Carpe Diem, Seize the Samples Uncertain "at the Moment" for Adaptive Batch Selection

Hwanjun Song, Minseok Kim, Sundong Kim, Jae-Gil Lee
Graduate School of Knowledge Service Engineering, KAIST, Korea

ABSTRACT

The accuracy of deep neural networks is significantly affected by how well mini-batches are constructed during the training step. In this paper, we propose a novel adaptive batch selection algorithm called **Recency Bias** that exploits the uncertain samples predicted inconsistently *in recent iterations*. The historical label predictions of each training sample are used to evaluate its predictive uncertainty within a *sliding window*. Then, the sampling probability for the next mini-batch is assigned to each training sample in proportion to its predictive uncertainty. By taking advantage of this design, *Recency Bias* not only accelerates the training step but also achieves a more accurate network. We demonstrate the superiority of *Recency Bias* by extensive evaluation on two independent tasks. Compared with existing batch selection methods, the results showed that *Recency Bias* reduced the test error by up to 20.97% in a fixed wall-clock training time. At the same time, it improved the training time by up to 59.32% to reach the same test error.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Supervised learning by classification**.

KEYWORDS

Batch Selection, Uncertain Sample, Acceleration, Convergence

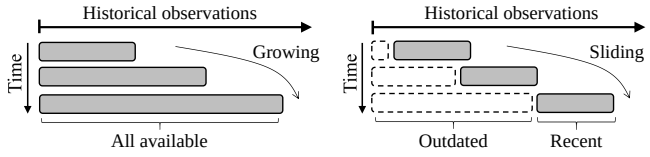
ACM Reference Format:
Hwanjun Song, Minseok Kim, Sundong Kim, Jae-Gil Lee. 2020. Carpe Diem, Seize the Samples Uncertain "at the Moment" for Adaptive Batch Selection. In *CIKM '20: ACM International Conference on Information and Knowledge Management, October 19–23, 2020, Online.* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

Deep neural networks (DNNs) have become one of the most popular methods for supervised learning tasks in that traditional machine learning is successfully superseded by recent deep learning in many applications [5, 20]. However, in return for this higher popularity and capability, the training of DNNs raises many challenges mainly because of their very high expressive power as well as complex structure. Thus, in this paper, we address an important issue in the training of DNNs, namely *batch selection*.

CIKM '20, October 19–23, 2020, Online

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00



(a) Growing Window. (b) Sliding Window.

Figure 1: Two forms of handling the time-series observations.

Stochastic gradient descent (SGD) for *randomly* selected mini-batch samples is commonly used to train DNNs. However, many recent studies have pointed out that the performance of DNNs is heavily dependent on how well the mini-batch samples are selected [3, 13, 27]. In earlier approaches, a sample's *difficulty* is employed to identify proper mini-batch samples, and these approaches achieve a more accurate and robust network [10] or expedite the training convergence of SGD [19]. However, the two opposing difficulty-based strategies, i.e., preferring *easy* samples [10, 16] versus *hard* samples [19, 27], work well in different situations. The former results in a network robust to outliers and noisy labels, but slows down the training process by causing small gradients [21, 26]; the latter accelerates the training process, but leads to poor generalization on test data by exacerbating the overfitting [19]. Thus, for practical reasons to cover more diverse situations, recent approaches begin to exploit a sample's *uncertainty* that indicates the consistency of previous predictions [3, 28, 29].

An important question here is how to evaluate a sample's uncertainty based on its historical predictions during the training process. Intuitively, because a series of historical predictions can be seen as a series of data indexed in chronological order, the uncertainty can be measured based on *two* forms of handling time-series observations: *(i)* a *growing window* (Figure 1(a)) that consistently increases the size of a window to use all available observations and *(ii)* a *sliding window* (Figure 1(b)) that maintains a window of a fixed size on the most recent observations by deleting outdated ones. While the state-of-the-art algorithm, *Active Bias* [3], adopts the growing window, we propose to use the sliding window in this paper.

In more detail, *Active Bias* recognizes uncertain samples based on the inconsistency of the predictions in the *entire* history of past SGD iterations. Then, it emphasizes such uncertain samples by choosing them with high probability for the next mini-batch. However, according to our experiments presented in Section 4.2, such uncertain samples *slowed down* the convergence speed of training, though they ultimately reduced the generalization error. This weakness is attributed to the inherent limitation of the growing window, where older observations could be too outdated [30]. In other words, the outdated predictions no longer represent a network's current behavior. As illustrated in Figure 2, when the label predictions of two samples were inconsistent for a long time, *Active Bias* invariably regards them as highly uncertain, although their recent label

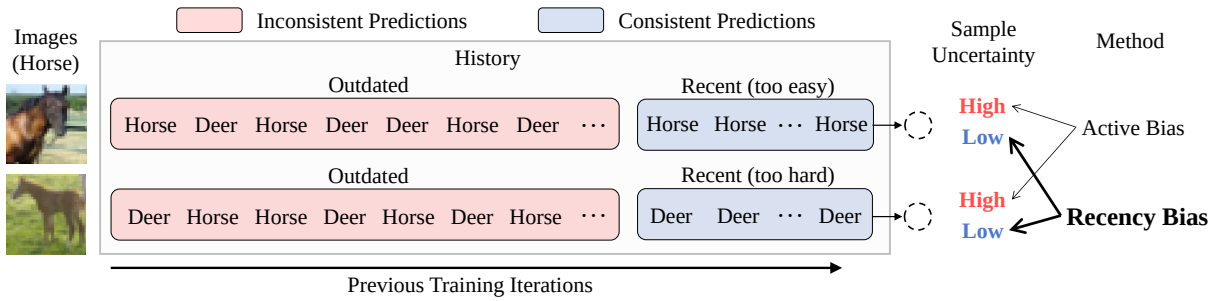


Figure 2: The difference in sample uncertainty estimated by *Active Bias* and *Recency Bias*.

predictions become consistent along with the network's training progress. This aspect evidently entails the risk of emphasizing uninformative samples—too easy or too hard—at the current moment, thereby slowing down the convergence speed of training.

Therefore, we propose a simple but effective batch selection method, called **Recency Bias**, that takes advantage of the *sliding window* to evaluate the uncertainty in *fresher* observations. As opposed to *Active Bias*, *Recency Bias* excludes the outdated predictions by managing a sliding window of a fixed size and picks up the samples predicted inconsistently within the sliding window. Thus, as shown in Figure 2, the two samples uninformative at the moment are no longer selected by *Recency Bias* simply because their recent predictions are consistent. Consequently, since informative samples are effectively selected throughout the training process, this strategy not only accelerates the training speed but also leads to a more accurate network.

To validate the superiority of *Recency Bias*, two popular convolutional neural networks (CNNs)[1] were trained for two independent tasks: image classification and fine tuning. We compared *Recency Bias* with not only the random batch selection (baseline) but also *Online Batch* and *Active Bias*, which are the two state-of-the-art adaptive batch selection methods. Our extensive experiments empirically confirmed the following promising results:

- **Uncertainty Estimation**: *Recency Bias* benefited from using the sliding window in estimating the uncertainty. It evaluated truly uncertain samples during the entire training process, while the growing window approach (*Active Bias*) misclassified many easy samples as uncertain at a late stage of training.
- **Accuracy and Efficiency**: *Recency Bias* provided a relative reduction in test error by up to 20.97% in a fixed wall-clock training time compared with three batch selection strategies. At the same time, it significantly reduced the execution time by up to 59.32% to reach the same test error.
- **Practicality**: The performance dominance of *Recency Bias* was consistent on two learning tasks based on various benchmark datasets. The impact on both convergence speed and generalization capability have a great potential to improve many other deep learning tasks.

