindermann led and managed the research.

Yarin Gal and Sebastian Farquhar advised the research.

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.

Algan, G. and Ulusoy, I. Image classification with deep learning in the presence of noisy labels: A survey. *Knowledge-Based Systems*, 215:106771, 2021.

Anil, R., Pereyra, G., Passos, A., Ormandi, R., Dahl, G. E., and Hinton, G. E. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

Baayen, R. H. *Word frequency distributions*, volume 18. Springer Science & Business Media, 2001.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural network. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1613–1622, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/blundell15.html.

Borsos, Z., Mutnỳ, M., and Krause, A. Coresets via bilevel optimization for continual learning and streaming. *arXiv preprint arXiv:2006.03875*, 2020.

Bottou, L. and LeCun, Y. Large scale online learning. *Advances in neural information processing systems*, 16:217–224, 2004.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, P., Liao, B. B., Chen, G., and Zhang, S. Understanding and utilizing deep neural networks trained with noisy labels. In *International Conference on Machine Learning*, pp. 1062–1070. PMLR, 2019.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

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. *International Conference on Learning Representations*, 2020.

Darlow, L. N., Crowley, E. J., Antoniou, A., and Storkey, A. J. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

Farquhar, S., Gal, Y., and Rainforth, T. On statistical bias in active learning: How and when to fix it. *arXiv preprint arXiv:2101.11665*, 2021.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

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.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32:103–112, 2019.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Jiang, A. H., Wong, D. L.-K., Zhou, G., Andersen, D. G., Dean, J., Ganger, G. R., Joshi, G., Kaminksy, M., Kozuch, M., Lipton, Z. C., et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019.

Johnson, T. B. and Guestrin, C. Training deep models faster with robust, approximate importance sampling. *Advances in Neural Information Processing Systems*, 31: 7265–7275, 2018.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Katharopoulos, A. and Fleuret, F. Biased importance sampling for deep neural network training. *arXiv preprint arXiv:1706.00043*, 2017.

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.

Kawaguchi, K. and Lu, H. Ordered sgd: A new stochastic optimization framework for empirical risk minimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 669–679. PMLR, 2020.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

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.

Kirsch, A., Van Amersfoort, J., and Gal, Y. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *arXiv preprint arXiv:1906.08158*, 2019.

Kirsch, A., Rainforth, T., and Gal, Y. Active learning under pool set distribution shift and noisy data. *CoRR*, abs/2106.11719, 2021. URL https://arxiv.org/abs/2106.11719.

Komatsuzaki, A. One epoch is all you need. *arXiv preprint arXiv:1906.06669*, 2019.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

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

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Li, M. and Sethi, I. K. Confidence-based active learning. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1251–1261, 2006.

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

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., and Galstyan, A. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

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.

Mukhoti, J., Kirsch, A., van Amersfoort, J., Torr, P. H., and Gal, Y. Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty. *arXiv preprint arXiv:2102.11582*, 2021.

Osawa, K., Swaroop, S., Jain, A., Eschenhagen, R., Turner, R. E., Yokota, R., and Khan, M. E. Practical deep learning with bayesian principles. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 4287–4299, 2019.

Paul, M., Ganguli, S., and Dziugaite, G. K. Deep learning on a data diet: Finding important examples early in training. *arXiv preprint arXiv:2107.07075*, 2021.

Phan, H. huyvnphan/pytorch_cifar10. Jan 2021. doi: 10.5281/zenodo.4431043.

Pleiss, G., Zhang, T., Elenberg, E., and Weinberger, K. Q. Identifying mislabeled data using the area under the margin ranking. *Advances in Neural Information Processing Systems*, 33:17044–17056, 2020.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pp. 4334–4343. PMLR, 2018.

Rolnick, D., Veit, A., Belongie, S., and Shavit, N. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

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

Settles, B. Active learning literature survey. 2009.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Tian, Y., Henaff, O. J., and Oord, A. v. d. Divide and contrast: Self-supervised learning from uncurated data. *arXiv preprint arXiv:2105.08054*, 2021.

Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wilson, A. G. and Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. Learning from massive noisy labeled data for image classification. In *CVPR*, 2015.

Yadav, C. and Bottou, L. Cold case: The lost mnist digits. In *Advances in Neural Information Processing Systems 32*, 2019.

