Learning to Auto Weight: Entirely Data-Driven and Highly Efficient Weighting Framework

Zhenmao Li,[1] **Yichao Wu,**[1] **Ken Chen,**[1]
Yudong Wu,[1] **Shunfeng Zhou,**[1] **Jiaheng Liu,**[2] **Junjie Yan**[1]
[1]SenseTime, [2]BUAA
{lizhenmao, wuyichiao, wuyudong, zhoushunfeng, yanjunjie}@sensetime.com
kenchen1024@gmail.com, liujiaheng@buaa.edu.cn

Abstract

Example weighting algorithm is an effective solution to the training bias problem, however, most previous typical methods are usually limited to human knowledge and require laborious tuning of hyperparameters. In this paper, we propose a novel example weighting framework called *Learning to Auto Weight* (LAW). The proposed framework finds step-dependent weighting policies adaptively, and can be jointly trained with target networks without any assumptions or prior knowledge about the dataset. It consists of three key components: *Stage-based Searching Strategy (3SM)* is adopted to shrink the huge searching space in a complete training process; *Duplicate Network Reward (DNR)* gives more accurate supervision by removing randomness during the searching process; *Full Data Update (FDU)* further improves the updating efficiency. Experimental results demonstrate the superiority of weighting policy explored by LAW over standard training pipeline. Compared with baselines, LAW can find a better weighting schedule which achieves much more superior accuracy on both biased CIFAR and ImageNet.

Introduction

Although the quantity of training samples is critical for current state-of-the-art deep neural networks (DNNs), the quality of data also has significant impacts on exerting the powerful capacity of DNNs on various tasks. For supervised learning, it is a common hypothesis that both training and test examples are drawn i.i.d. from the same distribution. However, during practical training, this assumption is not always valid, therefore, the training bias problems, mainly including label noise and class imbalance, are encountered frequently.

It is widely known that the sample weighting algorithm is an effective solution to the training bias problem. Although common techniques such as cost-sensitive learning (Lin et al. 2017) and curriculum learning (Bengio et al. 2009; Kumar, Packer, and Koller 2010) demonstrate the effectiveness of example reweighting, they are usually predefined for specific tasks with prior knowledge and require laborious tuning of hyperparameters. To alleviate this problem adaptively, a method of learning to auto-weight is probably effective, which searches weighting strategies for picking more valuable samples or filtering harmful samples. However, there are some inherently severe challenges if we want to make the searing process work well. **The first challenge** is the huge searching space caused by numerous iterations during the training process. Our objective is to search step dependent weighting strategies for making the accuracy on validation datasets as high as possible. Suppose a complete training process involves thousands of steps N, the number of possible weights for a sample is w_n, the batch size is B (typical 128), so the number of possible weights of a batch data in one step is $a_n = w_n^B$, the searching space of weights is $a_n^N = w_n^{BN}$, which is an enormous number for searching a good strategy. **The second one** is the randomness which is an implicit but harmful problem for searching good weighting strategies. The randomness can derive from different data combination, the random augmentation, different initialization of parameters, etc. Thus, given a strategy model for weighting strategies, using accuracies on validation datasets to update the strategy model may cause the searching process to fail easily. **Last but not least**, collecting the training samples for learning to auto weight is nontrivial. In practice, to obtain credible feedback signals, we need to conduct complete training processes, so that huge numbers of networks must be trained to convergence. Therefore, the process of searching a weighting strategy is time-consuming with low efficiency.

Based on the above analysis, in this paper, we propose a novel example weighting strategies searching framework to learn weighting strategies from data adaptively, which is modeled by the strategy model. For the first challenge, we simplify the searching process and divide the training process into a small number of stages N' (typical 20) consisting of successive iterations, so that the time steps for searching can be significantly limited to the number of stages $N' \ll N$. We call this method ***Stage-based Searching Strategy Method (3SM)***, which can shrink the searching space and reduce time costs significantly. Meanwhile, to solve the second challenge, we design a novel feedback signal measurement, called ***Duplicate Networks Reward (DNR)***, where a reference network is added to generate accuracy difference on the validation dataset as the feedback signal to remove the randomness. In this way, if we get a higher accuracy or lower accuracy,

the searching algorithm could focus on the quality of the weighting strategy, and the strategy model could be updated for better ones. Besides, to raise the efficiency in the last challenge, we utilize a data buffer to cache the samples for updating the strategy model, and make full of all the data in the buffer to optimize the strategy model for numbers of epochs. We call this updating method ***Full Data Update (FDU)***, which can improve the efficiency of updating the strategy model significantly and accelerate the weighting strategies searching. Experimental results demonstrate the superiority of weighting policy explored by LAW over standard training pipeline. Especially, compared with baselines, LAW can find a better weighting schedule which achieves much more superior accuracy in the noisy or imbalance CIFAR and ImageNet dataset.

