AUTOSAMPLING: SEARCH FOR EFFECTIVE DATA SAMPLING SCHEDULES

Anonymous authors
Paper under double-blind review

ABSTRACT

Data sampling acts as a pivotal role in training deep learning models. However, an effective sampling schedule is difficult to learn due to its inherent high-dimension as a hyper-parameter. In this paper, we propose the AutoSampling method to automatically learn sampling schedules for model training, which consists of the multi-exploitation step aiming for optimal local sampling schedules and the exploration step for the ideal sampling distribution. More specifically, we achieve sampling schedule search with shortened exploitation cycle to provide enough supervision. In addition, we periodically estimate the sampling distribution from the learned sampling schedules and perturb it to search in the distribution space. The combination of two searches allows us to learn a robust sampling schedule. We apply our AutoSampling method to a variety of image classification tasks illustrating the effectiveness of the proposed method.

1 INTRODUCTION

Data sampling policies can greatly influence the performance of model training in computer vision tasks. For instance, data resampling, reweighting or importance sampling methods promote better model performance by adjusting the training data frequency (Estabrooks et al., 2004; Weiss et al., 2007; Drummond et al., 2003; Bengio et al., 2009; Wang et al., 2019; Johnson & Guestrin, 2018; Katharopoulos & Fleuret, 2018; Byrd & Lipton, 2018). In addition, learning-based methods (Li et al., 2019; Jiang et al., 2017) aim to automatically select or re-weight training data based on feedback from a held-out validation set, and they are shown to be particularly effective for leaning on noisy datasets. These studies show that data are not equal and can be of different values to model training. However, existing works using handcrafted rules to determine how training data are sampled normally rely on some knowledge of the training data, such as their classes or loss values. Such knowledge may not be sufficient to fully determine the value of each data in the entire course of training, as their validity rely on certain assumptions on the dataset which may fail. For example, loss-guided samplers often require dataset to be clean enough to ensure robust training, for otherwise they can lead to sub-optimal training outcome (Shrivastava et al., 2016; Loshchilov & Hutter, 2015). Furthermore, previous methods only adjust sampling inside mini-batches, causing the sampling policy to be only locally optimal(Li et al., 2019; Jiang et al., 2017). Therefore, their effectiveness depends on the size of the mini-batches and is limited by the sampling schedules generated normally by a fixed uniform sampler. It is crucial to explore ways of finding robust sampling policy that requires little assumptions on the dataset and can reach a relatively global optimum.

We attempt to solve the above problems through directly learning a globally optimal sampler. Generally speaking, exploring effective sampling policy through learning, especially for large dataset such as ImageNet, remains an open and nontrivial problem, which we try to tackle in this work. While popular approaches such as deep reinforcement learning (Cubuk et al., 2018; Zhang et al., 2020), Bayesian optimization (Snoek et al., 2015) or simply random search (Bergstra & Bengio, 2012) have already been utilized to tune low-dimensional hyper-parameters like augmentation schedules, their applications in finding a good sampling schedule is not fully exploited. The difficulty of optimizing the sampling schedule lies in its high-dimension, which originates from the massive amount of training data. A typical image classification dataset consists of at least tens of thousand of training images. The dimension of a data augmentation policy is generally only in dozens, and it needs more than ten thousands of training runs (Cubuk et al., 2018) to sample enough long-term rewards to find an optimal augmentation policy. As such, optimizing high-dimensional hyper-parameter like the

sampling schedule may require orders of magnitude more rewards to gather and hence training runs, which would result in extremely slow convergence. Moreover, optimizing such high-dimensional hyper-parameter would also need to update a large agent or policy network offline, which is difficult to train and also contributes to the slow convergence.

To overcome the aforementioned challenges, we propose a data sampling policy search framework, named AutoSampling, to sufficiently learn an optimal sampling schedule. Unlike previous methods, which focus on collecting global rewards and updating hyper-parameters or agents offline, our AutoSampling method shortens the evaluation cycle and collects rewards online. Through this manner, we provide the search process with enough feedback to learn an effective sampling policy. Specifically, the AutoSampling employs a multi-exploitation-and-exploration structure. For the multi-exploitation process, we run evaluation in between mini-batches and record the optimal sampling schedules accordingly for a sufficient number of training iterations to supply enough rewards for learning. In the exploration process, we calculate the histogram of all training images in the recorded sampling schedule, from which we estimate the sampling distribution. Then we perturb the sampling distribution to search in distribution space, and use it to generate new mini-batches for further exploitation. By updating the sampling distribution online in a non-parametric fashion, we avoid updating a huge network which cause learning hard to converge. The proposed method dynamically learns the sampling schedule throughout the entire training process by iterating between two processes, adaptive to varying needs for certain data during different training stages.