Yi, K. and Wu, J. Probabilistic end-to-end noise correction for learning with noisy labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Zhang, J., Yu, H.-F., and Dhillon, I. S. Autoassist: A framework to accelerate training of deep neural networks. *arXiv preprint arXiv:1905.03381*, 2019.

Appendix

A. Steps required to reach a given test accuracy

Figs. 4 (vision) and 5 (NLP) show the number of steps required to reach a given test accuracy across several datasets for different selection methods. Interestingly, on CoLA (unbalanced and noisy), the uniform sampling baseline shows high variance across seeds, while RHO-LOSS works robustly across seeds.

Table 3 shows results for RHO-LOSS training without holdout data. Results are similar to Table 2. Here, we train the IL model without any holdout data. We split the training set $\mathcal{D}$ into two halves and train an IL model on each half. Each model computes the IL for the half of $\mathcal{D}$ that it was not trained on. (This is as in Fig. 2, row 3, except that previously we only used half of $\mathcal{D}$ and further split it into halves of the half.) Training two IL models costs no additional compute since each model is trained on half as much data compared to the default settings.

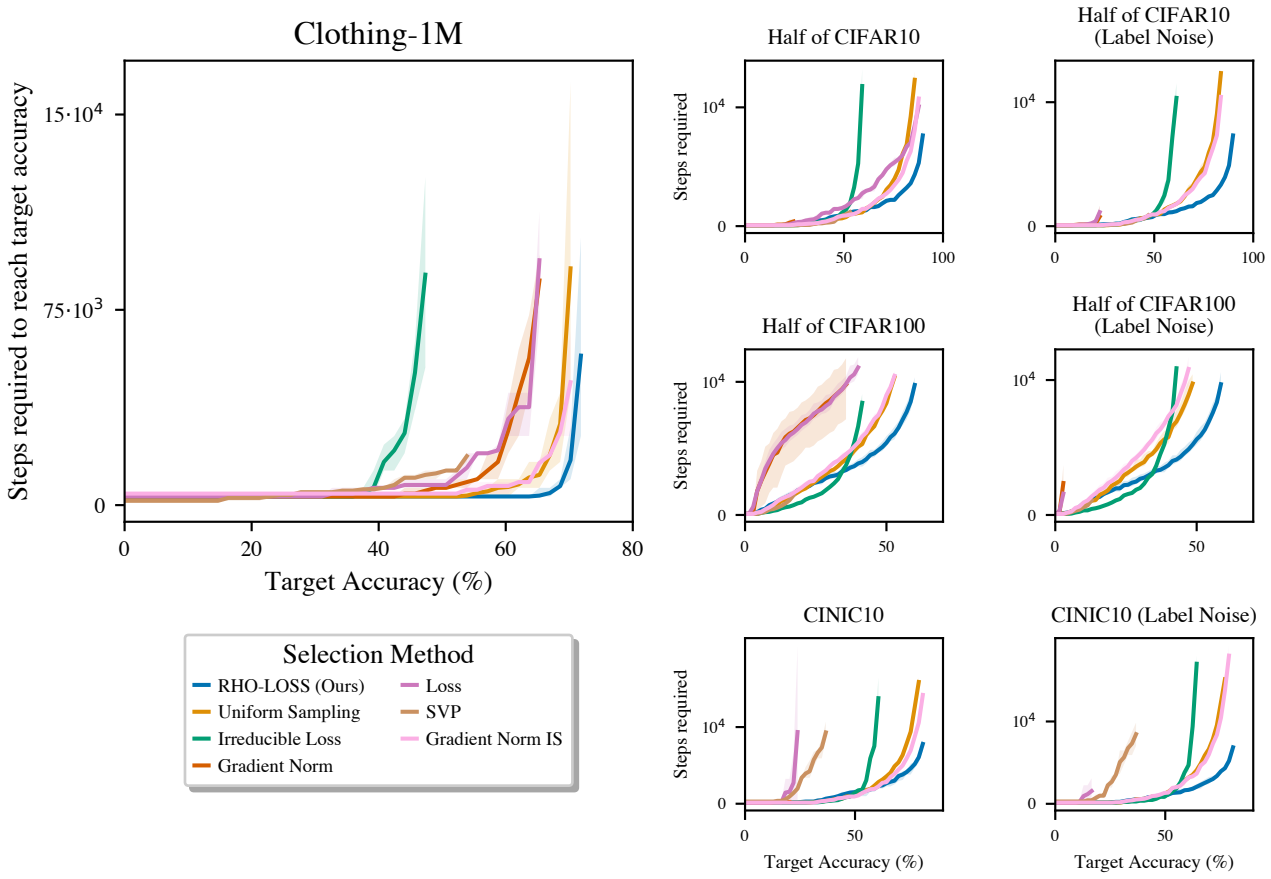