In the rest of the paper, Section 2 reviews related work, Section 3 proposes the *Recency Bias* algorithm, Section 4 presents the experiments, and Section 5 concludes the paper.

[1]The idea is applicable to the DNNs other than CNNs, and we leave this extension as the future work.

2 RELATED WORK

SGD is a popular optimization method, which has been extensively studied in the machine learning community [11, 18, 25]. Typically, at each training iteration, more than one training samples called a *mini-batch* is constructed from the training dataset and then employed to update the model parameter such that this parameter minimizes the empirical risk on the mini-batch. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be the entire training dataset composed of a sample x_i with its true label y_i, where N is the total number of training samples. Then, a straightforward strategy to construct a mini-batch $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{b}$ is to select b samples *uniformly at random* (i.e., $P(x_i|\mathcal{D}) = 1/N$) from the training dataset $\mathcal{D}$.

Because not all samples have an equal impact on training, many research efforts have been devoted to develop advanced *sampling schemes*. Bengio et al. [1] first took easy samples and then gradually increased the difficulty of samples using heuristic rules. Kumar et al. [16] determined the easiness of the samples using their prediction errors. Recently, Tsvetkov et al. [31] used Bayesian optimization to learn an optimal curriculum for training dense, distributed word representations. Sachan and Xing [24] emphasized that the right curriculum must introduce a small number of the samples dissimilar to those previously seen. Fan et al. [7] proposed a neural data filter based on reinforcement learning to select training samples adaptively. However, it is common for deep learning to emphasize *hard* samples because of the plethora of easy ones [13].

In light of this, Loshchilov and Hutter [19] proposed a *difficulty*-based sampling scheme, called *Online Batch*, that uses the rank of the loss computed from previous epochs. *Online Batch* sorts the previously computed losses of samples in descending order and exponentially decays the sampling probability of a sample according to its rank r. Then, the r-th ranked sample $x(r)$ is selected with the probability dropping by a factor of $\exp\left(\log(s_e)/N\right)$, where s_e is the *selection pressure* parameter that affects the probability gap between the most and the least important samples. When normalized to sum to 1.0, the probability $P(x(r)|\mathcal{D}; s_e)$ is defined by

$$P(x(r)|\mathcal{D}; s_e) = \frac{1/\exp\left(\log(s_e)/N\right)^r}{\sum_{j=1}^{N} 1/\exp\left(\log(s_e)/N\right)^j}. \quad (1)$$

However, it has been reported that *Online Batch* accelerates the convergence of training loss but deteriorates the generalization on test data because of the overfitting to hard training samples [19]. Thus, as confirmed in our experiment in Section 4.2, it only works well for easy datasets, where hard training samples rarely exist.

Most close to our work, Chang et al. [3] devised an *uncertainty*-based sampling scheme, called *Active Bias*, that chooses uncertain samples with high probability for the next batch. *Active Bias* maintains the history $\mathcal{H}_i^{t-1}$ that stores *all* $h(y_i|x_i)$ before the current iteration t (i.e., growing window), where $h(y_i|x_i)$ is the softmax probability of a given sample x_i for its true label y_i. Then, it measures the uncertainty of the sample x_i by computing the variance over *all* $h(y_i|x_i)$ in $\mathcal{H}_i^{t-1}$ and draws the next mini-batch samples based on the normalized probability $P(x_i|\mathcal{D}, \mathcal{H}_i^{t-1}; \epsilon)$ defined by

$$P(x_i|\mathcal{D}, \mathcal{H}_i^{t-1}; \epsilon) = \frac{\hat{std}(\mathcal{H}_i^{t-1}) + \epsilon}{\sum_{j=1}^{N} \big(\hat{std}(\mathcal{H}_j^{t-1}) + \epsilon\big)}, \quad \text{where } \hat{std}(\mathcal{H}_i^{t-1}) = \sqrt{var\big(h(y_i|x_i)\big) + \frac{var\big(h(y_i|x_i)\big)^2}{|\mathcal{H}_i^{t-1}| - 1}} \tag{2}$$

and ϵ is a smoothness constant to prevent the low variance samples from never being selected again. As mentioned earlier in Introduction, *Active Bias* slows down the training process compared to random batch selection because the oldest part in the history $\mathcal{H}_i^{t-1}$ no longer represents the current behavior of the network.

For the completeness of the survey, we include the recent studies on submodular batch selection. Joseph et al. [12] and Wang et al. [32] designed their own submodular objectives that cover diverse aspects, such as sample redundancy and sample representativeness, for more effective batch selection. Differently from their work, we explore the issue of truly uncertain samples in an orthogonal perspective. Our uncertainty measure can be easily injected into their submodular optimization framework as a measure of sample informativeness.

3 BATCH SELECTION VIA *RECENCY BIAS*

3.1 Problem Setting and Overview

In the standard training of DNNs by SGD, the parameter θ of the network is updated according to the descent direction of the expected empirical risk on the given mini-batch $\mathcal{B}$,

$$\theta_{t+1} = \theta_t - \eta \nabla \Big(\frac{1}{|\mathcal{B}|} \sum_{x_i \in \mathcal{B}} f_{x_i}(\theta_t) \Big), \tag{3}$$

where η is the given learning rate and $f_{x_i}(\theta_t)$ is the empirical risk of the sample x_i for the network parameterized by θ_t.

In this study, we modify the update rule to highlight uncertain samples during the training process. First, *Recency Bias* manages a sliding window of a fixed size, called the *label history* $\mathcal{H}$, which stores recent label predictions of all training samples. Second, *Recency Bias* evaluates the sample uncertainty based on the label history and then builds a sampling distribution $P(x|\mathcal{D}, \mathcal{H}; s_e)$ such that uncertain samples exhibit higher probabilities, where s_e is the selection pressure parameter. Let $\mathcal{U}$ be the uncertain mini-batch samples drawn according to the sampling distribution built at the current moment. Then, the modified update rule is

$$\theta_{t+1} = \theta_t - \eta \nabla \Big(\frac{1}{|\mathcal{U}|} \sum_{x_i \in \mathcal{U}} f_{x_i}(\theta_t) \Big), \text{where } \mathcal{U} \sim P(x|\mathcal{D}, \mathcal{H}; s_e). \tag{4}$$

Please note that *Recency Bias* iteratively refines the sampling distribution by using the latest label history *once per epoch* because an epoch is a widely-used learning cycle to measure the model changes [8, 19]. To update the network by Eq. (4), the main challenge is how to evaluate the uncertainty of training samples based on their label histories as well as how to assign the sampling probability for them to construct the next mini-batch. The two main challenges are detailed in the following section.

3.2 Batch Selection Methodology

3.2.1 Criterion of an Uncertain Sample. Intuitively, samples are uncertain if their *recent* label predictions are highly inconsistent because they are neither too easy nor too hard at the moment. Thus, we adopt the *predictive uncertainty* [28] in Definition 3.1 that uses the information entropy [2] to measure the inconsistency of recent label predictions. Here, the sample with high predictive uncertainty is regarded as uncertain and thus should be selected with high probability for the next mini-batch.

