

# Learning to Reweight Examples for Robust Deep Learning

Mengye Ren<sup>1,2</sup> Wenyuan Zeng<sup>1,2</sup> Bin Yang<sup>1,2</sup> Raquel Urtasun<sup>1,2</sup>

## Abstract

Deep neural networks have been shown to be very powerful modeling tools for many supervised learning tasks involving complex input patterns. However, they can also easily overfit to training set biases and label noises. In addition to various regularizers, example reweighting algorithms are popular solutions to these problems, but they require careful tuning of additional hyperparameters, such as example mining schedules and regularization hyperparameters. In contrast to past reweighting methods, which typically consist of functions of the cost value of each example, in this work we propose a novel meta-learning algorithm that learns to assign weights to training examples based on their gradient directions. To determine the example weights, our method performs a meta gradient descent step on the current mini-batch example weights (which are initialized from zero) to minimize the loss on a clean unbiased validation set. Our proposed method can be easily implemented on any type of deep network, does not require any additional hyperparameter tuning, and achieves impressive performance on class imbalance and corrupted label problems where only a small amount of clean validation data is available.

## 1. Introduction

Deep neural networks (DNNs) have been widely used for machine learning applications due to their powerful capacity for modeling complex input patterns. Despite their success, it has been shown that DNNs are prone to training set biases, i.e. the training set is drawn from a joint distribution  $p(x, y)$  that is different from the distribution  $p(x^v, y^v)$  of the evaluation set. This distribution mismatch could have many

different forms. Class imbalance in the training set is a very common example. In applications such as object detection in the context of autonomous driving, the vast majority of the training data is composed of standard vehicles but models also need to recognize rarely seen classes such as emergency vehicles or animals with very high accuracy. This will sometime lead to biased training models that do not perform well in practice.

Another popular type of training set bias is label noise. To train a reasonable supervised deep model, we ideally need a large dataset with high-quality labels, which require many passes of expensive human quality assurance (QA). Although coarse labels are cheap and of high availability, the presence of noise will hurt the model performance, e.g. Zhang et al. (2017) has shown that a standard CNN can fit any ratio of label flipping noise in the training set and eventually leads to poor generalization performance.

Training set biases and misspecification can sometimes be addressed with dataset resampling (Chawla et al., 2002), i.e. choosing the correct proportion of labels to train a network on, or more generally by assigning a weight to each example and minimizing a weighted training loss. The example weights are typically calculated based on the training loss, as in many classical algorithms such as AdaBoost (Freund & Schapire, 1997), hard negative mining (Malisiewicz et al., 2011), self-paced learning (Kumar et al., 2010), and other more recent work (Chang et al., 2017; Jiang et al., 2017).

However, there exist two contradicting ideas in training loss based approaches. In noisy label problems, we prefer examples with smaller training losses as they are more likely to be clean images; yet in class imbalance problems, algorithms such as hard negative mining (Malisiewicz et al., 2011) prioritize examples with higher training loss since they are more likely to be the minority class. In cases when the training set is both imbalanced and noisy, these existing methods would have the wrong model assumptions. In fact, without a proper definition of an unbiased test set, solving the training set bias problem is inherently ill-defined. As the model cannot distinguish the right from the wrong, stronger regularization can usually work surprisingly well in certain synthetic noise settings. Here we argue that in order to learn general forms of training set biases, it is necessary to have a small unbiased validation to guide training. It is actually

<sup>1</sup>Uber Advanced Technologies Group, Toronto ON, CANADA

<sup>2</sup>Department of Computer Science, University of Toronto, Toronto ON, CANADA. Correspondence to: Mengye Ren <mren3@uber.com>.

not uncommon to construct a dataset with two parts - one relatively small but very accurately labeled, and another massive but coarsely labeled. Coarse labels can come from inexpensive crowdsourcing services or weakly supervised data (Cordts et al., 2016; Russakovsky et al., 2015; Chen & Gupta, 2015).

Different from existing training loss based approaches, we follow a meta-learning paradigm and model the most basic assumption instead: *the best example weighting should minimize the loss of a set of unbiased clean validation examples that are consistent with the evaluation procedure*. Traditionally, validation is performed at the end of training, which can be prohibitively expensive if we treat the example weights as some hyperparameters to optimize; to circumvent this, we perform validation at *every* training iteration to dynamically determine the example weights of the current batch. Towards this goal, we propose an online reweighting method that leverages an additional small validation set and adaptively assigns importance weights to examples in every iteration. We experiment with both class imbalance and corrupted label problems and find that our approach significantly increases the robustness to training set biases.

## 2. Related Work

The idea of weighting each training example has been well studied in the literature. Importance sampling (Kahn & Marshall, 1953), a classical method in statistics, assigns weights to samples in order to match one distribution to another. Boosting algorithms such as AdaBoost (Freund & Schapire, 1997), select harder examples to train subsequent classifiers. Similarly, hard example mining (Malisiewicz et al., 2011), downsamples the majority class and exploits the most difficult examples. Focal loss (Lin et al., 2017) adds a soft weighting scheme that emphasizes harder examples.

Hard examples are not always preferred in the presence of outliers and noise processes. Robust loss estimators typically downweigh examples with high loss. In self-paced learning (Kumar et al., 2010), example weights are obtained through optimizing the weighted training loss encouraging learning easier examples first. In each step, the learning algorithm jointly solves a mixed integer program that iterates optimizing over model parameters and binary example weights. Various regularization terms on the example weights have since been proposed to prevent overfitting and trivial solutions of assigning weights to be all zeros (Kumar et al., 2010; Ma et al., 2017; Jiang et al., 2015). Wang et al. (2017) proposed a Bayesian method that infers the example weights as latent variables. More recently, Jiang et al. (2017) proposed to use a meta-learning LSTM to output the weights of the examples based on the training loss. Reweighting examples is also related to curriculum learning (Bengio et al., 2009), where the model reweights

among many available tasks. Similar to self-paced learning, typically it is beneficial to start with easier examples.

One crucial advantage of reweighting examples is robustness against training set bias. There has also been a multitude of prior studies on class imbalance problems, including using dataset resampling (Chawla et al., 2002; Dong et al., 2017), cost-sensitive weighting (Ting, 2000; Khan et al., 2015), and structured margin based objectives (Huang et al., 2016). Meanwhile, the noisy label problem has been thoroughly studied by the learning theory community (Natarajan et al., 2013; Angluin & Laird, 1988) and practical methods have also been proposed (Reed et al., 2014; Sukhbaatar & Fergus, 2014; Xiao et al., 2015; Azadi et al., 2016; Goldberger & Ben-Reuven, 2017; Li et al., 2017; Jiang et al., 2017; Vahdat, 2017; Hendrycks et al., 2018). In addition to corrupted data, Koh & Liang (2017); Muñoz-González et al. (2017) demonstrate the possibility of a dataset adversarial attack (i.e. dataset poisoning).

Our method improves the training objective through a weighted loss rather than an average loss and is an instantiation of meta-learning (Thrun & Pratt, 1998; Lake et al., 2017; Andrychowicz et al., 2016), i.e. learning to learn better. Using validation loss as the meta-objective has been explored in recent meta-learning literature for few-shot learning (Ravi & Larochelle, 2017; Ren et al., 2018; Lorraine & Duvenaud, 2018), where only a handful of examples are available for each class. Our algorithm also resembles MAML (Finn et al., 2017) by taking one gradient descent step on the meta-objective for each iteration. However, different from these meta-learning approaches, our reweighting method does not have any additional hyperparameters and circumvents an expensive offline training stage. Hence, our method can work in an online fashion during regular training.

## 3. Learning to Reweight Examples

In this section, we derive our model from a meta-learning objective towards an online approximation that can fit into any regular supervised training. We give a practical implementation suitable for any deep network type and provide theoretical guarantees under mild conditions that our algorithm has a convergence rate of  $O(1/\epsilon^2)$ . Note that this is the same as that of stochastic gradient descent (SGD).

