Adaptive Dataset Sampling by Deep Reinforcement Learning

Jaerin Lee*

***Abstract*— In most of the deep learning applications, the training dataset is partitioned into mini-batches before being fed into the trained network. Although the size and the quality of datasets have been continuously enhanced, sampling strategy of which have drawn much less attention of the communities. We argue that the generalization capability of a trained model can be improved by modifying the order of the samples of the training dataset. In this work, we propose an adaptive sampler that is aware of history of previous samples and the state of the neural network being trained. We utilize the recent advances of deep reinforcement learning to formalize a Markov Decision Process (MDP) of dataset sampling, and design a stochastic sampling policy using recurrent neural networks. Our method using adaptive sampler is both model-agnostic and task-agnostic, i.e., it can be applied to any deep supervised learning framework. We demonstrate the effectiveness of our adaptive sampler in CIFAR-10 image classification benchmark. An image classifier model trained with our sampler shows better test set accuracy than the one equipped with a random or a sequential sampler.**

I. Introduction

Deep learning is a data-driven framework in statistics and machine learning that extensively uses massively multi-parameter nonlinear function approximators to model the structure of data. These models, also referred to as artificial neural networks (ANNs), have been played a key role in recent advances in computer vision [10, 14], in natural language processing (NLP) [5, 20, 31], as well as in reinforcement learning [9, 17, 23]. A typical deep learning methodology is to fit a delicately designed ANN model with a given set of data by running a gradient-based iterative algorithm over a scalar objective.

In most of the applications, deep models are notoriously data-hungry and training them requires tremendous amount of data. For instance, ImageNet dataset for image classification task [21] contains one million images and exceeds 130 GB in total. Mahajan et al. [16] report that the same classifier model can achieve even better accuracy when trained with 3.5 billion images. On the other hand, GPT-2 [20], a state-of-the-art language model for NLP tasks, is trained with 8 million documents with a total of 40 GB of corpus, yet the authors of [20] report that the model is still suboptimal and requires more data for better performance.

Commercial off-the-shelf hardware that runs deep learning algorithm has limited memory capacity. Only a small fraction of the dataset can reside in the machine at the same time, considering the size of millions of parameters of a deep model and their gradients. Therefore, it is natural to partition the dataset into small batches of instances, and train the network with a single batch at a time. Introducing mini-batches adds randomness to the gradient update; however, the convergence is still guaranteed by the stochastic approximation theory [3].

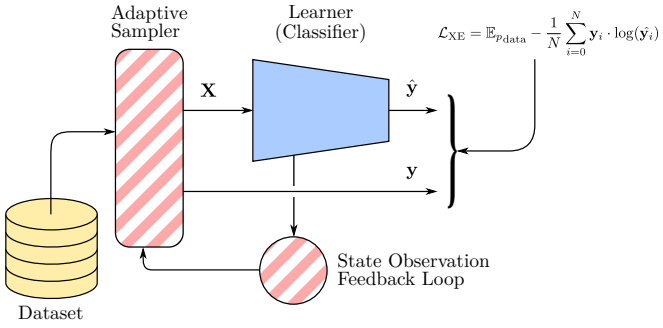

Fig. 1. **Overview of our adaptive sampling framework**. In each training iteration, the active sampler samples a mini-batch of instances from the given dataset and feed these into the learner network. The sampler receives the current state of the learner network and keeps track of that state and the sequence of previously sampled items. The learner network is trained independently to the sampler. Therefore, this framework can be used with any supervised deep learning application.

There have been continuous improvements in building larger and of higher quality datasets [1, 20, 21] in the field of deep learning. However, there are seldom a discourse on the significance of the sampling techniques. Choosing a mini-batch from a given dataset is almost always done by random shuffling [10, 17, 20]. In this work, we emphasize that determining how to sample is a key to provide generalization capability of the model, and to stabilize the training.