Definition 3.1. **(Predictive Uncertainty)** Given a neural network Φ parameterized by θ, let $\hat{y}_t = \Phi(x_i; \theta_t)$ be the predicted label of a sample x_i at time t and $\mathcal{H}_i(q) = \{\hat{y}_{t_1}, \hat{y}_{t_2}, \ldots, \hat{y}_{t_q}\}$ be the label history of the sample x_i that stores the predicted labels at the previous q times. The label history $\mathcal{H}_i(q)$ corresponds to the *sliding window* of size q to compute the uncertainty of the sample x_i. Next, based on $\mathcal{H}_i(q)$, the probability of the j-th label ($j \in \{1, 2, \ldots, k\}$) estimated as the label of the sample x_i is formulated by

$$p(y_i = j|x_i) = \frac{\sum_{\hat{y} \in \mathcal{H}_i(q)} [\hat{y} = j]}{|\mathcal{H}_i(q)|}, \tag{5}$$

where $[\cdot]$ is the Iverson bracket[2]. Then, to quantify the uncertainty of the sample x_i, the empirical entropy is used to define the *predictive uncertainty* $U(x_i)$ by

$$U(x_i) = \frac{1}{\log(k)} entropy\big(p(y_i|x_i)\big), \quad \text{where } entropy\big(p(y_i|x_i)\big) = -\sum_{j=1}^{k} p(y_i = j|x_i) \log p(y_i = j|x_i). \tag{6}$$

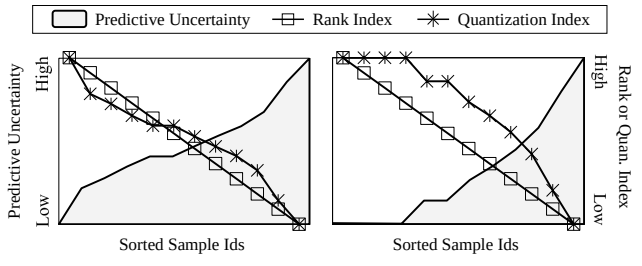
For k classes, to ensure $0 \le U(x_i) \le 1$, the empirical entropy is divided by $\log(k)$, which is the maximum value obtained when $\forall_{j \in \{1,\ldots,k\}} p(y_i = j|x_i) = 1/k$.

3.2.2 Sampling Probability for Mini-batch Construction. To construct next mini-batch samples, we assign the sampling probability according to the predictive uncertainty in Definition 3.1. Motivated by Loshchilov and Hutter [19], the sampling probability of a given sample x_i is exponentially decayed with its predictive uncertainty $U(x_i)$. In detail, we adopt the quantization method [4] and use the quantization index to decay the sampling probability. The index is obtained using the simple quantizer $Q(z)$ defined by

$$Q(z) = \lceil (1 - z)/\Delta \rceil, \ 0 \le z \le 1, \tag{7}$$

where Δ is the quantization step size. Compared with the rank-based index [19], the quantization index is known to well reflect the

[2]The Iverson bracket $[p]$ returns 1 if p is true; 0 otherwise.



(a) Early Stage (30%). (b) Later Stage (70%).

Figure 3: Comparison of the index computed by the rank-based and quantization-based approaches: (a) and (b) show the predictive uncertainty of training samples in CIFAR-100 with their indexes at the 30% and 70% of total training epochs, respectively.

difference in actual values [33]. Comparing two different epochs in Figure 3, although the predictive uncertainty of the samples are changed between the two epochs, the rank-based approach ignores this change by assigning a fixed index for the same rank, whereas our quantization approach assigns different indexes every moment by reflecting the change in actual values.

In Eq. (7), we set Δ to be $1/N$ such that the index is bounded to N (the total number of samples). Then, the sampling probability $P(x_i|\mathcal{D},\mathcal{H};s_e)$ is defined by

$$P(x_i|\mathcal{D},\mathcal{H};s_e) = \frac{1/\exp\big(\log(s_e)/N\big)^{Q(U(x_i))}}{\sum_{j=1}^{N} 1/\exp\big(\log(s_e)/N\big)^{Q(U(x_j))}}. \quad (8)$$

The higher the predictive uncertainty, the smaller the quantization index. Therefore, a higher sampling probability is assigned for uncertain samples by Eq. (8).

Meanwhile, it is known that using only some part of training data exacerbates the overfitting problem at a late stage of training [19, 35]. Thus, to alleviate the problem, we include more training samples as the training progresses by exponentially decaying the selection pressure s_e by

$$s_e = s_{e_0}\Big(\exp\big(\log(1/s_{e_0})/(e_{end}-e_0)\big)\Big)^{e-e_0}. \quad (9)$$

At each epoch e from e_0 to e_{end}, the selection pressure s_e exponentially decreases from s_{e_0} to 1. Because this technique gradually reduces the sampling probability gap between the most and the least uncertain samples, more diverse samples are selected for the next mini-batch at a later epoch. When the selection pressure s_e becomes 1, the mini-batch samples are randomly chosen from the entire dataset. The ablation study on the effect of decaying the selection pressure is presented in Section 4.4.

3.3 Convergence Guarantee

All adaptive batch selection strategies are widely known to follow the typical convergence guarantee of the vanilla SGD [23], as long as their sampling distributions are *strictly positive* and their gradient estimates are *unbiased* [9, 34]. Hence, we provide the theoretical evidence that *Recency Bias* satisfies the two aforementioned conditions for the convergence guarantee.

Lemma 3.2. *Let $P(x|\mathcal{D},\mathcal{H};s_e)$ be the sampling distribution of Recency Bias. Then, it is a strictly positive distribution.*

Proof. By Eq. (6) and Eq. (7), the index $Q(U(x))$ is bounded by

$$0 \le Q(U(x)) \le \lceil 1/\Delta \rceil. \quad (10)$$

Then, the lower bound of $P(x|\mathcal{D},\mathcal{H};s_e)$ is formulated by

$$0 < \frac{1}{\sum_{j=1}^{N} 1/\exp\big(\log(s_e)/N\big)^{Q(U(x_j))}} \le P(x|\mathcal{D},\mathcal{H};s_e). \quad (11)$$

Thus, the sampling distribution of *Recency Bias* is strictly positive because $P(x_i|\mathcal{D},\mathcal{H};s_e) > 0$ for all $x_i \in \mathcal{D}$. □

Lemma 3.3. *Let $\tilde{G}$ be the gradient estimate of Recency Bias. Then, $\tilde{G}$ is an unbiased estimator.*

Proof. Let $f_{x_i}(\theta)$ be the empirical risk of the sample x_i for the given network parameterized by θ. Then, because the mini-batch samples in *Recency Bias* are drawn according to $P(x|\mathcal{D},\mathcal{H};s_e)$, the gradient estimate $\tilde{G}$ is equivalent to that of the weighted empirical risk on the mini-batch $\mathcal{B}$ randomly drawn from $\mathcal{D}$ [9],

$$\tilde{G} = \nabla \sum_{x_i\in\mathcal{B}} w(x_i) f_{x_i}(\theta),\ w(x_i) = \frac{P(x_i|\mathcal{D},\mathcal{H};s_e)}{\sum_{x_j\in\mathcal{B}} P(x_j|\mathcal{D},\mathcal{H};s_e)}. \quad (12)$$

Here, $w(x_i)$ is a constant value by Eq. (8), and therefore the estimate $\tilde{G}$ is rewritten as

$$\tilde{G} = \sum_{x_i\in\mathcal{B}} w(x_i)\nabla f_{x_i}(\theta),\ \text{ where } \sum_{x_i\in\mathcal{B}} w(x_i) = 1. \quad (13)$$