### 3.1. From a meta-learning objective to an online approximation

Let  $(x, y)$  be an input-target pair, and  $\{(x_i, y_i), 1 \leq i \leq N\}$  be the training set. We assume that there is a small unbiased and clean validation set  $\{(x_i^v, y_i^v), 1 \leq i \leq M\}$ , and  $M \ll N$ . Hereafter, we will use superscript  $v$  to denote validation set and subscript  $i$  to denote the  $i^{th}$  data. We also assume

that the training set contains the validation set; otherwise, we can always add this small validation set into the training set and leverage more information during training.

Let  $\Phi(x, \theta)$  be our neural network model, and  $\theta$  be the model parameters. We consider a loss function  $C(\hat{y}, y)$  to minimize during training, where  $\hat{y} = \Phi(x, \theta)$ .

In standard training, we aim to minimize the expected loss for the training set:  $\frac{1}{N} \sum_{i=1}^N C(\hat{y}_i, y_i) = \frac{1}{N} \sum_{i=1}^N f_i(\theta)$ , where each input example is weighted equally, and  $f_i(\theta)$  stands for the loss function associating with data  $x_i$ . Here we aim to learn a reweighting of the inputs, where we minimize a weighted loss:

$$\theta^*(w) = \arg \min_{\theta} \sum_{i=1}^N w_i f_i(\theta), \quad (1)$$

with  $w_i$  unknown upon beginning. Note that  $\{w_i\}_{i=1}^N$  can be understood as training hyperparameters, and the optimal selection of  $w$  is based on its validation performance:

$$w^* = \arg \min_{w, w \geq 0} \frac{1}{M} \sum_{i=1}^M f_i^v(\theta^*(w)). \quad (2)$$

It is necessary that  $w_i \geq 0$  for all  $i$ , since minimizing the negative training loss can usually result in unstable behavior.

**Online approximation** Calculating the optimal  $w_i$  requires two nested loops of optimization, and every single loop can be very expensive. The motivation of our approach is to adapt online  $w$  through a single optimization loop. For each training iteration, we inspect the descent direction of some training examples locally on the training loss surface and reweight them according to their similarity to the descent direction of the validation loss surface.

For most training of deep neural networks, SGD or its variants are used to optimize such loss functions. At every step  $t$  of training, a mini-batch of training examples  $\{(x_i, y_i), 1 \leq i \leq n\}$  is sampled, where  $n$  is the mini-batch size,  $n \ll N$ . Then the parameters are adjusted according to the descent direction of the expected loss on the mini-batch. Let's consider vanilla SGD:

$$\theta_{t+1} = \theta_t - \alpha \nabla \left( \frac{1}{n} \sum_{i=1}^n f_i(\theta_t) \right), \quad (3)$$

where  $\alpha$  is the step size.

We want to understand what would be the impact of training example  $i$  towards the performance of the validation set at training step  $t$ . Following a similar analysis to Koh & Liang (2017), we consider perturbing the weighting by  $\epsilon_i$  for each

training example in the mini-batch,

$$f_{i,\epsilon}(\theta) = \epsilon_i f_i(\theta), \quad (4)$$

$$\hat{\theta}_{t+1}(\epsilon) = \theta_t - \alpha \nabla \sum_{i=1}^n f_{i,\epsilon}(\theta) \Big|_{\theta=\theta_t}. \quad (5)$$

We can then look for the optimal  $\epsilon^*$  that minimizes the validation loss  $f^v$  locally at step  $t$ :

$$\epsilon_t^* = \arg \min_{\epsilon} \frac{1}{M} \sum_{i=1}^M f_i^v(\theta_{t+1}(\epsilon)). \quad (6)$$

Unfortunately, this can still be quite time-consuming. To get a cheap estimate of  $w_i$  at step  $t$ , we take a single gradient descent step on a mini-batch of validation samples wrt.  $\epsilon_t$ , and then rectify the output to get a non-negative weighting:

$$u_{i,t} = -\eta \frac{\partial}{\partial \epsilon_{i,t}} \frac{1}{m} \sum_{j=1}^m f_j^v(\theta_{t+1}(\epsilon)) \Big|_{\epsilon_{i,t}=0}, \quad (7)$$

$$\tilde{w}_{i,t} = \max(u_{i,t}, 0). \quad (8)$$

where  $\eta$  is the descent step size on  $\epsilon$ .

To match the original training step size, in practice, we can consider normalizing the weights of all examples in a training batch so that they sum up to one. In other words, we choose to have a hard constraint within the set  $\{w : \|w\|_1 = 1\} \cup \{0\}$ .

$$w_{i,t} = \frac{\tilde{w}_{i,t}}{(\sum_j \tilde{w}_{j,t}) + \delta(\sum_j \tilde{w}_{j,t})}, \quad (9)$$

where  $\delta(\cdot)$  is to prevent the degenerate case when all  $w_i$ 's in a mini-batch are zeros, i.e.  $\delta(a) = 1$  if  $a = 0$ , and equals to 0 otherwise. Without the batch-normalization step, it is possible that the algorithm modifies its effective learning rate of the training progress, and our one-step look ahead may be too conservative in terms of the choice of learning rate (Wu et al., 2018). Moreover, with batch normalization, we effectively cancel the meta learning rate parameter  $\eta$ .

### 3.2. Example: learning to reweight examples in a multi-layer perceptron network

In this section, we study how to compute  $w_{i,t}$  in a multi-layer perceptron (MLP) network. One of the core steps is to compute the gradients of the validation loss wrt. the local perturbation  $\epsilon$ . We can consider a multi-layered network where we have parameters for each layer  $\theta = \{\theta_l\}_{l=1}^L$ , and at every layer, we first compute  $z_l$  the pre-activation, a weighted sum of inputs to the layer, and afterwards we apply a non-linear activation function  $\sigma$  to obtain  $\tilde{z}_l$  the post-activation:

$$z_l = \theta_l^\top \tilde{z}_{l-1}, \quad (10)$$

$$\tilde{z}_l = \sigma(z_l). \quad (11)$$

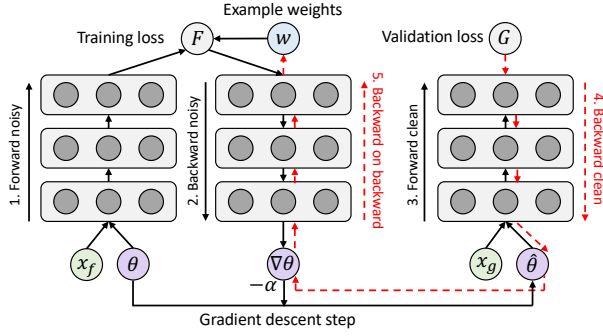


Figure 1. Computation graph of our algorithm in a deep neural network, which can be efficiently implemented using second order automatic differentiation.

During backpropagation, let  $g_l$  be the gradients of loss wrt.  $z_l$ , and the gradients wrt.  $\theta_l$  is given by  $\tilde{z}_{l-1} g_l^\top$ . We can further express the gradients towards  $\epsilon$  as a sum of local dot products.

$$\begin{aligned} & \frac{\partial}{\partial \epsilon_{i,t}} \mathbb{E} \left[ f^v(\theta_{t+1}(\epsilon)) \Big|_{\epsilon_{i,t}=0} \right] \\ & \propto -\frac{1}{m} \sum_{j=1}^m \frac{\partial f_j^v(\theta)}{\partial \theta} \Big|_{\theta=\theta_t}^\top \frac{\partial f_i(\theta)}{\partial \theta} \Big|_{\theta=\theta_t} \quad (12) \\ & = -\frac{1}{m} \sum_{j=1}^m \sum_{l=1}^L (z_{j,l-1}^v)^\top \tilde{z}_{i,l-1} (g_{j,l}^v)^\top g_{i,l}. \end{aligned}$$

Detailed derivations can be found in Appendix A. Eq. 12 suggests that the meta-gradient on  $\epsilon$  is composed of the sum of the products of two terms:  $z^\top z^v$  and  $g^\top g^v$ . The first dot product computes the similarity between the training and validation inputs to the layer, while the second computes the similarity between the training and validation gradient directions. In other words, suppose that a pair of training and validation examples are very similar, and they also provide similar gradient directions, then this training example is helpful and should be up-weighted, and conversely, if they provide opposite gradient directions, this training example is harmful and should be downweighted.