Figure 4: Vision datasets—gradient steps required to achieve a given test accuracy **(lower is better)**. **Left column:** The speedup of RHO-LOSS over uniform sampling is greatest on a large-scale web-scraped dataset with noisy labels. **Middle column:** Speedups are still substantial on clean datasets and RHO-LOSS still achieves higher final accuracy than all prior art. **Right column:** Applying 10% uniform label noise to training data degrades other methods but increases the speedup of our method. A step corresponds to lines $5-10$ in Algorithm 1. Lines correspond to means and shaded areas to minima and maxima across 3 random seeds. On CIFAR10/100, only half of the data is used for training (see text).

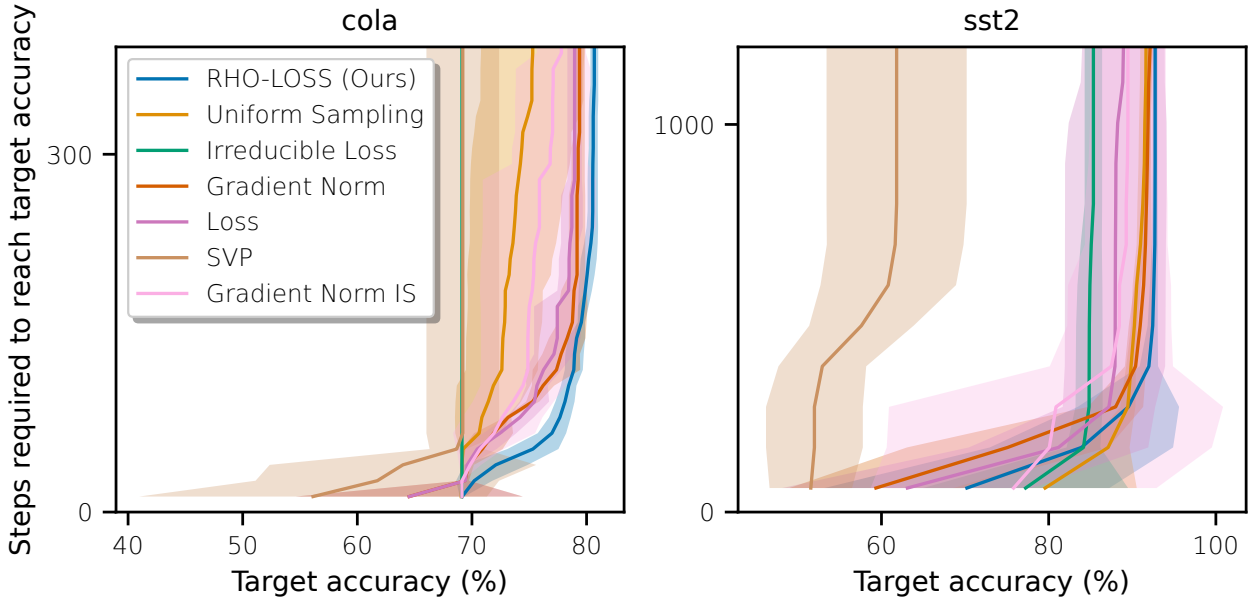

Figure 5: NLP datasets—gradient steps required to achieve a given test accuracy **(lower is better)**. **Left:** CoLA grammatical acceptibility classification. **Right:** SST2 sentiment classification. A step corresponds to lines $5-10$ in Algorithm 1. Lines correspond to means and shaded areas to standard deviations across 4 or more random seeds. Only half of the data is used for training (see text).

Table 3: **Epochs required to reach a given target test accuracy when using no holdout data** (lower is better). Final accuracy in parentheses. Results averaged across 2-3 seeds. Best performance in bold. RHO-LOSS performs best in both epochs required and final accuracy.