There are both empirical [26] and theoretical [19] evidence that the variance of the stochastic gradient of a mini-batch is inversely proportional to the size of the batch. Let θ_t be a network parameter at iteration t. Let $\mathcal{B}_t$ be a sampled mini-batch at t. Let $l(\cdot)$ be a true scalar loss function and let $l_{\mathcal{B}_t}(\cdot)$ be a scalar loss function calculated on the batch $\mathcal{B}_t$. The gradient $\nabla_\theta l(\theta_t)$ and $\nabla_\theta l_{\mathcal{B}_t}(\theta_t)$ have a following statistical relationship. The stochastic gradient estimator is an unbiased estimator for the true gradient, i.e.,

$$\mathbb{E}^{\mathcal{B}}\{\nabla_\theta l_{\mathcal{B}}(\theta_t)\} = \nabla_\theta l(\theta_t). \tag{1}$$

The variance of the stochastic gradient is defined as

$$\mathrm{Var}^{\mathcal{B}}(\nabla_\theta l_{\mathcal{B}}(\theta_t)) := \mathbb{E}^{\mathcal{B}}\|\nabla_\theta l_{\mathcal{B}}(\theta_t)\|^2 - \|\mathbb{E}^{\mathcal{B}}\nabla_\theta l_{\mathcal{B}}(\theta_t)\|^2. \tag{2}$$

Then, the ratio between the variance of the gradient from a mini-batch and that from the entire dataset is inversely proportional to the size of the batch [19, 26], i.e.,

$$\mathrm{Var}^{\mathcal{B}}(\nabla_\theta l_{\mathcal{B}}(\theta_t)) \propto \frac{|\mathcal{D}|^2}{|\mathcal{B}|}\mathrm{Var}(\nabla_\theta l(\theta_t)), \tag{3}$$

*The author is with the Department of Electrical and Computer Engineering, Seoul National University. Student ID: 2019-20239. ironjr@snu.ac.kr

†This work was conducted as a final project of Stochastic Control and Reinforcement Learning, Spring 2020 at Seoul National University.

where $\mathcal{D}$ is the training dataset. The randomness of the right-hand side comes from θ_t, which is derived from the sequence of mini-batches sampled up to iteration $t-1$.

CIFAR-10 [13] is a widely used benchmark in computer vision containing 50k training and 10k test images. In typical applications [10], the dataset is partitioned into mini-batches of size 128. Plugging the numbers in Eq. (3) gives a factor of 2.0×10^7. The huge size indicates that reducing the variance of the gradient from each mini-batch can help the stability of the training process. Introducing prior knowledge is an effective tool to leverage between bias-variance tradeoff in Bayesian statistics [8].

From this intuition we propose a novel adaptive sampler that is aware of both the state of the trained network and the sampling history when selecting the next mini-batch. Our method is model-agnostic, as well as task-agnostic, meaning that our sampler can be used in *any* supervised deep learning application regardless of the dataset, the model, the loss and the optimization algorithm. Our contributions are three-fold.

- We quantitatively show that the dataset sampling strategy is essential in reducing the generalization error in deep learning framework.
- We design the adaptive sampler using deep reinforcement learning. To the best of our knowledge, this is the first attempt to use such technology in the sampling of data for training the neural networks.
- We demonstrate the effectiveness of our sampling technique in practical deep learning problem. Our method outperforms random or sequential sampling techniques, highlighting the importance of history-aware sampling in training a deep model with stochastic approximation.

II. Related Work

A. Active Sampling in Machine Learning

Active sampling is a long-used technique [24] where a trained network tries to pose queries of data to learn in the next steps of the optimization algorithm. Its effectiveness has been tested on various machine learning problems. The most recent ones use active sampling to select a set of tasks in a multi-task reinforcement learning problem [25], to train a language model [2], or to perform active learning in image classification task [15]. However, these algorithms use predefined handcrafted rules to fetch the required data. Gal et al. [7] take a Bayesian approach to perform active learning for an image recognition task and reported a good performance gain. In this work, we suggest to exploit the recent advances of reinforcement learning to the active sampling problem in order to train a deep model. However, we have found that the name *adaptive* sampling better suits with our method instead of *active* sampling. The term *adaptive* emphasize the model-agnostic nature of our methodology.

B. Dataset Manipulation

Since the dataset is one of the major components consisting a deep learning framework, dataset manipulation is a readily investigated technique by deep learning communities. It has been discovered that the trained artificial neural networks can be exploited by perturbing an input by certain noises [27]. Many other works propose how to attack neural networks by modifying inputs, or to defend from those attacks by manipulating the training dataset [4]. Recently, dataset regularization strategies such as mixture [34], regional dropout [6], or performing both operations at the same time [33] are proposed to reduce the generalization error of convolutional neural networks. Dataset distillation [32] shows that the gradients with respect to the entire dataset can be emulated by extremely small number of samples. These are the examples of direct manipulation of the data samples to achieve better regularization or exploitation of the vulnerability of the network being trained. Here, we do *not* modify data points. Instead, by changing the order of the training data samples, we achieve better generalization of the model.