### 3.3. Implementation using automatic differentiation

In an MLP and a CNN, the unnormalized weights can be calculated based on the sum of the correlations of layerwise activation gradients and input activations. In more general networks, we can leverage automatic differentiation techniques to compute the gradient of the validation loss wrt. the example weights of the current batch. As shown in Figure 1, to get the gradients of the example weights, one needs to first unroll the gradient graph of the training batch, and then use backward-on-backward automatic differentiation to take a second order gradient

pass (see Step 5 in Figure 1). We list detailed step-by-step pseudo-code in Algorithm 1. This implementation can be generalized to any deep learning architectures and can be very easily implemented using popular deep learning frameworks such as TensorFlow (Abadi et al., 2016).

#### Algorithm 1 Learning to Reweight Examples using Automatic Differentiation

**Require:**  $\theta_0, \mathcal{D}_f, \mathcal{D}_g, n, m$

**Ensure:**  $\theta_T$

```

1: for  $t = 0 \dots T - 1$  do
2:    $\{X_f, y_f\} \leftarrow \text{SampleMiniBatch}(\mathcal{D}_f, n)$ 
3:    $\{X_g, y_g\} \leftarrow \text{SampleMiniBatch}(\mathcal{D}_g, m)$ 
4:    $\hat{y}_f \leftarrow \text{Forward}(X_f, y_f, \theta_t)$ 
5:    $\epsilon \leftarrow 0; l_f \leftarrow \sum_{i=1}^n \epsilon_i C(y_{f,i}, \hat{y}_{f,i})$ 
6:    $\nabla \theta_t \leftarrow \text{BackwardAD}(l_f, \theta_t)$ 
7:    $\hat{\theta}_t \leftarrow \theta_t - \alpha \nabla \theta_t$ 
8:    $\hat{y}_g \leftarrow \text{Forward}(X_g, y_g, \hat{\theta}_t)$ 
9:    $l_g \leftarrow \frac{1}{m} \sum_{i=1}^m C(y_{g,i}, \hat{y}_{g,i})$ 
10:   $\nabla \epsilon \leftarrow \text{BackwardAD}(l_g, \epsilon)$ 
11:   $\tilde{w} \leftarrow \max(-\nabla \epsilon, 0); w \leftarrow \frac{\tilde{w}}{\sum_j \tilde{w} + \delta (\sum_j \tilde{w})}$ 
12:   $\hat{l}_f \leftarrow \sum_{i=1}^n w_i C(y_{f,i}, \hat{y}_{f,i})$ 
13:   $\nabla \theta_t \leftarrow \text{BackwardAD}(\hat{l}_f, \theta_t)$ 
14:   $\theta_{t+1} \leftarrow \text{OptimizerStep}(\theta_t, \nabla \theta_t)$ 
15: end for
    
```

**Training time** Our automatic reweighting method will introduce a constant factor of overhead. First, it requires two full forward and backward passes of the network on training and validation respectively, and then another backward on backward pass (Step 5 in Figure 1), to get the gradients to the example weights, and finally a backward pass to minimize the reweighted objective. In modern networks, a backward-on-backward pass usually takes about the same time as a forward pass, and therefore compared to regular training, our method needs approximately  $3 \times$  training time; it is also possible to reduce the batch size of the validation pass for speedup. We expect that it is worthwhile to spend the extra time to avoid the irritation of choosing early stopping, finetuning schedules, and other hyperparameters.

### 3.4. Analysis: convergence of the reweighted training

Convergence results of SGD based optimization methods are well-known (Reddi et al., 2016). However it is still meaningful to establish a convergence result about our method since it involves optimization of two-level objectives (Eq. 1, 2) rather than one, and we further make some first-order approximation by introducing Eq. 7. Here, we show theoretically that our method converges to the critical point of the validation loss function under some mild conditions, and we also give its convergence rate. More detailed proofs can be found in the Appendix B, C.



**Definition 1.** A function  $f(x) : \mathbb{R}^d \rightarrow \mathbb{R}$  is said to be Lipschitz-smooth with constant  $L$  if

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \forall x, y \in \mathbb{R}^d.$$

**Definition 2.**  $f(x)$  has  $\sigma$ -bounded gradients if  $\|\nabla f(x)\| \leq \sigma$  for all  $x \in \mathbb{R}^d$ .

In most real-world cases, the high-quality validation set is really small, and thus we could set the mini-batch size  $m$  to be the same as the size of the validation set  $M$ . Under this condition, the following lemma shows that our algorithm always converges to a critical point of the validation loss. However, our method is not equivalent to training a model only on this small validation set. Because directly training a model on a small validation set will lead to severe overfitting issues. On the contrary, our method can leverage useful information from a larger training set, and still converge to an appropriate distribution favored by this clean and balanced validation dataset. This helps both generalization and robustness to biases in the training set, which will be shown in our experiments.

**Lemma 1.** Suppose the validation loss function is Lipschitz-smooth with constant  $L$ , and the train loss function  $f_i$  of training data  $x_i$  have  $\sigma$ -bounded gradients. Let the learning rate  $\alpha_t$  satisfies  $\alpha_t \leq \frac{2n}{L\sigma^2}$ , where  $n$  is the training batch size. Then, following our algorithm, the validation loss always monotonically decreases for any sequence of training batches, namely,

$$G(\theta_{t+1}) \leq G(\theta_t), \quad (13)$$

where  $G(\theta)$  is the total validation loss

$$G(\theta) = \frac{1}{M} \sum_{i=1}^M f_i^v(\theta_{t+1}(\epsilon)). \quad (14)$$

Furthermore, in expectation, the equality in Eq. 13 holds only when the gradient of validation loss becomes 0 at some time step  $t$ , namely  $\mathbb{E}_t[G(\theta_{t+1})] = G(\theta_t)$  if and only if  $\nabla G(\theta_t) = 0$ , where the expectation is taking over possible training batches at time step  $t$ .

Moreover, we can prove the convergence rate of our method to be  $O(1/\epsilon^2)$ .

**Theorem 2.** Suppose  $G$ ,  $f_i$  and  $\alpha_t$  satisfy the aforementioned conditions, then Algorithm 1 achieves  $\mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \epsilon$  in  $O(1/\epsilon^2)$  steps. More specifically,

$$\min_{0 \leq t \leq T} \mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \frac{C}{\sqrt{T}}, \quad (15)$$

where  $C$  is some constant independent of the convergence process.

## 4. Experiments

To test the effectiveness of our reweighting algorithm, we designed both class imbalance and noisy label settings, and a combination of both, on standard MNIST and CIFAR benchmarks for image classification using deep CNNs.<sup>1</sup>

### 4.1. MNIST data imbalance experiments

We use the standard MNIST handwritten digit classification dataset and subsample the dataset to generate a class imbalance binary classification task. We select a total of 5,000 images of size  $28 \times 28$  on class 4 and 9, where 9 dominates the training data distribution. We train a standard LeNet on this task and we compare our method with a suite of commonly used tricks for class imbalance: 1) PROPORTION weights each example by the inverse frequency 2) RESAMPLE samples a class-balanced mini-batch for each iteration 3) HARD MINING selects the highest loss examples from the majority class and 4) RANDOM is a random example weight baseline that assigns weights based on a rectified Gaussian distribution:

$$w_i^{\text{md}} = \frac{\max(z_i, 0)}{\sum_i \max(z_i, 0)}, \quad \text{where } z_i \sim \mathcal{N}(0, 1). \quad (16)$$

To make sure that our method does not have the privilege of training on more data, we split the balanced validation set of 10 images directly from the training set. The network is trained with SGD with a learning rate of 1e-3 and mini-batch size of 100 for a total of 8,000 steps.