Dataset	Target Acc	Uniform	RHO-LOSS
CIFAR10	80%	39	**17**
	90%	177 (90.8%)	**47 (92.2%)**
CIFAR100	50%	47	**22**
	65%	142 (67.8%)	**87 (68.1%)**
CINIC10	70%	37	**26**
	80%	146 (80.1%)	**70 (82.1%)**

B. Experiment Details

Architectures. We experiment with various architectures in Figs. 1 and 2 (row 4). In all other figures and tables, we use the following architectures: For experiments on QMNIST, we use a multi-layer perceptron with 2 hidden layers and 512 units in each hidden layer. For experiments on CIFAR-10, CIFAR-100 and CINIC-10, we use a variant of ResNet-18 (He et al., 2016). We adapted the ResNet18 to 32x32 images by modifying the architecture to remove the downsampling effect. We replaced the spatial downsampling of a strided convolution and max pooling in the original ResNet18, with a convolutional layer with 64 filters and a kernel size of 3x3. We also removed the average pooling at the end of the ResNet18. This ResNet18 variant is similar to Resnet20, just with more filters. For experiments on Clothing-1M, following the experimental set-up of Yi & Wu (2019), the target model is a ResNet-50 pre-trained on ImageNet. The irreducible loss model is a ResNet-18 with random initialisation. The multiple target architectures in Fig 2 were adapted from (Phan, 2021). For NLP datasets, we use a pretrained ALBERT v2 (Lan et al., 2019).

Hyperparameters. *Vision*: All models are trained using the AdamW optimizer with default PyTorch hyperparameters (β_1=0.9, β_2=0.999, and weight decay of 0.01, learning rate 0.001), a $n_b = 32$ (64 for CINIC-10) $n_B = 320$ (640 for CINIC-10), meaning we select $\frac{n_b}{n_B} = 10\%$ of points. *NLP*: ALBERT v2 was trained using the AdamW optimizer with a learning rate as indicated in the original paper ($2 \cdot 10^{-5}$) and weight decay of 0.02. We finetuned all weights, not just the final layer. the batch size n_b was 32, $n_B = 320$, meaning we select $\frac{n_b}{n_B} = 10\%$ of points. We use between 2 and 10 seeds for each experiment.

Data augmentation. On CIFAR-10, CIFAR-100, and CINIC-10, we train using data augmentation (random crop and horizontal flip), both for training the IL model, and in the main training runs. Remember that we only compute the irreducible losses once at the start of training, to save compute (Algorithm 1). We use the un-augmented images for this as we found that using augmented images makes little difference to performance but costs more compute.

Irreducible loss model training. The irreducible loss models are trained on holdout sets (not test sets, see dataset description in main text). For each dataset, we select the irreducible loss model checkpoint from the epoch with *lowest holdout loss* on $\mathcal{D}$ (as opposed to highest accuracy); we find that this improves performance while also saving compute as the holdout loss typically reaches its minimum early in training.

BatchNorm. Like many deep-learning methods, RHO-LOSS interacts with BatchNorm (Ioffe & Szegedy, 2015) since the loss of a given point is affected by other points in the same batch. **Important:** We compute the BatchNorm statistics for selection and model update separately. For selection (line 5-8 in Algorithm 1), the statistics are computed across the large batch B_t. For training (line 9-10), the statistics are computed across the small batch b_t. These choices can affect performance a lot. For new datasets, we recommend to vary how the batchnorm statistics are computed during selection (trying both train mode and eval mode) and choose the option that works best.

C. Robustness to Noise

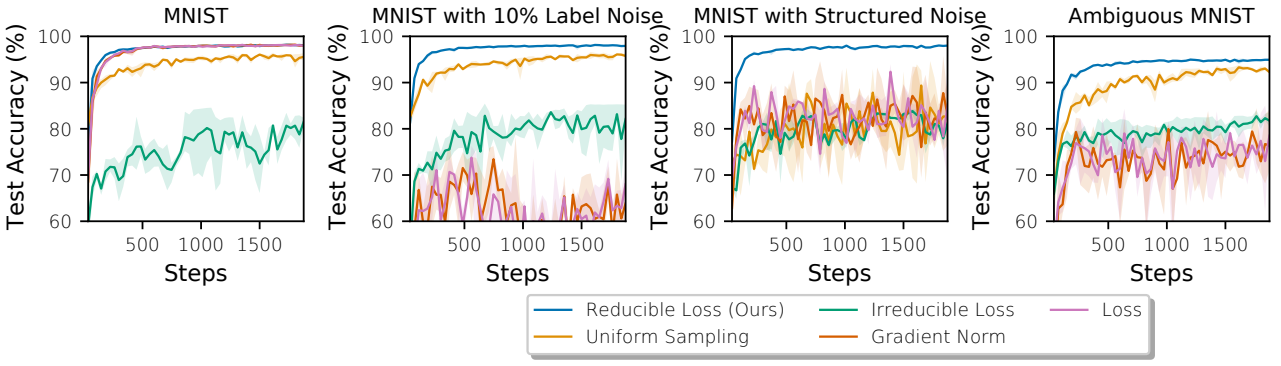

Figure 6: RHO-LOSS is robust to a variety of label noise patterns, while other selection methods degrade. A step corresponds to lines $6-11$ in Algorithm 1. Lines correspond to means and shaded areas to minima and maxima across 3 random seeds.