So far as we know, we are the first to explore finding robust sampling schedules through learning without any condition(such as cleanness, imbalance or noisiness) on the dataset. We apply AutoSampling to a variety of image classification tasks to search for robust sampling schedules. The experiment results on CIFAR-10/100 (Krizhevsky, 2009) and ImageNet datasets demonstrate the effectiveness of our method.

2 Background

2.1 Related Work

Data sampling is of great significance to deep learning, and has been extensively studied recently. Both human-designed rules and learning-based methods have been applied to find sampling schedules for strong training performance. Approaches with human-designed rules take pre-defined heuristic rules to modify the frequency and order by which training data is presented. In particular, one intuitive method is to resample or reweight data according to their frequencies, difficulties or importance in training (Estabrooks et al., 2004; Weiss et al., 2007; Drummond et al., 2003; Bengio et al., 2009; Lin et al., 2017; Shrivastava et al., 2016; Loshchilov & Hutter, 2015; Wang et al., 2019; Johnson & Guestrin, 2018; Katharopoulos & Fleuret, 2018; Byrd & Lipton, 2018). These methods have been widely used for building robust sampling schedules in imbalanced training or hard mining problems. However, they are often restricted to certain tasks and datasets based on which they are proposed, and their ability to generalize to a broader range of tasks with different data distribution may be limited. In another word, these methods often inexplicitly assume certain conditions on the dataset, such as cleanness or imbalance. Furthermore, these methods lack in flexibility to select training data to fulfill training needs at different learning stages.

In addition, some learning-based methods have been proposed for finding suitable sampling schemes automatically. Deep reinforcement learning has already been applied to find good training data sampler (Fan et al., 2017), but its heavy computational requirement limits its usage on large dataset like ImageNet (Deng et al., 2009). Learning-based methods are also utilized to automatically reweight data during training (Li et al., 2019; Jiang et al., 2017; Ren et al., 2018), but their effectiveness is mainly examined on noisy datasets. Whether or not they can generalize over tasks of other datasets still remain untested.

As for hyper-parameter tuning, popular approaches such as deep reinforcement learning (Cubuk et al., 2018; Zhang et al., 2020), Bayesian optimization (Snoek et al., 2015) or simply random search (Bergstra & Bengio, 2012) have already been utilized to tune low-dimensional hyper-parameters like data augmentation schedules, dropout rates or learning rates and proven to be effective. Nevertheless, they have not been adopted to find good sampling schedule due to its inherent high dimensionality. Some recent works tackle the challenge of optimizing high-dimensional hyper-parameter.

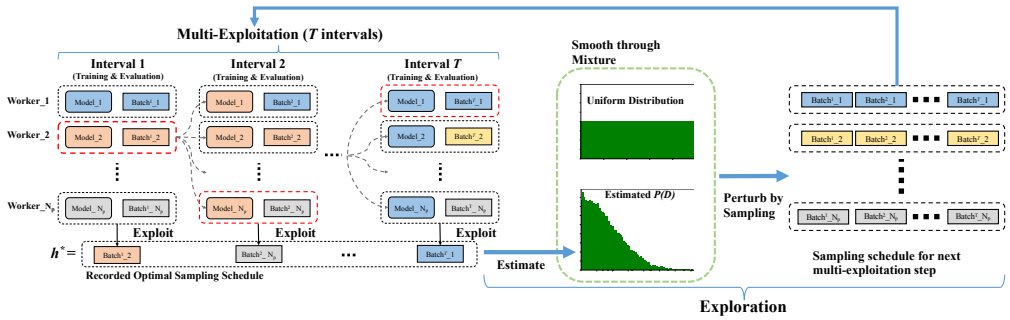

Figure 1: Overview of AutoSampling, which consists of two parts: a) The multi-exploitation step, illustrated by the left half, is the process of learning optimal sampling schedule locally. The same color of model for each worker indicates that the same model weight is cloned into it. b) The exploration step, shown by the right half, is to search in the sampling distribution space. Specifically, we estimate the sampling distribution from the schedules collected in the multi-exploitation step and perturb it to generate new sampling schedules for all workers.