Figure 2 plots the test error rate across various imbalance ratios averaged from 10 runs with random splits. Note that our method significantly outperforms all the baselines. With class imbalance ratio of 200:1, our method only reports a small increase of error rate around 2%, whereas other methods suffer terribly under this setting. Compared with resampling and hard negative mining baselines, our approach does not throw away samples based on its class or training loss - as long as a sample is helpful towards the validation loss, it will be included as a part of the training loss.

### 4.2. CIFAR noisy label experiments

Reweighting algorithm can also be useful on datasets where the labels are noisy. We study two settings of label noise here:

- UNIFORMFLIP: All label classes can uniformly flip to any other label classes, which is the most studied in the literature.
- BACKGROUNDFLIP: All label classes can flip to a single background class. This noise setting is very

<sup>1</sup>Code released at: <https://github.com/uber-research/learning-to-reweight-examples>

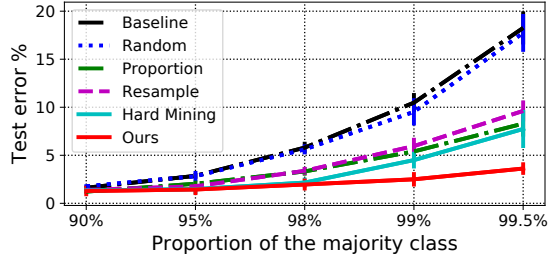


Figure 2. MNIST 4-9 binary classification error using a LeNet on imbalanced classes. Our method uses a small balanced validation split of 10 examples.

realistic. For instance, human annotators may not have recognized all the positive instances, while the rest remain in the background class. This is also a combination of label imbalance and label noise since the background class usually dominates the label distribution.

We compare our method with prior work on the noisy label problem.

- REED, proposed by [Reed et al. \(2014\)](#), is a bootstrapping technique where the training target is a convex combination of the model prediction and the label.
- S-MODEL, proposed by [Goldberger & Ben-Reuven \(2017\)](#), adds a fully connected softmax layer after the regular classification output layer to model the noise transition matrix.
- MENTORNET, proposed by [Jiang et al. \(2017\)](#), is an RNN-based meta-learning model that takes in a sequence of loss values and outputs the example weights. We compare numbers reported in their paper with a base model that achieves similar test accuracy under 0% noise.

In addition, we propose two simple baselines: 1) RANDOM, which assigns weights according to a rectified Gaussian (see Eq. 16); 2) WEIGHTED, designed for BACKGROUNDFLIP, where the model knows the oracle noise ratio for each class and reweights the training loss proportional to the percentage of clean images of that label class.

**Clean validation set** For UNIFORMFLIP, we use 1,000 clean images in the validation set; for BACKGROUNDFLIP, we use 10 clean images per label class. Since our method uses information from the clean validation, for a fair comparison, we conduct an additional finetuning on the clean data based on the pre-trained baselines. We also study the effect on the size of the clean validation set in an ablation study.

Table 1. CIFAR UNIFORMFLIP under 40% noise ratio using a WideResNet-28-10 model. Test accuracy shown in percentage. Top rows use only noisy data, and bottom uses additional 1000 clean images. “FT” denotes fine-tuning on clean data.

MODEL	CIFAR-10	CIFAR-100
BASILINE	$67.97 \pm 0.62$	$50.66 \pm 0.24$
REED-HARD	$69.66 \pm 1.21$	$51.34 \pm 0.17$
S-MODEL	$70.64 \pm 3.09$	$49.10 \pm 0.58$
MENTORNET	76.6	56.9
RANDOM	$86.06 \pm 0.32$	$58.01 \pm 0.37$
USING 1,000 CLEAN IMAGES		
CLEAN ONLY	$46.64 \pm 3.90$	$9.94 \pm 0.82$
BASILINE +FT	$78.66 \pm 0.44$	$54.52 \pm 0.40$
MENTORNET +FT	78	59
RANDOM +FT	$86.55 \pm 0.24$	$58.54 \pm 0.52$
OURS	<b><math>86.92 \pm 0.19</math></b>	<b><math>61.34 \pm 2.06</math></b>

**Hyper-validation set** For monitoring training progress and tuning baseline hyperparameters, we split out another 5,000 hyper-validation set from the 50,000 training images. We also corrupt the hyper-validation set with the same noise type.

**Experimental details** For REED model, we use the best  $\beta$  reported in [Reed et al. \(2014\)](#) ( $\beta = 0.8$  for hard bootstrapping and  $\beta = 0.95$  for soft bootstrapping). For the S-MODEL, we explore two versions to initialize the transition weights: 1) a smoothed identity matrix; 2) in background flip experiments we consider initializing the transition matrix with the confusion matrix of a pre-trained baseline model (S-MODEL +CONF). We find baselines can easily overfit the training noise, and therefore we also study early stopped versions of the baselines to provide a stronger comparison. In contrast, we find early stopping not necessary for our method.

To make our results comparable with the ones reported in MENTORNET and to save computation time, we exchange their Wide ResNet-101-10 with a Wide ResNet-28-10 (WRN-28-10) ([Zagoruyko & Komodakis, 2016](#)) with dropout 0.3 as our base model in the UNIFORMFLIP experiments. We find that test accuracy differences between the two base models are within 0.5% on CIFAR datasets under 0% noise. In the BACKGROUNDFLIP experiments, we use a ResNet-32 ([He et al., 2016](#)) as our base model.

We train the models with SGD with momentum, at an initial learning rate 0.1 and a momentum 0.9 with mini-batch size 100. For ResNet-32 models, the learning rate decays  $\times 0.1$  at 40K and 60K steps, for a total of 80K steps. For WRN and early stopped versions of ResNet-32 models, the learning rate decays at 40K and 50K steps, for a total of 60K steps. Under regular 0% noise settings, our base ResNet-32 gets 92.5% and 68.1% classification accuracy on CIFAR-10 and

Table 2. CIFAR BACKGROUNDFLIP under 40% noise ratio using a ResNet-32 model. Test accuracy shown in percentage. Top rows use only noisy data, and bottom rows use additional 10 clean images per class. “+ES” denotes early stopping; “FT” denotes fine-tuning.

MODEL	CIFAR-10	CIFAR-100
BASILINE	59.54 $\pm$ 2.16	37.82 $\pm$ 0.69
BASILINE +ES	64.96 $\pm$ 1.19	39.08 $\pm$ 0.65
RANDOM	69.51 $\pm$ 1.36	36.56 $\pm$ 0.44
WEIGHTED	79.17 $\pm$ 1.36	36.56 $\pm$ 0.44
REED SOFT +ES	63.47 $\pm$ 1.05	38.44 $\pm$ 0.90
REED HARD +ES	65.22 $\pm$ 1.06	39.03 $\pm$ 0.55
S-MODEL	58.60 $\pm$ 2.33	37.02 $\pm$ 0.34
S-MODEL +CONF	68.93 $\pm$ 1.09	46.72 $\pm$ 1.87
S-MODEL +CONF +ES	79.24 $\pm$ 0.56	54.50 $\pm$ 2.51
USING 10 CLEAN IMAGES PER CLASS		
CLEAN ONLY	15.90 $\pm$ 3.32	8.06 $\pm$ 0.76
BASILINE +FT	82.82 $\pm$ 0.93	54.23 $\pm$ 1.75
BASILINE +ES +FT	85.19 $\pm$ 0.46	55.22 $\pm$ 1.40
WEIGHTED +FT	85.98 $\pm$ 0.47	53.99 $\pm$ 1.62
S-MODEL +CONF +FT	81.90 $\pm$ 0.85	53.11 $\pm$ 1.33
S-MODEL +CONF +ES +FT	85.86 $\pm$ 0.63	55.75 $\pm$ 1.26
OURS	<b>86.73 <math>\pm</math> 0.48</b>	<b>59.30 <math>\pm</math> 0.60</b>

100, and the WRN-28-10 gets 95.5% and 78.2%. For the finetuning stage, we run extra 5K steps of training on the limited clean data.

