Machine Learning (2020) 109:1837–1853
https://doi.org/10.1007/s10994-020-05903-6

Ada-boundary: accelerating DNN training via adaptive boundary batch selection

Hwanjun Song[1] · **Sundong Kim**[2] · **Minseok Kim**[1] · **Jae-Gil Lee**[1]

Received: 3 December 2019 / Revised: 21 June 2020 / Accepted: 11 August 2020 /
Published online: 4 September 2020

Abstract
Neural networks converge faster with help from a smart batch selection strategy. In this regard, we propose *Ada-Boundary*, a novel and simple adaptive batch selection algorithm that constructs an effective mini-batch according to the learning progress of the model. Our key idea is to exploit *confusing* samples for which the model cannot predict labels with high confidence. Thus, samples near the current decision boundary are considered to be the most effective for expediting convergence. Taking advantage of this design, *Ada-Boundary* maintained its dominance for various degrees of training difficulty. We demonstrate the advantage of *Ada-Boundary* by extensive experimentation using CNNs with five benchmark data sets. *Ada-Boundary* was shown to produce a relative improvement in test errors by up to 31.80% compared with the baseline for a fixed wall-clock training time, thereby achieving a faster convergence speed.

Keywords Batch selection · Acceleration · Convergence · Decision boundary

1 Introduction

Deep neural networks (DNNs) have achieved remarkable performance in many fields, especially, in computer vision and natural language processing (Goodfellow et al. 2016). Nevertheless, as the size of data set grows, the training step via stochastic gradient descent

Editors: Ira Assent, Carlotta Domeniconi, Aristides Gionis, Eyke Hüllermeier.

✉ Jae-Gil Lee
jaegil@kaist.ac.kr

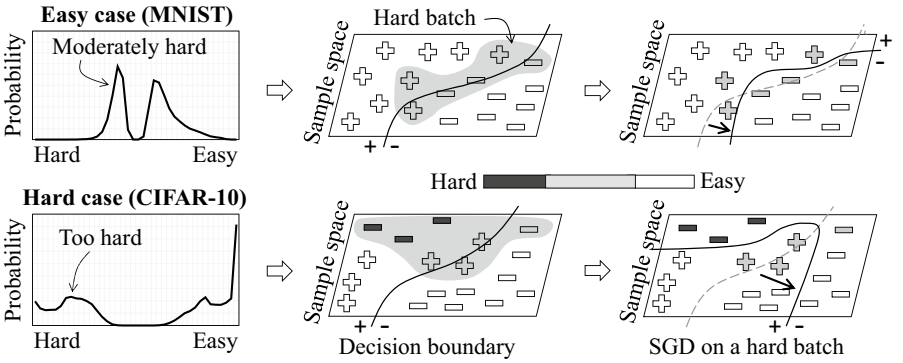
Hwanjun Song
songhwanjun@kaist.ac.kr

Sundong Kim
sundong@ibs.re.kr

Minseok Kim
minseokkim@kaist.ac.kr

[1] Graduate School of Knowledge Service Engineering, KAIST, Daejeon, Korea

[2] Institute for Basic Science, Daejeon, Korea



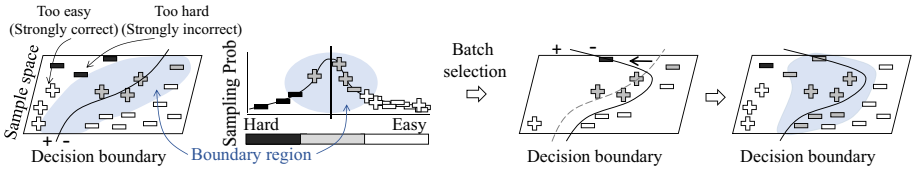
(a) Difficulty distribution. **(b)** Hard sample oriented training.

Fig. 1 Analysis on hard batch selection strategy: **a** shows the true sample distribution according to the difficulty computed by Eq. (1) at the training accuracy of 60%. An easy data set (MNIST) does not have "too hard" samples but "moderately hard" samples colored in gray, whereas a relatively hard data set (CIFAR-10) has many "too hard" samples colored in black. **b** Shows the result of SGD on a hard batch. The moderately hard samples are informative to update a model, but the too hard samples make the model overfit to themselves

(SGD) based on mini-batches suffers from extremely high computational cost, which is mainly due to *slow convergence*. The common approaches for expediting convergence include SGD variants (Zeiler 2012; Kingma and Ba 2015) that maintain individual learning rates for parameters, and batch normalization (Ioffe and Szegedy 2015) that stabilizes the gradient variance.

Recently, considering the fact that not all samples have an equal impact on training, many studies have attempted to design sampling schemes based on sample importance (Wu et al. 2017; Fan et al. 2017; Katharopoulos and Fleuret 2018). *Curriculum learning* (Bengio et al. 2009), inspired by human learning, is one of the representative methods for speeding up the training step by gradually increasing the difficulty level of the training samples. In contrast, deep learning studies focus on giving *higher* weights to *harder* samples during the entire training process. When the model requires many epochs for convergence, it is known to converge faster with the batches of hard samples than with randomly selected batches (Schaul et al. 2016; Loshchilov and Hutter 2016; Gao and Jojic 2017; Song et al. 2020). There are various criteria for judging the hardness of a sample, e.g., the rank of the loss computed from previous epochs (Loshchilov and Hutter 2016).

Here, a natural question arises: **Does the "hard" batch selection always speed up DNN training?** Our answer is **partially, yes**: It is helpful only when training an *easy* data set. According to our in-depth analysis, as demonstrated in Fig. 1a, the *hardest* samples in a hard data set (e.g., CIFAR-10) were *too hard* to learn. They were highly likely to sway the decision boundary bias towards themselves, as shown in Fig. 1b. On the other hand, in an easy data set (e.g., MNIST), the *hardest* samples, though they were only moderately hard, provided useful information for training. In practice, it has been reported that hard batch selection successfully speed up training only for the easy MNIST data set (Loshchilov and Hutter 2016; Gao and Jojic 2017), and our experiments presented in Sect. 5 also confirmed the previous findings. This limitation calls for a new sampling scheme that supports both easy and hard data sets.



(a) Before update. **(b)** After update.