(MacKay et al., 2019) uses structured best-response functions and (Jonathan Lorraine, 2019) achieve this goal through the combinations of the implicit function theorem and efficient inverse Hessian approximations. However, they have not been tested on the task of optimizing sampling schedules

2.2 Population Based Training

Hyper-parameter tuning task can be framed as a bi-level optimization problem with the the following objective functions,

$$\begin{aligned} &\min_{h \in \mathcal{H}} \mathcal{L}(\theta^*, h) \\ &\text{subject to } \theta^* = \arg\max_{\theta \in \Theta} \text{eval}(\theta, h) \end{aligned} \tag{1}$$

where θ represents the model weight and $h = (h_1, h_2, \cdots, h_T)$ is the hyper-parameter schedule for T training intervals. Population based training (PBT) (Jaderberg et al., 2017) solves the bi-level optimization problem by using a population $\mathcal{P}$ of child models in parallel with different hyper-parameter schedules initialized:

$$\mathcal{P} = \{(\theta_i, h_i, t)\}_{i=1}^{N_p} \tag{2}$$

where θ_i, h_i respectively represents the child model weight, the corresponding hyper-parameter schedule for the training interval t on worker i, and N_p is the number of workers. PBT proceeds in intervals, which usually consists of several epochs of model training. During the interval, the population number of models are trained in parallel to finish the lower-level optimization of model weights θ_i. Between intervals, an exploit-and-explore procedure is adopted to conduct the upper-level optimization of the hyper-parameter schedule. At the exploit step for interval t, child models are evaluated on a held-out validation dataset, and the best performing hyper-parameter setting $\mathbf{h}_\mathbf{t}^*$ are recorded, and the weight of the top-performing model are broadcasted to all child models, that is,

$$\begin{aligned} h_t^*, \theta_t^* = &\arg\max_{p_i = (\theta_i, h_i, t) \in \mathcal{P}} \text{eval}(\theta_i, h_i) \\ &\theta^* \to \theta_i, i = 1, \cdots, N_p \end{aligned} \tag{3}$$

At the explore step, new hyper-parameter schedules for the next interval $t+1$ are initialized with different random seeds on all workers to search in the hyper-parameter space. The next exploit-and-explore cycle will then be continued. In the end, the top-performing hyper-parameter schedule $h^* = (h_1^*, h_2^*, \cdots, h_T^*)$ can be obtained, which can be transferred to other models.

PBT is applied to tune low-dimenisal hyper-parameters such as data augmentation schedules (Ho et al., 2019; Jaderberg et al., 2017). However, it cannot be directly used for finding sampling strategies due to the high dimension. For instance, in PBA (Ho et al., 2019) 3200 epochs of training are needed to optimize 60 hyper-parameters for data augmentation, and a linear up-scaling to learning sampling schedule in CIFAR-100 (Krizhevsky, 2009) would require 2.67 million epochs, which are practically infeasible.

Algorithm 1: The Multi-Exploitation Step

Input: Training dataset D, population $\mathcal{P}$, number of workers N_p, number of exploitation intervals T, exploitation interval length N_s
Initialize $\mathbf{H}^* \leftarrow ()$
for $t = 1$ **to** T **do**
  for $j = 1$ **to** N_s **do**
    for $(\theta_i, h_{t,i}, t) \in \mathcal{P}$ **do**
      $\theta_i \leftarrow \bigtriangledown\mathcal{L}(\theta_i, h_{t,i})$ ▷ update the weight of child model i
    end for
    $h_t^*, \theta_t^* = \arg\max_{\mathcal{P}} \text{eval}(\theta_i, h_i)$
    $\mathbf{H}^* \leftarrow \mathbf{H}^* + h_t^*$ ▷ update the sampling for child model i
    for $i = 1$ **to** N_p **do**
      $\theta_i \leftarrow \theta_t^*$ ▷ clone the optimal weight
    end for
  end for
end for
Return $\mathbf{H}^*, \mathcal{P}$

3 AUTOSAMPLING WITH SEARCHING

In this section, we introduce our AutoSampling, which leverages the PBT framework to search for robust sampling schedules. Unlike PBT method which applies the exploitation and exploration at the same time, our AutoSampling adopts a multi-exploitation-and-exploration structure. In the multi-exploitation step, we take the exploitation step between mini-batches for sufficient feedback collection and update to search locally. In the exploration loop, we estimate the sampling distribution using sampling schedule updates accumulated during the multi-exploitation step, slightly perturb it for all workers to search globally in the distribution space, and generate new training mini-batches with the changed distribution. AutoSampling starts with the uniform distribution on all workers with different random seeds, and takes certain number of explorations to reach an ideal sampling distribution and the corresponding sampling schedule, which can be transferred to other models. It should also be pointed out that the concrete exploit and explore steps in AutoSampling are also different from those of PBT. The overview of our method is illustrated in Fig.1.