We report the average test accuracy for 5 different random splits of clean and noisy labels, with 95% confidence interval in Table 1 and 2. The background classes for the 5 trials are [0, 1, 3, 5, 7] (CIFAR-10) and [7, 12, 41, 62, 85] (CIFAR-100).

### 4.3. Results and Discussion

The first result that draws our attention is that “Random” performs surprisingly well on the UNIFORMFLIP benchmark, outperforming all historical methods that we compared. Given that its performance is comparable with Baseline on BACKGROUNDFLIP and MNIST class imbalance, we hypothesize that random example weights act as a strong regularizer and under which the learning objective on UNIFORMFLIP is still consistent.

Regardless of the strong baseline, our method ranks the top on both UNIFORMFLIP and BACKGROUNDFLIP, showing our method is less affected by the changes in the noise type. On CIFAR-100, our method wins more than 3% compared to the state-of-the-art method.

**Understanding the reweighting mechanism** It is beneficial to understand how our reweighting algorithm contributes to learning more robust models during training. First, we use a pre-trained model (trained at half of the total iterations without learning rate decay) and measure the example weight distribution of a randomly sampled batch

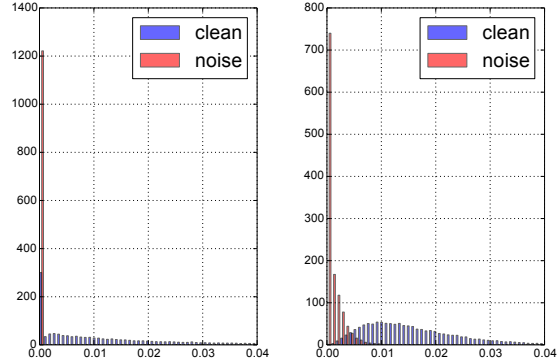


Figure 3. Example weights distribution on BACKGROUNDFLIP. Left: a hyper-validation batch, with randomly flipped background noises. Right: a hyper-validation batch containing only on a single label class, with flipped background noises, averaged across all non-background classes.

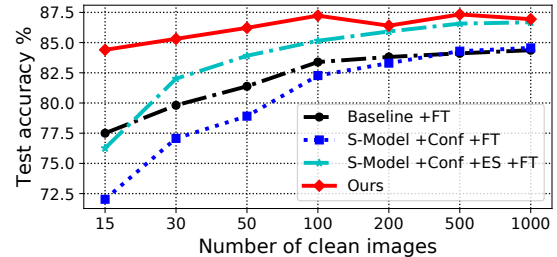


Figure 4. Effect of the number of clean images used, on CIFAR-10 with 40% of data flipped to label 3. “ES” denotes early stopping.

of validation images, which the model has never seen. As shown in the left figure of Figure 3, our model correctly pushes most noisy images to zero weights. Secondly, we conditioned the input mini-batch to be a single non-background class and randomly flip 40% of the images to the background, and we would like to see how well our model can distinguish clean and noisy images. As shown in Figure 3 right, the model is able to reliably detect images that are flipped to the background class.

**Robustness to overfitting noise** Throughout experimentation, we find baseline models can easily overfit to the noise in the training set. For example, shown in Table 2, applying early stopping (“ES”) helps the classification performance of “S-Model” by over 10% on CIFAR-10. Figure 6 compares the final confusion matrices of the baseline and the proposed algorithm, where a large proportion of noise transition probability is cleared in the final prediction. Figure 7 shows training curves on the BACKGROUNDFLIP experiments. After the first learning rate decay, both “Baseline” and “S-Model” quickly degrade their validation performance due to overfitting, while our model remains the same validation

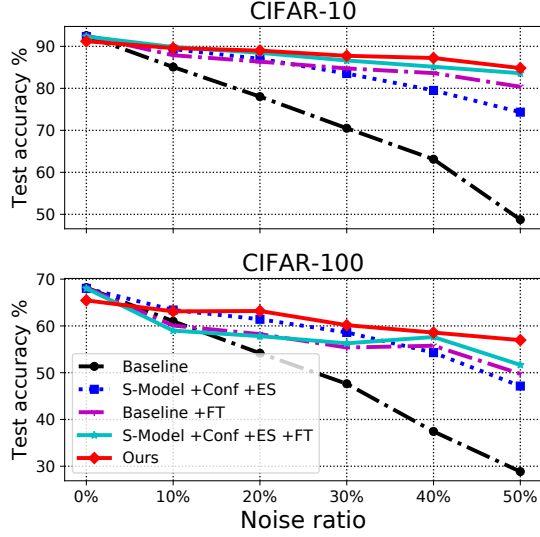


Figure 5. Model test accuracy on imbalanced noisy CIFAR experiments across various noise levels using a base ResNet-32 model. “ES” denotes early stopping, and “FT” denotes finetuning.

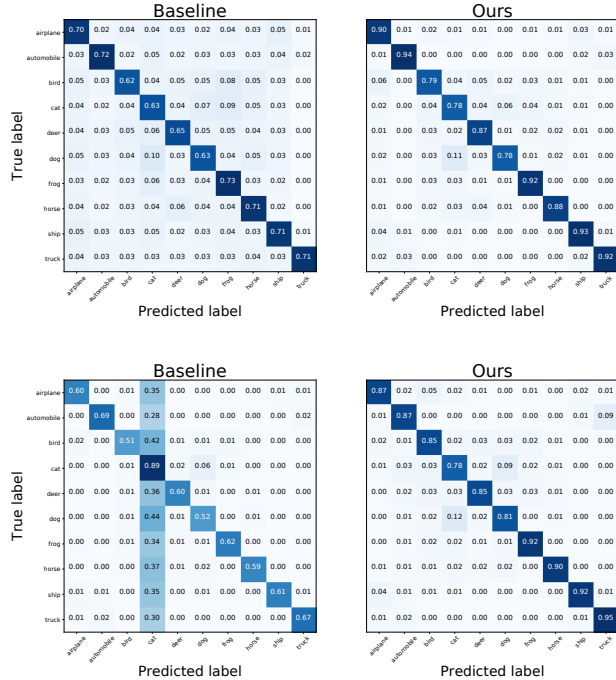


Figure 6. Confusion matrices on CIFAR-10 UNIFORMFLIP (top) and BACKGROUNDFLIP (bottom)

accuracy until termination. Note that here “S-Model” knows the oracle noise ratio in each class, and this information is not available in our method.

**Impact of the noise level** We would like to investigate how strongly our method can perform on a variety of noise levels. Shown in Figure 5, our method only drops 6%

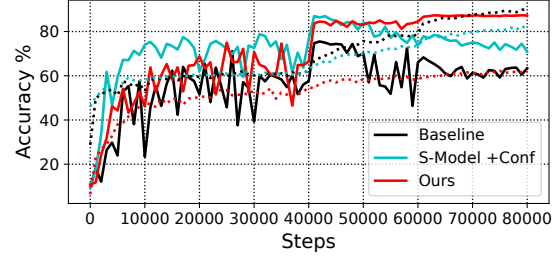


Figure 7. Training curve of a ResNet-32 on CIFAR-10 BACK-GROUNDFLIP under 40% noise ratio. Solid lines denote validation accuracy and dotted lines denote training. Our method is less prone to label noise overfitting.

accuracy when the noise ratio increased from 0% to 50%; whereas the baseline has dropped more than 40%. At 0% noise, our method only slightly underperforms baseline. This is reasonable since we are optimizing on the validation set, which is strictly a subset of the full training set, and therefore suffers from its own subsample bias.