That is, $\tilde{G}$ is the weighted combination of the gradient estimate $\nabla f_{x_i}(\theta)$ of each training sample x_i, which is an unbiased estimate of the true gradient,

$$\mathbb{E}[\nabla f_{x_i}(\theta)] = \sum_{x_i\in\mathcal{D}} \frac{1}{|\mathcal{D}|}\nabla f_{x_i}(\theta) = \frac{1}{|\mathcal{D}|}\nabla \sum_{x_i\in\mathcal{D}} f_{x_i}(\theta). \quad (14)$$

Thus, the gradient estimate of *Recency Bias* naturally becomes an unbiased estimator due to the linearity of expectation. □

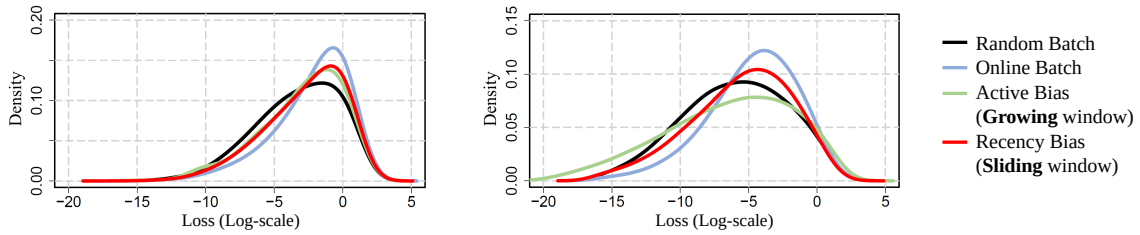
Theorem 3.4. *The SGD by Recency Bias is guaranteed to converge. More specifically, supposing that $\theta_* = \text{argmin}_\theta f_{\mathcal{D}}(\theta)$, where $f_{\mathcal{D}}(\theta)$ is the empirical risk of the model parameterized by θ for all $x_i \in \mathcal{D}$, the empirical risk of Recency Bias at iteration t satisfies the convergence rate of $O(1/\sqrt{t})$ [25],*

$$\mathbb{E}[f_{\mathcal{D}}(\theta_t)] - f_{\mathcal{D}}(\theta_*) = O(1/\sqrt{t}). \quad (15)$$

Proof. Done by Lemma 3.2 and Lemma 3.3. □

3.4 Algorithm Pseudocode

Algorithm 1 describes the overall procedure of *Recency Bias*. The algorithm requires a warm-up period of γ epochs because the quantization index for each sample is not confirmed yet. During the warm-up period, which should be at least q epochs ($\gamma \ge q$) to obtain the label history of size q, randomly selected mini-batch samples are used for the network update (Lines 13–14). After the warm-up period, the algorithm decays the selection pressure s_e and updates not only the quantization index but also the sampling probability in a batch at the beginning of each epoch (Lines 5–10). Subsequently,



(a) Early Stage (30%). (b) Late Stage (70%).

Figure 4: The loss distribution of mini-batch samples selected by four batch selection methods: (a) and (b) show the loss distribution at the 30% and 70% of total training epochs, respectively.

Algorithm 1 *Recency Bias* Algorithm

INPUT: $\mathcal{D}$: data, *epochs*, b: batch size, q: window size, s_{e_0}: initial selection pressure, γ: warm-up
OUTPUT: θ_t: model parameter

1: $t \leftarrow 1$; $\theta_t \leftarrow$ Initialize the model parameter;
2: $\mathcal{H} \leftarrow$ Initialize the label history of size q;
3: **for** $i = 1$ **to** *epochs* **do**
4: /* **Sampling Probability Derivation** */
5: **if** $i > \gamma$ **then**
6: $s_e \leftarrow$ Decay_Pressure(s_{e_0}, i); /* Decaying s_e by Eq. (9) */
7: /* Update the index and sampling probability in a batch */
8: **for** $m = 1$ **to** N **do**
9: $q\_dict[x_m] = Q(U(x_m))$; /* By Eq. (7) */
10: $p\_table \leftarrow$ Compute_Prob(q_dict, s_e); /* By Eq. (8) */
11: /* **Network Training** */
12: **for** $j = 1$ **to** N/b **do**
13: **if** $i \leq \gamma$ **then** /* Warm-up */
14: $\{(x_l, y_l)\}_{l=1}^{b} \leftarrow$ Randomly select mini-batch samples;
15: **else** /* Adaptive batch selection */
16: $\{(x_l, y_l)\}_{l=1}^{b} \leftarrow$ Select mini-batch samples by p_table;
17: $losses, labels \leftarrow$ Inference($\{(x_l, y_l)\}_{l=1}^{b}, \theta_t$); /* Forward */
18: $\theta_{t+1} \leftarrow$ SGD($losses$, θ_t); /* Backward */
19: /* History update using a sliding window */
20: $\mathcal{H} \leftarrow$ Update_Label_History($\mathcal{H}, labels$);
21: $t \leftarrow t + 1$;
22: **return** θ_t;

the uncertain samples are selected for the next mini-batch according to the updated sampling probability (Lines 15–16), and then the label history is updated along with the network update (Lines 17–21). The entire procedure repeats for a given number of epochs.

Overall, the key technical novelty of *Recency Bias* is to incorporate the notion of a *sliding window* rather than a growing window into adaptive batch selection, thereby improving both training speed and generalization error.

Time Complexity: The main "additional" cost of *Recency Bias* is the derivation of the sampling probability for each sample (Lines 5–10). Because only simple mathematical operations are needed per sample, its time complexity is linear to the number of samples (i.e., $O(N)$), which is negligible compared with that of the forward and backward steps of a complex network (Lines 17–18). Therefore, we contend that *Recency Bias* does *not* add the complexity of an underlying optimization algorithm.

Table 1: Summary of datasets used for the two independent tasks in Sections 4.2 and 4.3.

Dataset	# Training	# Testing	# Classes	Resolution
MNIST [17]	60,000	10,000	10	28×28
CIFAR-10 [15]	50,000	10,000	10	32×32
CIFAR-100 [15]	50,000	10,000	100	32×32
MIT-67 [22]	5,360	1,340	67	256×256
FOOD-100 [14]	7,000	3,000	100	256×256