Fig. 2 Key idea of *Ada-Boundary*: **a** shows the sampling process of *Ada-Boundary*, **b** shows the results of an SGD iteration on the boundary samples

In this paper, we propose a novel and simple adaptive batch selection strategy, called ***Ada-Boundary***, that accelerates training and is better generalized to hard data sets. As opposed to existing hard batch selection, *Ada-Boundary* picks up the samples with the most appropriate difficulty, considering the learning progress of the model. The samples near the *current* decision boundary are selected with high probability, as shown in Fig. 2a. Intuitively speaking, the samples far from the decision boundary are not that helpful because they are either too hard or too easy: those on the incorrect side are too hard, and those on the correct side are too easy. This is the reason why we regard the confusing samples around the decision boundary, which are *moderately hard*, to have the appropriate difficulty level.

Overall, the key idea of *Ada-Boundary* is to use the distance of a sample to the decision boundary as the hardness of the sample. The beauty of this design is that it does *not* require human intervention. The current decision boundary should be directly influenced by the learning progress of the model. The decision boundary of a DNN moves towards eliminating incorrect samples as the training step progresses, so the difficulty of the samples near the decision boundary gradually increases as the model is learned. Then, the decision boundary continually updates to identify the *confusing* samples, as illustrated in Fig. 2b. This approach accelerates convergence by providing samples suited to the model at every SGD iteration, and it is less prone to incur an overfitting issue.

We conducted extensive experiments to demonstrate the superiority of *Ada-Boundary*. A popular convolutional neural network (CNN)[1] model was trained on five benchmark data sets for the image classification task. Compared to *random batch* selection, *Ada-Boundary* produced a relative improvement in test errors by up to 31.80% for a fixed wall-clock training time. Compared to the two state-of-the-art algorithms, *online batch* (Loshchilov and Hutter 2016) and *active bias* (Chang et al. 2017), it respectively improved the test error by up to 8.14% and 10.07% within the same time frame. Moreover, *Ada-Boundary* is well-generalized for different gradient optimizers and CNN models.

2 Related work

There have been numerous attempts to understand which samples contribute the most during training. Curriculum learning (Bengio et al. 2009), inspired by the perceived way that humans and animals learn, first takes easy samples and then gradually increases the

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

difficulty of samples using a manual method. Self-paced learning (Kumar et al. 2010) uses prediction error to determine the easiness of samples in order to alleviate the limitation of curriculum learning. The researchers assumed that importance was determined by how easy the samples were. However, easiness does not sufficiently determine when a sample should be introduced to a learner (Gao and Jojic 2017).

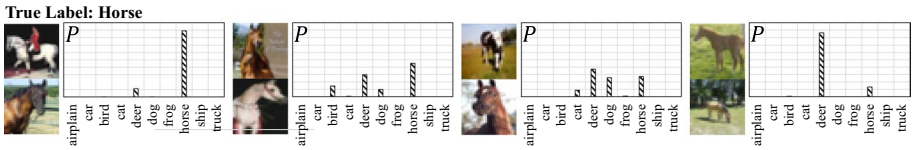
Recently, Tsvetkov et al. (2016) used Bayesian optimization to optimize a curriculum for training dense, distributed word representations. Sachan and Xing (2016) emphasized that the right curriculum not only has to arrange data samples in order of difficulty, but also must introduce a small number of samples that are dissimilar to the samples previously seen. Shrivastava et al. (2016) proposed a hard-example mining method to eliminate several heuristics and hyperparameters commonly used to select hard examples. However, these algorithms are designed to support only a designated task, such as natural language processing or object detection. The neural data filter proposed by Fan et al. (2017) is orthogonal to our work because it aims to filter redundant samples from streaming data. As mentioned earlier, *Ada-Boundary* generally follows the philosophy of curriculum learning because it exploits the samples with the most appropriate difficulty at the current training progress.

More closely related to adaptive batch selection, Loshchilov and Hutter (2016) stored the history of losses for previously seen samples, and computed sampling probability based on loss rank. The sample probability to be selected for the subsequent mini-batch was exponentially decayed with its rank. This allowed the samples with low ranks (i.e., high losses) to be considered more frequently for the subsequent mini-batch. Gao and Jojic (2017)'s work is similar to that of Loshchilov and Hutter (2016) except that gradient norms are used instead of losses to compute the probability. In contrast to curriculum learning, both methods focus on hard samples only for training. Also, they ignored the difference in actual losses or gradient norms by transforming the values to ranks. Similar to our work, the usefulness of exploiting uncertain samples was witnessed by *active bias* (Chang et al. 2017) for a different purpose. Their main contribution lies in producing a more accurate and robust model by choosing samples with high prediction variance, whereas ours lies in training faster by using confusing samples that have softmax distributions of low variance. According to our experiments presented in Sect. 5.1, the samples selected by *active bias* slowed down the convergence in training loss, though they reduced the generalization error.

To complement this survey, we mention work done to accelerate the optimization process of algorithms based on importance sampling. Needell et al. (2014) re-weighted the obtained gradients by the inverse of their sampling probabilities to reduce the variance. Schmidt et al. (2015) biased the sampling toward the Lipschitz constant to quickly find the solution to a strongly-convex optimization problem arising from the training of conditional random fields.

3 *Ada-Boundary* Components

The main challenge for *Ada-Boundary* is to evaluate how close a sample is to the decision boundary. In this section, we introduce a novel distance measure, and present a method for computing the sampling probability based on the measure.



(a) Strongly correct. **(b)** Weakly correct. **(c)** Weakly incorrect. **(d)** Strongly incorrect.

Fig. 3 Classification of CIFAR-10 samples using the softmax distribution at the training accuracy of 90%. If the prediction probability of the true label is the highest, the prediction is correct; otherwise, incorrect. If the highest probability dominates the distribution, the model's confidence is strong; otherwise, weak

3.1 Sample distance based on Softmax distribution

To evaluate the distance from the sample to the decision boundary, we note that the softmax distribution, which is the output of the softmax layer in neural networks, clearly indicates how confidently the model predicts the true label, as demonstrated in Fig. 3.

Let $h(y|x_i;\theta^t)$ be the softmax distribution for a given sample x_i over $y \in \{1, 2, \ldots, k\}$ labels, where θ^t is the parameter of a neural network at time t. Then, the distance from a sample x_i with the true label y_i to the decision boundary of the neural network with θ^t is defined by the directional distance function,

$$\begin{aligned} dist(x_i, y_i;\theta^t) &= \overbrace{sign(x_i, y_i)}^{\text{correctness}} \cdot \overbrace{std\big(h(y|x_i;\theta^t)\big)}^{\text{confidence}}, \\ \text{where } \; sign(x_i, y_i) &= \begin{cases} +1, & argmax_{y\in\{1,\ldots,k\}} h(y|x_i;\theta^t) = y_i \\ -1, & otherwise. \end{cases} \end{aligned} \tag{1}$$

More specifically, the function consists of two terms related to the direction and magnitude of the distance, determined by the model's correctness and confidence, respectively. The correctness is determined by verifying whether the label with the highest probability matches the true label y_i, and the confidence is computed by the standard deviation of the softmax distribution. Intuitively, the standard deviation is a nice indicator of confidence because the value gets closer to zero when the model is confused.