3.1 MULTI-EXPLOITATION BY SEARCHING IN THE DATA SPACE

In the multi-exploitation step, we aim to search locally in the data space for the optimal sampling schedule. Specifically, we wish to learn the hyper-parameters in T training intervals. In each interval, there are a population $\mathcal{P}$ of N_p child models. Denote $\mathbf{h}_{t,i}$ as the training data in the tth training interval for the ith child model. When all of the T training intervals for the ith child model is considered, we have $\mathbf{H}_i = \{\mathbf{h}_{t,i} | t = 1, \ldots, T\} = \{x_1, \cdots, x_N\}$, where N is the number of training data for the multi-exploitation step. Each interval consists of a certain number of training iterations that is also equivalent to a certain number of training mini-batches. AutoSampling is expected to produce a sequence of training samples, denoted by $\mathbf{H}^*$, so that a given model is optimally trained. The population $\{\mathbf{H}_i\}$ forms the local search space, from which we aim to search for an optimal sampling schedule $\mathbf{H}^*$.

Given the population $\mathcal{P}$, we train them in parallel on N_p workers. Once an interval of data $\mathbf{h}_{t,i}$ containing N_s training samples have been used for training, we take the exploit step and evaluate all child models. We record the top-performing weight and sub-schedule for the current interval t, in particular,

$$h_t^*, \theta_t^* = \underset{p_i = (\theta_i, h_i, t) \in \mathcal{P}}{\arg\max} \text{eval}(\theta_i, h_{t,i}) \tag{4}$$

On the other hand, we update all child model weights of $\mathcal{P}$ by cloning into them with the top-performing weight θ_t^* so we can continue searching based on the more promising child. We will continue the exploit steps through the whole training process, and output the recorded optimal sampling schedule $\mathbf{H}^* = \{h_1^*, h_2^*, \cdots, h_T^*\}$. By using exploitation interval of mini-batches rather than epochs or even entire training runs adopted by earlier methods, AutoSampling may yield a better and more robust sampling schedule in one multi-exploitation step. We describe the multi-exploitation with details in Alg.1.

Algorithm 2: Search based AuoSampling

Input: Training dataset D, population size N_p
Initialize $H^* \leftarrow ()$, $P(D) \leftarrow$ uniform(D) and initialize child models $\theta_1, \cdots, \theta_{N_p}$
while not end of training **do**
  for $i = 1$ **to** N_p **do**
    Sample h_i from Mixture($\log(P(D) + \beta)$, $N_u \times$ uniform(D))
  end for
  Initialize $\mathcal{P} = \{(\theta_i, h_i, t)\}_{i=1}^{N_p}$
  $\mathbf{H}^*, \mathcal{P} \leftarrow$ Alg.1
  Estimate $P(D)$ according to Equation (5)
  Update $P(D)$ according to Equation (6)
  $H^* \leftarrow H^* + \mathbf{H}^*$
end while
Return H^*, $P(D)$

3.2 Exploration by Searching in Sampling Distribution Space

In the exploration, we intend to apply the explore step to search in sampling distribution space. We first record the optimal sampling schedule h^* produced by the multi-exploitation step. Then, we estimate the underlying sampling distribution $P(D)$ from the data histogram, that is, for $x \in D$

$$P(x) = \frac{count(x \in \mathbf{H}^*)}{\sum_{x \in D} count(x \in \mathbf{H}^*)} \tag{5}$$

where $count(x \in \mathbf{H}^*)$ denotes the number of x's appearances in $\mathbf{H}^*$. In the explore step, we further perturb the $P(D)$ for each worker and generate the corresponding sampling schedules for the later exploitation step. We introduce perturbations into the generated schedules by simply sampling from the multinomial distribution $P(D)$ using different random seeds. However, in our experiments, we observe that the distribution produced by $P(D)$ tends to be extremely skewed and a majority of the data actually have zero frequencies. Such skewness causes highly imbalanced training mini-batches, and therefore destabilizes subsequent model training.

Distribution Smoothing To tackle the above issue, we first smooth $P(D)$ through the logarithmic function, and then apply a probability mixture with uniform distributions. In particular for the dataset D,

$$P'(D) = Mixture(\log(P(D) + \beta), N_u \times \text{uniform}(D)) \tag{6}$$

where $\beta \geq 1$ is the smoothing factor and $N_u \times$ uniform(D) denotes N_u uniform multinomial distributions on the dataset D. The smoothing through the $\log$ function can greatly reduce the skewness, however, $\log(P(D) + \beta)$ may still contain zero probabilities for some training data, resulting in their absence in training and causing unstable training. Therefore, we further smooth it through an probability mixture with N_u uniform distribution uniform(D) to ensure presence of all data. This is equivalent to adding N_u epochs of training data to the batch sampled by $P'(D)$ and shuffling the union, which also adds more perturbations into the new sampling schedules. Once we have new diverse sampling schedules for the population, we then proceed to the exploit step.

We continue this alternation between multi-exploitation and exploration steps until the end of training. Note that to generate sampling schedule for the first multi-exploitation run, we initialize $P(D)$ to be an uniform multinomial distribution. We output a sequence of optimal sampling schedules $H^* = (\mathbf{H}_1^*, \cdots, \mathbf{H}_N^*)$ for N alternations. The entire process is illustrated in details in Alg.2.