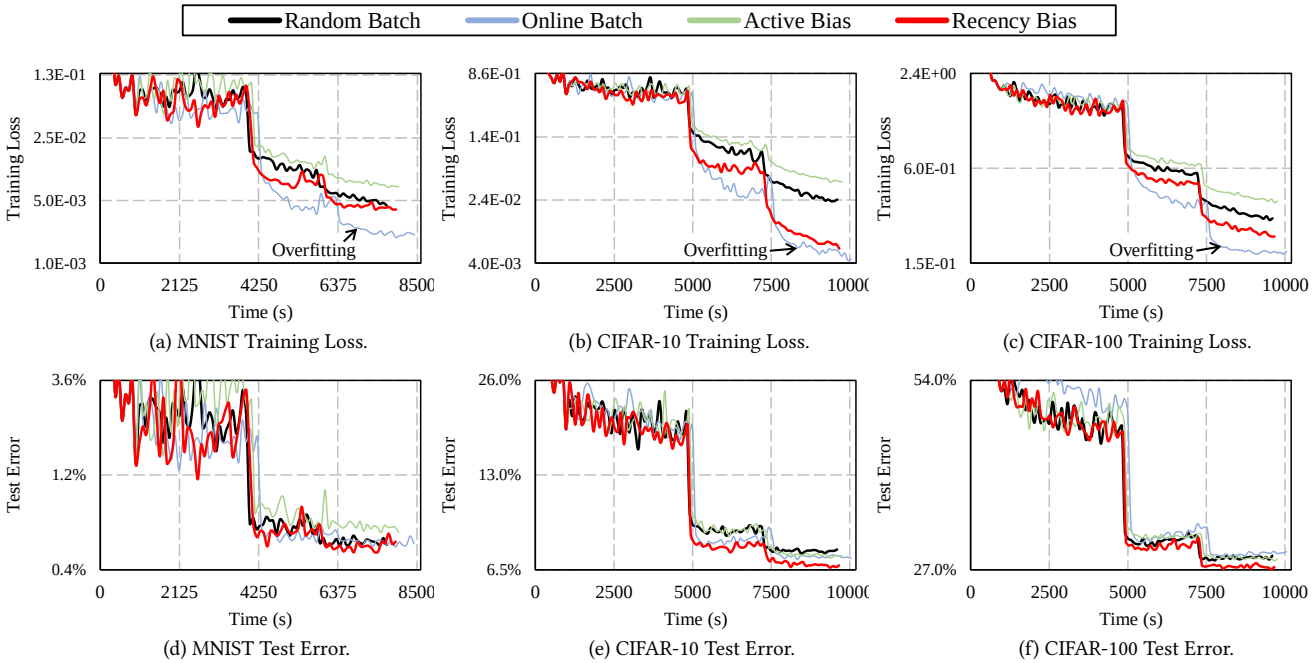
4 EVALUATION

We empirically show the improvement of *Recency Bias* over not only *Random Batch* (baseline) but also *Online Batch* [19] and *Active Bias* [3], which are two state-of-the-art adaptive batch selections. In particular, we elaborate on the effect of the sliding window approach (*Recency Bias*) compared with the growing window approach (*Active Bias*). *Random Batch* selects next mini-batch samples uniformly at random from the entire dataset. *Online Batch* selects hard samples based on the rank of the loss computed from previous epochs. *Active Bias* selects uncertain samples with high variance of true label probabilities in the growing window. All the algorithms were implemented using TensorFlow 1.15 and executed using a single NVIDIA Titan Volta GPU.

Image classification and fine-tuning tasks were performed to validate the superiority of *Recency Bias*. Because fine-tuning is used to quickly adapt to a new dataset, it is suitable to reap the benefit of fast training speed. In support of reliable evaluation, we repeated every task *thrice* and reported the average and standard error of the best test errors. The *best test error* in a given time has been widely used for the studies on fast and accurate training [13, 19]. All the datasets used for experiments are summarized in Table 1.

4.1 Analysis on Selected Mini-batch Samples

For an in-depth analysis on selected samples, we plot the loss distribution of mini-batch samples selected from CIFAR-10 by four different methods in Figure 4. *(i)* The distribution of *Online Batch* is the most skewed toward high loss by the design principle of selecting hard samples. *(ii) Active Bias* emphasizes *moderately hard* samples at an early training stage in considering that its loss distribution lies between those of *Random Batch* and *Online Batch*. However, owing to the outdated predictions caused by the growing window, the proportion of easy samples with low loss increases at a late training stage. These easy samples, which are misclassified as uncertain at



(a) MNIST Training Loss. (b) CIFAR-10 Training Loss. (c) CIFAR-100 Training Loss.

(d) MNIST Test Error. (e) CIFAR-10 Test Error. (f) CIFAR-100 Test Error.

Figure 5: Convergence curves of four batch selection methods using "DenseNet with momentum" (log-scale).

Table 2: The best test errors (%) of four batch selection methods using DenseNet.

Optimizer	Momentum in Figure 5			SGD in Figure 8 (Appendix A)		
Method	MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
Random Batch	0.53 ± 0.03	7.33 ± 0.09	27.95 ± 0.16	1.23 ± 0.03	14.86 ± 0.09	40.15 ± 0.06
Online Batch	0.51 ± 0.01	7.00 ± 0.10	28.39 ± 0.25	0.77 ± 0.02	13.52 ± 0.02	40.72 ± 0.12
Active Bias	0.62 ± 0.03	7.07 ± 0.04	27.87 ± 0.11	**0.68 ± 0.02**	14.21 ± 0.25	42.87 ± 0.05
Recency Bias	**0.49 ± 0.02**	**6.60 ± 0.02**	**27.05 ± 0.19**	0.99 ± 0.06	**13.18 ± 0.11**	**38.65 ± 0.11**

Table 3: The best test errors (%) of four batch selection methods using ResNet.

Optimizer	Momentum in Figure 9 (Appendix A)			SGD in Figure 10 (Appendix A)		
Method	MNIST	CIFAR-10	CIFAR-100	MNIST	CIFAR-10	CIFAR-100
Random Batch	0.64 ± 0.04	10.22 ± 0.12	33.20 ± 0.07	1.16 ± 0.03	12.73 ± 0.09	40.07 ± 0.16
Online Batch	0.67 ± 0.05	10.06 ± 0.05	33.38 ± 0.01	0.89 ± 0.03	12.18 ± 0.08	40.69 ± 0.09
Active Bias	0.61 ± 0.04	10.55 ± 0.08	34.19 ± 0.07	**0.80 ± 0.01**	13.51 ± 0.07	45.62 ± 0.07
Recency Bias	**0.61 ± 0.01**	**9.79 ± 0.04**	**32.43 ± 0.04**	0.97 ± 0.03	**11.63 ± 0.09**	**38.94 ± 0.14**

that stage, tend to make the convergence of training slow down. *(iii)* In contrast to *Active Bias*, by virtue of the sliding window, the distribution of *Recency Bias* lies between those of *Random Batch* and *Online Batch* regardless of the training stage. Consequently, *Recency Bias* continues to highlight the moderately hard samples, which are likely to be informative, during the training process.