C. Reinforcement Learning in Computer Vision

Although computer vision and reinforcement learning have different origins, there are number of works that merge the ideas and benefits from both fields. In computer vision communities, reinforcement learning is used for search tasks on which gradients are hard to obtain. In network architecture search (NAS) [18, 28, 35, 36], a recurrent policy network seeks for an optimal structure for a neural network to solve a specific task on a certain dataset. The policy is trained with various reinforcement learning algorithms policy gradient methods [22]. Our method receives current performance of the trained network, which is a similar feedback obtained by controllers of NAS algorithms. Therefore, the reward and advantage function is borrowed from such works, especially from Pham et al. [18].

III. Problem Statement

A deep learning framework for supervised learning consists of four components: a dataset $\mathcal{D}$, a function approximator $\Phi(\,\cdot\,;\theta)$ parametrized by θ, a scalar loss function $l(\,\cdot\,)$, and an optimization algorithm A. The labeled dataset $\mathcal{D} \coloneqq \{(\mathbf{X}_i, y_i) \mid i = 1, 2, \ldots, N\}$ is a finite set of pairs of data points $\mathbf{X}_i$ and corresponding labels y_i. We call the function approximator $\Phi^{(t)} \colon \mathcal{X} \to \ell$ at each iteration t as the *learner* network. It maps each data point $\mathbf{X}_i \in \mathcal{X}$ to a fixed-length logit vector $\mathbf{l}_i^{(t)} \in \ell$. It has internal state fully characterized by its parameters θ_t at iteration t. In practice, the stochastic gradient-based optimization algorithm A tries to minimize the loss l with respect to the learner network parameter θ. At the beginning, θ is initialized with an arbitrary value $\theta \leftarrow \theta_0$. At iteration t of the algorithm A, a sampler S samples a mini-batch $\mathcal{B}_t$ of instances from the dataset $\mathcal{D}$. The parameter θ is updated by the gradient of the loss, estimated by the samples in the batch, i.e.,

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \nabla_\theta l_{\mathcal{B}_t}(\theta_t), \tag{4}$$

where α_t is a learning rate determined by A.

We model the adaptive sampler S_{ada} as a parametrized function $S_{\text{ada}}(\,\cdot\,;\theta_{\text{s},t})$. In the *problem of optimal sampling*, it is required to find an optimal parameter θ_{s}^* for the sampler

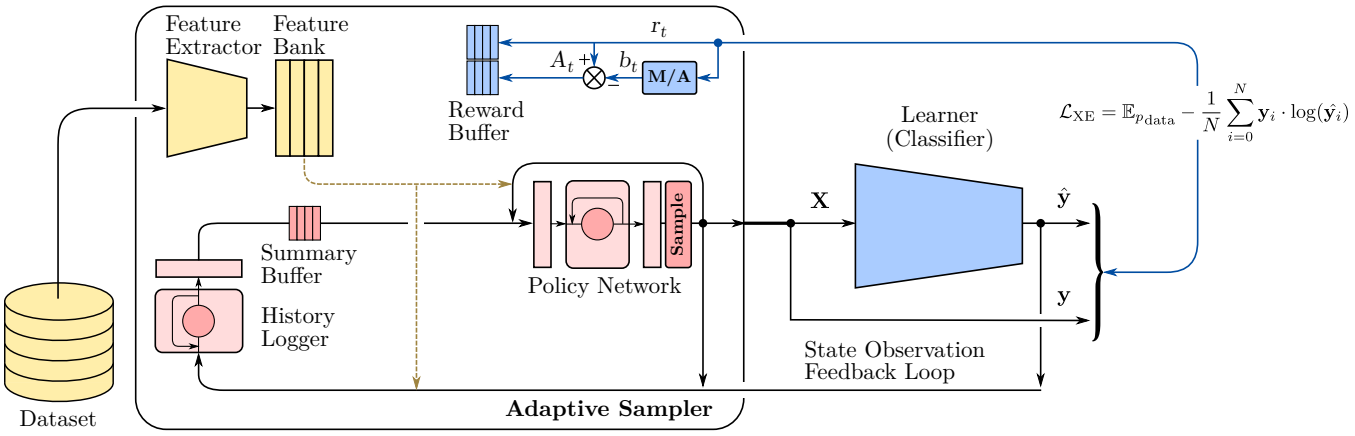

Fig. 2. **Architecture of our adaptive sampler**. The adaptive sampler consists of three networks: a feature extractor, a history logger, and a policy network. Feature extractor maps the entire dataset into a set of fixed-length feature vectors *prior* to the training of a model. Every training iteration of the learner network, stochastic policy network generates logits for each available training set index. Then, the items for the next mini-batch is sampled from the probabilistic distribution generated by the logits. Feature vector of previously sampled samples and the output logit from the learner network is concatenated into a state vector. The history logger keeps track of this states with a recurrent neural network to summarize the recent history of sampled items and activations of the learner. This history summarization is then fed into the policy network for next batches of items. Reward signals for the policy update is calculated from the validation loss of the learner network, which is occasionally obtained by sampling from the separate validation set.