One might argue that the cross-entropy loss, $H(p, q) = -p(x_i)\log(q(x_i))$ where $p(x_i)$ and $q(x_i)$ are the true and softmax distributions for x_i, can be adopted for the distance function. However, because $p(x_i)$ is formulated as a one-hot true label vector, the cross-entropy loss cannot capture the prediction probability for *false* labels, which is an important factor in confusing samples.

Another advantage is that our distance function is *bounded* as opposed to the loss. For k labels, the maximum value of $std(h(y|x_i;\theta^t))$ is $k^{-1}\sqrt{(k-1)}$ when $h(m|x_i;\theta^t) = 1$ and $\forall_{l\neq m} h(l|x_i;\theta^t) = 0$. Thus, $dist(x_i, y_i;\theta^t)$ is bounded by

$$-k^{-1}\sqrt{k-1} \leq dist(x_i, y_i;\theta^t) \leq k^{-1}\sqrt{k-1}. \tag{2}$$

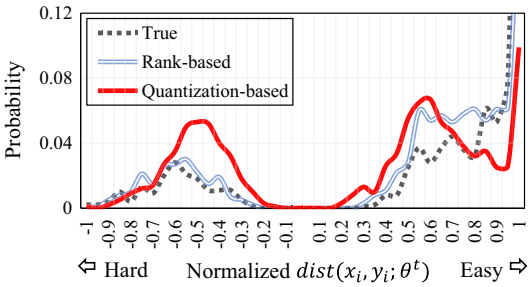


Fig. 4 Sample distribution according to the normalized $dist(x_i, y_i; \theta^t)$ on Fashion-MNIST data set at the training accuracy of 80%. The distributions of mini-batch samples selected by the rank-based approach and the quantization-based approach are plotted together with the true sample distribution

3.2 Sampling probability based on quantization index

The rank-based approach introduced by Loshchilov and Hutter (2016) is a common way to assign the sampling probability of being selected for the next mini-batch. This approach sorts the samples by a certain importance measure in descending order, and exponentially decays the sampling probability of a given sample according to its rank. Let N denote the total number of samples. Then, each r-th ranked sample is selected with the probability $p(r)$, which drops by a factor of $\exp(\log(s_e)/N)$. Here, 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 that the r-th ranked sample is selected is defined by

$$p(r) = \frac{1/\exp(\log(s_e)/N)^r}{\sum_{j=1}^{N} 1/\exp(\log(s_e)/N)^j}. \tag{3}$$

In the existing rank-based approach, the rank of a sample is determined by $|dist(x_i, y_i; \theta^t)|$ in ascending order, because it is inversely proportional to the sample importance. However, if the mass of the true sample distribution is skewed to one side (e.g., easy side) as shown in Fig. 4, the mini-batch samples are selected with high probability from the skewed side rather than from around the decision boundary where $|dist(x_i, y_i; \theta^t)|$ is very small. This problem was attributed to the unconditionally fixed probability of a given rank. In other words, samples with similar ranks are selected with similar probabilities, regardless of the magnitude of the distance values.

To incorporate the impact of distance into batch selection, we adopt the quantization method (Gray and Neuhoff 1998; Chen and Wornell 2001) and use the quantization index q instead of rank r. Let Δ be the quantization step size and d be the output of the function $dist(x_i, y_i; \theta^t)$ of a given sample x_i. Then, the index q is obtained by the simple quantizer $Q(d)$,

$$q = Q(d), \quad Q(d) = \lceil |d|/\Delta \rceil. \tag{4}$$

The quantization index gets larger as sampling moves away from the decision boundary. In addition, the difference between two indexes reflects the difference in the actual distances.

In Eq. (4), we set Δ to be $k^{-1}\sqrt{k-1}/N$ such that the index q is bounded to N (the total number of samples) by Eq. (2). Then, the sampling probability of a given sample x_i with the true label y_i is defined by

$$p(x_i, y_i) = \frac{1/\exp\left(\log(s_e)/N\right)^{Q(dist(x_i, y_i; \theta^t))}}{\sum_{j=1}^{N} 1/\exp\left(\log(s_e)/N\right)^{Q(dist(x_j, y_j; \theta^t))}}. \tag{5}$$

As shown in Fig. 4, our quantization-based method produces a *well-balanced* distribution, even if the true sample distribution is skewed.

4 *Ada-Boundary* Algorithm

4.1 Main proposed algorithm

Algorithm 1 describes the overall procedure of *Ada-Boundary*. The input to the algorithm consists of samples of size N (i.e., training data set), a mini-batch of size b, the selection pressure s_e, and the threshold γ used to decide the warm-up period. In the early stages of training, because the quantization index for each sample is not confirmed yet, the algorithm requires a warm-up period of γ epochs. Randomly selected mini-batch samples are used for the warm-up period (Lines 6–7), and their quantization indexes are updated (Lines 12–18). After the warm-up epochs, the algorithm computes the sampling probability of each sample by Eq. (5) and selects mini-batch samples based on the probability (Lines 8–11). Then, the quantization indexes are updated in the same way (Lines 12–18). Here, we compute the indexes using the previous model with θ^t after every SGD step, rather than the latest model with θ^{t+1}, in order to reuse the previously computed softmax distributions; in addition, we asynchronously update the indexes of the samples that are only included in the mini-batch to prevent the additional forward propagation of the entire sample, which would induce a high computational cost.