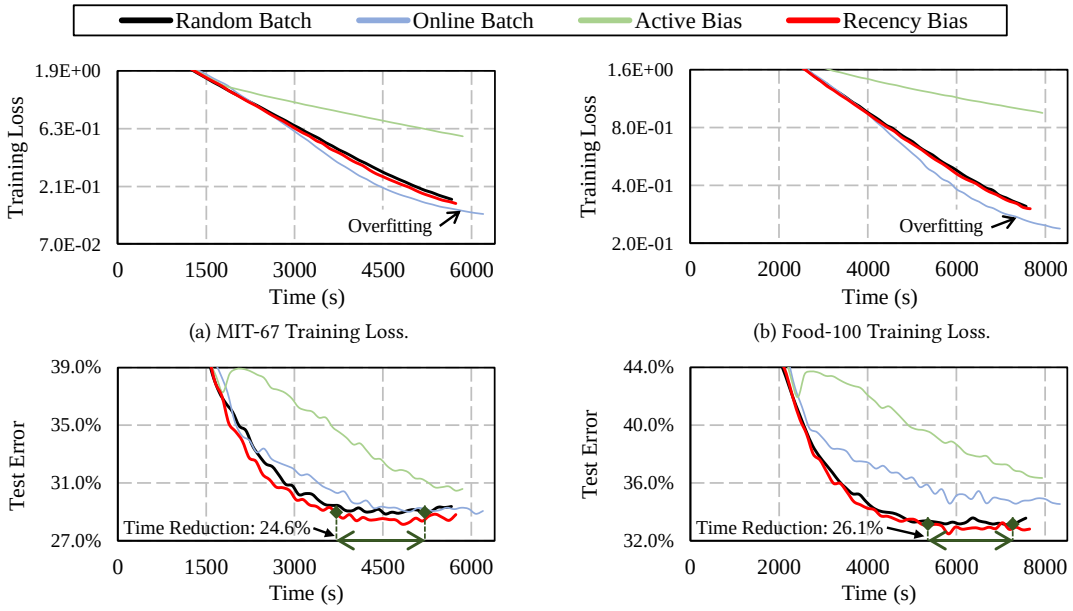
4.2 Task I: Image Classification

4.2.1 Experiment Setting. We trained DenseNet (L=40, k=12) and ResNet (L=50) with a momentum optimizer and an SGD optimizer on *three* benchmark datasets: MNIST (10 classes)[3], classification of handwritten digits [17], and CIFAR-10 (10 classes)[4] and CIFAR-100 (100 classes)[3], classification of a subset of 80 million categorical images [15]. For the classification task, we used data augmentation, batch normalization, a dropout of 0.2, a momentum of 0.9, and a batch size of 128. Regarding the algorithm parameters, we fixed the window size $q = 10$ and the initial selection pressure $s_{e_0} = 100$,[5] which were the best values found by the grid search in Section 4.5. The warm-up epoch γ was set to be 10. To reduce the performance variance caused by randomly initialized model parameters, all parameters were shared by all methods during the warm-up period. Regarding the training schedule, we trained the network for 40,000

[3] http://yann.lecun.com/exdb/mnist

[4] https://www.cs.toronto.edu/~kriz/cifar.html

[5] *Online Batch* also used the same decaying strategy.



(a) MIT-67 Training Loss.

(b) Food-100 Training Loss.

(c) MIT-67 Test Error

(d) Food-100 Test Error.

Figure 6: Convergence curves for fine-tuning on two benchmark datasets.

iterations and used an initial learning rate of 0.1, which was divided by 10 at 50% and 75% of the total number of training iterations.

4.2.2 Results. Figure 5 shows the convergence curves of training loss and test error for four batch selection methods using DenseNet and a momentum optimizer. In order to highlight the improvement of *Recency Bias* over the baseline (*Random Batch*), their lines are dark colored. The best test errors in Figures 5(d), 5(e), and 5(f) are summarized on the left side of Table 2.

In general, *Recency Bias* achieved the most accurate network while accelerating the training process on all datasets. The training loss of *Recency Bias* converged faster (Figures 5(a), 5(b), and 5(c)) without the increase in the generalization error, thereby achieving the lowest test error (Figures 5(d), 5(e), and 5(f)). In contrast, the test error of *Online Batch* was not the best even if its training loss converged the fastest among all methods. As the training difficulty increased from CIFAR-10 to CIFAR-100, the test error of *Online Batch* became even worse than that of *Random Batch*. That is, emphasizing hard training samples rather worsened the generalization capability of the network for the hard dataset (i.e., CIFAR-100) because of the overfitting. On the other hand, *Recency Bias* expedited the training step as well as achieved the lowest test error even for the hard dataset. Meanwhile, *Active Bias* was prone to make the network better generalized on test data. In CIFAR-10, despite its highest training loss, the test error of *Active Bias* was better than that of *Random Batch*. However, *Active Bias* slowed down the training process because of the limitation of growing windows, as discussed in Section 4.1. We note that, although both *Recency Bias* and *Active Bias* exploited uncertain samples, only *Recency Bias* based on sliding windows succeeded to not only speed up the training process but also reduce the generalization error. Quantitatively, *Recency Bias* achieved a significant reduction in test error of 3.22%–9.96% compared with *Random Batch*, 4.72%–5.71% compared with *Online Batch*, and 2.94%–20.97% compared with *Active Bias*.

Table 4: The best test errors (%) of four batch selection strategies using DenseNet in Figure 6.

Method	MIT-67	Food-100
Random Batch	28.85 ± 0.26	33.07 ± 0.60
Online Batch	28.87 ± 0.55	34.54 ± 0.24
Active Bias	30.46 ± 0.31	36.35 ± 0.47
Recency Bias	**28.02 ± 0.32**	**32.47 ± 0.61**

The results of the best test error for ResNet or an SGD optimizer are summarized in Tables 2 and 3 (see Appendix A for more details). Regardless of a neural network and an optimizer, *Recency Bias* achieved the lowest test error except in MNIST with an SGD optimizer. The improvement of *Recency Bias* over the others was higher with an SGD optimizer than with a momentum optimizer.

4.3 Task II: Fine-Tuning

4.3.1 Experiment Setting. Our second experiment was fine-tuning a pretrained network on new datasets. Because there exist many powerful pretrained networks on large datasets, this task is an important application to verify the usefulness of fast training. We prepared DenseNet (L=121, k=32) previously trained on ImageNet [6] and then fine-tuned the network on *two* benchmark datasets: MIT-67 (67 classes)[6], classification of indoor scenes [22], and Food-100 (100 classes)[7], classification of popular foods in Japan [14]. As summarized in Table 1, all training and testing images in both datasets were resized to 256×256, which is the original input size of the pretrained network. For the fine-tuning task, the network was

[6] http://web.mit.edu/torralba/www/indoor.html
[7] http://foodcam.mobi/dataset100.html

Table 5: Recency Bias's reduction in training time over other batch selection methods.

Method	MIT-67	FOOD-100
Random Batch	$(5,218-3,936)/5,218\times100=\mathbf{24.57}\%$	$(7,263-5,365)/7,263\times100=\mathbf{26.13}\%$
Online Batch	$(6,079-3,823)/6,079\times100=\mathbf{37.11}\%$	$(8,333-3,685)/8,333\times100=\mathbf{55.78}\%$
Active Bias	$(5,738-3,032)/5,738\times100=\mathbf{47.16}\%$	$(7,933-3,227)/7,933\times100=\mathbf{59.32}\%$

Table 6: The converged training loss of *Recency Bias* with the four strategies of decaying the selection pressure.

Metric	Training Loss	
Decaying Strategy	CIFAR-10	CIFAR-100
Strategy 1 ($s_e = 10$)	0.0059	0.2172
Strategy 2 ($s_e = 100$)	0.0036	0.1757
Strategy 3 ($s_e = 10 \rightarrow 1$)	0.0120	0.2539
Strategy 4 ($s_e = 100 \rightarrow 1$)	0.0060	0.2207

Table 7: The best test error (%) of *Recency Bias* with the four strategies of decaying the selection pressure.

Metric	Test Error	
Decaying Strategy	CIFAR-10	CIFAR-100
Strategy 1 ($s_e = 10$)	6.81 ± 0.02	27.84 ± 0.17
Strategy 2 ($s_e = 100$)	6.62 ± 0.04	28.19 ± 0.15
Strategy 3 ($s_e = 10 \rightarrow 1$)	7.13 ± 0.03	27.47 ± 0.22
Strategy 4 ($s_e = 100 \rightarrow 1$)	6.60 ± 0.02	27.05 ± 0.19

trained end-to-end for 50 epochs with a batch size 32 and a constant learning rate 2×10^{-4} after replacing the last classification layer. The other configurations were the same as those in Section 4.2.