Note that unlike population-based augmentation (Ho et al., 2019), which carefully normalizes the ranking between workers to update an augmentation agent, the proposed method requires only recording the top schedules and updates the sampler in a non-parametric fashion.

4 Experiments

In this section, we present comprehensive experiments on various datasets to illustrate the performance of AutoSampling, and also demonstrate the process of progressively learning better sampling distribution.

Table 1: Performance on CIFAR-100 using different configurations of AutoSampling and baselines. Worker is the number of worker used and Step is the exploitation interval in terms of batches.

NETWORK	WORKER	STEP	EXPLORATION TYPE	TOP1(%)
RESNET18 (ZHANG ET AL., 2019)	-	-	-	78.34±0.05
RESNET18	1	-	UNIFORM	78.46±0.035
RESNET18	20	80 BATCHES	RANDOM	78.76±0.003
RESNET18	20	20 BATCHES	RANDOM	78.99±0.003
RESNET18	80	20 BATCHES	RANDOM	79.09±0.017
RESNET18	20	20 BATCHES	MIXTURE	**79.44**±0.020
RESNET50 (JIN ET AL., 2019)	-	-	-	79.34
RESNET50	1	-	UNIFORM	79.70±0.023
RESNET50	20	80 BATCHES	RANDOM	80.55±0.129
RESNET50	20	20 BATCHES	RANDOM	81.05±0.064
RESNET50	80	20 BATCHES	RANDOM	81.19±0.072
RESNET50	20	20 BATCHES	MIXTURE	**81.53**±0.088
DENSENET121	1	-	UNIFORM	80.13±0.028
DENSENET121	20	80 BATCHES	RANDOM	80.62±0.694
DENSENET121	20	20 BATCHES	RANDOM	**81.11**±0.127
DENSENET121	80	20 BATCHES	RANDOM	81.08±0.021
DENSENET121	20	20 BATCHES	MIXTURE	80.97±0.006

Table 2: Experiments on CIFAR-10.

NETWORK	EXPLORATION TYPE	TOP1(%)
RESNET18	UNIFORM	93.01±0.009
RESNET18	RANDOM	**95.86**±0.003
RESNET18	MIXTURE	95.80±0.018
RESNET50	UNIFORM	93.60±0.004
RESNET50	RANDOM	**96.10**±0.002
RESNET50	MIXTURE	96.09±0.070

Table 3: Experiments on ImageNet.

NETWORK	EXPLORATION TYPE	TOP1(%)
RESNET18	UNIFORM	70.38
RESNET18	RANDOM	72.07
RESNET18	MIXTURE	**72.91**
RESNET34	UNIFORM	74.09
RESNET34	RANDOM	76.11
RESNET34	MIXTURE	**76.92**

4.1 ABLATION STUDY

For this part, we gradually build up and test components of AutoSampling on CIFAR-100, and then examine their performance on CIFAR-10 and ImageNet datasets. The training implementation details and computational complexity can be found in Appendix A.1.

Adding Workers To look into the influence of the worker number, we conduct experiments using worker numbers of 1, 20, 80 respectively with the same setting of $N_s = 80$ and the random exploration method. With the worker number of 1, the experiment is simply the normal model training using stochastic gradient descent. To show the competitiveness of our baselines, we also include state-of-the-art results on CIFAR-100 with ResNet-18 and ResNet-50 (Zhang et al., 2019; Jin et al., 2019). We notice significant performance gain using the worker number of 20 for ResNet-18, ResNet-50 and DenseNet-121 (He et al., 2015; Huang et al., 2017), as illustrated in Table 1. However, we note that increasing worker number from 20 to 80 only brings marginal performance gains across various model structures, as shown in Table 1. Therefore, we set the worker number to be 20 for the rest of the experiments.

Shortening Exploitation Intervals To study the effects of the shortened exploitation interval, we run experiments using different exploitation intervals of 20 and 80 batches respectively. As shown in Table 1, models with the shorter exploitation interval of 20 perform better than the one with the longer exploitation interval across all three network structures. This result adheres to our intuition that shorter exploitation interval can encourage the sampler to accumulate more rewards to learn better sampling schedules. For the rest of this section we keep the exploitation interval of $N_s = 20$.

Adding Exploration Method We further add mixture as the exploration method to see the effects of learning the underlying sampling distribution, and completing the proposed method. As shown in Table 1, with ResNet-18 and ResNet-50 we push performance higher with the mixture exploration, and outperform the baseline method by about 1 and 1.8 percentage on CIFAR-100 respectively. However, we found that it is not true in the case of DenseNet-121 and this case may be attributed to the bigger capacity of DenseNet-121.