Algorithm 1 *Ada-Boundary* Algorithm

INPUT: N samples, $numEpoch$, b: mini-batch size, s_e: selection pressure, γ: warm-up period

1: $t \leftarrow 1$;
2: $\theta^t \leftarrow$ Initialize the model parameter;
3: $q\_dict \leftarrow \{\}$; /* Dictionary for quantization indexes */
4: **for** $i = 1$ **to** $numEpoch$ **do**
5: **for** $j = 1$ **to** N/b **do**
6: **if** $i \leq \gamma$ **then** /* Warm-up */
7: $\{(x_1, y_1), \ldots, (x_b, y_b)\} \leftarrow$ Randomly select next mini-batch samples;
8: **else** /* Adaptive batch selection */
9: /* By Eq. (5) */
10: $prob\_table \leftarrow Compute\_Prob(q\_dict, s_e)$;
11: $\{(x_1, y_1), \ldots, (x_b, y_b)\} \leftarrow$ Select next mini-batch samples based on $prob\_table$;
12: /* Forward */
13: $loss \leftarrow Get\_Loss(\{(x_1, y_1), \ldots, (x_b, y_b)\}, \theta^t)$;
14: /* Asynchronous index update by Eq. (4) */
15: **for** $m = 1$ **to** b **do**
16: $q\_dict[x_m] = Q(dist(x_m, y_m; \theta^t))$;
17: /* Backward */
18: $\theta^{t+1} \leftarrow SGD\_Step(loss, \theta^t)$;
19: $t \leftarrow t + 1$;

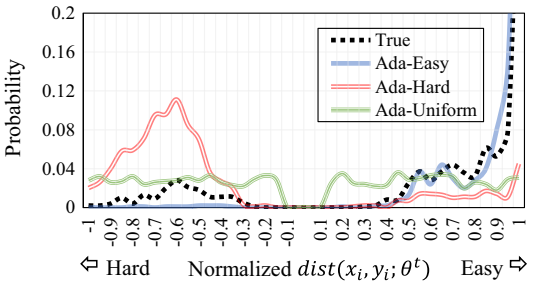


Fig. 5 The distributions of mini-batch samples selected by the two variants in the same configuration as Fig. 4

4.2 Variants of *Ada-Boundary* for comparison

For a more sophisticated analysis of sampling, we present two heuristic sampling strategies: (1) *Ada-Hard* is similar to the existing hard batch strategy (Loshchilov and Hutter 2016), but it uses our distance function instead of the loss. That is, *Ada-Hard* focuses on the samples far from the decision boundary in the *negative* direction; (2) *Ada-Uniform* is designed to select samples with a wide range of difficulty, so it samples uniformly over the distance range regardless of the sample distribution.

We modified a few lines of Algorithm 1 to implement the two variants. In detail, for *Ada-Hard*, the quantization index q should be small for the sample located far in the negative direction. Thus, *Ada-Hard* can be implemented by modifying the quantizer $Q(d)$ in Line 16 of Algorithm 1. When we set $\Delta = k^{-1}\sqrt{k-1/N}$ to make index q bound to N, the quantizers of *Ada-Hard* are defined by

$$Q(d) = \begin{cases} \lceil d/2\Delta \rceil + N/2, & \text{if } d \geq 0 \\ \lfloor d/2\Delta \rfloor + N/2 + 1, & \text{otherwise.} \end{cases} \tag{6}$$

Ada-Uniform can be implemented by using $F^{-1}(x)$ to compute the sampling probability in Line 10 of Algorithm 1, where $F(x)$ is the empirical sample distribution according to the sample's distance to the decision boundary. Note that the computational cost of *Ada-Uniform* is much higher than those of *Ada-Boundary* and *Ada-Hard* because of the computation for the empirical distribution, which requires the linear time complexity to the total number of training samples (i.e., $O(N)$) in *every* update iteration.

Figure 5 shows the distributions of mini-batch samples drawn by these two variants. The distribution of *Ada-Hard* is skewed to the hard side, and that of *Ada-Uniform* tends to be uniform.

5 Evaluation

To validate the superiority of *Ada-Boundary*, we performed an image classification task on *five* benchmark data sets with varying difficulty levels: MNIST (easy),[2] classification of handwritten digits (LeCun 1998), with 60,000 training and 10,000 testing images; Fashion-MNIST (relatively easy),[3] classification of various clothing (Xiao et al. 2017), with

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

[3] https://github.com/zalandoresearch/fashion-mnist.

60,000 training and 10,000 testing images; CIFAR-10[4] (relatively hard) and CIFAR-100 (hard),[4] classification of a subset of 80 million categorical images (Krizhevsky et al. 2014), with 50,000 training and 10,000 testing images; Tiny-ImageNet (hard),[5] classification of a subset of large-scale categorical images (Krizhevsky et al. 2012), with 100,000 training and 10,000 testing images. We did not apply any data augmentation or pre-processing procedures.

We experimentally analyzed the performance improvement of *Ada-Boundary* compared with not only *random batch* selection but also *four* different adaptive batch selection algorithms: *random batch* selection selects the next batch uniformly at random from the entire data set; *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; *Ada-Hard* and *Ada-Uniform*, which are the two variants of *Ada-Boundary* introduced in Sect. 4.2. All the algorithms were implemented using TensorFlow 2.1.0[6] and excuted using eight NVIDIA Titan Volta GPU. For reproducibility, we provide the source code at https://github.com/kaist-dmlab/Ada-Boundary.

For the classification task, we mainly used a densely connected neural network (DenseNet) (Huang et al. 2017), which is widely known to achieve good generalization performance on data sets with varying difficulty levels (Song et al. 2019). In support of reliable evaluation, we repeated every task *five* times and reported the average with its standard error. That is, the training loss (or test error) was averaged for all the trials at every epoch. To compare the convergence speed among the methods, we plotted the averaged training loss and test error for an equivalent wall-clock training time. Besides, because the *best test error* in a given time has been widely used for the studies on fast and accurate training (Loshchilov and Hutter 2016; Chang et al. 2017), we reported the average of the best test errors in tabular form.

5.1 Analysis on hard data sets

We used six batch selection strategies to train a DenseNet on the hard data sets: CIFAR-10, CIFAR-100, and Tiny-ImageNet. Specifically, we trained a DenseNet ($L = 25$, $k = 12$) with a momentum optimizer. We used batch normalization (Ioffe and Szegedy 2015), a momentum of 0.9, and a batch size of 128. As for the algorithm parameters, we used the best selection pressure s_e, obtained from $s_e = \{2, 8, 32\}$ (see Sect. 5.5 for details), and set the warm-up threshold γ to 15. Technically, a small γ is enough to warm-up, but to reduce the performance variance caused by randomly initialized parameters, we used a larger γ and shared the model parameters for all strategies during the warm-up period. For *online batch* selection, we recomputed all the losses across every epoch to reflect the latest losses. Regarding the training schedule, following the experimental setup of Huang et al. (2017), we trained the model for 100 epochs and used an initial learning rate of 0.1, which was divided by 5 at 50% and 75% of the total number of training iterations. Because the baseline strategy required about 2,300 seconds for the two CIFAR data sets and 14,500 seconds for the Tiny-ImageNet data set, we excluded the result of other strategies beyond those times.

[4] https://www.cs.toronto.edu/~kriz/cifar.html.
[5] https://www.kaggle.com/c/tiny-imagenet.
[6] https://www.tensorflow.org/versions/r2.1/api_docs.