4.3.2 Results on Test Error. Figure 6 shows the convergence curves of training loss and test error for the fine-tuning task on MIT-67 and Food-100. The best test errors in Figure 6 are summarized in Table 4. Overall, all convergence curves showed similar trends to those of the classification task in Figure 5. Only *Recency Bias* converged faster than *Random Batch* in both training loss and test error. *Online Batch* converged the fastest in training loss, but its test error was rather higher than *Random Batch* owing to the overfitting. *Active Bias* converged the slowest in both training loss and test error. Quantitatively, compared with *Random Batch*, *Recency Bias* reduced the test error by 2.88% and 1.81% in MIT-67 and Food-100, respectively.

4.3.3 Results on Training Time. Moreover, to assess the performance gain in training time, we computed the reduction in the training time taken to reach the same error. For example, in Figure 6(c), the best test error of 28.85% achieved in 5,218 seconds by *Random Batch* could be achieved only in 3,936 seconds by *Recency Bias*; thus, *Recency Bias* improved the training time by 24.6%. Table 5 summarizes the reduction in the training time of *Recency Bias* over three other batch selection methods. Notably, *Recency Bias* improved the training time by 24.57%–47.16% and 26.13%–59.32% in fine-tuning MIT-67 and FOOD-100 datasets, respectively.

4.4 Ablation Study on Selection Pressure

To examine the effect of decaying the selection pressure, we conducted additional ablation experiments on *four* different strategies of decaying the s_e value, as follows:

1. $\boldsymbol{s_e = 10}$: A *wide range* of uncertain training samples were selected during the *entire* training process because s_e was set to be a small constant value.
2. $\boldsymbol{s_e = 100}$: Differently to Strategy 1, only *highly* uncertain training samples were highlighted during the *entire* training process because s_e was set to be a large constant value.
3. $\boldsymbol{s_e = 10 \rightarrow 1}$: This strategy starts from Strategy 1, but, as the training progresses, more diverse training samples were chosen regardless of their uncertainty due to the exponential decay of s_e from 10 to 1.
4. $\boldsymbol{s_e = 100 \rightarrow 1}$: This strategy is similar to Strategy 3, but the initial value of s_e was made much larger to further emphasize highly uncertain samples during the training process.

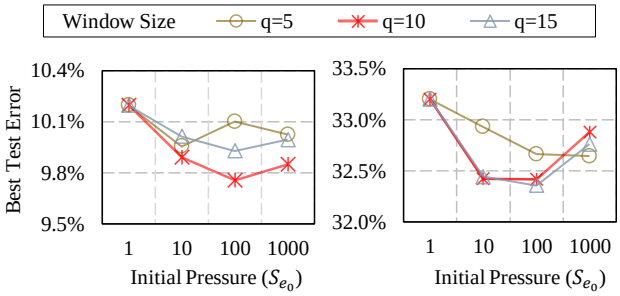
DenseNet (L=40, k=12) was trained on two CIFAR datasets using *Recency Bias* with these four strategies. A momentum optimizer was used, and the other configurations were the same as those in Section 4.2. Tables 6 and 7 summarize the converged training loss and the best test error, respectively, of the four strategies.

A lower training loss was generally achieved when the selection pressure was not decayed (see Strategy 1 and Strategy 2 in Table 6) because moderately hard training samples were consistently emphasized during the entire training process. Nevertheless, because using only a *part* of training data exacerbated the overfitting problem as mentioned in Section 3.2.2, the test error of Strategy 1 (or Strategy 2) was generally worse than that of Strategy 3 (or Strategy 4) as shown in Table 7. The overfitting problem may become severer with Strategy 2 than Strategy 1 as witnessed by the test error for CIFAR-100. As opposed to when using a constant selection pressure, a larger initial value was beneficial when decaying the selection pressure. That is, Strategy 4 achieved the lower test error than Strategy 3 in both datasets. Overall, the best test error was always achieved by Strategy 4, which corroborates the importance of decaying the selection pressure to exploit more diverse training samples especially at a late stage of training.

4.5 Hyperparameter Selection

Recency Bias receives the two hyperparameters: *(i)* the initial selection pressure s_{e_0} that determines the sampling probability gap between the most and the least uncertain samples and *(ii)* the window size q that determines how many recent label predictions are involved in predicting the uncertainty. To decide the best hyperparameters, we trained ResNet (L=50) on CIFAR-10 and CIFAR-100 with a momentum optimizer. For hyperparameters selection, the two hyperparameters were chosen in a grid $s_{e_0} \in \{1, 10, 100, 1000\}$ and $q \in \{5, 10, 15\}$.

Figure 7 shows the test errors of *Recency Bias* obtained by the grid search on the two datasets. Regarding the initial selection pressure



(a) CIFAR-10. (b) CIFAR-100.

Figure 7: Grid search on CIFAR-10 and CIFAR-100 datasets using ResNet.

s_{e_0}, the lowest test error was typically achieved when the s_{e_0} value was 100. As for the window size q, the test error was almost always the lowest when the q value was 10. Similar trends were observed for the other combinations of a neural network and an optimizer. Therefore, in all experiments, we set s_{e_0} to be 100 and q to be 10.

5 CONCLUSION

In this paper, we presented a novel adaptive batch selection algorithm called *Recency Bias* that emphasizes predictively uncertain samples for accelerating the training of neural networks. Toward this goal, the predictive uncertainty of each sample is evaluated using its *recent* label predictions managed by a sliding window of a fixed size. Then, uncertain samples *at the moment* are selected with high probability for the next mini-batch. We conducted extensive experiments on both classification and fine-tuning tasks. The results showed that *Recency Bias* is effective in reducing the training time as well as the best test error. It was worthwhile to note that using *all* historical observations to estimate the uncertainty has the side effect of slowing down the training process. Overall, a merger of uncertain samples and sliding windows greatly improves the power of adaptive batch selection.