Our contributions are listed as follows:

1) We propose a novel example weighting strategy searching framework called LAW, which can learn weighting policy from data adaptively. LAW can find good sample weighting schedules that achieve higher accuracy in the contaminated and imbalance datasets without any extra information about the label noises.
2) We propose Stage-based Searching Strategy Method (3SM) to shrink the huge searching space in a complete training process.
3) We design novel Duplicate Networks Reward (DNR) that removes the data randomness and make the process of searching weighting strategies more effectively.
4) We propose Full Data Update (FDU) to make full use of all the data in the buffer to make the searching strategy process more efficient.

Related Work

Curriculum Learning: Inspired by that humans learn much better when putting the examples in a meaningful order (like from easy level to difficult level), (Bengio et al. 2009) formalizes a training strategy called curriculum learning which promotes learning with examples of increasing difficulty. This idea has been empirically verified and applied in a variety of areas (Kumar, Packer, and Koller 2010; Lee and Grauman 2011; Supancic and Ramanan 2013; Jiang et al. 2014a; Graves et al. 2017). Self-paced method (Kumar, Packer, and Koller 2010) defines the curriculum by considering the easy items with small losses in early stages and add items with large losses in the later stages. (Peng, Li, and Wang 2019) builds a more efficient batch selection method based on typicality sampling, where the typicality is estimated by the density of each sample. (Jiang et al. 2014b) formalizes the curriculum with preference to both easy and diverse samples. (Alain et al. 2015) reduces gradient variance by the sampling proposal proportional to the L2-norm of the gradient. Curriculums in the existing literature are usually determined by heuristic rules and thus require laborious tuning of hyperparameters.

Weighting: The practice of weighting each training example has been well investigated in the previous studies. Weighting algorithms mainly solve two kinds of problems: label noise and class imbalance. If models can converge to the optimal solution on the training set with coarse labels, there could be large performance gaps on the test set. This phenomenon has also been explained in (Zhang et al. 2016; Neyshabur et al. 2017; Arpit et al. 2017). Various regularization terms on the example weights have been proposed to prevent overfitting to on corrupted labels (Ma et al. 2017; Jiang et al. 2015). Recently, Jiang et al. propose MentorNet (Jiang et al. 2017), which provides a weighting scheme for StudentNet to focus on the sample whose label is probably correct. However, to acquire a proper MentorNet, it is necessary to give extra information such as the correct labels on a dataset during training. On the other hand, the class imbalance is usually caused by the cost and difficulty in collecting rarely seen classes. Kahn and Marshall (Kahn and Marshall 1953) propose importance sampling which assigns weights to samples to match one distribution to another. Lin et al. (Lin et al. 2017) propose focal loss to address the class imbalance by adding a soft weighting scheme that emphasizes harder examples. Other techniques such as cost-sensitive weighting (Khan et al. 2018) are also useful for class imbalance problems. Previous methods usually require prior-knowledge to determine a specified weighting mechanism, the performance will deteriorate if we cannot get accurate descriptions of the dataset. To learn from the data, Ren et al. (Ren et al. 2018) propose a novel meta-learning algorithm that learns to assign weights to training examples based on their gradient directions.

Learning to Auto Weight

In this section, we first demonstrate the formulation of learning to auto weight as a problem of searching the best strategy to pick valuable samples for training. It's also a bilevel optimization problem of training a classification model and a strategy model for weighting samples. Second, we explain the framework to solve the optimization problem. In the end, we describe the learning to search the best weighting strategy in detail.

Problem Formulation

Unlike the standard SGD training process, which treats every sample equally in all batches and all training stages, LAW tries to find the best strategy to weight samples in different batches and steps for better accuracy on validation datasets. In this paper, we model the strategy as a sample weighting function $\mathcal{K}(f, \theta)$ parameterized by θ, where the f is the feature of one sample. In different training steps $t = 1, 2, 3, ...T$, there are different weighting functions denoted by $\mathcal{K}_t(f, \theta_t)$ of different weighting strategies. Given an train dataset $\mathcal{X}_{train} = \{(x_i, y_i)|i = 1, 2, 3...N_{train}\}$ and a validation dataset $\mathcal{X}_{val} = \{(x_i, y_i)|i = 1, 2, 3...N_{val}\}$, we need to train a network $\mathcal{M}(\cdot, w)$ parameterized by w for the classification or other tasks. The purpose of LAW is to find a weighting strategy making a training network achieve better accuracy in all steps, and this can be realized by maximizing the cumulative validation accuracy associated with the network $\mathcal{M}(\cdot, w)$, which is trained to minimize corresponding