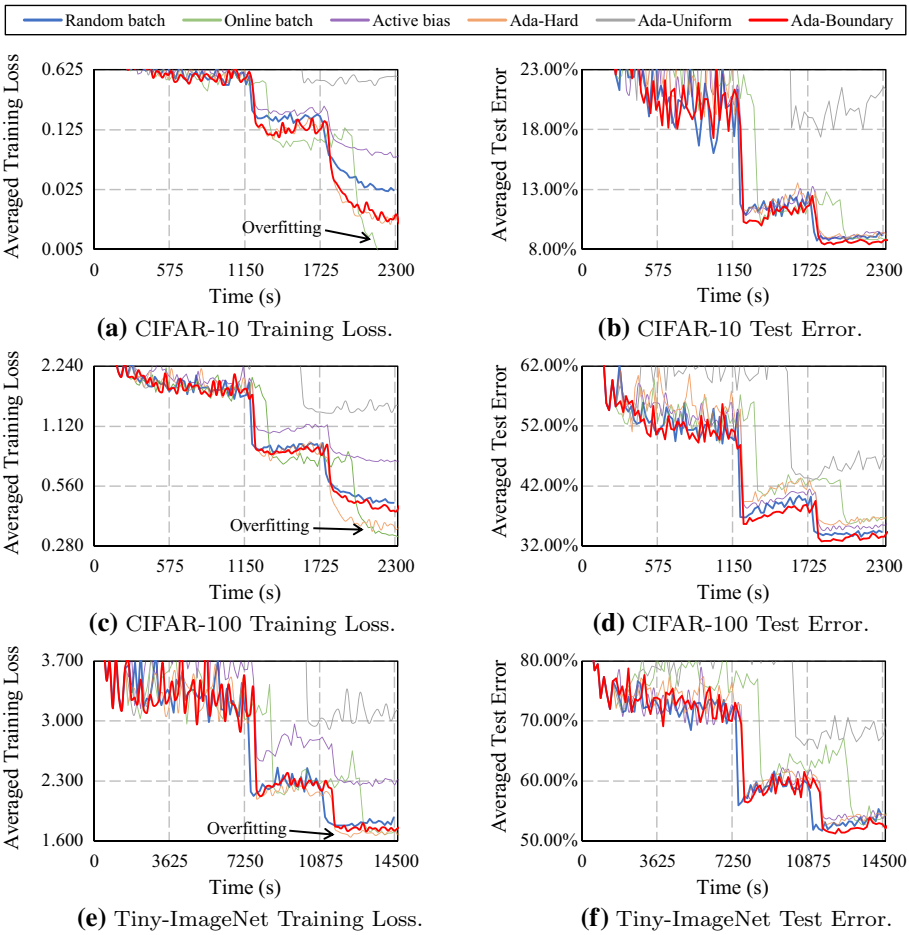


Fig. 6 Convergence curves of six batch selection strategies using **DenseNet with momentum** on CIFAR-10 and CIFAR-100

Table 1 The best test error (%) of six batch selection strategies using DenseNet on three hard data sets in Fig. 6

Method	CIFAR-10	CIFAR-100	Tiny-ImageNet
Random batch	8.71 ± 0.02	33.54 ± 0.11	51.67 ± 0.41
Online batch	8.78 ± 0.09	35.64 ± 0.03	52.66 ± 0.37
Active bias	8.87 ± 0.09	34.48 ± 1.30	53.42 ± 0.10
Ada-Hard	8.90 ± 0.10	35.40 ± 0.50	52.63 ± 1.19
Ada-Uniform	17.37 ± 0.34	43.08 ± 0.083	65.37 ± 0.45
Ada-Boundary	**8.38 ± 0.07**	**32.74 ± 0.10**	**51.17 ± 0.49**

The lowest values are highlighted in bold

Figure 6 shows the convergence curves of training loss and test error for six batch selection strategies on three hard data sets: CIFAR-10, CIFAR-100, and Tiny-ImageNet. In

order to improve legibility, only the curves for the baseline and proposed strategies are dark colored. The best test errors in Fig. 6 are summarized in Table 1. We conducted a convergence analysis of the six batch selection strategies, as follows:

- **CIFAR-10** (relatively hard): Except *Ada-Uniform* and *active bias*, all adaptive batch selections achieved faster convergence speed than *random batch* selection in training loss, but only *Ada-Boundary* converged faster than *random batch* selection in test error. This means that the strategy focused on hard sample results in the overfitting to "too hard" samples, which is indicated by a larger converged test error. Meanwhile, *active bias* was prone to make the network better generalized on test data, considering its test error comparable to that of *random batch* selection despite its much higher training loss. That is, *active bias* resulted in better generalization, but slowed down the training process. Quantitatively, *Ada-Boundary* achieved test error relatively lower by 3.79% (8.71% → 8.38%) than *random batch* selection. In contrast, the test error of *Ada-Hard*, *online batch* selection, and *active bias* was relatively higher by 2.18% (8.71% → 8.90%), 0.80% (8.71% → 8.78%), and 1.84% (8.71% → 8.87%).
- **CIFAR-100** (hard): In both training loss and test error, the convergence curves of all strategies showed similar trends to those of CIFAR-10. However, as the training difficulty increased from CIFAR-10 to CIFAR-100, the overfitting of *Ada-Hard* and *online batch* selection was further exacerbated. This emphasizes the need to consider the samples with appropriate difficulty rather than hard samples. Compared with *random batch* selection, *Ada-Boundary* achieved test error relatively lower by 2.39% (33.54% → 32.74%). On the other hand, the test error of *Ada-Hard*, *online batch* selection, and *active bias* was relatively higher by 5.55% (33.54% → 35.40%), 6.26% (33.54% → 35.64%), and 2.80% (33.54% → 34.48%).
- **Tiny-ImageNet** (hard): The convergence trend was consistent even when the data set became larger and harder. Only *Ada-Boundary* achieved test error relatively lower by 0.97% (51.67% → 51.17%) than *random batch* selection. On the other hand, the test error of *Ada-Hard*, *online batch* selection, and *active bias* was relatively higher by 1.86% (51.67% → 52.63%), 1.92% (51.67% → 52.66%), and 3.39% (51.67% → 53.42%).

In all the cases, the large performance gap between *Ada-Uniform* and other methods were attributed to the high computational cost for updating its empirical sampling distribution and the over-sampling of too hard samples owing to the plethora of easy ones.

