Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting

Jun Shu1, Qi Xie1, Lixuan Yi1, Qian Zhao1, Sanping Zhou1, Zongben Xu1, and Deyu Meng*2,1

1Xi’an Jiaotong University
2The Macau University of Science and Technology
*Corresponding author:dymeng@mail.xjtu.edu.cn

Abstract

Current deep neural networks (DNNs) can easily overfit to biased training data with corrupted labels or class imbalance. Sample re-weighting strategy is commonly used to alleviate this issue by designing a weighting function mapping from training loss to sample weight, and then iterating between weight recalculating and classifier updating. Current approaches, however, need manually pre-specify the weighting function as well as its additional hyper-parameters. It makes them fairly hard to be generally applied in practice due to the significant variation of proper weighting schemes relying on the investigated problem and training data. To address this issue, we propose a method capable of adaptively learning an explicit weighting function directly from data. The weighting function is an MLP with one hidden layer, constituting a universal approximator to almost any continuous functions, making the method able to fit a wide range of weighting functions including those assumed in conventional research. Guided by a small amount of unbiased meta-data, the parameters of the weighting function can be finely updated simultaneously with the learning process of the classifiers. Synthetic and real experiments substantiate the capability of our method for achieving proper weighting functions in class imbalance and noisy label cases, fully complying with the common settings in traditional methods, and more complicated scenarios beyond conventional cases. This naturally leads to its better accuracy than other state-of-the-art methods. Source code is available at https://github.com/xjtushujun/meta-weight-net.

1 Introduction

DNNs have recently obtained impressive good performance on various applications due to their powerful capacity for modeling complex input patterns. However, DNNs can easily overfit to biased training data, like those containing corrupted labels or with class imbalance, leading to their poor performance in generalization in such cases. This robust deep learning issue has been theoretically illustrated in multiple literatures [4, 5, 6, 7, 8, 9].

In practice, however, such biased training data are commonly encountered. For instance, practically collected training samples always contain corrupted labels [10,11,12,13,14,15,16,17]. A typical example is a dataset roughly collected from a crowdsourcing system [18] or search engines [19,20], which would possibly yield a large amount of noisy labels. Another popular type of biased training data is those with class imbalance. Real-world datasets are usually depicted as skewed distributions, with a long-tailed configuration. A few classes account for most of the data, while most classes are

---

1We call the training data biased when they are generated from a joint sample-label distribution deviating from the distribution of evaluation/test set [1].

We propose to automatically learn an explicit loss-weight function, parameterized by an MLP from data. The main idea is to parameterize the weighting function as an MLP (multilayer perceptron) network with only one hidden layer (as shown in Fig. 1(c)), called Meta-Weighting-Net, which is theoretically a universal approximator for almost any continuous function \(f\), and then use a small unbiased validation set (meta-data) to guide the training of all its parameters. The explicit form of the weighting function can be finally attained from evaluation/test ones, is thus an important while challenging issue in machine learning [1, 21].

Sample reweighting approach is a commonly used strategy against this robust learning issue. The main methodology is to design a weighting function mapping from training loss to sample weight (with hyper-parameters), and then iterates between calculating weights from current training loss values and minimizing weighted training loss for classifier updating. There exist two entirely contradictory ideas for constructing such a loss-weight mapping. One makes the function monotonically increasing as depicted in Fig. 1(a), i.e., enforce the learning to more emphasize samples with larger loss values since they are more like to be uncertain hard samples located on the classification boundary. Typical methods of this category include AdaBoost [22, 23], hard negative mining [24] and focal loss [25]. This sample weighting manner is known to be necessary for class imbalance problems, since it can prioritize the minority class with relatively higher training losses.

On the contrary, the other methodology sets the weighting function as monotonically decreasing, as shown in Fig. 1(b) to take samples with smaller loss values as more important ones. The rationality lies on that these samples are more likely to be high-confident ones with clean labels. Typical methods include self-paced learning (SPL) [26], iterative reweighting [27, 17] and multiple variants [28, 29, 30]. This weighting strategy has been especially used in noisy label cases, since it inclines to suppress the effects of samples with extremely large loss values, possibly with corrupted incorrect labels.