S_{ada} that draws the sequence of samples $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{T-1}$ from the training datset $\mathcal{D}$ that minimizes the generalization error of the learner network Φ. The adaptive sampler should utilize the information of the previous and current state of the learner Φ, the history $\mathcal{H}$ of sampled items, and the training dataset $\mathcal{D}$. Therefore, our goal is to learn a sampler S_{ada} with a history-dependent policy π to maximize the generalization capability of a trained network Φ.

We model this problem as a finite-horizon, discounted Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$. Here, $\mathcal{S}$ is a state space determined by the state of the learner Φ and the sampling history $\mathcal{H}$. The action space $\mathcal{A}$ is defined as a set of all indices of the dataset $\mathcal{D}$, i.e., if $N \coloneqq |\mathcal{D}|$, then $\mathcal{A} = \{1, 2, \ldots, N\}$. A state transition probability $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{S})$ is determined by the deep learning framework $(\mathcal{D}, \Phi, l, A)$. A reward function $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ carries the information of current generalization error of the learner network and is rigorously defined in Sec. V. Finally, $\gamma \in (0, 1]$ is a discount factor.

The optimal stochastic policy $\pi\colon \mathcal{S} \to \mathbb{P}(\mathcal{A})$ is determined by solving the maximization problem of the expected cumulative reward

$$J(\pi) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)\right]. \tag{5}$$

In this work, we utilize a policy gradient method to solve the problem online.

IV. Design Considerations

There are several desiderata need be concerned before resolving the structure of the sampler. The first and the most important one is the appropriate approximation of the state space. As mentioned in Sec. III, the state space in our problem is a set of pairs $(\theta_t, \mathcal{H}_t)$, where t is the index of the learner's training iteration. The sampling history $\mathcal{H}_t$ is a collection of actions having been executed from the beginning of the training sequence, i.e., $\mathcal{H}_t \coloneqq \{a_0, a_1, \ldots, a_{t-1}\}$. To keep track of this growing sequence, some types of recurrent modules that summarize a sequence of varying length into a fixed-length vector are required. Moreover, the state θ_t of the learner network $\Phi(\cdot; \theta_t)$ at iteration t is often a set of millions of parameters correlated through complex interrelationships. It is practically infeasible to track the changes of all the parameters of the learner network in the sampler. In the simplest form, a deep model Φ can be viewed as a black-box function with input/output interface. A proper approximation of the state of such function is then a set of pairs of inputs and respective outputs. Therefore, we introduce a recurrent policy network that receives input items $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{|\mathcal{B}|}$ and output logits $\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_{|\mathcal{B}|}$ generated by the learner $\Phi(\,\cdot\,; \theta_t)$ at iteration t.

Secondly, indices of the items in the dataset rarely have any semantic meaning. Hence, the history of indices have little useful information to be used in our sampler; it is best to use a sequence of items directly as the history of samples. On the other extreme, each instance of the dataset is generally too large and unstructured to be directly handled in the submodules of the sampler. A feature extractor is, therefore, introduced to summarize each item into a fixed-length vector.

Although recurrent modules are necessary to store the history into a compact form, information in these module have limited lifetime [31]. Longer sequences are beneficial, since they give more hints about the current state of the learner; however, they also increase memory demand due to the nature of the backpropagation algorithm. In situations where stochastic approximation is desired, only a fraction of the dataset can be tracked by the sampler. Therefore, the sampler structure and the schedule of its policy update are constrained by memory budget, and the hidden states of the recurrent units are reset periodically. Update of the sampler policy need be slower than the update of the learner

network, for the reward signal should properly reflect the effect of changes the policy has made. Intermediate states of the sampler are required to be buffered to support slow gradient update of the policy network. Due to the same memory constraint, the stored sampler states should be cleared periodically.