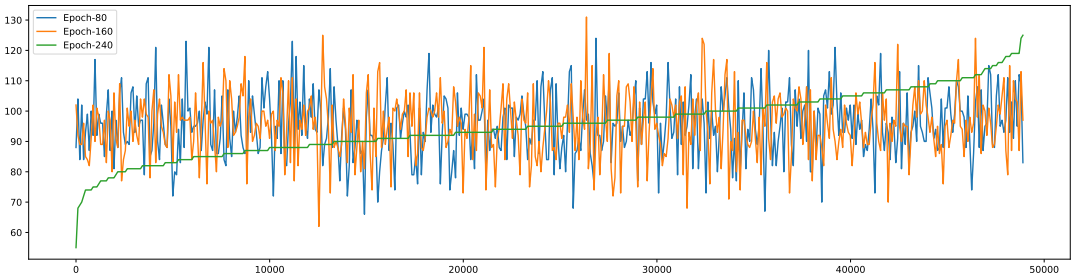

Figure 2: The comparison between histograms estimated from the sampling schedules of Epoch 80, 160 and 240 from CIFAR-100 training with ResNet-18. We divide the training data into 500 segments and calculate the histograms from frequencies of data appearances of all segments. We reorder the X-axis based on the ranking of frequencies from epoch 240.

Table 4: Static vs dynamic sampling schedule on CIFAR-100 (%)

NETWORK	SAMPLING TYPE		
	UNIFORM	STATIC	DYNAMIC
RESNET18	78.46±0.035	78.80±0.007	**79.44**±0.020
RESNET50	79.70±0.023	80.21±0.014	**81.53**±0.088

Generalization Over Datasets In addition, we experiment on other datasets. We report the results on CIFAR10 in Table 2 and the results of ResNet-18, ResNet-34 on ImageNet in Table 3. For CIFAR-10, we notice that the mixture and random exploration methods are comparable while both outperforming the uniform baseline, and we believe it is due to the simplicity of the dataset. In the more challenging ImageNet, the mixture exploration outperforms the random exploration by a clear margin. We also compare our AutoSampling with some recent non-uniform sampling methods on CIFAR-100, which can be found in Appendix A.2

4.2 STATIC VS DYNAMIC SCHEDULES

We aim to see if the final sampling distribution estimated by our AutoSampling is sufficient to produce robust sampling schedules. In another word, we wish to know training with the AutoSampling is either a process of learning a robust sampling distribution, or a process of dynamically adjusting the sampling schedule for optimal training.

To this end, we conduct training using different sampling schedules. First, we calculate the sampling distribution estimated throughout the learning steps of AutoSampling, and use it to generate the sampling schedule of a full training process, which we denote as STATIC. Moreover, we denote the sampling schedule learned using AutoSampling as DYNAMIC, since AutoSampling dynamically adjust the sampling schedule alongside the training process. Finally, we denote the baseline method as UNIFORM, which uses the sampling schedule generated from uniform distribution.

We report results on CIFAR-100 with ResNet-18 and ResNet-50 in Table 5. Model trained with STATIC sampling schedules exceed the baseline UNIFORM significantly, indicating the superiority of the learned sampling distribution over the uniform distribution. It shows the ability of AutoSampling to learn good sampling distribution. Nonetheless, note that models trained with DYNAMIC sampling schedules outperforms models trained with STATIC, by a margin bigger than the one between STATIC and UNIFORM. This result shows the fact that despite the AutoSampling's capability of learning good sampling distribution, its flexibility during training matters even more. Moreover, this phenomenon also indicates that models at different stages of learning process may require different sampling distributions to achieve optimal training. One single sampling distribution, even gradually estimated using AutoSampling, seems incapable of covering the needs from different learning stages. We plot the sampling histograms estimated from schedules of different learning stages with ResNet-18 on CIFAR-100 in Fig.2, showing the differences between optimized sampling distributions from different epochs.

4.3 ANALYZING SAMPLING SCHEDULES LEARNED BY AUTOSAMPLING

To further investigate the sampling schedule learned by AutoSampling, we review the images at the tail and head part of the sampling spectrum. In particular, given a sampling schedule learned we rank all images based on their appearances in training. Training images at the top and bottom of the order

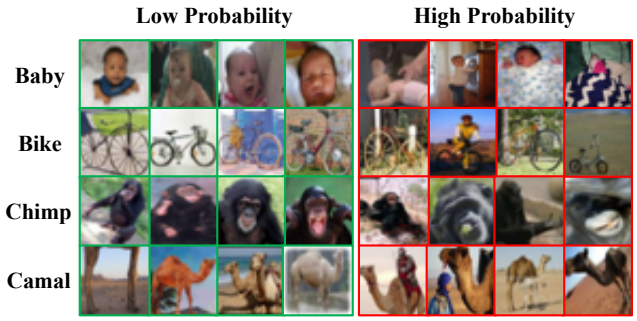

Figure 3: Example images on the head and tail of the sampling spectrum. The images on the left are the ones with low sampling probability, while the images on the right more likely to be sampled. We obtain these images using AutoSampling with the ResNet-18 model on CIFAR-100.