In this set of experiments, we evaluate the performance of different selection methods under a variety of noise patterns on QMNIST (MNIST with extra holdout data) and variations thereof. We use this dataset because it has little label noise in its original form, allowing us to test the effect of adding noise. Firstly, we add uniform label noise to 10% of training points. Secondly, we add structured label noise that affects easily confused classes. We follow (Rolnick et al., 2017) and flip the labels of the four most frequently confused classes (in the confusion matrix of a trained model) with 50% probability. For example, a 2 is often confused with a 5; thus we change the label of all 2s to 5s with 50% probability. Thirdly, we leverage the natural noise distribution of MNIST by using AmbiguousMNIST (Mukhoti et al., 2021) as the training set. AmbiguousMNIST contains a training set with 60k generated ambiguous digits that have more than one plausible label. While selecting with loss and gradient norm trains accelerates training on the MNIST training set, their performance degrades on all three types of noise distributions (Figure 6).

D. Irreducible Holdout Loss Approximation

In this appendix section, we examine one of the key approximations made in the theory section. To arrive at Eq. (3), we used the approximation $L[y \mid x; \mathcal{D}_{\text{ho}}] \approx L[y \mid x; \mathcal{D}_{\text{ho}}, \mathcal{D}_{\text{t}}]$. In words, we approximated the cross-entropy loss of a model trained on the data points acquired so far $\mathcal{D}_{\text{t}}$ and the holdout dataset $\mathcal{D}_{\text{ho}}$, with the cross-entropy loss of a model trained only on the holdout set. This approximation saves a lot of compute: rather than having to recompute the term with every change of $\mathcal{D}_t$, it is now sufficient to compute it once at the start of training.

We have already highlighted the impact of the approximation on points selected when training on QMNIST in Section 4.1. In our main experiment setting—using neural networks trained with gradient descent—we empirically find that the approximation does not reduce speed of target model training or final target model accuracy (Table 4). This finding holds across a range of datasets (CIFAR-10, CIFAR-100, CINIC-10). Updating the irreducible loss model on $\mathcal{D}_{\text{t}}$ seems empirically not necessary.

Indeed, the approximation actually has two desirable properties when used for neural networks trained with gradient descent. We will first describe why we expect these desirable properties, and then show that they indeed appear.

First, let us restate both selection functions:

Original selection function:

$$\arg\max_{(x,y)\in B_t} L[y \mid x; \mathcal{D}_{\text{t}}] - L[y \mid x; \mathcal{D}_{\text{ho}}, \mathcal{D}_{\text{t}}].$$

Approximated selection function:

$$\arg\max_{(x,y)\in B_t} L[y \mid x; \mathcal{D}_{\text{t}}] - L[y \mid x; \mathcal{D}_{\text{ho}}].$$

Desirable property 1: The approximation prevents repeated selection of undesirable points. When using SGD instead of Bayesian updating, the original selection function can acquire undesired points repeatedly. Let's say that we

Table 4: Number of epochs required to reach a given target test accuracy across several datasets. Results averaged across 2-3 random seeds. NR indicates that the target accuracy was not reached.

Dataset	Target acc	approximated selection function $L[y \mid x; \mathcal{D}_{\text{t}}] - L[y \mid x; \mathcal{D}_{\text{ho}}]$	original selection function $L[y \mid x; \mathcal{D}_{\text{t}}] - L[y \mid x; \mathcal{D}_{\text{ho}}, \mathcal{D}_{\text{t}}]$
CIFAR10	60%	18	13
	75%	30	24
	90%	102	NR, but reaches 88% in 157 epochs
CIFAR100	30%	35	21
	45%	58	NR, but reaches 43% in 61 epochs
	60%	123	NR
CINIC10	55%	12	12
	65%	19	21
	75%	32	NR, but reaches 74% in 68 epochs