V. Adaptive Sampler Design

A. Controller Design

Implemented as an iterator, at each call, the sampler S_{ada} returns an index of a single training example. The sampler is called $|\mathcal{B}_t|$ times every training iteration of the learner to fetch a mini-batch of samples. It receives learner's output logits of the current batch and have full access to the training dataset $\mathcal{D}$. The input/output interfaces of our adaptive sampler makes the module model-agnostic and also task-agnostic. The choice of the dataset sampler is independent of the choice of the learner model. This does not affect the overall training algorithm, since from the learner's perspective, only the order of the data samples changes. Our adaptive sampler S_{ada} has three major components: a *feature extractor* ϕ, a *history logger* η, and a *policy network* π.

Feature Extractor. For each labeled instance $(\mathbf{X}_i, y_i) \in \mathcal{D}$, a feature extractor $\phi(\cdot;\theta_{\text{f}})$ maps each data point $\mathbf{X}_i$ to a fixed-length feature vector $\mathbf{f}_i$, i.e., $\mathbf{f}_i \coloneqq \phi(\mathbf{X}_i;\theta_{\text{f}})$. This feature is used as a representative of the fetched index. However, training a feature extractor in the sampler is redundant to the training of the learner network. This increases the amount of gradients to be stored for the policy update by a large margin. Inspired by the work of Ulyanov et al. [29], we assume that *untrained* convolutional neural network can be used for a feature extractor of an image dataset. Thus, the feature extractor ϕ is *not* updated after the initialization. Another advantage of this setting is that the feature extraction can be done *prior* to the entire learning sequence. We generate the feature vectors for all the samples from the dataset to construct a feature bank. The features required in the training time can be queried to the bank.

History Logger. The history logger $\eta(\,\cdot\,;\theta_{\text{h}})$ is a recurrent neural network with internal hidden state $\mathbf{h}_{\text{h},t}$ at time t. The module receives image feature of the currently fetched mini-batch $\mathbf{f}_{\mathcal{B}_t} = (\mathbf{f}_{t,1}, \mathbf{f}_{t,2}, \ldots, \mathbf{f}_{t,|\mathcal{B}_t|})$ and the corresponding logit tensor $\mathbf{l}_{\mathcal{B}_t} \coloneqq \Phi(\mathbf{X}_{\mathcal{B}_t};\theta_t)$ generated by the learner network Φ and returns a fixed-length summarization $\mathbf{s}_{t+1}$, which acts as the next state supplied to the policy network, i.e.,

$$\mathbf{s}_{t+1} = \eta(\texttt{concat}(\mathbf{f}_{\mathcal{B}_t}, \mathbf{l}_{\mathcal{B}_t});\theta_{\text{h}}). \tag{6}$$

The history logger is implemented with two-layer LSTM cells followed by a single linear decoder.

Policy Network. The sampling policy $\pi(\cdot;\theta_{\text{p}})$ is a stochastic policy implemented with a recurrent neural network with hidden state $\mathbf{h}_{\text{p},t}$ at time t. In each call to the sampler, the policy network receives a concatenation of the feature vector of previously sampled item $\mathbf{f}_{t+1,i}$ and the summary $\mathbf{s}_t$ from the history logger. The output is a logit vector $\mathbf{l}^{\pi}_{t+1,i+1}$ indicating the probability of selecting each training sample for $(i+1)$-th item of batch $\mathcal{B}_{t+1}$.

$$\mathbf{l}^{\pi}_{t+1,i+1} = \pi(\texttt{concat}(\mathbf{s}_t, \mathbf{f}_{t+1,i});\theta_{\text{p}}). \tag{7}$$

The sampling probability of the item $a_j \in \mathcal{A}$ is given by the categorical distribution generated from the logit $\mathbf{l}^{\pi}_{t+1,i+1}$,

$$\pi_{t+1,i+1}(a_j|\mathbf{s}_t, \mathbf{f}_{t+1,i}) \coloneqq [\texttt{softmax}(\mathbf{l}^{\pi}_{t+1,i+1})]_j. \tag{8}$$

The policy network consists of a single linear encoder followed by two-layer LSTM cell and a single linear decoder. The output dimension of the decoder is the number of available actions, i.e., the number of samples in the training dataset $\mathcal{D}$.