Table 5: Transfer of sampling distributions learned using three model structures onto ResNet-50 on CIFAR-100 (%). UNIFORM denotes the baseline result using uniform sampling distribution.

NETWORK	SAMPLING SCHEDULE SOURCE			
	UNIFORM	RESNET18	RESNET50	DENSENET121
RESNET50	79.70±0.023	80.27±0.014	80.21±0.014	80.47±0.194

are extracted, corresponding to high and low probabilities of being sampled respectively. In Fig.3, we show 4 classes of exemplary images. Conforming to our presumption, the sampling probability seems to indicate the difficulty of each training image, as we can see from Fig.3. The images of low probability tend to have clearer imagery features enabling easy recognition, while on the other hand the images of high probability tend to be more obscure. In addition, we notice the images of low probability also contain low quality images. For instance, in Fig.3 the leftmost image of CAMAL class contains only legs and is not helpful. This shows that AutoSampling can potentially rule out problematic training data for better training.

Furthermore, we examine the transfer ability of sampling distributions learned by AutoSampling to other network structures. Specifically, we run training on ResNet-50 (He et al., 2015) using STATIC sampling schedule generated by three distributions learned by AutoSampling on three different model structures. As shown in Table 5, using sampling schedules learned by AutoSampling from other structures, we demonstrate similar improvement over the UNIFORM baseline. This result, combined with the above observation on images of different sampling probability, indicates that there may exist a common optimal sampling schedule, which is determined by the intrinsic property of the data rather than the model being optimized. Our AutoSampling is an effort to gradually converge to such an optimal schedule.

4.4 DISCUSSIONS

The experimental results and observations from Section 4.2 and 4.3 shred light on the possible existence of an optimal sampling schedule, which relies only on the intrinsic property of the data and the learning stage of the model, regardless of the specific model structure. Once obtained, the optimal sampling schedule can be generalized over other model structures for robust training. Our AutoSampling represents a practical way to approximate such an optimal sampling schedule through learning. Although AutoSampling requires relatively large amount of computing resources to find a robust sampler, we want to point out that the efficiency of our method can be improved through better training techniques. Moreover, the fascinating possibility of an optimal sampling schedule may indicate more efficient sampling heuristics if one can effectively determine data significance based on its property and the training stage.

5 CONCLUSIONS

In this paper, we introduce a new search based AutoSampling scheme to overcome the issue of insufficient rewards for optimizing high-dimensional sampling hyper-parameter by utilizing a shorter period of reward collection to ensure enough reward. We use a shortened exploitation interval to search in the local data space and provide sufficient rewards. For the exploration step, we estimate sampling distribution from searched sampling schedule and perturb it to search in the distribution space. We test our method on various datasets with different networks, and it consistently outperforms the baseline methods a lot across different benchmarks.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012. ISSN 1532-4435.

Jonathon Byrd and Zachary C. Lipton. Weighted risk minimization & deep learning. *CoRR*, abs/1812.03372, 2018. URL http://arxiv.org/abs/1812.03372.

Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018. URL http://arxiv.org/abs/1805.09501.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Chris Drummond, Robert C Holte, et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, volume 11, pp. 1–8. Citeseer, 2003.

Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational intelligence*, 20(1):18–36, 2004.

Yang Fan, Fei Tian, Tao Qin, Jiang Bian, and Tie-Yan Liu. Learning what data to learn. *CoRR*, abs/1702.08635, 2017. URL http://arxiv.org/abs/1702.08635.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Daniel Ho, Eric Liang, Ion Stoica, Pieter Abbeel, and Xi Chen. Population based augmentation: Efficient learning of augmentation policy schedules. *CoRR*, abs/1905.05393, 2019. URL http://arxiv.org/abs/1905.05393.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL http://arxiv.org/abs/1711.09846.

Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Regularizing very deep neural networks on corrupted labels. *CoRR*, abs/1712.05055, 2017. URL http://arxiv.org/abs/1712.05055.

Xiao Jin, Baoyun Peng, Yichao Wu, Yu Liu, Jiaheng Liu, Ding Liang, Junjie Yan, and Xiaolin Hu. Knowledge distillation via route constrained optimization. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

Tyler B Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 7265–7275. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7957.