Although these sample reweighting methods help improve the robustness of a learning algorithm on biased training samples, they still have evident deficiencies in practice. On the one hand, current methods need to manually set a specific form of weighting function based on certain assumptions on training data. This, however, tends to be infeasible when we know little knowledge underlying data or the label conditions are too complicated, like the case that the training set is both imbalanced and noisy. On the other hand, even when we specify certain weighting schemes, like focal loss [25] or SPL [26], they inevitably involve hyper-parameters, like focusing parameter in the former and age parameter in the latter, to be manually preset or tuned by cross-validation. This tends to further raise their application difficulty and reduce their performance stability in real problems.

To alleviate the aforementioned issue, this paper presents an adaptive sample weighting strategy to automatically learn an explicit weighting function from data. The main idea is to parameterize the weighting function as an MLP (multilayer perceptron) network with only one hidden layer (as shown in Fig. 1(c)), called Meta-Weighting-Net, which is theoretically a universal approximator for almost any continuous function [31], and then use a small unbiased validation set (meta-data) to guide the training of all its parameters. The explicit form of the weighting function can be finally attained specifically suitable to the learning task.

In summary, this paper makes the following three-fold contributions:

1) We propose to automatically learn an explicit loss-weight function, parameterized by an MLP from data in a meta-learning manner. Due to the universal approximation capability of this weight net, it can finely fit a wide range of weighting functions including those used in conventional research.
2) Experiments verify that the weighting functions learned by our method highly comply with manually preset weighting manners used in tradition in different training data biases, like class imbalance and noisy label cases as shown in Fig. 1(d) and 1(e), respectively. This shows that the weighting scheme learned by the proposed method inclines to help reveal deeper understanding for data bias insights, especially in complicated bias cases where the extracted weighting function is with complex tendencies (as shown in Fig. 1(f)).

3) The insights of why the proposed method works can be well interpreted. Particularly, the updating equation for Meta-Weight-Net parameters can be explained by that the sample weights of those samples better complying with the meta-data knowledge will be improved, while those violating such meta-knowledge will be suppressed. This tallies with our common sense on the problem: we should reduce the influence of those highly biased ones, while emphasize those unbiased ones.

The paper is organized as follows. Section 2 presents the proposed meta-learning method as well as the detailed algorithm and analysis of its convergence property. Section 3 discusses related work. Section 4 demonstrates experimental results and the conclusion is finally made.

2 The Proposed Meta-Weight-Net Learning Method

2.1 The Meta-learning Objective

Consider a classification problem with the training set \( \{ x_i, y_i \}_{i=1}^{N} \), where \( x_i \) denotes the \( i \)-th sample, \( y_i \in \{ 0, 1 \}^c \) is the label vector over \( c \) classes, and \( N \) is the number of the entire training data. \( f(x, w) \) denotes the classifier, and \( w \) denotes its parameters. In current applications, \( f(x, w) \) is always set as a DNN. We thus also adopt DNN, and call it the classifier network for convenience in the following.

Generally, the optimal classifier parameter \( w^* \) can be extracted by minimizing the loss \( \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i, w)) \) calculated on the training set. For notation convenience, we denote that \( L_i^{\text{train}}(w) = \ell(y_i, f(x_i, w)) \). In the presence of biased training data, sample re-weighting methods enhance the robustness of training by imposing weight \( V(L_i^{\text{train}}(w); \Theta) \) on the \( i \)-th sample loss, where \( V(\ell; \Theta) \) denotes the weight net, and \( \Theta \) represents the parameters contained in it. The optimal parameter \( w \) is calculated by minimizing the following weighted loss:

\[
w^*(\Theta) = \arg \min_w L^{\text{train}}(w; \Theta) \triangleq \frac{1}{N} \sum_{i=1}^{N} V(L_i^{\text{train}}(w); \Theta)L_i^{\text{train}}(w). \quad (1)
\]

Meta-Weight-Net: Our method aims to automatically learn the hyper-parameters \( \Theta \) in a meta-learning manner. To this aim, we formulate \( V(L_i(w); \Theta) \) as a MLP network with only one hidden layer containing 100 nodes, as shown in Fig. 1(e). We call this weight net as Meta-Weight-Net or MW-Net for easy reference. Each hidden node is with ReLU activation function, and the output is with the Sigmoid activation function, to guarantee the output located in the interval of \([0, 1]\). Albeit simple, this net is known as a universal approximator for almost any continuous function [31], and thus can fit a wide range of weighting functions including those used in conventional research.

