Searching to Exploit Memorization Effect in Learning with Noisy Labels

Quanming Yao [1] **Hansi Yang** [2] **Bo Han** [3 4] **Gang Niu** [4] **James T. Kwok** [5]

Abstract

Sample selection approaches are popular in robust learning from noisy labels. However, how to properly control the selection process so that deep networks can benefit from the memorization effect is a hard problem. In this paper, motivated by the success of automated machine learning (AutoML), we model this issue as a function approximation problem. Specifically, we design a domain-specific search space based on general patterns of the memorization effect and propose a novel Newton algorithm to solve the bi-level optimization problem efficiently. We further provide theoretical analysis of the algorithm, which ensures a good approximation to critical points. Experiments are performed on benchmark data sets. Results demonstrate that the proposed method is much better than the state-of-the-art noisy-label-learning approaches, and also much more efficient than existing AutoML algorithms.

1. Introduction

Deep networks have enjoyed huge empirical success in a wide variety of tasks, such as image processing, speech recognition, language modeling and recommender systems (Goodfellow et al., 2016). However, this highly counts on the availability of large amounts of quality data, which may not be feasible in practice. Instead, many large data sets are collected from crowdsourcing platforms or crawled from the internet, and the obtained labels are noisy (Patrini et al., 2017). As deep networks have large learning capacities, they will eventually overfit the noisy labels, leading to poor generalization performance (Zhang et al., 2016; Arpit et al., 2017; Jiang et al., 2018).

To reduce the negative effects of noisy labels, a number of methods have been recently proposed (Sukhbaatar et al., 2015; Reed et al., 2015; Patrini et al., 2017; Ghosh et al., 2017; Malach & Shalev-Shwartz, 2017; Liu & Tao, 2015; Cheng et al., 2020). They can be grouped into three main categories. The first one is based on estimating the label transition matrix, which captures how correct labels are flipped to the wrong ones (Sukhbaatar et al., 2015; Reed et al., 2015; Patrini et al., 2017; Ghosh et al., 2017). However, this can be fragile to heavy noise and is unable to handle a large number of labels (Han et al., 2018). The second type is based on regularization (Miyato et al., 2016; Laine & Aila, 2017; Tarvainen & Valpola, 2017). However, since deep networks are usually over-parameterized, they can still completely memorize the noisy data given sufficient training time (Zhang et al., 2016).

The third approach, which is the focus in this paper, is based on selecting (or weighting) possibly clean samples in each iteration for training (Jiang et al., 2018; Han et al., 2018; Yu et al., 2019; Wang et al., 2019). Intuitively, by making the training data less noisy, better performance can be obtained. Representative methods include the MentorNet (Jiang et al., 2018) and Co-teaching (Han et al., 2018; Yu et al., 2019). Specifically, MentorNet uses an additional network to select clean samples for training of a StudentNet. Co-teaching improves MentorNet by simultaneously maintaining two networks with identical architectures during training, and each network is updated using the small-loss samples from the other network.

In sample selection, a core issue is how many small-loss samples are to be selected in each iteration. While discarding a lot of samples can avoid training with noisy labels, dropping too many can be overly conservative and lead to lower accuracy (Han et al., 2018). Co-teaching uses the observation that deep networks usually learn easy patterns before overfitting the noisy samples (Zhang et al., 2016; Arpit et al., 2017). This *memorization effect* has been widely seen in various deep networks (Patrini et al., 2017; Ghosh et al., 2017; Han et al., 2018). Hence, during the early stage of training, Co-teaching drops very few samples as the network will not memorize the noisy data. As training proceeds, the network starts to memorize the noisy data.