5.2 Analysis on easy data sets

We also trained a DenseNet ($L = 25$, $k = 12$) with momentum on the easy data sets: MNIST and Fashion-MNIST. We used the same experimental configuration as in Sect. 5.1, except for the training schedule. Generally, because a small learning rate without decay was preferred for easy data sets (Loshchilov and Hutter 2016; Gao and Jojic 2017), we used a constant learning rate of 0.01 over 80 epochs. Here, the baseline strategy required about 1,880 seconds for all cases.

Figure 7 shows the convergence curves of training loss and test error for six batch selection strategies on MNIST and Fashion-MNIST, and the best test errors in Fig. 7 are summarized in Table 2. We conducted a convergence analysis of the six batch selection strategies, as follows:

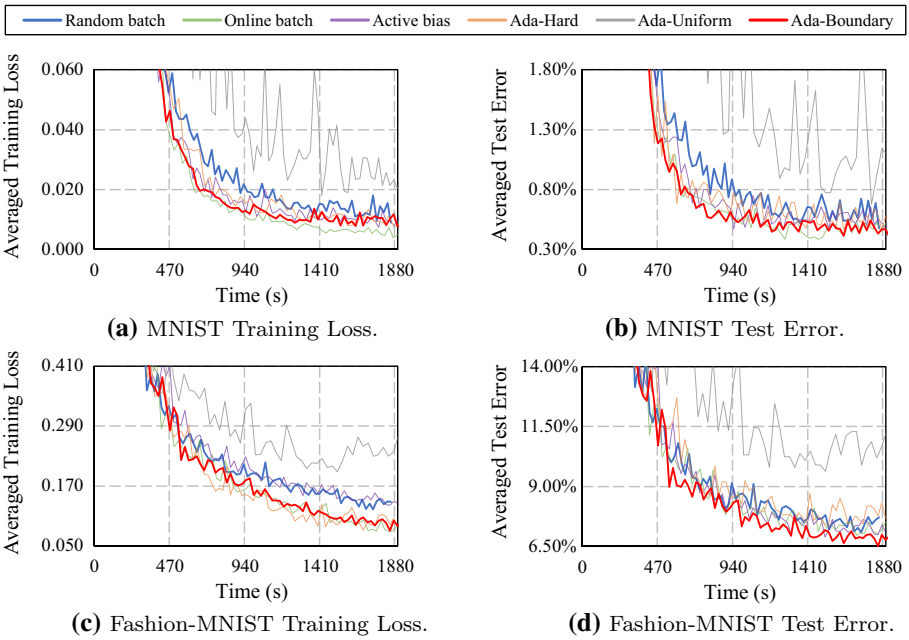


Fig. 7 Convergence curves of six batch selection strategies using **DenseNet with momentum** on MNIST and Fashion-MNIST

Table 2 The best test error (%) of six batch selection strategies using DenseNet on two easy data sets in Fig. 7

Method	MNIST	Fashion-MNIST
Random batch	0.48 ± 0.02	7.08 ± 0.02
Online batch	**0.39 ± 0.02**	6.95 ± 0.04
Active bias	0.46 ± 0.01	6.88 ± 0.08
Ada-Hard	0.46 ± 0.01	7.36 ± 0.17
Ada-Uniform	0.68 ± 0.06	9.64 ± 0.25
Ada-Boundary	0.41 ± 0.02	**6.51 ± 0.04**

The lowest values are highlighted in bold

- **MNIST** (easy): As we clarified in Sect. 1, the hard batch selections, *Ada-Hard* and *online batch* selection, worked well in the easy MNIST data set. They converged faster than *random batch* selection in both training loss and test error. *Ada-Boundary* showed a fast convergence comparable to that of *online batch* selection; the absolute difference of test error between them was only 0.02%, which was almost negligible. Quantitatively, *Ada-Boundary*, *Ada-Hard*, *online batch* selection, and *active bias* achieved test error relatively lower by 14.58% (0.48% → 0.41%), 4.17% (0.48% → 0.46%), 18.75% (0.48% → 0.39%), and 4.17% (0.48% → 0.46%) than *random batch* selection, respectively.
- **Fashion-MNIST** (relatively easy): In both training loss and test error, *Ada-Boundary* achieved significantly faster convergence speed than *random batch* selection. *Ada-Hard* and *online batch* selection tended to weakly overfit to "too hard" samples. Their test