Meta learning process. The parameters contained in MW-Net can be optimized by using the meta learning idea [32][33][34][35]. Specifically, assume that we have a small amount unbiased meta-data set (i.e., with clean labels and balanced data distribution) \( \{ x_i^{\text{meta}}, y_i^{\text{meta}} \}_{i=1}^{M} \), representing the meta-knowledge of ground-truth sample-label distribution, where \( M \) is the number of meta-samples and \( M \ll N \). The optimal parameter \( \Theta^* \) can be obtained by minimizing the following meta-loss:

\[
\Theta^* = \arg \min_{\Theta} L^{\text{meta}}(w^*(\Theta)) \triangleq \frac{1}{M} \sum_{i=1}^{M} L_i^{\text{meta}}(w^*(\Theta)), \quad (2)
\]

where \( L_i^{\text{meta}}(w) = \ell(y_i^{\text{meta}}, f(x_i^{\text{meta}}, w)) \) is calculated on meta-data.

2.2 The Meta-Weight-Net Learning Method

Calculating the optimal \( \Theta^* \) and \( w^* \) require two nested loops of optimization. Here we adopt an online strategy to update \( \Theta \) and \( w \) through a single optimization loop, respectively, to guarantee the efficiency of the algorithm.
Formulating learning manner of classifier network. As general network training tricks, we employ SGD to optimize the training loss (1). Specifically, in each iteration of training, a mini-batch of training samples \( \{ (x_i, y_i) \, | \, i = 1 \leq i \leq n \} \) is sampled, where \( n \) is the mini-batch size. Then the updating equation of the classifier network parameter can be formulated by moving the current \( w(t) \) along the descent direction of the objective loss in Eq. (1) on a mini-batch training data:

\[
\hat{w}(t)(\Theta) = w(t) - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla w L_{train}(w(t); \Theta) \bigg|_{w(t)}.
\]

where \( \alpha \) is the step size.

Algorithm 1: The MW-Net Learning Algorithm

**Input:** Training data \( D \), meta-data set \( \hat{D} \), batch size \( n, m \), max iterations \( T \).

**Output:** Classifier network parameter \( w(T) \)

1: Initialize classifier network parameter \( w(0) \) and Meta-Weight-Net parameter \( \Theta(0) \).
2: for \( t = 0 \) to \( T - 1 \) do
3: \( \{x, y\} \leftarrow \text{SampleMiniBatch}(D, n) \).
4: \( \{x^{(meta)}, y^{(meta)}\} \leftarrow \text{SampleMiniBatch}(\hat{D}, m) \).
5: Formulate the classifier learning function \( \hat{w}(t)(\Theta) \) by Eq. (3).
6: Update \( \Theta(t+1) \) by Eq. (4).
7: Update \( w(t+1) \) by Eq. (5).
8: end for

Updating parameters of Meta-Weight-Net: After receiving the feedback of the classifier network parameter updating formulation \( \hat{w}(t)(\Theta) \) from the Eq. (3), the parameter \( \Theta \) of the Meta-Weight-Net can then be readily updated guided by Eq. (2), i.e., moving the current parameter \( \Theta(t) \) along the objective gradient of Eq. (2) calculated on the meta-data:

\[
\Theta(t+1) = \Theta(t) - \beta \frac{1}{m} \sum_{i=1}^{m} \nabla \Theta L_{meta}(\hat{w}(t)(\Theta)) \bigg|_{\Theta(t)}.
\]

where \( \beta \) is the step size.

Updating parameters of classifier network: Then, the updated \( \Theta(t+1) \) is employed to ameliorate the parameter \( w \) of the classifier network, i.e.,

\[
w(t+1) = w(t) - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla w L_{train}(w(t); \Theta(t+1)) \bigg|_{w(t)}.
\]

The MW-Net Learning algorithm can then be summarized in Algorithm 1 and Fig. 2 illustrates its main implementation process (steps 5-7). All computations of gradients can be efficiently implemented by automatic differentiation techniques and generalized to any deep learning architectures of classifier network. The algorithm can be easily implemented using popular deep learning frameworks like PyTorch [36]. It is easy to see that both the classifier network and the MW-Net gradually ameliorate their parameters during the learning process based on their values calculated in the last step, and the weights can thus be updated in a stable manner, as clearly shown in Fig. 6.