**Size of the clean validation set** When the size of the clean validation set grows larger, fine-tuning on the validation set will be a reasonable approach. Here, we make an attempt to explore the tradeoff and understand when fine-tuning becomes beneficial. Figure 4 plots the classification performance when we varied the size of the clean validation on BACKGROUNDFLIP. Surprisingly, using 15 validation images for all classes only results in a 2% drop in performance, and the overall classification performance does not grow after having more than 100 validation images. In comparison, we observe a significant drop in performance when only fine-tuning on these 15 validation images for the baselines, and the performance catches up around using 1,000 validation images (100 per class). This phenomenon suggests that in our method the clean validation acts more like a regularizer rather than a data source for parameter fine-tuning, and potentially our method can be complementary with fine-tuning based method when the size of the clean set grows larger.

## 5. Conclusion

In this work, we propose an online meta-learning algorithm for reweighting training examples and training more robust deep learning models. While various types of training set biases exist and manually designed reweighting objectives have their own bias, our automatic reweighting algorithm shows superior performance dealing with class imbalance, noisy labels, and both. Our method can be directly applied to any deep learning architecture and is expected to train end-to-end without any additional hyperparameter search. Validating on every training step is a novel setting and we show that it has links with model regularization, which can be a fruitful future research direction.



## References

- Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek Gordon, Steiner, Benoit, Tucker, Paul A., Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI*, 2016.
- Andrychowicz, Marcin, Denil, Misha, Colmenarejo, Sergio Gomez, Hoffman, Matthew W., Pfau, David, Schaul, Tom, and de Freitas, Nando. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems, NIPS*, 2016.
- Angluin, Dana and Laird, Philip. Learning from noisy examples. *Machine Learning*, 2(4):343–370, Apr 1988. ISSN 1573-0565.
- Azadi, Samaneh, Feng, Jiashi, Jegelka, Stefanie, and Darrell, Trevor. Auxiliary image regularization for deep cnns with noisy labels. In *Proceedings of the 4th International Conference on Learning Representation, ICLR*, 2016.
- Bengio, Yoshua, Louradour, Jérôme, Collobert, Ronan, and Weston, Jason. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML*, 2009.
- Chang, Haw-Shiuan, Learned-Miller, Erik G., and McCallum, Andrew. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems, NIPS*, 2017.
- Chawla, Nitesh V., Bowyer, Kevin W., Hall, Lawrence O., and Kegelmeyer, W. Philip. SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16:321–357, 2002.
- Chen, Xinlei and Gupta, Abhinav. Webly supervised learning of convolutional networks. In *Proceedings of the 2015 IEEE International Conference on Computer Vision, ICCV*, 2015.
- Cordts, Marius, Omran, Mohamed, Ramos, Sebastian, Rehfeld, Timo, Enzweiler, Markus, Benenson, Rodrigo, Franke, Uwe, Roth, Stefan, and Schiele, Bernt. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.
- Dong, Qi, Gong, Shaogang, and Zhu, Xiatian. Class rectification hard mining for imbalanced deep learning. In *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, 2017.
- Finn, Chelsea, Abbeel, Pieter, and Levine, Sergey. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.
- Freund, Yoav and Schapire, Robert E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.
- Goldberger, Jacob and Ben-Reuven, Ehud. Training deep neural-networks using a noise adaptation layer. In *Proceedings of the 5th International Conference on Learning Representation, ICLR*, 2017.
- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.
- Hendrycks, Dan, Mazeika, Mantas, Wilson, Duncan, and Gimpel, Kevin. Using trusted data to train deep networks on labels corrupted by severe noise. *CoRR*, abs/1802.05300, 2018.
- Huang, Chen, Li, Yining, Loy, Chen Change, and Tang, Xiaoou. Learning deep representation for imbalanced classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.
- Jiang, Lu, Meng, Deyu, Zhao, Qian, Shan, Shiguang, and Hauptmann, Alexander G. Self-paced curriculum learning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.
- Jiang, Lu, Zhou, Zhengyuan, Leung, Thomas, Li, Li-Jia, and Fei-Fei, Li. Mentornet: Regularizing very deep neural networks on corrupted labels. *CoRR*, abs/1712.05055, 2017.
- Kahn, Herman and Marshall, Andy W. Methods of reducing sample size in monte carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953.
- Khan, Salman Hameed, Bennamoun, Mohammed, Sohel, Ferdous Ahmed, and Togneri, Roberto. Cost sensitive learning of deep feature representations from imbalanced data. *CoRR*, abs/1508.03422, 2015.
- Koh, Pang Wei and Liang, Percy. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.
- Kumar, M. Pawan, Packer, Benjamin, and Koller, Daphne. Self-paced learning for latent variable models. In

- Advances in Neural Information Processing Systems, NIPS*, 2010.
- Lake, Brenden M., Ullman, Tomer D., Tenenbaum, Joshua B., and Gershman, Samuel J. Building machines that learn and think like people. *Behav Brain Sci*, 40: e253, Jan 2017.
- Li, Yuncheng, Yang, Jianchao, Song, Yale, Cao, Liangliang, Luo, Jiebo, and Li, Li-Jia. Learning from noisy labels with distillation. In *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, 2017.
- Lin, Tsung-Yi, Goyal, Priya, Girshick, Ross B., He, Kaiming, and Dollár, Piotr. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, 2017.
- Lorraine, Jonathan and Duvenaud, David. Stochastic hyperparameter optimization through hypernetworks. *CoRR*, abs/1802.09419, 2018.
- Ma, Fan, Meng, Deyu, Xie, Qi, Li, Zina, and Dong, Xuanyi. Self-paced co-training. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.
- Malisiewicz, Tomasz, Gupta, Abhinav, and Efros, Alexei A. Ensemble of exemplar-svms for object detection and beyond. In *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, 2011.
- Muñoz-González, Luis, Biggio, Battista, Demontis, Ambra, Paudice, Andrea, Wongrassamee, Vasin, Lupu, Emil C., and Roli, Fabio. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS*, 2017.
- Natarajan, Nagarajan, Dhillon, Inderjit S., Ravikumar, Pradeep, and Tewari, Ambuj. Learning with noisy labels. In *Advances in Neural Information Processing Systems, NIPS*, 2013.
- Ravi, Sachin and Larochelle, Hugo. Optimization as a model for few-shot learning. In *Proceedings of the 5th International Conference on Learning Representations, ICLR*, 2017.
- Reddi, Sashank J., Hefny, Ahmed, Sra, Suvrit, Póczos, Barnabás, and Smola, Alexander J. Stochastic variance reduction for nonconvex optimization. In *Proceedings of the 33rd International Conference on Machine Learning, ICML*, 2016.
- Reed, Scott E., Lee, Honglak, Anguelov, Dragomir, Szegedy, Christian, Erhan, Dumitru, and Rabinovich, Andrew. Training deep neural networks on noisy labels with bootstrapping. *CoRR*, abs/1412.6596, 2014.
- Ren, Mengye, Triantafillou, Eleni, Ravi, Sachin, Snell, Jake, Swersky, Kevin, Tenenbaum, Joshua B., Larochelle, Hugo, and Zemel, Richard S. Meta learning for few-shot semi-supervised classification. In *Proceedings of the 6th International Conference on Learning Representations, ICLR*, 2018.
- Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision, IJCV*, 115(3):211–252, 2015.
- Sukhbaatar, Sainbayar and Fergus, Rob. Learning from noisy labels with deep neural networks. *CoRR*, abs/1406.2080, 2014.
- Thrun, Sebastian and Pratt, Lorien. *Learning to Learn*. Springer, 1998.
- Ting, Kai Ming. A comparative study of cost-sensitive boosting algorithms. In *Proceedings of the 17th International Conference on Machine Learning, ICML*, 2000.
- Vahdat, Arash. Toward robustness against label noise in training deep discriminative neural networks. In *Advances in Neural Information Processing Systems, NIPS*, 2017.
- Wang, Yixin, Kucukelbir, Alp, and Blei, David M. Robust probabilistic modeling with bayesian data reweighting. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.
- Wu, Yuhuai, Ren, Mengye, Liao, Renjie, and Grosse, Roger B. Understanding short-horizon bias in stochastic meta-optimization. In *Proceedings of the 6th International Conference on Learning Representations, ICLR*, 2018.
- Xiao, Tong, Xia, Tian, Yang, Yi, Huang, Chang, and Wang, Xiaogang. Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2015.
- Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. In *Proceedings of the British Machine Vision Conference, BMVC*, 2016.
- Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding deep learning requires rethinking generalization. In *Proceedings of the 5th International Conference on Learning Representations, ICLR*, 2017.