David Duvenaud Jonathan Lorraine, Paul Vicol. Optimizing millions of hyperparameters by implicit differentiation. *proceedings of AISATS*, 11, 2019.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. *CoRR*, abs/1803.00942, 2018. URL http://arxiv.org/abs/1803.00942.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Zhenmao Li, Yichao Wu, Ken Chen, Yudong Wu, Shunfeng Zhou, Jiaheng Liu, and Junjie Yan. LAW: learning to auto weight. *CoRR*, abs/1905.11058, 2019. URL http://arxiv.org/abs/1905.11058.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. *CoRR*, abs/1511.06343, 2015. URL http://arxiv.org/abs/1511.06343.

Matthew MacKay, Paul Vicol, Jon Lorraine, and Roger Grosse David Duvenaud. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *proceedings of ICLR*, abs/1903.03088, 2019. URL https://arxiv.org/abs/1903.03088.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *CoRR*, abs/1803.09050, 2018. URL http://arxiv.org/abs/1803.09050.

Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2171–2180, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/snoek15.html.

Yiru Wang, Weihao Gan, Jie Yang, Wei Wu, and Junjie Yan. Dynamic curriculum learning for imbalanced data classification. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

Gary M Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? *Dmin*, 7(35-41):24, 2007.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

Michael R. Zhang, James Lucas, Geoffrey E. Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. *CoRR*, abs/1907.08610, 2019. URL http://arxiv.org/abs/1907.08610.

Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxdUySKvS.

A APPENDIX

A.1 IMPLEMENTATION DETAILS

Experiments on CIFAR We use the same training configuration for both CIFAR-100 and CIFAR-10 datasets, which both consist of 50000 training images. In particular, for model training we use the base learning rate of 0.1 and a step decay learning rate schedule where the learning rate is divided by 10 after each 60 epochs. We run the experiments for 240 epochs. In addition, we set the training batch size to be 128 per worker, and each worker is for one GPU card.

We run the explore step for each $N_u + 1$ epochs with $N_u = 3$, but note that we take the first explore step after the initial 20 epochs to better accumulate enough rewards. The experiments require 4800 epochs of training for 20 workers, and roughly 14 hours of training time.

Experiments on ImageNet For ImageNet which consists of 1.28 million training images, we adopted the base learning rate of 0.2 and a cosine decay learning rate schedule. We run the experiments with 100 epochs of training. For each worker we utilize eight GPU cards and a total batch size of 512. Eight workers are used for all ImageNet experiments, and the rest of the setting adheres to that of CIFAR experiments. In addition, we utilize FP16 computation to achieve faster training, which has almost no drop in accuracy in practice. The experiments require 800 epochs of training for 8 workers, and roughly 4 days of training time.

A.2 COMPARISON WITH EXISTING SAMPLING METHODS

To better illustrate the effectiveness of our AutoSampling method, we conduct experiments in comparison with recent non-uniform sampling methods DLIS (Johnson & Guestrin, 2018) and RAIS (Katharopoulos & Fleuret, 2018). DLIS (Johnson & Guestrin, 2018) achieves faster convergence by selecting data reducing gradient norm variance, while RAIS (Katharopoulos & Fleuret, 2018) does so through approximating the ideal sampling distribution using robust optimization. The comparison is recorded in Table 6.

First, we run AutoSampling using Wide Resnet-28-2 (Zagoruyko & Komodakis, 2016) on CIFAR-100 with the training setting aligned roughly to (Katharopoulos & Fleuret, 2018). AutoSampling achievs improvement of roughly 3 percentage points (73.37$\pm$1.09% $\rightarrow$ 76.24$\pm$1.02%), while Katharopoulos & Fleuret shows improvement of 2 percentage points (66.0% $\rightarrow$ 68.0 %). Second, we report the comparison between AutoSampling and RAIS on CIFAR-100. Johnson & Guestrin shows no improvement (76.4% $\rightarrow$ 76.4 %) on accuracy and 0.027 (0.989 $\rightarrow$ 0.962) decrease in validation loss, while our method shows improvement of 0.008 (78.6% $\rightarrow$ 79.4%) on accuracy and 0.014 (0.886 $\rightarrow$ 0.872) decrease in validation loss. As such, our method demonstrates significant improvements over existing non-uniform sampling methods.

Table 6: Comparisons between AutoSampling and existing sampling methods on CIFAR-100

Methods	Network	Baseline (%)	With method (%)	Improvement (%)
DLIS	WRN-28-2	66.0	68.0	2.0
AutoSampling (ours)	WRN-28-2	73.37$\pm$1.09	76.24$\pm$1.02	**2.87**
RAIS	ResNet18	76.4	76.4	0.0
AutoSampling (ours)	ResNet18	78.46$\pm$0.035	79.44$\pm$0.020	**0.98**