[1]4Paradigm Inc (Hong Kong) [2]Department of Electrical Engineering, Tshinghua University [3]Department of Computer Science, Hong Kong Baptist University [4]RIKEN Center for Advanced Intelligence Project [5]Department of Computer Science and Engineering, Hong Kong University of Science and Technology. Correspondence to: Quanming Yao <yaoquanming@4paradigm.com>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

This is avoided in Co-teaching by gradually dropping more samples according to a pre-defined schedule. Empirically, this signiificantly improves the network's generalization performance on noisy labels (Jiang et al., 2018; Han et al., 2018). However, it is unclear if its manally-designed schedule is "optimal". Moreover, the schedule is not data-dependent, but is the same for all data sets. Manually finding a good schedule for each and every data set is clearly very time-consuming and infeasible.

Motivated by the recent success of automated machine learning (AutoML) (Hutter et al., 2018; Yao & Wang, 2018), in this paper we propose to exploit the memorization effect automatically using AutoML. We first formulate the learning of schedule as a bi-level optimization problem, similar to that in neural architecture search (NAS) (Zoph & Le, 2017). A search space for the schedule is designed based on the learning curve behaviors shared by deep networks. This space is expressive, and yet compact with only a small number of hyperparameters. However, computing the gradient is difficult as sample selection is a discrete operator. To avoid this problem and perform efficient search, we propose to use stochastic relaxation (Geman & Geman, 1984) together with Newton's method to capture information from both the model and optimization objective. Convergence analysis is provided, and extensive experiments are performed on benchmark datasets. Empirically, the proposed method outperforms state-of-the-art methods, and can select a higher proportion of clean samples than other sample selection methods. Ablation studies show that the chosen search space is appropriate, and the proposed search algorithm is faster than popular AutoML search algorithms in this context.

Notation. In the sequel, scalars are in lowercase letters, vectors are in lowercase boldface letters, and matrices are in uppercase boldface letters. The gradient of a function $\mathcal{J}$ is denoted $\nabla \mathcal{J}$, and $\|\cdot\|$ denotes the ℓ_2-norm of a vector.

2. Related work

2.1. Automated Machine Learning (AutoML)

Recently, AutoML has shown to be very useful in the design of machine learning models (Hutter et al., 2018; Yao & Wang, 2018). Two of its important ingredients are:

1. *Search space*, which needs to be specially designed for each AutoML problem (Baker et al., 2017; Liu et al., 2019; Zhang et al., 2020). It should be general (so as to cover existing models), yet not too general (otherwise searching in this space will be expensive).
2. *Search algorithms*: Two types are popularly used. The first includes derivative-free optimization methods, such as reinforcement learning (Zoph & Le, 2017; Baker et al., 2017), genetic programming (Xie & Yuille, 2017), and Bayesian optimization (Bergstra et al., 2011; Snoek et al., 2012). The second type is gradient-based, and updates the parameters and hyperparameters in an alternating manner. On NAS problems, gradient-based methods are usually more efficient than derivative-free methods (Liu et al., 2019; Akimoto et al., 2019; Yao et al., 2020).

2.2. Learning from Noisy Labels

The state-of-the-arts usually combat noisy labels by sample selection (Jiang et al., 2018; Han et al., 2018; Malach & Shalev-Shwartz, 2017; Yu et al., 2019; Wang et al., 2019), which only uses the "clean" samples (with relatively small losses) from each mini-batch for training. The general procedure is shown in Algorithm 1. Let f be the classifier to be learned. At the tth iteration, a subset $\mathcal{D}_f$ of small-loss samples are selected from the mini-batch $\mathcal{D}$ (step 3). These "clean" samples are then used to update the network parameters in step 4.