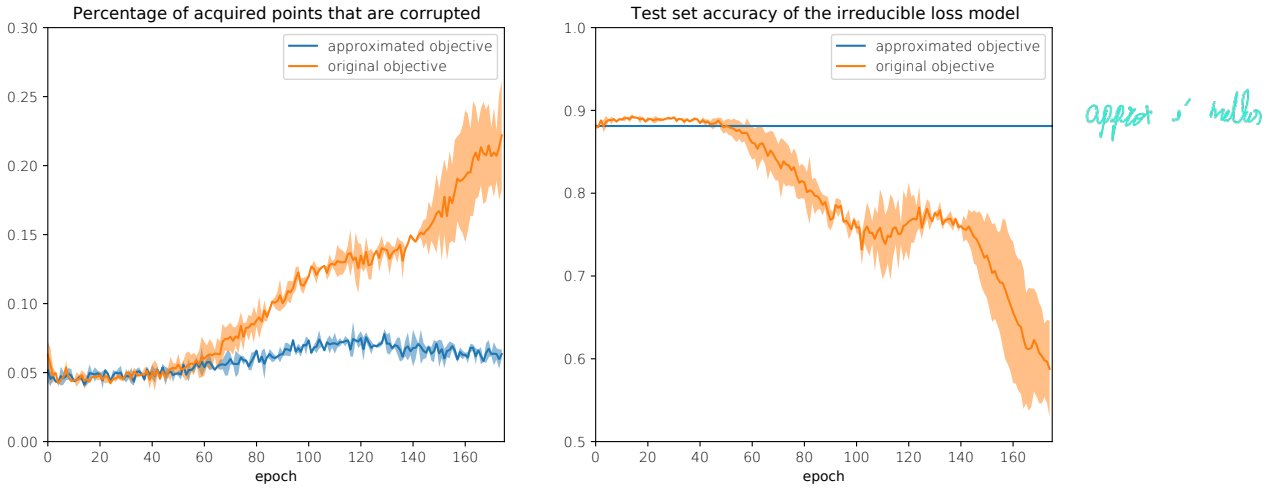

Figure 7: Desired properties of the irreducible loss model approximation. **Left.** The approximated selection function selects fewer corrupted points later on in training. **Right.** The test set accuracy of the irreducible loss model deteriorates over time if it is updated on $\mathcal{D}_t$. With the approximation, the irreducible loss is not updated during target model training. Results on CIFAR-10 with 20% of data points corrupted with uniform label noise. Shaded areas represent standard deviation across three different random seeds.

acquire, for whatever reason, a noisy, redundant, or irrelevant point. We only take one gradient step each time we acquire a (batch of) point(s), meaning the training loss (first term in the selection function) will on each only decrease somewhat. In the original selection function, the second term will also decrease somewhat, meaning that the difference between the first and second term may remain large. In the approximated selection function, the second term is constant, the difference between first and second term will thus likely decrease more than under the original selection function. Under the approximated selection function, we are thus less likely to acquire undesired points again, if we have acquired them in earlier epochs.

Desirable property 2: The approximation prevents deterioration of the irreducible loss model over time. With both selection functions, we compute the second term of the selection function with an "irreducible loss model", which we train on a holdout set before we start target model training. In the target model training, we (greedily) acquire the points that most improve the loss of the target model (on the holdout set). We thus deliberately introduce bias into the data selection. However, this bias is tailored to the target model and may not be suitable for the irreducible loss model. As a simplifying example, consider a target model early in training, which has not yet learnt a certain class, and an irreducible loss model, which has learnt that class. Data points in that class will have high training loss, low irreducible loss, and will be acquired often. This, however, is not useful for the irreducible loss model, and might lead to decreased accuracy on data points from other classes. With the approximation, this can't happen. The described failure mode could likely also be alleviated by more sophisticated training schemes for the irreducible loss model, such as periodically mixing in data points from the holdout set. However, such training schemes would require even more compute and/or overhead.
We find empirically that both desired properties of the approximation indeed manifest themselves. In Fig. 7, we train a target model (Resnet-18) on CIFAR-10, with 20% of the data points corrupted by uniform label noise. The approximated selection function leads to faster target model training (the approximated selection function needs 80 epochs to reach the same target model accuracy that the original selection function reaches in 100 epochs) and higher final accuracy than the original selection function (88.6% vs 86.1%). Indeed, the original selection function leads to acquiring more corrupted points, especially later in training (Fig. 7, left), and the accuracy *of the irreducible loss model* deteriorates over time (Fig. 7, right). We tuned the learning rate of the irreducible loss model to 0.01 times that of the target model. Without this adjustment, the results look similar but the original selection function performs worse.