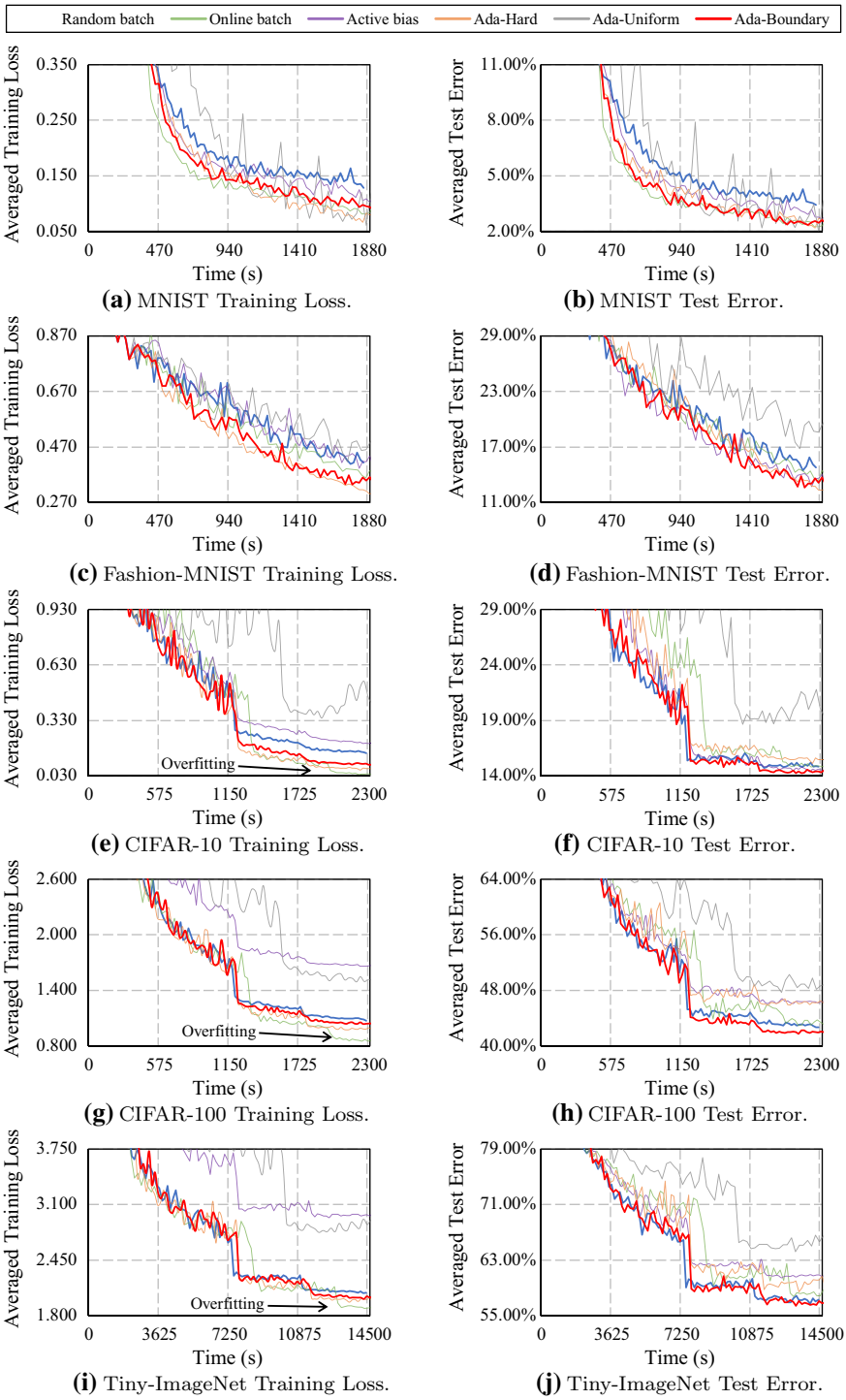
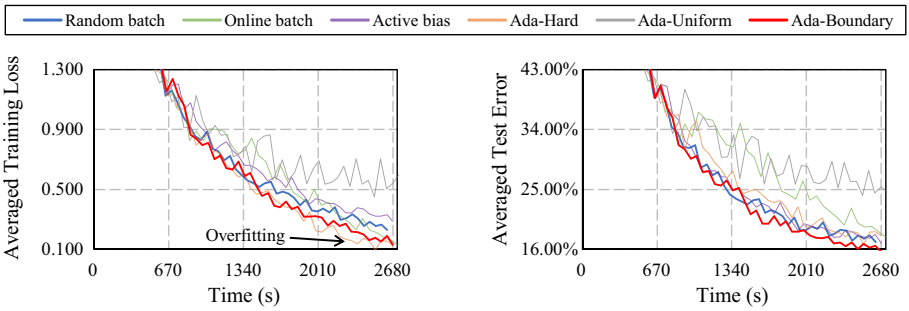


Fig. 8 Convergence curves of six batch selection strategies using **DenseNet with SGD**



(a) CIFAR-10 Training Loss. **(b)** CIFAR-10 Test Error.

Fig. 9 Convergence curves of six batch selection strategies using **WideResNet with SGD** on CIFAR-10

error approached that of *random batch* selection, even when they converged much faster in training loss. In summary, *Ada-Boundary*, *online batch* selection, and *active bias* achieved test error relatively lower by 8.05% (7.08% → 6.51%), 1.84% (7.08% → 6.95%), and 2.82% (7.08% → 6.88%) than *random batch* selection, respectively. On the other hand, *Ada-Hard* was relatively higher by 3.80% (7.08% → 7.36%).

5.3 Generalization of the gradient optimizer

To validate the generality of the optimizer, we repeated the experiment in Sects. 5.1 and 5.2 using the SGD optimizer. Figure 8 shows the convergence curves of six batch selection strategies with the SGD optimizer on all the data sets, and we conducted their convergence analysis as follows:

- **MNIST** (easy): All adaptive batch selection strategies converged faster than *random batch* selection in both training loss and test error. The convergence curve of *Ada-Uniform* tended to fluctuate.
- **Fashion-MNIST** (relatively easy): Except *Ada-Uniform*, all strategies showed comparable performance in both training loss and test error, which were slightly faster than *random batch* selection.
- **CIFAR-10** (relatively hard): Except for *Ada-Uniform* and *active bias*, all adaptive batch selections converged significantly faster than *random batch* selection in training loss. However, due to the overfitting to "too hard" samples, only *Ada-Boundary* achieved much faster convergence speed than other adaptive batch selections in test error.
- **CIFAR-100** (hard): *Ada-Boundary* converged faster than *random batch* selection in both training loss and test error. In contrast, *Ada-Hard* and *online batch* selection suffered from the overfitting issue in test error, even when they converged faster than *random batch* selection in training loss. much faster convergence speed than other adaptive batch selections in test error.
- **Tiny-ImageNet** (hard): Again, similar to the CIFAR data sets, *Ada-Boundary* achieved the lowest test error while expediting the convergence of training loss. *Active bias* converged the slowest in training error, but its test error was comparable to that of the hard batch selections.

In summary, only *Ada-Boundary* succeeded in increasing the convergence speed for all data sets, regardless of the difficulty level. Quantitatively, compared with *random batch* selection,

Table 3 The converged training loss of *Ada-Boundary* with varying s_e in the same configuration as in Sects. 5.1 and 5.2

Method	Converged training loss			
	MNIST	Fashion-MNIST	CIFAR-10	CIFAR-100
$s_e = 2.0$	0.0087	0.1122	0.0106	0.4171
$s_e = 8.0$	0.0083	0.0986	0.0103	0.3520
$s_e = 32.0$	0.0077	0.0803	0.0083	0.3399

Table 4 The best test error (%) of *Ada-Boundary* with varying s_e in the same configuration as in Sects. 5.1 and 5.2

Method	Best test error			
	MNIST	Fashion-MNIST	CIFAR-10	CIFAR-100
$s_e = 2.0$	0.452 ± 0.017	6.870 ± 0.037	$\mathbf{8.383 \pm 0.068}$	$\mathbf{32.740 \pm 0.099}$
$s_e = 8.0$	0.425 ± 0.010	6.795 ± 0.128	8.460 ± 0.097	32.934 ± 0.064
$s_e = 32.0$	$\mathbf{0.412 \pm 0.016}$	$\mathbf{6.505 \pm 0.043}$	8.693 ± 0.111	33.543 ± 0.082

The lowest values are highlighted in bold

Ada-Boundary achieved a significant reduction in test error of 31.80% (3.43% → 2.34%), 12.85% (14.47% → 12.61%), 3.26% (14.72% → 14.24%), 2.11% (42.63% → 41.73%), and 0.58% (56.70% → 56.37%) in MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, and Tiny-ImageNet.