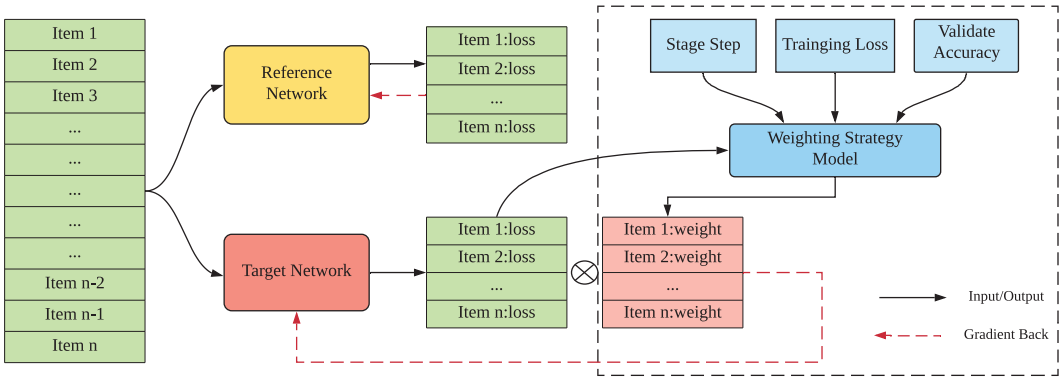


Figure 1: The sample weighting framework of LAW.

losses. Thus, the objective of LAW is:

$$\max_{\theta} \mathcal{J}(\theta) = \sum_{t} acc(w_t^*) \tag{1}$$

$$acc(w_t^*) = \frac{1}{N_{val}} \sum_{(\hat{x},\hat{y})\in\mathcal{X}_{val}} \delta(\mathcal{M}(\hat{x}, w_t^*), \hat{y}) \tag{2}$$

$$\text{s.t.} \quad w_t^* = \arg\min_{w} \sum_{(x,y)\in\mathcal{X}_{train}} \frac{\mathcal{K}_t(f,\theta_t)\mathcal{L}(\mathcal{M}(x,w),y)}{N_{train}},$$

where the δ is an impulse function, whose value is 1 when $\mathcal{M}(\hat{x}, w_t^*)$ is equal to $\hat{y}$ and 0 otherwise, $\mathcal{L}$ denotes the loss function. There are different weighting strategies for different training steps. For example, in early steps, the network may favor easy samples while in later stages, hard examples may need be considered much more. To tackle this problem, we sample weighting strategies according to the $\mathcal{K}_t(f,\theta_t)$, then train classification networks to obtain the best models $\mathcal{M}(\cdot, w^*)$. Next, the corresponding accuracies provide signals to optimize the $\mathcal{K}_t(f,\theta_t)$. Thus, the weighting strategy would be found under these two optimization processes interactively.

Weighting Framework

Our framework is illustrated in Figure 1, where the left half side constitutes the training procedure on classification networks and right side constitutes the weighting procedure by the strategy model, named Weighting Strategy Model defined as $\mathcal{K}(f,\theta)$ in . In every training step, we sample a batch of data consisting of n data items and feed them to two networks with identical architectures, Reference Network and Target Network. Reference Network is trained by a general training procedure without any weighting strategy and Target Network is trained by the weighting strategy from the Weighting Strategy Model. We collect the training state of Target Network, including Stage Step, Training Loss and Validation Accuracy (the details will be elaborated later) with some features of data items like losses. Then Weighting Strategy Model outputs weights for every data item. Thus, a weighted loss mean is calculated by an element multiplication between the losses and corresponding weights. In the end, the weighted loss mean is utilized to optimize Target Network, which is different from the optimization of Reference Network.

Stage-based Searching Strategy Method (3SM): For a standard SGD training process, one iteration consists of a forward and backward propagation based on the input mini-batch of samples, and the whole training process usually contains large numbers of successive iterations before getting the final model. In this process, the weights of feeding samples may have influences on the accuracy in the end, especially for biased training datasets. However, on account of the thousands of iteration steps, it's tricky and inefficiency to search a strategy based on one iteration step. Therefore, we uniformly divide the training process to a small number of stages, where one weighting strategy keeps unchanged for all iteration steps in one stage until the start of the next stage. To clarify this, we use $T = 1, 2, 3...T_{max}$ to denote the stage, and $t = 1, 2, 3...t_{max}$ to denote iteration steps of training a network inside one stage. So the number of total iteration steps is $T_{max} \times t_{max}$. Weighting Strategy Model outputs weights to weight all samples from $t = 1$ to $t = (t_{max} - 1)$, where there is no update for the strategy model. While at $t = t_{max}$, we calculate feedback signals and update the strategy model by the Full Data Update (the details will be elaborated later). The 3SM is illustrated in Figure 2.