Algorithm 1 General procedure on using sample selection to combat noisy labels.

```
1: for $t = 0, \dots, T-1$ do
2:     draw a mini-batch $\mathcal{D}$ from $\mathcal{D}$;
3:     select $R(t)$ small-loss samples $\mathcal{D}_f$ from $\mathcal{D}$ based on network's predictions;
4:     update network parameter using $\mathcal{D}_f$;
5: end for
```

3. Proposed Method

3.1. Motivation

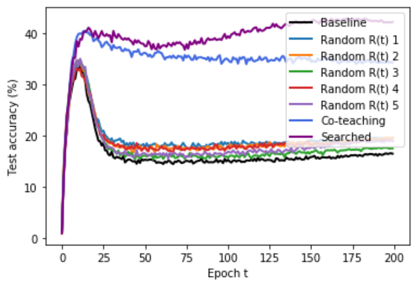
In step 3 of Algorithm 1, $R(\cdot)$ controls how many samples are selected into $\mathcal{D}_f$. As can be seen from Figure 1(a), its setting is often critical to the performance, and random $R(t)$ schedules have only marginal improvements over directly training on the whole noisy data set (denoted "Baseline" in the figure) (Han et al., 2018; Ren et al., 2018). Moreover, while having a large $R(\cdot)$ can avoid training with noisy labels, dropping too many samples can lead to lower accuracy, as demonstrated in Table 8 of (Han et al., 2018).

Based on the memorization effect in deep networks (Zhang et al., 2016), Co-teaching (Han et al., 2018) (and its variant Co-teaching+ (Yu et al., 2019)) designed the following schedule:

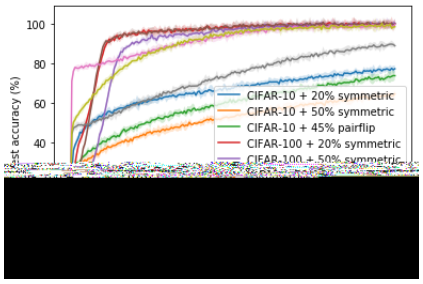
$$R(t) = 1 - \tau \cdot \min((t/t_k)^c, 1), \tag{1}$$

where τ, c and t_k are some hyperparameters. As can be seen from Figure 1(a), it can significantly improve the performance over random schedules.

While $R(\cdot)$ is critical and that it is important to exploit the memorization effect, it is unclear if the schedule in

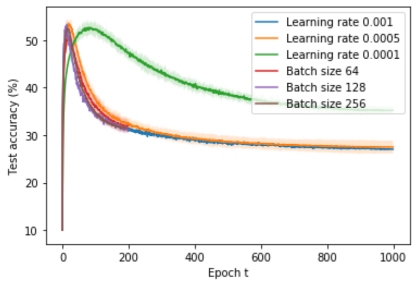


(a) Impact of $R(t)$.



(b) Different data sets (training accuracy).

(c) Different data sets (testing accuracy).



(d) Different architectures.

(e) Different optimizers.

(f) Different optimizer settings.

Figure 1. Training and testing accuracies on CIFAR-10, CIFAR-100, and MNIST using various architectures, optimizers, and optimizer settings. The detailed setup is in Appendix **??**.

(1) is "optimal". Moreover, the same schedule is used by Co-teaching on all the data sets. This is expected to be suboptimal, but it is hard to find $R(\cdot)$ for each and every data set manually. This motivates us to formulate the design of $R(\cdot)$ as an AutoML problem that searches for a good $R(\cdot)$ automatically (Section 3.2). The two important ingredients of AutoML, namely, search space and search algorithm, will then be described in Sections 3.3 and 3.4, respectively.

3.2. Formulation as an AutoML Problem

Let the noisy training (resp. clean validation) data set be $\mathcal{D}_{\text{tr}}$ (resp. $\mathcal{D}_{\text{val}}$), the training (resp. validation) loss be $\mathcal{L}_{\text{tr}}$ (resp. $\mathcal{L}_{\text{val}}$), and f be a neural network with model parameter w. We formulate the design of $R(\cdot)$ in Algorithm 1 as the following AutoML problem:

$$R^* = \arg\min_{R(\cdot)\in\mathcal{F}} \mathcal{L}_{\text{val}}(f(\boldsymbol{w}^*; R), \mathcal{D}_{\text{val}}), \quad (2)$$

$$\text{s.t. } \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathcal{L}_{\text{tr}}(f(\boldsymbol{w}; R), \mathcal{D}_{\text{tr}}). \quad (3)$$

where $\mathcal{F}$ is the search space of $R(\cdot)$.