E. Experimental Details for Assessing Impact of Approximations

Dataset: QMNIST, with uniform label noise applied to 10% of the dataset. Batch size of 1000 is used.

Models: Deep Ensemble contains 5 3-layer MLP's with 512 hidden units. The weaker irreducible loss model is an MLP with 256 hidden units.

Training: For Approximation 0, we use a deep ensemble for both models. The irreducible loss model is trained to convergence on $\mathcal{D}_{\text{ho}}$. Then the target model and the irreducible model are used to acquire 10% of points each batch using the selection function. They are then trained to convergence on each batch of points acquired. The irreducible loss model is trained on $\mathcal{D}_{\text{ho}} \cup \mathcal{D}_{\text{t}}$, while the target model is only trained on $\mathcal{D}_{\text{t}}$. We train for a maximum of 5 epochs, which often is to convergence, to enable a fair comparison to further approximations. For Approximation 1a, the deep ensembles are replaced with single MLPs. The training regime remains the same. We compare the approximations over the first epoch. To compare Approximation 1b to 0, and for all further approximations, we increase the size of the dataset five-fold, by duplicating samples in QMNIST. This means for approximation 1b, we have 5x the data that we have for Approximation 1a, but with increased redundancy. We train the model in Approximation 1b by taking a single gradient step per datapoint, with the larger dataset. On the other hand, we train the model for Approximation 0 (still to convergence or 5 epochs) on the standard dataset size. By doing this, Approximation 0 and 1b have taken the equivalent number of gradient steps, at the time-steps where we are tracking the reducible loss of points selected, enabling a fair comparison between the approximations. The irreducible loss models are trained on $\mathcal{D}_{\text{ho}} \cup \mathcal{D}_{\text{t}}$ in their respective set-ups. To compare Approximation 2 to Approximation 0, we compare updating the irreducible loss model with a single gradient on each set of acquired points, to not updating the irreducible loss model on $\mathcal{D}_{\text{t}}$ at all. To isolate effect of not updating, we utilise the same initial irreducible loss model. To compare Approximation 3, we simply train a small irreducible model (one with 256 hidden units) and follow the same training regime as Approximation 2.

F. Ablation of percentage selected

Our method has a hyperparameter, the percentage $\frac{n_b}{n_B}$ of evaluated points which are selected for training. In the experiments above, this parameter was set to 0.1. We have not tuned this parameter, as we aim to analyse how well our method works "out of the box". In fact, on 2/3 datasets, performance further improves with other values of this parameter. Adjusting this percentage should allow practitioners to specify their preferred tradeoff between training time and computation, where a low percentage typically corresponds to a lower training time and greater compute cost. For these experiments, we kept $n_b = 32$ and adapt n_B accordingly. The percentage $\frac{n_b}{n_B}$ of datapoints selected per batch has different effects across datasets as shown in Fig. 8.

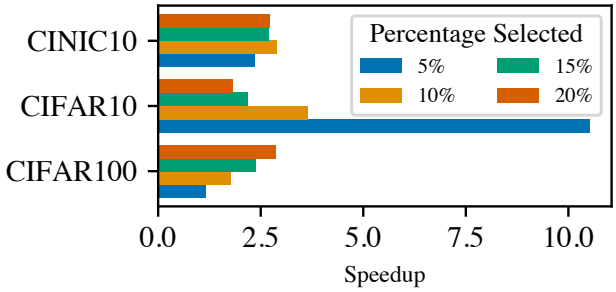

Figure 8: Varying the percent of data points selected in each training batch. Average over 3 random seeds.

G. Active Learning Baselines

We compare our method to typical methods used in the Active Learning (AL) literature. Note that our method is label-aware, while active learning acquires datapoints without using label information. We consider the following baselines, which select the top-k points using an acquisiton function, $\alpha(x)$:

- Bayesian Active Learning by Disagreement (Houlsby et al., 2011) with $\alpha(x) = \mathrm{H}[y \mid x, \mathcal{D}_t] - \mathbb{E}_{\mathrm{p}(\theta \mid \mathcal{D}_t)} [\mathrm{H}[y \mid x, \theta]]$.