\[\text{Notice that } \Theta \text{ here is a variable instead of a quantity, which makes } \hat{w}(t)(\Theta) \text{ a function of } \Theta \text{ and the gradient in Eq. (4) be able to be computed.}\]
2.3 Analysis on the Weighting Scheme of Meta-Weight-Net

The computation of Eq. (4) by backpropagation can be rewritten as

$$
\Theta^{(t+1)} = \Theta^{(t)} + \frac{\alpha \beta}{n} \sum_{j=1}^{n} \left( \frac{1}{m} \sum_{i=1}^{m} G_{ij} \right) \frac{\partial V(L_{\text{train}}(w(t)); \Theta)}{\partial \Theta} \bigg|_{\Theta^{(t)}},
$$

where $G_{ij} = \frac{\partial L_{\text{meta}}(\Theta)}{\partial w} \left|_{w(t)} \right. \cdot \frac{\partial L_{\text{train}}(w)}{\partial w} \left|_{w(t)} \right.$. Neglecting the coefficient $ \frac{1}{m} \sum_{i=1}^{m} G_{ij}$, it is easy to see that each term in the sum orient to the ascend gradient of the weight function $V(L_{\text{train}}(w(t)); \Theta)$.

Suppose the loss function $V(L_{\text{train}}(w(t)); \Theta)$ is Lipschitz smooth with constant $L$, and $V(\cdot)$ is differential with a $\delta$-bounded gradient and twice differential with its Hessian bounded by $B$, and the loss function $L$ have $\rho$-bounded gradients with respect to training/meta data. Let the learning rate $\alpha_t$ satisfies $\alpha_t = \min\{\frac{1}{t}, \frac{1}{T}\}$, for some $k > 0$, such that $\frac{k}{T} < 1$, and $\beta_t, 1 \leq t \leq N$ is a monotone descent sequence, $\beta_t = \min\{\frac{1}{T}, \frac{1}{\sigma^2 \sqrt{T}}\}$ for some $c > 0$, such that $\sqrt{\sigma T} \geq L$ and $\sum_{t=1}^{\infty} \beta_t \leq \infty$, $\sum_{t=1}^{\infty} \beta_t^2 \leq \infty$. Then the proposed algorithm can achieve $E[\|G(\Theta^{(t)})\|_2^2] \leq c$ in $O(1/\epsilon^2)$ steps. More specifically,

$$
\min_{0 \leq t \leq T} E[\|\nabla L_{\text{meta}}(\Theta^{(t)})\|_2^2] \leq O\left(\frac{C}{\sqrt{T}}\right),
$$

where $C$ is some constant independent of the convergence process, and $\sigma$ is the variance of drawing uniformly mini-batch sample at random.

Theorem 2. The conditions in Theorem 1 hold, then we have:

$$
\lim_{t \to \infty} E[\|\nabla L_{\text{train}}(w(t); \Theta^{(t+1)})\|_2^2] = 0.
$$

2.4 Convergence of the MW-Net Learning Algorithm

Our algorithm involves optimization of two-level objectives, and therefore we show theoretically that our method converges to the critical points of both the meta and training loss function under some mild conditions in Theorem 1 and 2 respectively. The proof is listed in the supplementary material.

Theorem 1. Suppose the loss function $\ell$ is Lipschitz smooth with constant $L$, and $V(\cdot)$ is differential with a $\delta$-bounded gradient and twice differential with its Hessian bounded by $B$, and the loss function $\ell$ have $\rho$-bounded gradients with respect to training/meta data. Let the learning rate $\alpha_t$ satisfies $\alpha_t = \min\{\frac{1}{t}, \frac{1}{T}\}$, for some $k > 0$, such that $\frac{k}{T} < 1$, and $\beta_t, 1 \leq t \leq N$ is a monotone descent sequence, $\beta_t = \min\{\frac{1}{T}, \frac{1}{\sigma^2 \sqrt{T}}\}$ for some $c > 0$, such that $\sqrt{\sigma T} \geq L$ and $\sum_{t=1}^{\infty} \beta_t \leq \infty$, $\sum_{t=1}^{\infty} \beta_t^2 \leq \infty$. Then the proposed algorithm can achieve $E[\|G(\Theta^{(t)})\|_2^2] \leq c$ in $O(1/\epsilon^2)$ steps. More specifically,

$$
\min_{0 \leq t \leq T} E[\|\nabla L_{\text{meta}}(\Theta^{(t)})\|_2^2] \leq O\left(\frac{C}{\sqrt{T}}\right),
$$

where $C$ is some constant independent of the convergence process, and $\sigma$ is the variance of drawing uniformly mini-batch sample at random.