B. Reward and Advantage Functions

Our definition of an optimal sampler requires the same performance metric as network architecture search [18, 35] to generate reward signals: the validation loss of the learner network. To evaluate the validation error throughout the training phase, the original training set $\mathcal{D}$ is split into a validation set $\mathcal{D}_{\text{val}}$ and a new training set $\mathcal{D}_{\text{train}}$. Every T_1 iterations of the learner update, the learner network is fixed and the sampler sequentially samples items from the validation set. Validation loss $l^{(t)}_{val}$ of the learner network is calculated by the same loss function l used for training the learner, and the ground-truth labels of the validation set. The reward is calculated from the averaged validation loss.

$$r_t \coloneqq c_r \exp\{-2l^{(t)}_{val}\}, \tag{9}$$

where, $c_r = 80$ is a constant factor.

Using a single reward directly in the policy gradient update leads to unnecessarily large variance [22]. Therefore, it is common to define a baseline and calculate the advantage as the difference between the current reward and the baseline [30]. Since our policy network is intrinsically sequential, we use moving average of the reward as the baseline.

$$b_t \coloneqq \lambda b_{t-1} + (1-\lambda) r_t. \tag{10}$$

Here, $\lambda = 0.9$ controls the rate of decay of the received reward. The advantage A_t at iteration t is calculated as,

$$A_t \coloneqq r_t - b_t. \tag{11}$$

We use Adam [12] optimizer to update our policy. Because Adam optimizer is a stochastic gradient *descent* solver, we define a policy *loss* to calculate the negative policy gradient for gradient ascent. The final policy loss l^{π} is composed of two terms: the inverse validation loss l^{π}_{val}, and the regularization term R.

$$\begin{aligned} l^{\pi}_{\text{val},t} &= \log \pi_t(a_t|\mathbf{s}_t) \cdot A_t(\mathbf{s}_t, a_t), \\ R_t &= -c_H \cdot \pi_t(a_t|\mathbf{s}_t) \log \pi_t(a_t|\mathbf{s}_t), \\ l^{\pi}_t &= -(l^{\pi}_{\text{val},t} + R_t). \end{aligned} \tag{12}$$

The regularization term R encourages the sampling distribution to have high entropy in order to prevent neglecting samples of the dataset. The constant $c_H = 1 \times 10^{-4}$ controls the amount of regularization applied to the policy.

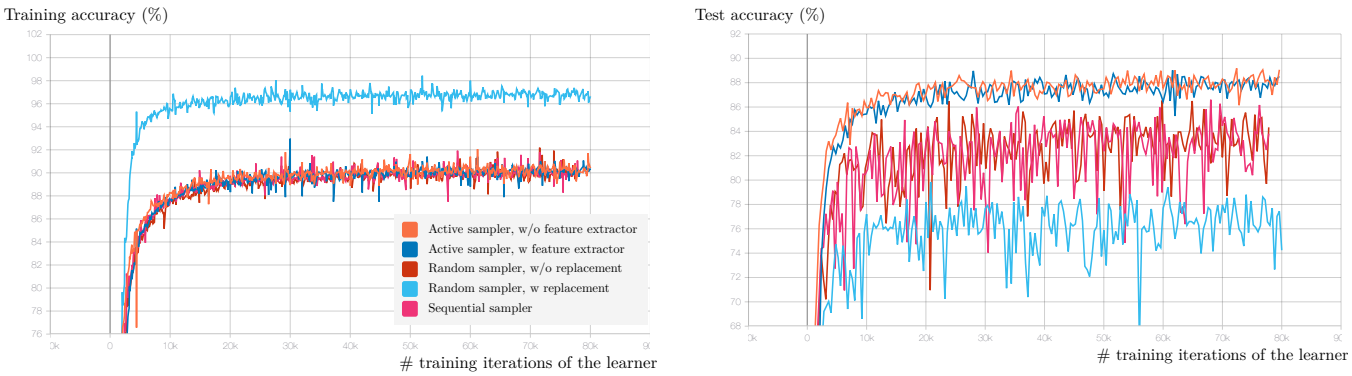

Fig. 3. **Training and test accuracy of the classifier trained with different samplers**. As in Table I the network trained with our adaptive sampler shows the highest test set accuracy on CIFAR-10 benchmark. It is worth notable that random sampling with allowing replacement shows the highest accuracy gap between training set and the test set, which is worse than the result from the same random sampler without replacement. This indicates that the sampling order is more important than the long term distribution of the samples to reduce the generalization error.