In the first step $t = 1$, we collect informations of the Target Network as training phase descriptor s_T in every stage, such as current training stage T, smoothed historical training loss l_{smooth}, smoothed historical validating accuracy acc_{smooth}. To improve the expression capacity, we embed the number of the current training stage to a d dimension vector: $T \rightarrow \mathbf{e}_T \in R^d$, which is initialized using Gaussian initialization and is optimized as a part of Weighting Strategy Model in every searching step. When the current training stage is T, we define an intermediate network as $\mu(s_T|\theta^\mu)$ parameterized by θ^μ and the output of $\mu(s_T|\theta^\mu)$ is the parameter θ_T in $\mathcal{K}_T(f,\theta_T)$, where f is items' features descriptor. The features we used are listed as follows:

Training Loss: One practicable descriptor is the training loss, which is frequently utilized in the curriculum learning, hard example mining and self-paced methods. The loss can be used to describe the difficulty level of a sample and sometimes it would be helpful to drop out the samples with large

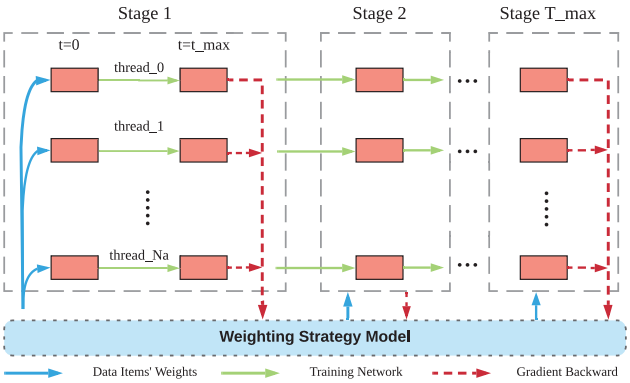


Figure 2: Illustration of stage-based weighting strategy searching. We conduct numbers of threads in parallel and synchronize the gradients of the weighting strategy model.

losses when there are outliers.

Entropy: For classification tasks, the entropy of predicted probabilities demonstrates the hardness of the input samples. Hard examples tend to have large entropy while samples with small and stable entropy are probably easy.

Density: The density of one item reveals how informative or typical the sample is, which can be used to measure the importance of samples. Obviously, samples with large density should be paid more attention to. To make it simple, we calculate similarity matrices using samples' logits **Lo** defined as $\mathbf{Lo}^T\mathbf{Lo}$ in one batch, and for each sample, we average it's similarities with other samples to approximate the density.

Label: We can also use the label on account of that the label information would help to remove some bias in the dataset like class imbalance.

In addition, we normalize the features to make the learning process more stable and the learned strategy more general for other datasets and networks. Once the features are extracted, the weights of items are defined as:

$$\mathcal{K}_T(f, \theta_T) = 1 + \tanh(\theta_T f + b), \tag{3}$$

$$\theta_T = \mu(s_T|\theta^\mu), \tag{4}$$

$$s_T = [\mathbf{e}_T, l_{smooth}, acc_{smooth}], \tag{5}$$

$$f = [loss, entropy, density, label], \tag{6}$$

For $t = 0, 1, 2...t_{max}$ in stage T, the gradients of the loss weighted by $\mathcal{K}_T(f, \theta_T)$ will be propagated back to update parameters of the Target Network $\mathcal{M}(\cdot, w)$:

$$w_t = w_{t-1} - \eta\nabla_w[\mathcal{K}_T \times \mathcal{L}_b(w_{t-1}, x_b, y_b)], \tag{7}$$

where η is the learning rate for the network parameters, b is the batch size, $\mathcal{L}_b$ denotes the batched loss which takes three inputs: the current network parameters w_{t-1} and a mini-batched data x_b, y_b.

Learning to Search Strategy

Considering that the validation accuracy is non-differentiable with respect to θ in $\mathcal{K}(f, \theta)$, it is a tricky problem to calculate the gradient of validation accuracy with respect to θ. To address this optimization problem, we utilize the method in DDPG (Lillicrap et al. 2015) to solve the searching problem approximately. Specifically, we add an extra network defined as $Q(s_T, \theta_T|\theta^Q)$ parameterized by θ^Q to estimate the objective in Equation 1, where the θ_T is defined same as $\mathcal{K}_T(f, \theta_T)$. Thus, if the $Q(s_T, \theta_T|\theta^Q)$ estimates the objective precisely, we can choose θ_T to maximum the objective, that is, improving the accuracy on validation datasets.

Duplicate Networks Reward (DNR): Randomness is an implicit but harmful problem for searching good weighting strategies. The randomness can be from the SGD optimizer, different data combination in one batch, data orders between batches, random augmentation of inputs, some random operation like dropping out in a network architecture, different initialization of parameters, and numerical precision in hardware et al. The randomness above could disturb weighting strategy searching because of too many factors that influence the accuracy in the end, so it's hard for the learning algorithm to decide what makes the accuracy higher. Thus, the feedback signal (it's also called reward) for updating the strategy model, must be designed to remove the randomness so that the learning method can apply credits to those better weighting strategies. Therefore, In each episode, we train two networks, one for searching strategies called Target Network and the other for comparison called Reference Network. There are some issues should be noted: The first is that the two network architectures is completely identical; The second is that the initialization of parameters is completely identical as we copy one network's parameters to another network directly; The third is that the data inputs at every step are completely identical to remove data randomness. In iteration steps updating the strategy model, we calculate accuracies of the Target Network and Reference Network on validation datasets and the reward is defined as the difference of accuracy between them. We call the reward from two identical networks Duplicate Networks Reward (DNR). In this way, if we get a higher accuracy or lower accuracy, the searching algorithm could put enough credit on better weighting strategies so that the strategy model could be updated forward better ones. What's more, Considering that the reward is not important in early stages, we add different weights from weakness to mightiness on rewards at different stages:

$$reward = r_w * (acc_{target} - acc_{reference}), \tag{8}$$

$$r_w = exp(k * \frac{ce}{ne}) * s, \tag{9}$$

where ce is current epoch, ne is total number of epochs of training process and k, s is the scale adjustment rate.