Theorem 2. The conditions in Theorem 1 hold, then we have:

$$
\lim_{t \to \infty} E[\|\nabla L_{\text{train}}(w(t); \Theta^{(t+1)})\|_2^2] = 0.
$$

3 Related Work

Sample Weighting Methods. The idea of reweighting examples can be dated back to dataset resampling [40][41], which pre-evaluates the sample weights as a pre-processing step by using certain prior knowledge on the task or data. To make the sample weights fit data more flexibly, more recent researchers focused on re-designing a weighting function mapping from training loss to sample weight, and dynamically ameliorate weights during training process [43][44]. There are mainly two manners to design the weighting function. One is to make it monotonically increasing, specifically effective in class imbalance case. Typical methods include the boosting algorithm (like AdaBoost [22]) and multiple of its variations [45], hard example mining [24] and focal loss [25], which impose larger weights to ones with larger loss values. On the contrary, another series of methods specify the weighting function as monotonically decreasing, especially used in noisy label cases. For example, SPL [26] and its extensions [28][29], iterative reweighting [27][17] and other recent work [46][40], pay more focus on easy samples with smaller losses. The limitation of these methods are that they all need to manually pre-specify the form of weighting function as well as their hyper-parameters, raising their difficulty to be readily used in real applications.

*Derivation can be found in supplementary materials.*
Meta Learning Methods. Inspired by meta-learning developments [47, 48, 49, 37, 50], recently some methods were proposed to learn an adaptive weighting scheme from data to make the learning more automatic and reliable. Typical methods along this line include FWL [51], learning to teach [52, 32] and MentorNet [21] methods, whose weight functions are designed as a Bayesian function approximator, a DNN with attention mechanism, a bidirectional LSTM network, respectively. Instead of only taking loss values as inputs as classical methods, the weighting functions they used (i.e., the meta-learner), however, are with much more complex forms and required to input complicated information (like sample features). This makes them not only hard to succeed good properties possessed by traditional methods, but also to be easily reproduced by general users.

A closely related method, called L2RW [1], adopts a similar meta-learning mechanism compared with ours. The major difference is that the weights are implicitly learned there, without an explicit weighting function. This, however, might lead to unstable weighting behavior during training and unavailability for generalization. In contrast, with the explicit yet simple Meta-Weight-Net, our method can learn the weight in a more stable way, as shown in Fig. 6 and can be easily generalized from a certain task to related other ones (see in the supplementary material).

Other Methods for Class Imbalance. Other methods for handling data imbalance include: [53, 54] tries to transfer the knowledge learned from major classes to minor classes. The metric learning based methods have also been developed to effectively exploit the tailed data to improve the generalization ability, e.g., triple-header loss [55] and range loss [56].

Other Methods for Corrupted Labels. For handling noisy label issue, multiple methods have been designed by correcting noisy labels to their true ones via a supplemental clean label inference step [11, 14, 57, 13, 21, 1, 5]. For example, GLC [13] proposed a loss correction approach to mitigate the effects of label noise on DNN classifiers. Other methods along this line include the Reed [58], Co-training [16], D2L [59] and S-Model [12].

4 Experimental Results

To evaluate the capability of the proposed algorithm, we implement experiments on data sets with class imbalance and noisy label issues, and real-world dataset with more complicated data bias.

4.1 Class Imbalance Experiments

We use Long-Tailed CIFAR dataset [60], that reduces the number of training samples per class according to an exponential function \( n = n_i \mu^i \), where \( i \) is the class index, \( n_i \) is the original number of training images and \( \mu \in (0, 1) \). The imbalance factor of a dataset is defined as the number of training samples in the largest class divided by the smallest. We trained ResNet-32 [61] with softmax cross-entropy loss by SGD with a momentum 0.9, a weight decay \( 5 \times 10^{-4} \), an initial learning rate 0.1. The learning rate of ResNet-32 is divided by 10 after 80 and 90 epoch (for a total 100 epochs), and the learning rate of WN-Net is fixed as \( 10^{-5} \). We randomly selected 10 images per class in validation set as the meta-data set. The compared methods include: 1) BaseModel, which uses a softmax cross-entropy loss to train ResNet-32 on the training set; 2) Focal loss [25] and Class-Balanced [60] represent the state-of-the-arts of the predefined sample reweighting techniques; 3) Fine-tuning, fine-tune the result of BaseModel on the meta-data set; 4) L2RW [1], which leverages an additional meta-dataset to adaptively assign weights on training samples.