- (Average) conditional entropy, $\alpha(x) = \mathbb{E}_{\mathrm{p}(\theta|\mathcal{D}_t)}\left[\mathrm{H}[y \mid x, \theta]\right]$, where the average is taken over the model parameter posterior.
- (Average predictive) entropy, $\alpha(x) = \mathrm{H}[y \mid x, \mathcal{D}_t]$.
- Loss minus conditional entropy $\alpha(x) = L[y \mid x, \theta] - \mathbb{E}_{\mathrm{p}(\theta|\mathcal{D}_t)}\left[\mathrm{H}[y \mid x, \theta]\right]$. This uses the (average) conditional entropy as an estimate of how noisy datapoint x is—points with high noise are deprioritized. Compared to RHO-LOSS, it replaces the IL with the conditional entropy. This acquisition function uses the label and therefore cannot be used for active learning.

We additionally compare our method to uniform sampling. We run all baselines on MNIST and CIFAR10. Note that several of these active learning baselines consider epistemic uncertainty; that is, uncertainty in predictions driven by uncertainty in the model parameters. This mandates performing (approximate) Bayesian inference. We use Monte-Carlo Dropout(Gal & Ghahramani, 2016) to perform approximate inference. For MNIST, we use an 2 hidden layer MLP with 512 hidden units per hidden layer, and a dropout probability of a 0.5. For experiments on CIFAR10, we use a small-scale CNN with dropout probability 0.05 (the dropout probability follows (Osawa et al., 2019)).

Fig. 9 shows training curves for our method, uniform sampling, and the active learning baselines. Our method accelerates training across both datasets. The active learning methods accelerate training for MNIST but not for CIFAR10. This highlights that active learning methods, if naively applied to online batch selection, may not accelerate model training.

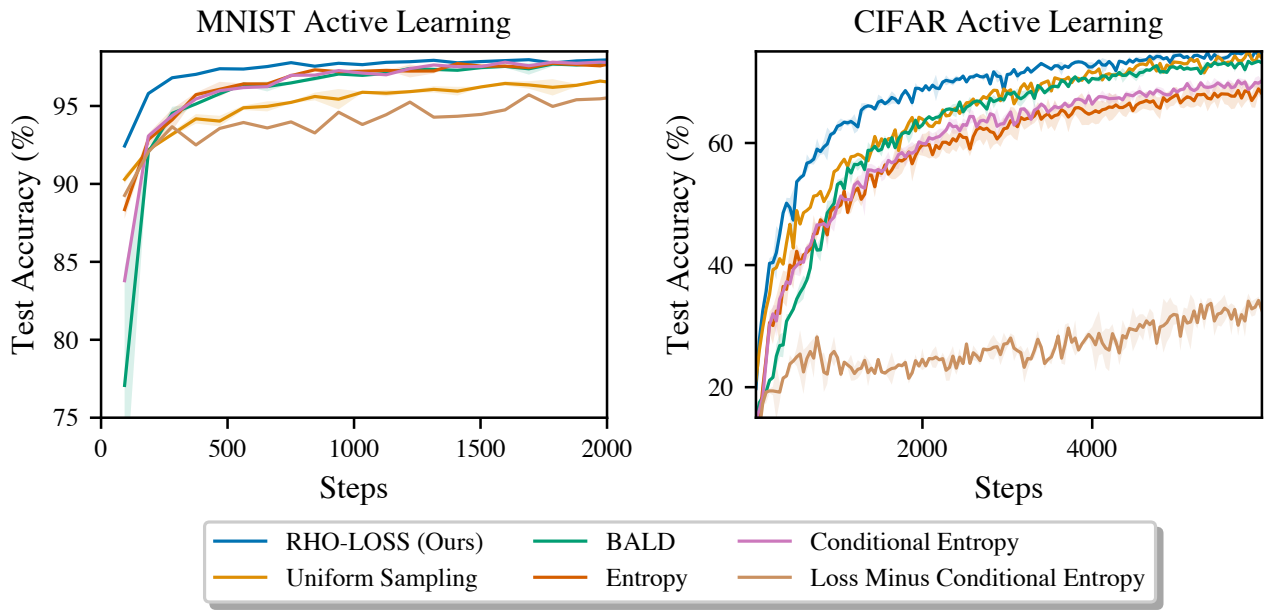

Figure 9: Training curves for several active learning baselines on the MNIST and CIFAR10 datasets.