5.4 Generalization of the model

To show the generality of the model, we trained a WideResNet 16-8 (Zagoruyko and Komodakis 2016) on the CIFAR-10 data set for 50 epochs. For the experiment, we used the SGD optimizer and a constant learning rate of 0.01. The other configurations were the same as in Sect. 5.1. Figure 9 shows the convergence curves of six batch selection strategies on CIFAR-10. Here, *Ada-Boundary* also outperformed other batch selection strategies in both training loss and test error. *Ada-Boundary* significantly reduced the test error by 8.38% (17.06% → 15.63%) compared with *random batch* selection. Owing to the overfitting issue, *Ada-Hard* converged slower than *Ada-Boundary* in test error, even when it achieved low training loss comparable to that of *Ada-Boundary*. The slow convergence speed of *online batch* selection in test error is explained by the same reasoning.

5.5 Impact of selection pressure s_e

The selection pressure s_e determines how strongly the boundary samples are selected. The larger the s_e value, the larger the sampling probabilities of the boundary samples, so more boundary samples are chosen for the next mini-batch. On the other hand, as smaller s_e value brings *Ada-Boundary* closer to *random batch* selection. To analyze the impact of the selection pressure s_e and determine the best value, we trained a DenseNet ($L = 25$, $k = 12$) with momentum on our four benchmark data sets with varying s_e values in the same configuration as outlined in Sects. 5.1 and 5.2.

Tables 3 and 4 respectively show the converged training loss and the best test error of *Ada-Boundary* with varying s_e values on four benchmark data sets. For training loss, the convergence speed was accelerated as the s_e value increased. That is, lower training loss was achieved with a larger s_e value (see Table 3). Similarly, for test error, this trend was observed on easy data sets (see MNIST and Fashion-MNIST in Table 4). However, the overexposure to the boundary samples when the large s_e was used incurred an overfitting issue on the hard data sets (see CIFAR-10 and CIFAR-100 in Table 4). When using a large s_e value, the test error increased even when the training loss decreased. This means that the overexposure to only some part of training samples is not beneficial for the generalization of overall training in hard data sets. Therefore, we used $s_e = 32.0$ for easy data sets, and $s_e = 2.0$ for hard data sets in all experiments.

6 Conclusion and future work

In this paper, we proposed a novel and simple adaptive batch selection algorithm, *Ada-Boundary*, that presents the most appropriate samples according to the learning progress of the model. Toward this goal, we defined the distance from a sample to the decision boundary and introduced a quantization method for selecting the samples near the boundary with high probability. We performed extensive experimentation using a DenseNet for five benchmark data sets with varying difficulty levels. The results showed that *Ada-Boundary* significantly accelerated the training process, and was better generalized for hard data sets. When training an easy data set, *Ada-Boundary* showed a fast convergence comparable to that of the state-of-the-art algorithm; when training hard data sets, only *Ada-Boundary* converged significantly faster than *random batch* selection as well as the state-of-the-art algorithm.

The most exciting benefit of *Ada-Boundary* is its potential to reduce the time needed to train a DNN. This becomes more important as the size and complexity of the data increases and can be boosted with recent advances of hardware technology. It can be easily implemented into various optimizers owing to its simplicity. Our immediate future work is to apply *Ada-Boundary* to other types of DNNs, such as RNN (Mikolov et al. 2010) and LSTM (Hochreiter and Schmidhuber 1997), which have a neural structure completely different from that of CNN. In addition, we plan to investigate the relationship between the power of a DNN and the improvement in *Ada-Boundary*.

Acknowledgements This work was partly supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (Ministry of Science and ICT) (No. 2017R1E1A1A01075927) and Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korea Government (MSIT) (No. 2020-0-00862, DB4DL: High-Usability and Performance In-Memory Distributed DBMS for Deep Learning).

References

Bengio, Y., J. Louradour, R. Collobert, & J. Weston (2009). Curriculum learning. In *International Conference on Machine Learning (ICML)*, pp. 41–48.

Chang, H.-S., E. Learned-Miller, & A. McCallum (2017). Active Bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1002–1012.

Chen, B., & Wornell, G. W. (2001). Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *Transactions on Information Theory*, *47*(4), 1423–1443.
Fan, Y., Tian, F., Qin, T., & Liu, T.-Y. (2017). Neural data filter for bootstrapping stochastic gradient descent. In *International Conference on Learning Representation (ICLR)*.
Gao, T., & Jojic, V. (2017). Sample importance in training deep neural networks. https://openreview.net/forum?id=r1IRctqxg.
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge: MIT Press.
Gray, R. M., & Neuhoff, D. L. (1998). Quantization. *Transactions on Information Theory*, *44*(6), 2325–2383.
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.
Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708.
Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pp. 448–456.
Katharopoulos, A., & Fleuret, F. (2018). Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning (ICML)*, pp. 2525–2534.
Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representation (ICLR)*.
Krizhevsky, A., Nair, V., & Hinton, G. (2014). CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105.
Kumar, M. P., Packer, B., & Koller, D. (2010). Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1189–1197.
LeCun, Y. (1998). The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist.
Loshchilov, I., & Hutter, F. (2016). Online batch selection for faster training of neural networks. In *International Conference on Learning Representation (ICLR)*.
Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pp. 1045–1048.
Needell, D., Ward, R., & Srebro, N. (2014). Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1017–1025.
Sachan, M., & Xing, E. (2016). Easy questions first? A case study on curriculum learning for question answering. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 453–463.
Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representation (ICLR)*.
Schmidt, M., Babanezhad, R., Ahmed, M., Defazio, A., Clifton, A., & Sarkar, A. (2015). Non-uniform stochastic average gradient method for training conditional random fields. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 819–828.
Shrivastava, A., Gupta, A., & Girshick, R. (2016). Training region-based object detectors with online hard example mining. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 761–769.
Song, H., Kim, M., & Lee, J.-G. (2019). SELFIE: Refurbishing unclean samples for robust deep learning. In *International Conference on Machine Learning*, pp. 5907–5915.
Song, H., Kim, M., Kim, S., & Lee, J.-G. (2020). Carpe diem, seize the samples uncertain "at the moment" for adaptive batch selection. In *International Conference on Information and Knowledge Management (CIKM)*.
Tsvetkov, Y., Faruqui, M., Ling, W., MacWhinney, B., & Dyer, C. (2016). Learning the curriculum with bayesian optimization for task-specific word representation learning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 130–139.
Wu, C.-Y., Manmatha, R., Smola, A. J., & Krähenbühl, P. (2017). Sampling matters in deep embedding learning. In *International Conference on Computer Vision (ICCV)*, pp. 2840–2848.
Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747.
Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. arXiv:1605.07146.
Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. arXiv:1212.5701.