Table 1 shows the classification accuracy of ResNet-32 on the test set and confusion matrices are displayed in Fig. 8 (more details are listed in the supplementary material). It can be observed that:

1) Our algorithm evidently outperforms other competing methods on datasets with class imbalance, showing its robustness in such data bias case; 2) When imbalance factor is 1, i.e., all classes are with same numbers of samples, fine-tuning runs best, and our method still attains a comparable performance; 3) When imbalance factor is 200 on long-tailed CIFAR-100, the smallest class has only two samples. An extra fine-tuning achieves performance gain, while our method still perform well in such extreme data bias.

To understand the weighing scheme of MW-Net, we depict the tendency curve of weight with respect to loss by the learned MW-Net in Fig. 1(d), which complies with the classical optimal weighting manner to such data bias, i.e., larger weights should be imposed on samples with relatively large losses, which are more likely to be minority class sample.
Table 1: Test accuracy (%) of ResNet-32 on long-tailed CIFAR-10 and CIFAR-100, and the best and the second best results are highlighted in **bold** and *italic bold*, respectively.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Long-Tailed CIFAR-10</th>
<th>Long-Tailed CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>BaseModel</td>
<td>65.68</td>
<td>70.36</td>
</tr>
<tr>
<td>Focal Loss</td>
<td>65.29</td>
<td>70.38</td>
</tr>
<tr>
<td>Class-Balanced</td>
<td>68.89</td>
<td>74.57</td>
</tr>
<tr>
<td>L2RW</td>
<td>66.51</td>
<td>74.91</td>
</tr>
<tr>
<td>Ours</td>
<td>68.91</td>
<td>75.21</td>
</tr>
</tbody>
</table>

Table 2: Test accuracy comparison on CIFAR-10 and CIFAR-100 of WRN-28-10 with varying noise rates under uniform noise. Mean accuracy (±std) over 5 repetitions are reported (‘—’ means the method fails).

<table>
<thead>
<tr>
<th>Dataset / Noise Rate</th>
<th>BaseModel</th>
<th>Reed-Balanced</th>
<th>S-Model</th>
<th>Focal Loss</th>
<th>Co-teaching</th>
<th>D2L Fine-tuning</th>
<th>MentorNet</th>
<th>L2RW</th>
<th>GLC</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>60%</td>
<td>62.82±0.31</td>
<td>61.79±0.31</td>
<td>65.11±0.14</td>
<td>68.39±0.25</td>
<td>70.87±0.24</td>
<td>64.86±0.13</td>
<td>68.17±0.13</td>
<td>64.32±0.14</td>
<td>69.20±0.28</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>60.77±0.23</td>
<td>58.78±0.35</td>
<td>60.49±0.29</td>
<td>70.86±0.14</td>
<td>70.84±0.34</td>
<td>64.00±0.35</td>
<td>67.12±0.14</td>
<td>66.00±0.00</td>
<td>66.55±0.34</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>53.12±0.33</td>
<td>53.75±0.36</td>
<td>53.04±0.28</td>
<td>70.48±0.25</td>
<td>69.05±0.26</td>
<td>64.23±0.34</td>
<td>65.10±0.24</td>
<td>66.34±0.00</td>
<td>66.74±0.35</td>
</tr>
</tbody>
</table>

| CIFAR-100            | 60%       | 55.24±0.34    | 58.91±0.30 | 58.92±0.26 | 64.12±0.34 | 66.10±0.24 | 62.77±0.24 | 64.24±0.24 | 63.91±0.00 | 64.59±0.34 |
|                      | 40%       | 52.32±0.33    | 57.11±0.40 | 58.32±0.26 | 64.12±0.34 | 66.10±0.24 | 62.77±0.24 | 64.24±0.24 | 63.91±0.00 | 64.59±0.34 |
|                      | 20%       | 50.13±0.33    | 56.99±0.40 | 58.32±0.26 | 64.12±0.34 | 66.10±0.24 | 62.77±0.24 | 64.24±0.24 | 63.91±0.00 | 64.59±0.34 |