## A. Reweighting in an MLP

We show the complete derivation below on calculating the example weights in an MLP network.

$$\frac{\partial}{\partial \epsilon_{i,t}} \mathbb{E} [f^v(\theta_{t+1}(\epsilon))] \Big|_{\epsilon_{i,t}=0} \quad (17)$$

$$= \frac{1}{m} \sum_{j=1}^m \frac{\partial}{\partial \epsilon_{i,t}} f_j^v(\theta_{t+1}(\epsilon)) \Big|_{\epsilon_{i,t}=0} \quad (18)$$

$$= \frac{1}{m} \sum_{j=1}^m \frac{\partial f_j^v(\theta)}{\partial \theta} \Big|_{\theta=\theta_t}^\top \frac{\partial \theta_{t+1}(\epsilon_{i,t})}{\partial \epsilon_{i,t}} \Big|_{\epsilon_{i,t}=0} \quad (19)$$

$$\propto - \frac{1}{m} \sum_{j=1}^m \frac{\partial f_j^v(\theta)}{\partial \theta} \Big|_{\theta=\theta_t}^\top \frac{\partial f_i(\theta)}{\partial \theta} \Big|_{\theta=\theta_t} \quad (20)$$

$$= - \frac{1}{m} \sum_{j=1}^m \sum_{l=1}^L \frac{\partial f_j^v}{\partial \theta_l} \Big|_{\theta_l=\theta_{l,t}}^\top \frac{\partial f_i}{\partial \theta_l} \Big|_{\theta_l=\theta_{l,t}} \quad (21)$$

$$= - \frac{1}{m} \sum_{j=1}^m \sum_{l=1}^L \text{vec} \left( \tilde{z}_{j,l-1}^v g_{j,l}^{v\top} \right)^\top \text{vec} \left( \tilde{z}_{i,l-1} g_{i,l}^\top \right) \quad (22)$$

$$= - \frac{1}{m} \sum_{j=1}^m \sum_{l=1}^L \sum_{p=1}^{D_1} \sum_{q=1}^{D_2} \tilde{z}_{j,l-1}^v g_{j,l,q}^v \tilde{z}_{i,l-1,p} g_{i,l,p} g_{i,l,q} \quad (23)$$

$$= - \frac{1}{m} \sum_{j=1}^m \sum_{l=1}^L \sum_{p=1}^{D_1} \tilde{z}_{j,l-1}^v \tilde{z}_{i,l-1,p} \sum_{q=1}^{D_2} g_{j,l,q}^v g_{i,l,q} \quad (24)$$

$$= - \frac{1}{m} \sum_{j=1}^m \sum_{l=1}^L (\tilde{z}_{j,l-1}^{v\top} \tilde{z}_{i,l-1}) (g_{j,l}^{v\top} g_{i,l}). \quad (25)$$

## B. Convergence of our method

This section provides the proof for Lemma 1.

**Lemma.** Suppose the validation loss function is Lipschitz-smooth with constant  $L$ , and the train loss function  $f_i$  of training data  $x_i$  have  $\sigma$ -bounded gradients. Let the learning rate  $\alpha_t$  satisfies  $\alpha_t \leq \frac{2n}{L\sigma^2}$ , where  $n$  is the training batch size. Then, following our algorithm, the validation loss always monotonically decreases for any sequence of training batches, namely,

$$G(\theta_{t+1}) \leq G(\theta_t), \quad (26)$$

where  $G(\theta)$  is the total validation loss

$$G(\theta) = \frac{1}{M} \sum_{i=1}^M f_i^v(\theta_{t+1}(\epsilon)). \quad (27)$$

Furthermore, in expectation, the equality in Eq. 26 holds only when the gradient of validation loss becomes 0 at some time step  $t$ , namely  $\mathbb{E}_t [G(\theta_{t+1})] = G(\theta_t)$  if and only if  $\nabla G(\theta_t) = 0$ , where the expectation is taking over possible training batches at time step  $t$ .

*Proof.* Suppose we have a small validation set with  $M$  clean data  $\{x_1, x_2, \dots, x_M\}$ , each associating with a validation loss function  $f_i(\theta)$ , where  $\theta$  is the parameter of the model. The overall validation loss would be,

$$G(\theta) = \frac{1}{M} \sum_{i=1}^M f_i(\theta). \quad (28)$$

Now, suppose we have another  $N - M$  training data,  $\{x_{M+1}, x_{M+2}, \dots, x_N\}$ , and we add those validation data into this set to form our large training dataset  $T$ , which has  $N$  data in total. The overall training loss would be,

$$F(\theta) = \frac{1}{M} \sum_{i=1}^N f_i(\theta). \quad (29)$$

For simplicity, since  $M \ll N$ , we assume that the validation data is a subset of the training data. During training, we take a mini-batch  $B$  of training data at each step, and  $|B| = n$ . Following some similar derivation as Appendix A, we have the following update rules:

$$\theta_{t+1} = \theta_t - \frac{\alpha_t}{n} \sum_{i \in B} \max \{ \nabla G^\top \nabla f_i, 0 \} \nabla f_i, \quad (30)$$

where  $\alpha_t$  is the learning rate at time-step  $t$ . Since all gradients are taken at  $\theta_t$ , we omit  $\theta_t$  in our notations.

Since the validation loss  $G(\theta)$  is Lipschitz-smooth, we have

$$G(\theta_{t+1}) \leq G(\theta_t) + \nabla G^\top \Delta \theta + \frac{L}{2} \|\Delta \theta\|^2. \quad (31)$$

Plugging our updating rule (Eq. 30),

$$G(\theta_{t+1}) \leq G(\theta_t) - I_1 + I_2, \quad (32)$$

where,

$$\begin{aligned} I_1 &= \frac{\alpha_t}{n} \sum_{i \in B} \max \{ \nabla G^\top \nabla f_i, 0 \} \nabla G^\top \nabla f_i \\ &= \frac{\alpha_t}{n} \sum_{i \in B} \max \{ \nabla G^\top \nabla f_i, 0 \}^2, \end{aligned} \quad (33)$$

and,

$$I_2 = \frac{L}{2} \left\| \frac{\alpha_t}{n} \sum_{i \in B} \max \{ \nabla G^\top \nabla f_i, 0 \} \nabla f_i \right\|^2 \quad (34)$$

$$\leq \frac{L}{2} \frac{\alpha_t^2}{n^2} \sum_{i \in B} \left\| \max \{ \nabla G^\top \nabla f_i, 0 \} \nabla f_i \right\|^2 \quad (35)$$

$$= \frac{L}{2} \frac{\alpha_t^2}{n^2} \sum_{i \in B} \max \{ \nabla G^\top \nabla f_i, 0 \}^2 \|\nabla f_i\|^2 \quad (36)$$

$$\leq \frac{L}{2} \frac{\alpha_t^2}{n^2} \sum_{i \in B} \max \{ \nabla G^\top \nabla f_i, 0 \}^2 \sigma^2. \quad (37)$$

The first inequality (Eq. 35) comes from the triangle inequality. The second inequality (Eq. 37) holds since  $f_i$  has  $\sigma$ -bounded gradients. If we denote  $\mathcal{T}_t = \sum_{i \in B} \max\{\nabla G^\top \nabla f_i, 0\}^2$ , where  $t$  stands for the time-step  $t$ , then

$$G(\theta_{t+1}) \leq G(\theta_t) - \frac{\alpha_t}{n} \mathcal{T}_t \left(1 - \frac{L\alpha_t\sigma^2}{2n}\right). \quad (38)$$

Note that by definition,  $\mathcal{T}_t$  is non-negative, and since  $\alpha_t \leq \frac{2n}{L\sigma^2}$ , it follows that  $G(\theta_{t+1}) \leq G(\theta_t)$  for any  $t$ .