Full Data Update (FDU): Collecting the training samples for learning to auto weight is nontrivial. In practice, only if conducting complete training processes, we can obtain credible reward signals. In this way, huge numbers of networks must be trained to convergence. Hence, for updating the strategy model, we utilize a buffer to cache transitions data (Lillicrap et al. 2015), which is defined by a tuple $(s_T, \theta_T, r_T, s_{T+1}, Done)$, where T is the number of stage, s_T is defined in Equation 5, r_T is the reward in stage T, and $Done$ is a bool flag indicating whether the training procedure is finished or not. Since the time step is based on the stage,

we calculate the reward and update the strategy model only if the iteration step is t_{max} in one stage. Instead of sampling a batch of transition data in one time step, we utilize all the data in the buffer to update the strategy model for numbers of epochs, so that the strategy model can take full advantage of the cached data to learn useful knowledge. We call this update method Full Data Update (FDU). FDU can improve the efficiency of updating the strategy model significantly and accelerate the weighting strategies searching. As illustrated in Figure 2, Multiple classification networks and the strategy models are trained in different threads in parallel simultaneously and the number of threads is N_a. The parameters of the strategy models are shared among all the threads, and we synchronize gradients when updating the parameters of the strategy model in the last step in one stage. On the contrary, the parameters of the classification networks are optimized independently without sharing with others.

Given $\mu(s|\theta^\mu)$ parameterized by θ^μ and $Q(s,\theta|\theta^Q)$ parameterized by θ^Q, we have:

$$\nabla_{\theta^Q} Q = \frac{1}{N_a}\sum_{n=1}^{N_a}\frac{1}{B}\sum_{i=1}^{B}\nabla_{\theta^Q} L_Q(Q(s_i,\theta_i)), \tag{10}$$

$$L_Q(Q(s_i,\theta_i)) = (y_i - Q(s_i,\theta_i))^2, \tag{11}$$

$$y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}|\theta^\mu)), \tag{12}$$

$$\nabla_{\theta^\mu}\mu \approx \frac{1}{N_a}\sum_{n=1}^{N_a}\frac{1}{B}\sum_{i=1}^{B}\nabla_\theta Q(s_i,\theta|\theta^Q)|_{\theta=\mu(s_i)}\nabla_{\theta^\mu}\mu(s_i|\theta^\mu), \tag{13}$$

where B is batch size. The algorithm to update the strategy model is illustrated in Algorithm 1. The algorithm details of LAW are list in the Algorithm 2.

Algorithm 1 LAW: Update the strategy model

Input:
the buffer R, number of epochs En, batch size B;
for $e = 1, En$ **do**
  Shuffle all the data in R
  for each B transitions data **do**
    Update $Q(s,\theta|\theta^Q)$ one step as in Equation 10
    Update $\mu(s|\theta^\mu)$ one step as in Equation 13
  end for
end for

Algorithm 2 LAW:Learning to auto weight on one thread

Input: Training data D, batch size B, number of total training procedures L, number of total steps in one training procedure N, number of steps in one stage K;
Randomly initialize θ^μ in $\mu(s|\theta^\mu)$ and θ^Q in $Q(s,a|\theta^Q)$;
for $episode = 1, L$ **do**
  Random initialize w_t in Target Network M_t
  Copy w_t to w_r in Reference Network M^r: $w^r \leftarrow w_t$
  for $t = 1, N$ **do**
    if $t \mod K == 0$ **then**
      Calculate θ_t according to Equation 4
    else
      $\theta_t = \theta_{t-1}$
    end if
    Get data items' weights according to Equation 3
    Optimize Reference Network one step
    Optimize Target Network one step according to Equation 7
    if $t \mod K == 0$ **then**
      Compute the reward by Equation 8
      if $t \geq K$ **then**
        Cache the tuple $(s_{t-K}, \theta_t, r_t, s_t, Done)$ in the buffer
      end if
      Update the strategy model according to Algorithm 1
    end if
  end for
end for
Output: The strategy model

Experiments

Implementation Details

We demonstrate the effectiveness of LAW on image classification dataset CIFAR-10, CIFAR-100 (Krizhevsky and Hinton 2009), and ImageNet (Deng et al. 2009).