4.2 Corrupted Label Experiment

We study two settings of corrupted labels on the training set: 1) **Uniform noise**. The label of each sample is independently changed to a random class with probability $p$ following the same setting in [2]. 2) **Flip noise**. The label of each sample is independently flipped to similar classes with total probability $p$. In our experiments, we randomly select two classes as similar classes with equal probability. Two benchmark datasets are employed: CIFAR-10 and CIFAR-100 [62]. Both are popularly used for evaluation of noisy labels [59][61].1000 images with clean labels in validation set are randomly selected as the meta-data set. We adopt a Wide ResNet-28-10 (WRN-28-10) [63] for uniform noise and ResNet-32 [61] for flip noise as our classifier network models [4].

The comparison methods include: **BaseModel**, referring to the similar classifier network utilized in our method, while directly trained on the biased training data; the robust learning methods Reed [58], S-Model [12], SPL [26], Focal Loss [25], Co-teaching [16], D2L [59]; Fine-tuning, fine-tuning the result of **BaseModel** on the meta-data with clean labels to further enhance its performance; typical meta-learning methods MentorNet [21], L2RW [1], GLC [15]. We also trained the baseline network only on 1000 meta-images. The performance are evidently worse than the proposed method due to the neglecting of the knowledge underlying large amount of training samples. We thus have not involved its results in comparison.

All the baseline networks were trained using SGD with a momentum 0.9, a weight decay $5 \times 10^{-4}$ and an initial learning rate 0.1. The learning rate of classifier network is divided by 10 after 36 epoch and 38 epoch (for a total of 40 epochs) in uniform noise, and after 40 epoch and 50 epoch (for a total of 60 epochs) in flip noise. The learning rate of WN-Net is fixed at $10^{-3}$. We repeated the experiments 5 times with different random seeds for network initialization and label noise generation.

We report the accuracy averaged over 5 repetitions for each series of experiments and each competing method in Tables 2 and 3. It can be observed that our method gets the best performance across almost all datasets and all noise rates, except the second for 40% Flip noise. At 0% noise cases (unbiased ones), our method performs only slightly worse than the BaseModel. For other corrupted label cases, the superiority of our method is evident. Besides, it can be seen that the performance gaps between ours and all other competing methods increase as the noise rate is increased from 40% to 60% under uniform noise. Even with 60% label noise, our method can still obtain a relatively high classification accuracy, and attains more than 15% accuracy gain compared with the second best result for CIFAR100 dataset, which indicates the robustness of our methods in such cases.

---

[4] We have tried different classifier network architectures as classifier networks under each noise setting to show our algorithm is suitable to different deep learning architectures. We show this effect in Fig 4 verifying the consistently good performance of our method in two classifier network settings.
The results are summarized in Table 4, which shows that the proposed method achieves the best performance. Fig. 10 plots the tendency curve of the learned MW-Net function, which reveals abundant data insights. Specifically, when the loss is with relatively small values, the weighting function inclines to increase with loss, meaning that it tends to more emphasize hard margin samples with informative knowledge for classification; while when the loss gradually changes large, the
weighting function begins to monotonically decrease, implying that it tends to suppress noise labels samples with relatively large loss values. Such complicated essence cannot be finely delivered by conventional weight functions.

5 Conclusion

We have proposed a novel meta-learning method for adaptively extracting sample weights to guarantee robust deep learning in the presence of training data bias. Compared with current reweighting methods that require to manually set the form of weight functions, the new method is able to yield a rational one directly from data. The working principle of our algorithm can be well explained and the procedure of our method can be easily reproduced ( Appendix A provide the Pytorch implement of our algorithm (less than 30 lines of codes)), and the completed training code is avriable at https://github.com/xjtushujun/meta-weight-net). Our empirical results show that the propose method can perform superior in general data bias cases, like class imbalance, corrupted labels, and more complicated real cases. Besides, such an adaptive weight learning approach is hopeful to be employed to other weight setting problems in machine learning, like ensemble methods and multi-view learning.

Acknowledgments

This research was supported by the China NSFC projects under contracts 61661166011, 11690011, 61603292, 61721002,U1811461. The authors would also like to thank anonymous reviewers for their constructive suggestions on improving the paper, especially on the proofs and theoretical analysis of our paper.

References


Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. Learning from noisy labels with distillation. In ICCV, 2017.