VI. Experiment

A. Implementation Details

We demonstrate our adaptive sampler in the image classification task. We use 18-layer ResNet model [10] designed for CIFAR-10 dataset [13] as the learner network. Network architecture and related hyperparameters are brought from an *unofficial* implementation[1]. The 50k training set of CIFAR-10 dataset is split into 6.4k validation set and 43.6k training set. Only this smaller training set is used for calculating the gradient update for the learner network. Normalized training and validation images are padded with 4 pixels to each borders and randomly cropped to obtain 32×32 image patches. The patches are then applied to a random horizontal flip before the sampling. Training the learner network is done with stochastic gradient descent solver. We use momentum of 0.9 and weight decay of 5×10^{-4}. The learning rate is fixed to 0.1. We stop the training after 80k iterations.

For the policy network, we use widths of features and historical summaries to be 32. The feature extractor is a four-layer convolutional neural network. Each convolution in the feature extractor is followed by a batch normalization [11] and a leaky ReLU with slope 0.2. All convolutions have kernel size of 3 and stride of 2 except the front convolution, which has stride of 1. Size of the hidden state of all the recurrent units in the history logger and the policy network is 32. Adam optimizer [12] with default hyperparameters is used for training the sampler module. Both the history logger and the policy network are trained; however, the feature extractor is *not* trained and is fixed to the initial state. We provide two different versions of the sampler, the one with and the one without the feature extractor to see the effectiveness of this settings. For the sampler without feature extractor, we simply set all the components of feature in the feature banks to one. Learning rate for the policy optimizer is fixed to 3.5×10^{-4}. We evaluate the validation loss every 20 iterations of the training phase. The calculated rewards, baselines, and advantages are kept in a fixed size buffer.

[1]https://github.com/kuangliu/pytorch-cifar

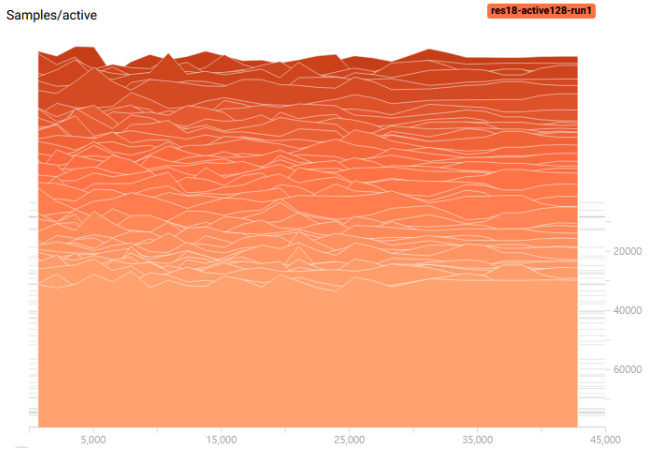

Fig. 4. **Histogram of short-term distributions of the samples fetched by our adaptive sampler**. At first, the generated samples have skewed distribution due to poor initialization of the policy network. In the late stage of the training, the validation error becomes small, and therefore, the entropy regularization term dominates the policy loss, resulting in more uniform distributions. This can be confirmed from the results of Fig. 5.

Every 100 iterations of the training phase of the learner network, the policy is updated for one step with five previous rewards and advantages. After the policy update, we clear all the buffers and reset the hidden state of the history logger and the policy network as zero vectors for memory efficiency.

B. Quantitative Results

Table I shows the quantitative comparison between our adaptive sampler and naïve static alternatives. Our method outperforms random sampling and sequential sampling in terms of both training and test accuracy. The results can also be seen in Fig. 3. The network trained with our adaptive sampler not only achieves higher generalization capability, but also shows less variance in the test accuracy, indicating that the training process is also stabilized.

It is notable that random sampling with replacement fails to generalize, while the same technique without replacement performs better. This indicates that the sampling order of the dataset is important in successful training of a deep model. The long term distribution of samples is less significant

Fig. 5. **Policy loss over time**. The graph shows that our policy objective is well-posed and the sampler module is trained successfully.

TABLE I
RESULTS ON CIFAR-10

Sampling method	Training acc. (%)	Test acc. (%)
Random, w/ Replacement	96.64	79.85
Random, w/o Replacement	90.07	86.53
Sequential	90.07	86.60
Adaptive, w/o feature extractor	90.43	**89.18**
Adaptive, w feature extractor	90.33	89.02

in determining the network performance. Fig. 4 shows the generated distribution of samples sampled from our policy over time. Our adaptive sampling strategy generates uniform distribution of samples like the sequential and the random sampling techniques. The results also suggest that the long-term sampling distribution is less important than the short-term sampling order. Lastly, the results shows that the sequential sampling performs on par with the random sampling, and this is because the original data indices are shuffled well.