CIFAR: CIFAR-10 and CIFAR-100 consist of 50,000 training and 10,000 validation color images of 32×32 resolution with 10 classes and 100 classes receptively. They are balanced datasets where each class holds the same number of images. To search the weighting strategy, we use a part of the training dataset like 20,000 for training and 5,000 for validation. While for testing the strategy, we use all samples. General pre-processing steps are used in training, such as zero-padding with 4 pixels, random crops with size 32×32, random flips and standardizing the data. All the networks are trained to convergence from scratch utilizing SGD optimizer with a batch-size of 128. The weight decay is set to $2e-5$ and the momentum is set to 0.9. The initial learning rate is 0.1, and then the learning rate is divided by 10 when the stage is 10,13 and 16. The total number of training epochs is 200, thus the classification network would be harmed by biased datasets if no weighting strategy was applied.

ImageNet: ImageNet dataset contains 1.28 million training images and 50,000 validation images with 1,000 classes. We also sample a small part of the training dataset for searching the weighting strategy. In detail, the sampled images coverage all the classes, and contain 100 images per class. That is, the total number of the sampled dataset is 100,000. The process of training ImageNet follows the convention for state-of-art ImageNet models. Concretely, the pre-processing includes random flipping, random size crop to 224×224 and standardization with mean channel subtraction. We use synchronous SGD with a momentum rate of 0.9 and $2e-5$ weight decay. The step learning rate scheme that decreases the initial learning rate 0.1 by a factor 0.1 every 30 epoch is utilized. We also add a warmup stage of 2 epochs. The batch size is set to 1024 and the number of total training epochs is 100.

For Weight Strategy Model, the $\mu(s|\theta^{\mu})$ and $Q(s,\theta|\theta^{Q})$ are modeled using MLP with 4 layers, which is optimized by Adam (Kingma and Ba 2014) with learning rates 10^{-5} and 10^{-4} respectively. We utilize 8 threads for searching weighting strategies in parallel to make this process more efficiently. Multiple threads can boost the searching process and 8 is an empirical value. When the N_a is 4, the computational complexity is lower, but the searching process slows down compared with 8 threads. On the other hand, 16 threads raise the computational complexity and the searching process will be time-consuming. For every complete training, we divide the total number of training epochs uniformly to 20 stages. What's more, we set a warmup process considering that weighting strategies of early training stages do not have much impact on the accuracy in the end. That is, we don't utilize any weighting strategy in early training stages but train networks in a general procedure which treats every sample equally. After the warmup, the strategy model is optimized at every stage to search proper strategies to focus on important samples by weighting.

Results of effects on noisy labels compatibility

For noisy datasets, the label of each image in the training dataset is independently changed to a uniform random class with probability p, where p is set to 0.4. The labels of images in the validation dataset remain unchanged for evaluation.

CIFAR Results: On the CIFAR-10 and CIFAR-100, we evaluate two ResNet (He et al. 2016) networks, one small 18-layer ResNet, one large 101-layer ResNet and a medium WRN-28-10 (Zagoruyko and Komodakis 2016) network.

And we illustrate the test accuracy of our LAW for different networks on CIFAR in Table 1. The baseline is the base model that treats every sample equally in every batch. Compared with baseline models, our LAW exhibits a significant accuracy improvements for all the networks, which reveals that our LAW filters the noise data effectively.

To show the effectiveness on noisy datasets further, we also perform experiments compared with popular methods with WRN-28-10 utilizing human knowledge or meta-learning, including Self-paced (Kumar, Packer, and Koller 2010), MentorNet (Jiang et al. 2017), L2RW (Ren et al. 2018), Focal loss (Lin et al. 2017), and CO-teaching(Han et al. 2018). All methods are evaluated under the same setting described in Implementation Details except the number of total epochs is 120 and the learning rate is divided by 10 when the stage is 18 and 19. As shown in Table 2, our LAW obviously outperforms any other methods.

Table 1: The top-1 accuracy on CIFAR with noise rate $p = 0.4$ of LAW for different networks.

Method	CIFAR-10	CIFAR-100
ResNet-18(Base)	79.44	55.18
ResNet-18(LAW)	**86.33**	**61.27**
WRN-28-10(Base)	76.77	53.11
WRN-28-10(LAW)	**89.73**	**68.23**
ResNet-101(Base)	75.57	52.20
ResNet-101(LAW)	**88.90**	**68.01**

Table 2: The top-1 accuracy on noisy CIFAR with a 0.4 noise fraction for WRN-28-10 compared with other methods.

Method	CIFAR-10-Noisy(%)	CIFAR-100-Noisy(%)
Self-paced	86.21	46.23
MentorNet	87.56	65.56
L2RW	87.04	62.45
Focal Loss	74.12	50.71
Co-teaching	74.84	46.45
LAW	**89.73**	**68.23**

Table 3: The top-1 accuracy of on noisy ImageNet with noise rate $p = 0.4$ of our LAW for different networks.