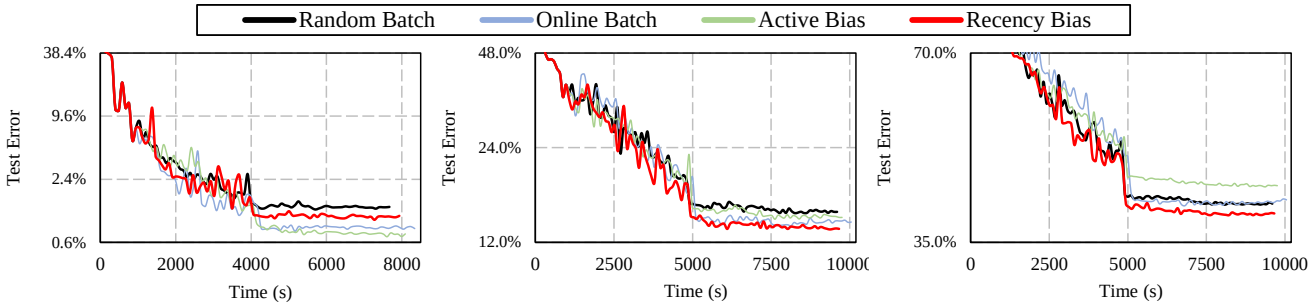
REFERENCES

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*. 41–48.
[2] David Chandler. 1987. *Introduction to modern statistical mechanics*. Oxford University Press.
[3] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. 2017. Active Bias: Training more accurate neural networks by emphasizing high variance samples. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 1002–1012.
[4] Brian Chen and Gregory W Wornell. 2001. Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information Theory* 47, 4 (2001), 1423–1443.
[5] Yixuan Chen, Jie Sui, Liang Hu, and Wei Gong. 2019. Attention-residual network with CNN for rumor detection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1121–1130.
[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.
[7] Yang Fan, Fei Tian, Tao Qin, and Tie-Yan Liu. 2017. Neural Data Filter for Bootstrapping Stochastic Gradient Descent. In *Proceedings of the 5th International Conference on Learning Representation*.
[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
[9] Siddharth Gopal. 2016. Adaptive sampling for SGD by exploiting side information. In *Proceedings of the 33rd International Conference on Machine Learning*. 364–372.
[10] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Proceedings of the 32rd International Conference on Neural Information Processing Systems*. 8527–8537.
[11] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*. 315–323.
[12] KJ Joseph, Krishnakant Singh, Vineeth N Balasubramanian, et al. 2019. Submodular batch selection for training deep neural networks. In *Proceedings of 28th International Joint Conference on Artificial Intelligence*. 2677–3683.
[13] Angelos Katharopoulos and François Fleuret. 2018. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning*. 2525–2534.
[14] Y. Kawano and K. Yanai. 2014. Food image recognition with deep convolutional features. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*.
[15] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.
[16] M Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*. 1189–1197.
[17] Yann LeCun. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist.
[18] Xiangyang Liu, Bingkun Wei, Fanhua Shang, and Hongying Liu. 2019. Loopless Semi-Stochastic Gradient Descent with Less Hard Thresholding for Sparse Learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 881–890.
[19] Ilya Loshchilov and Frank Hutter. 2016. Online batch selection for faster training of neural networks. In *Proceedings of the 4th International Conference on Learning Representation*.
[20] Lei Mei, Pengjie Ren, Zhumin Chen, Liqiang Nie, Jun Ma, and Jian-Yun Nie. 2018. An attentive interaction network for context-aware recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 157–166.
[21] Deyu Meng, Qian Zhao, and Lu Jiang. 2015. What objective does self-paced learning indeed optimize? *arXiv preprint arXiv:1511.06049* (2015).
[22] Ariadna Quattoni and Antonio Torralba. 2009. Recognizing indoor scenes. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. 413–420.
[23] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.
[24] Mrinmaya Sachan and Eric Xing. 2016. Easy questions first? A case study on curriculum learning for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 453–463.
[25] Ohad Shamir and Tong Zhang. 2013. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *Proceedings of the 30th International Conference on Machine Learning*. 71–79.
[26] Yanyao Shen and Sujay Sanghavi. 2019. Learning with bad training data via iterative trimmed loss minimization. In *Proceedings of the 36th International Conference on Machine Learning*. 5739–5748.
[27] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. 2016. Training region-based object detectors with online hard example mining. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. 761–769.
[28] Hwanjun Song, Minseok Kim, and Jae-Gil Lee. 2019. SELFIE: Refurbishing unclean samples for robust deep learning. In *International Conference on Machine Learning*. 5907–5915.
[29] Hwanjun Song, Minseok Kim, Dongmin Park, and Jae-Gil Lee. 2019. Prestopping: How does early stopping help generalization against label noise? *arXiv preprint arXiv:1911.08059* (2019).
[30] Luis Torgo. 2011. *Data mining with R: learning with case studies*. Chapman and Hall/CRC.
[31] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Brian MacWhinney, and Chris Dyer. 2016. Learning the curriculum with Bayesian optimization for task-specific word representation learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 130–139.
[32] Shengjie Wang, Wenruo Bai, Chandrashekhar Lavania, and Jeff Bilmes. 2019. Fixing Mini-batch Sequences with Hierarchical Robust Partitioning. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*. 3352–3361.
[33] Bernard Widrow, Istvan Kollar, and Ming-Chang Liu. 1996. Statistical theory of quantization. *IEEE Transactions on instrumentation and measurement* 45, 2 (1996), 353–361.
[34] Peilin Zhao and Tong Zhang. 2015. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32rd International Conference on Machine Learning*. 1–9.
[35] Tianyi Zhou and Jeff Bilmes. 2018. Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity. In *Proceedings of the 6th International Conference on Learning Representation*.

A GENERALIZATION OF *RECENCY BIAS*

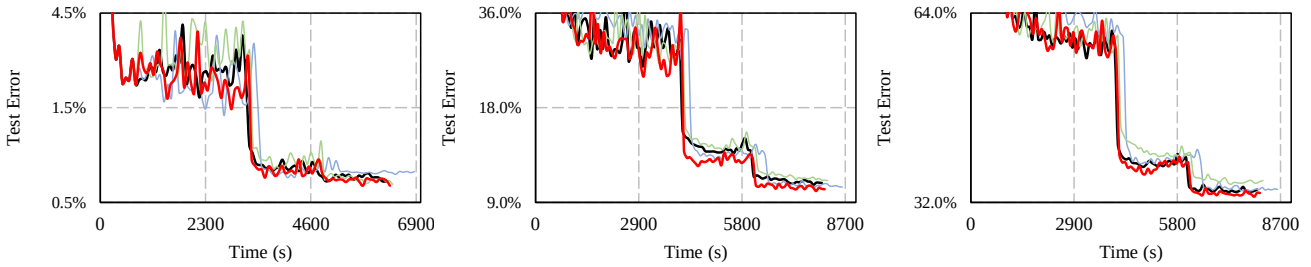
Figures 8, 9, and 10 show the convergence curves of test error for the four batch selection strategies using "DenseNet and an SGD optimizer" (see the right side of Table 2), "ResNet and a momentum optimizer" (see the left side of Table 3), and "ResNet and an SGD optimizer" (see the right side of Table 3), respectively.

The performance dominance of *Recency Bias* was generally consistent regardless of an optimizer and a network architecture. Except in MNIST with an SGD optimizer, *Recency Bias* achieved a significant reduction in test error of 2.32%–19.51% compared with *Random Batch*, 2.51%–8.96% compared with *Online Batch*, and 5.15%–14.64% compared with *Active Bias*, respectively.



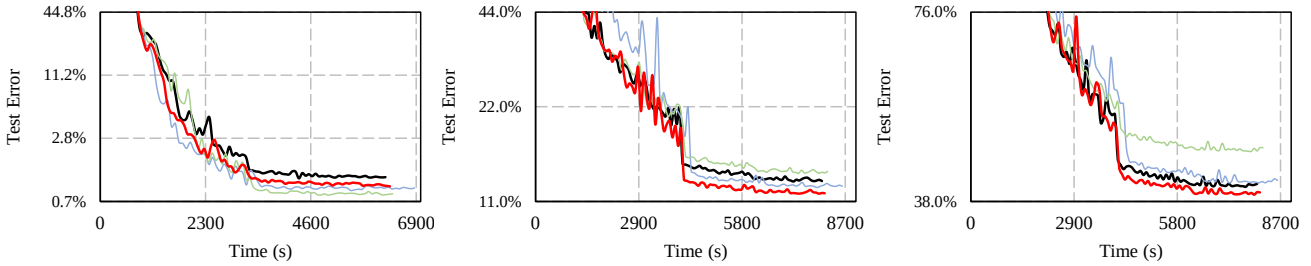
(a) MNIST Test Error. (b) CIFAR-10 Test Error. (c) CIFAR-100 Test Error.

Figure 8: Convergence curves of four batch selection strategies using "DenseNet with SGD" (log-scale).



(a) MNIST Test Error. (b) CIFAR-10 Test Error. (c) CIFAR-100 Test Error.

Figure 9: Convergence curves of four batch selection strategies using "ResNet with momentum" (log-scale).



(a) MNIST Test Error. (b) CIFAR-10 Test Error. (c) CIFAR-100 Test Error.

Figure 10: Convergence curves of four batch selection strategies using "ResNet with SGD" (log-scale).