C. Feasibility of the Design

Fig. 5 shows changes of the policy loss during the training phase. Despite the seldom updates, the policy loss is converged as the training proceeds. Unfortunately, however, the existence of feature extractor does not increase the test accuracy. The untrained module exhibits negative impact to the generalization error. Therefore, in the future work, application of transfer learning may be tried to alleviate the issue.

VII. CONCLUSION

We emphasized the importance of sampling strategy in training an artificial neural network in a deep learning framework. We proposed a novel approach to the sampling of the dataset. By deep reinforcement learning and the method of policy gradient, we constructed an effective adaptive sampler that is both model-agnostic and task-agnostic, and therefore, is applicable to general data-driven machine learning problems. We demonstrated our method on image classifciation benchmark, and shows clear improvements on both training and, especially, test set accuracy. We also found out that applying our method increases the stability of the trianing process. This clearly demonstrates that history-dependent sampling of data boosts the performance of a deep model.

REFERENCES

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "YouTube-8M: A large-scale video classification benchmark," *arXiv preprints*, Sep. 2016. arXiv: 1609.08675 [cs.CV].

[2] Y. Bengio and J. Senecal, "Adaptive importance sampling to accelerate training of a neural probabilistic language model," *IEEE Transactions on Neural Networks*, vol. 19, no. 4, pp. 713–722, 2008.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. Athena Scientific, Oct. 1996.

[4] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprints*, Sep. 2018. arXiv: 1810.00069 [cs.LG].

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprints*, Oct. 2018. arXiv: 1810.04805 [cs.CL].

[6] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with Cutout," *arXiv preprints*, Aug. 2017. arXiv: 1708.04552 [cs.CV].

[7] Y. Gal, R. Islam, and Z. Ghahramani, "Deep Bayesian active learning with image data," in *NIPS Workshop*, 2016. arXiv: 1703.02910 [cs.LG].

[8] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, third. Chapman and Hall/CRC, Oct. 2013.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018. arXiv: 1801.01290 [cs.LG].

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016. arXiv: 1512.03385 [cs.CV].

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015. arXiv: 1502.03167 [cs.LG].

[12] D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization," in *ICLR*, 2015.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009, Technical Report.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*, 2012.

[15] X. Li and Y. Guo, "Adaptive active learning for image classification," in *CVPR*, 2013.

[16] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten, "Exploring the limits of weakly supervised pretraining," in *ECCV*, 2018. arXiv: 1805.00932 [cs.CV].

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[18] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J., "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.

[19] X. Qian and D. Klabjan, "The impact of the mini-batch size on the variance of gradients in stochastic gradient descent," *arXiv e-prints*, Apr. 2020. arXiv: 2004.13146 [math.OC].

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, Feb. 2019.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015. arXiv: 1409.0575 [cs.CV].

[22] S. R. S. and B. A. G., *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, Nov. 2018.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprints*, Jul. 2017. arXiv: 1707.06347 [cs.LG].

[24] B. Settles, *Active Learning (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. Morgan & Claypool Publishers, Jun. 2012.

[25] S. Sharma, A. Jha, P. Hegde, and B. Ravindran, "Learning to multi-task by active sampling," in *ICLR*, 2018. arXiv: 1702.06053 [cs.NE].

[26] S. L. Smith and Q. V. Le, "A Bayesian perspective on generalization and stochastic gradient descent," in *ICLR*, 2018. arXiv: 1710.06451 [cs.LG].

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *ICLR*, 2014. arXiv: 1312.6199 [cs.CV].

[28] M. Tan and Q. V. Le, "EfficientNet: rethinking model scaling for convolutional neural networks," in *ICML*, 2019. arXiv: 1905.11946 [cs.LG].

[29] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *CVPR*, 2018. arXiv: 1711.10925 [cs.CV].

[30] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *AAAI*, 2016. arXiv: 1509.06461 [cs.LG].

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017. arXiv: 1706.03762 [cs.CL].

[32] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *arXiv preprints*, Nov. 2018. arXiv: 1811.10959 [cs.LG].

[33] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "CutMix: regularization strategy to train strong classifiers with localizable features," in *ICCV*, 2019. arXiv: 1905.04899 [cs.CV].

[34] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: beyond empirical risk minimization," in *ICLR*, 2018. arXiv: 1710.09412 [cs.LG].

[35] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICML*, 2017. arXiv: 1611.01578 [cs.LG].

[36] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018. arXiv: 1707.07012 [cs.CV].