Method	Noisy ImageNet(%)
ResNet-18(Base)	64.7
ResNet-18(LAW)	**65.2**
ResNet-34(Base)	65.4
ResNet-34(LAW)	**66.5**
ResNet-50(Base)	71.9
ResNet-50(LAW)	**73.7**
MobileNet-V2(Base)	59.3
MobileNet-V2(LAW)	**60.3**

ImageNet Results: We perform our LAW method on several popular networks on ImageNets: ResNet-18, ResNet-34, ResNet-50, and Mobilenetv2 (Sandler et al. 2018) to demonstrate the effectiveness on large dataset. As the same with CIFAR, we also construct a noisy dataset with probability 0.4. These networks including small, medium and heavy architectures, are evaluated to illustrate the generalization of our LAW method. Table 3 shows the top-1 test accuracy on validation set. All the experiments are conducted following the same setting and all the networks are trained from scratch. Considering the high cost of training process on ImageNet, we sample a small part of the dataset and train the weighting model only on ResNet-18, and train the small dataset in one GPU for searching the weighting strategies, then we transfer the learned weighting strategies to other networks. To test the learned strategy, we train the networks on ImageNet using synchronous SGD. As can be seen in the table, on the noisy ImageNet, our LAW improves the accuracy obviously. Similar to the results on noisy CIFAR, we also find that it can achieve more significant improvement when the network is heavier.

Effects on imbalance data

To evaluate effects of our LAW on the imbalance data, On CIFAR-10, we make an imbalance dataset by random discarding 96% of samples with the label of 0 and 1, while keeping the others the same as origin. That is, for the classes of 0 and 1, we only utilize 200 images on CIFAR-10 to train a network while 5,000 for other classes. On CIFAR-100, for the classes of 0 to 9, we utilize 50 images on CIFAR-100 to train a network while 500 for other classes. As for net-

Table 4: The top-1 accuracy on imbalance CIFAR of our LAW

Method	CIFAR-10-Imbalance(%)	CIFAR-100-Imbalance(%)
Vgg-19(Base)	84.88	59.33
Vgg-19(LAW)	**85.33**	**60.20**
ResNet-18(Base)	85.93	60.24
ResNet-18(LAW)	**88.44**	**61.94**

works, we train VGG-19 (Simonyan and Zisserman 2014) and ResNet-18 with the same setting described in Implementation Details above. It can be seen in the Table 4 that the weighting strategies explored by LAW can deal with this problem well.

To compare with other methods, we use Long-Tailed CIFAR dataset (Cui et al. 2019) and set the imbalance rate to 100, that is, the number of the largest class is 100 times the number of the smallest class. Other popular methods include Focal loss (Lin et al. 2017), Class-Balanced (Cui et al. 2019), L2RW (Ren et al. 2018), and we conduct experiments with ResNet-32 following the training schedule above except the number of total epochs is 100 and the learning rate is divided by 10 when the stage is 14 and 18.

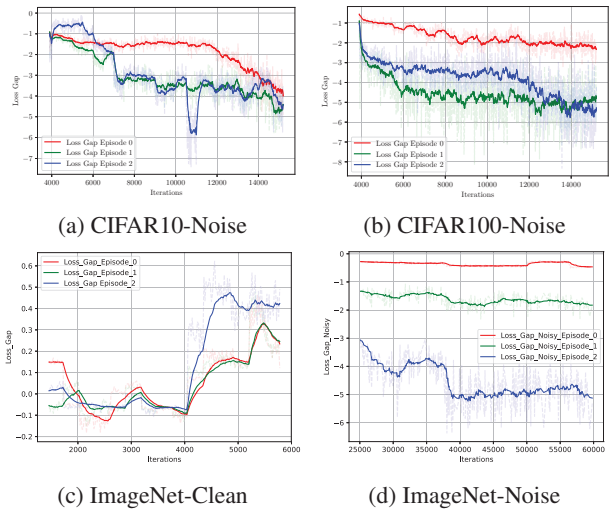
Table 5: The top-1 accuracy of ResNet-32 on imbalance CIFAR compared with other methods

Method	CIFAR-10-Imbalance(%)	CIFAR-100-Imbalance(%)
Focal Loss	71.34	39.41
Class-Balanced	74.6	40.16
L2RW	74.25	41.23
LAW	**76.34**	**43.61**

Analysis

In this section, we perform several analyses to illustrate the effectiveness of learned strategies by our LAW.

The curves of losses gap are shown in Figure 3, where the loss gap is defined as the mean of items' losses in one batch between the network trained with the learned weighting strategies and the network trained without any weighting strategy. For noisy datasets, tendencies of loss gaps are amazingly consistent for all datasets. Figure 3a, Figure 3b and Figure 3d illustrate loss gaps in noisy CIFAR-10, CIFAR-100 and ImageNet respectively. Apparently, the loss gaps are all under zero, which demonstrates that the learned weighting strategy from LAW can distinguish those data instances with corrupted labels and reduces the weights of them. For both CIFAR and ImageNet, the final accuracy of target networks is significantly higher than that of reference networks. It shows that LAW can find a much effective weighting schedule to find noisy data instances and filter them. To make the comparison more clearly, we also perform our LAW on the clean ImageNet and draw the loss gap in Figure 3c. Figure 3c demonstrates that the most of loss gap is near the 0 value and above 0 value in the later stage, which is contrary to the results on noisy ImageNet. That is, our LAW can find the noisy samples effectively and filter them in the early stages so that the damage from noise can be eliminated.