Next, we prove  $\mathbb{E}_t[\mathcal{T}_t] = 0$  if and only if  $\nabla G = 0$ , and  $\mathbb{E}_t[\mathcal{T}_t] > 0$  if and only if  $\nabla G \neq 0$ , where the expectation is taken over all possible training batches at time step  $t$ . It is obvious that when  $\nabla G = 0$ ,  $\mathbb{E}_t[\mathcal{T}_t] = 0$ . If  $\nabla G \neq 0$ , from the inequality below, we firstly know that there must exist a validation example  $x_{j,0 \leq j \leq M}$  such that  $\nabla G^\top \nabla f_j > 0$ ,

$$0 < \|\nabla G\|^2 = \nabla G^\top \nabla G = \frac{1}{M} \sum_{i=1}^M \nabla G^\top \nabla f_i. \quad (39)$$

Secondly, there is a non-zero possibility  $p$  to sample a training batch  $B$  such that it contains this data  $x_j$ . Also noticing that  $\mathcal{T}_t$  is a non-negative random variable, we have,

$$\begin{aligned} \mathbb{E}_t[\mathcal{T}_t] &\geq p \sum_{i \in B} \max\{\nabla G^\top \nabla f_i, 0\}^2 \\ &\geq p \max\{\nabla G^\top \nabla f_j, 0\}^2 \\ &= p (\nabla G^\top \nabla f_j)^2 > 0. \end{aligned} \quad (40)$$

Therefore, if we take expectation over the training batch on both sides of Eq. 38, we can conclude that,

$$\mathbb{E}_t[G(\theta_{t+1})] \leq G(\theta_t), \quad (41)$$

where the equality holds if and only if  $\nabla G = 0$ . This finishes our proof for Lemma 1.  $\square$

### C. Convergence rate of our method

This section provides proof for Theorem 2.

**Theorem.** Suppose  $G$ ,  $f_i$  and  $\alpha_t$  satisfy the aforementioned conditions, then Algorithm 1 achieves  $\mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \epsilon$  in  $O(1/\epsilon^2)$  steps. More specifically,

$$\min_{0 \leq t \leq T} \mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \frac{C}{\sqrt{T}}, \quad (42)$$

where  $C$  is some constant independent of the convergence process.

*Proof.* From the proof of Lemma 1, we have

$$\begin{aligned} &\frac{\alpha_t}{n} \left(1 - \frac{L\alpha_t\sigma^2}{2n}\right) \mathbb{E}_{0 \sim t}[\mathcal{T}_t] \\ &\leq \mathbb{E}_{0 \sim t-1}[G(\theta_t)] - \mathbb{E}_{0 \sim t}[G(\theta_{t+1})]. \end{aligned} \quad (43)$$

If we let  $\alpha_t$  to be a constant  $\alpha < \frac{2n}{L\sigma^2}$  (or a decay positive sequence upper bounded by  $\alpha$ ), and let  $\kappa = \left(1 - \frac{L\alpha\sigma^2}{2n}\right) \alpha/n > 0$ , then we have,

$$\begin{aligned} \kappa \sum_{t=0}^T \mathbb{E}_{0 \sim t}[\mathcal{T}_t] &\leq \mathbb{E}_0[G(\theta_0)] - \mathbb{E}_{0 \sim T}[G(\theta_{T+1})] \\ &\leq G(\theta_0) - G(\theta^*), \end{aligned} \quad (44)$$

where  $G(\theta^*)$  is the global minimum of function  $G$ . Therefore, it is obvious to see that there exist a time-step  $0 \leq \tau \leq T$  such that,

$$\mathbb{E}_{0 \sim \tau}[\mathcal{T}_\tau] \leq \frac{G(\theta_0) - G(\theta^*)}{\kappa T}. \quad (45)$$

We next prove that for this time-step  $\tau$ , the gradient square  $\mathbb{E}_{0 \sim \tau-1}[\|\nabla G(\theta_\tau)\|^2]$  is smaller than  $O(1/\sqrt{T})$ . Considering such  $M$  training batches  $B_1, B_2, \dots, B_M$  such that  $B_i$  is guaranteed to contain  $x_i$ . We know that those batches have non-zero sampling probability, denoted as  $p_1, p_2, \dots, p_M$ . We also denote  $p = \min\{p_1, p_2, \dots, p_M\}$ . Now, we have,

$$M \mathbb{E}_{0 \sim \tau}[\mathcal{T}_\tau] = \mathbb{E}_{0 \sim \tau-1} \left[ M \mathbb{E}_\tau[\mathcal{T}_\tau] \right] \quad (46)$$

$$\geq \mathbb{E}_{0 \sim \tau-1} \left[ \sum_{k=1}^M p_k \sum_{i \in B_k} \max\{\nabla G^\top \nabla f_i, 0\}^2 \right] \quad (47)$$

$$\geq p \mathbb{E}_{0 \sim \tau-1} \left[ \sum_{k=1}^M \sum_{i \in B_k} \max\{\nabla G^\top \nabla f_i, 0\}^2 \right] \quad (48)$$

$$\geq p \mathbb{E}_{0 \sim \tau-1} \left[ \sum_{i=1}^M \max\{\nabla G^\top \nabla f_i, 0\}^2 \right] \quad (49)$$

$$= p \sum_{i=1}^M \mathbb{E}_{0 \sim \tau-1} [\max\{\nabla G^\top \nabla f_i, 0\}^2] \quad (50)$$

$$\geq p \sum_{i=1}^M \left( \mathbb{E}_{0 \sim \tau-1} [\max\{\nabla G^\top \nabla f_i, 0\}] \right)^2 \quad (51)$$

$$\geq \frac{p}{M} \left( \sum_{i=1}^M \mathbb{E}_{0 \sim \tau-1} [\max\{\nabla G^\top \nabla f_i, 0\}] \right)^2 \quad (52)$$

The inequality in Eq. 47 comes from the non-negativeness of  $\mathcal{T}_t$ , the inequality in Eq. 51 comes from the property of expectation, and the final inequality in Eq. 52 comes from



the Cauchy-Schwartz inequality. Therefore,

$$\begin{aligned}
 & \sum_{i=1}^M \mathbb{E}_{0 \sim \tau-1} [\max\{\nabla G^\top \nabla f_i, 0\}] \\
 & \leq M \sqrt{\frac{(G(\theta_0) - G(\theta^*))}{p\kappa}} \sqrt{\frac{1}{T}},
 \end{aligned} \tag{53}$$

and so,

$$\begin{aligned}
 & \mathbb{E}_{0 \sim \tau-1} [\|\nabla G(\theta_\tau)\|^2] \\
 & = \mathbb{E}_{0 \sim \tau-1} [\nabla G^\top \nabla G] \\
 & = \mathbb{E}_{0 \sim \tau-1} \left[ \nabla G^\top \left( \frac{\sum_{i=0}^M \nabla f_i}{M} \right) \right] \\
 & \leq \frac{1}{M} \sum_{i=1}^M \mathbb{E}_{0 \sim \tau-1} [\max\{\nabla G^\top \nabla f_i, 0\}] \\
 & \leq \sqrt{\frac{G(\theta_0) - G(\theta^*)}{p\kappa}} \sqrt{\frac{1}{T}}.
 \end{aligned} \tag{54}$$

Therefore, we can conclude that our algorithm can always achieve  $\min_{0 \leq t < T} \mathbb{E} [\|\nabla G(\theta_t)\|^2] \leq O(\sqrt{1/T})$  in  $T$  steps, and this finishes our proof of Theorem 2.  $\square$