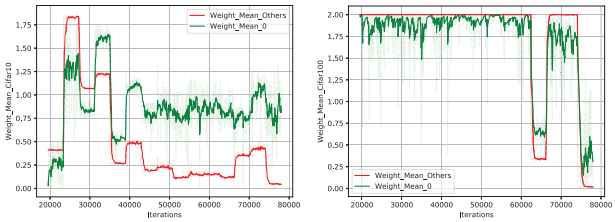


(a) CIFAR10-Noise (b) CIFAR100-Noise

(c) ImageNet-Clean (d) ImageNet-Noise

Figure 3: The loss mean gap between target network and reference network in random three episodes.

On imbalance datasets, we calculate the weight mean of samples from classes with a small number of images and the weight mean of other classes in one batch along with training steps. As showed in Figure 4a and 4b, on both datasets, the weights of the samples of label 0 is obvious larger than the samples of other labels. What interesting is that in early steps, the weights of small classes and other classes are close to each other, but in the later stage, the difference of two weight becomes larger. Our LAW can find samples of small classes and increase their weights. The policy explored by LAW can deal with imbalance problems well.



(a) CIFAR10-Imbalance (b) CIFAR100-Imbalance

Figure 4: The weight means of classes with a small number of images (Weight Mean 0) and other classes with an original number of images (Weight Mean Others) on CIFAR.

Conclusion

In this paper, we propose a novel example weighting framework called LAW, which can learn weighting policy from data adaptively. Experimental results demonstrate the superiority of weighting policy explored by LAW over standard training pipeline.

References

Alain, G.; Lamb, A.; Sankar, C.; Courville, A.; and Bengio, Y. 2015. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*.

Arpit, D.; Jastrzebski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M. S.; Maharaj, T.; Fischer, A.; Courville, A.; Bengio, Y.; et al. 2017. A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*.

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

Cui, Y.; Jia, M.; Lin, T.-Y.; Song, Y.; and Belongie, S. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9268–9277.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Graves, A.; Bellemare, M. G.; Menick, J.; Munos, R.; and Kavukcuoglu, K. 2017. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1311–1320. JMLR. org.

Han, B.; Yao, Q.; Yu, X.; Niu, G.; Xu, M.; Hu, W.; Tsang, I.; and Sugiyama, M. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *NIPS*, 8527–8537.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Jiang, L.; Meng, D.; Mitamura, T.; and Hauptmann, A. G. 2014a. Easy samples first: Self-paced reranking for zero-example multimedia search. In *Proceedings of the 22nd ACM international conference on Multimedia*, 547–556. ACM.

Jiang, L.; Meng, D.; Yu, S.-I.; Lan, Z.; Shan, S.; and Hauptmann, A. 2014b. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems*, 2078–2086.

Jiang, L.; Meng, D.; Zhao, Q.; Shan, S.; and Hauptmann, A. G. 2015. Self-paced curriculum learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Jiang, L.; Zhou, Z.; Leung, T.; Li, L.-J.; and Fei-Fei, L. 2017. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. *arXiv preprint arXiv:1712.05055*.

Kahn, H., and Marshall, A. W. 1953. Methods of reducing sample size in monte carlo computations. *Journal of the Operations Research Society of America* 1(5):263–278.

Khan, S. H.; Hayat, M.; Bennamoun, M.; Sohel, F. A.; and Togneri, R. 2018. Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE transactions on neural networks and learning systems* 29(8):3573–3587.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Kumar, M. P.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, 1189–1197.

Lee, Y. J., and Grauman, K. 2011. Learning the easy things first: Self-paced visual category discovery. In *CVPR 2011*, 1721–1728. IEEE.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, 2980–2988.

Ma, F.; Meng, D.; Xie, Q.; Li, Z.; and Dong, X. 2017. Self-paced co-training. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2275–2284. JMLR. org.

Neyshabur, B.; Bhojanapalli, S.; McAllester, D.; and Srebro, N. 2017. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, 5947–5956.

Peng, X.; Li, L.; and Wang, F.-Y. 2019. Accelerating minibatch stochastic gradient descent using typicality sampling. *arXiv preprint arXiv:1903.04192*.

Ren, M.; Zeng, W.; Yang, B.; and Urtasun, R. 2018. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, 4331–4340.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Supancic, J. S., and Ramanan, D. 2013. Self-paced learning for long-term tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2379–2386.

Zagoruyko, S., and Komodakis, N. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; and Vinyals, O. 2016. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.