Similar to the AutoML problems of auto-sklearn (Feurer et al., 2015) and NAS (Zoph & Le, 2017; Liu et al., 2019; Yao et al., 2020), this is also a bi-level optimization problem (Colson et al., 2007). At the outer level (subproblem (2)), a good $R(\cdot)$ is searched based on the validation set. At the lower level (subproblem (3)), we find the model parameters using the training set.

3.3. Designing the Search Space $\mathcal{F}$

In Section 3.3.1, we first discuss some observations from the learning curves of deep networks. These are then used in the design of an appropriate search space for $R(\cdot)$ in Section 3.3.2.

3.3.1. Observations from Learning Curves

Figures 1(b)-1(f) show the training and validation set accuracies obtained on the MNIST, CIFAR-10, CIFAR-100 data sets, which are corrupted with different types and levels of label noise (symmetric flipping 20%, symmetric flipping 50%, and pair flipping 45%), using a number of architectures (ResNet (He et al., 2016), DenseNet (Huang et al., 2017) and small CNN models in (Yu et al., 2019)), optimizers (SGD (Bottou, 2010), Adam (Kingma & Ba, 2014) and RMSProp (Hinton et al., 2012)) and optimizer settings (learning rate and batch size).

As can be seen, the training accuracy always increases as training progresses (Figure 1(b)), while the testing accuracy first increases and then slowly drops due to over-fitting (Figure 1(c)). Note that this pattern is independent of the network architecture (Figure 1(d)), choice of optimizer (Figure 1(e)), and hyperparameter (Figure 1(f)).

Recall that deep networks usually learn easy patterns first before memorizing and overfitting the noisy samples (Arpit et al., 2017). From (1) and Figure 1, we have the following observations on $R(t)$:

- During the initial phase when the learning curve rises, the deep network is plastic and can learn easy patterns from the data. In this phase, one can allow a larger $R(t)$ as there is little risk of memorization. Hence, at time $t = 0$, we can set $R(0) = 1$ and the entire noisy data set is used.
- As training proceeds and the learning curve has peaked, the network starts to memorize and overfit the noisy samples. Hence, $R(t)$ should then decrease. As can be seen from Figure 1(a), this can significantly improve the network's generalization performance on noisy labels.
- Finally, as the network gets less plastic and in case $R(t)$ drops too much at the beginning, it may be useful to allow $R(t)$ to slowly increase so as to enable learning.

The above motivates us to impose the following prior knowledge on the search space $\mathcal{F}$ of $R(\cdot)$. An example $R(\cdot)$ is shown in Figure 2.

Assumption 1 (A Prior on $\mathcal{F}$)**.** *The shape of $R(\cdot)$ should be opposite to that of the learning curve. Besides, as in (Han et al., 2018; Yu et al., 2019), it is natural to have $R(t) \in [0, 1]$ and $R(0) = 1$.*

3.3.2. Imposing Prior Knowledge

To allow efficient search, the search space has to be small but not too small. To achieve this, we impose the prior knowledge proposed in Section 3.3.1 on $\mathcal{F}$. Specifically, we use k basis functions (f_i's) whose shapes follow Assumption 1 (shown in Table 1 and Figure 2). The exact choice of these basis functions is not important. The search space for $R(\cdot)$ is then defined as:

$$\mathcal{F} \equiv \left\{ R(t) = \sum_{i=1}^{k} \alpha_i f_i(t; \beta_i) : \sum_i \alpha_i = 1, \alpha_i \geq 0 \right\}, \quad (4)$$

where β_i is the hyperparameter associated with basis function f_i. In the experiments, we set all β_i's to be in the range $[0, 1]$. Let $\boldsymbol{\alpha} = \{\alpha_i\}$, $\boldsymbol{\beta} = \{\beta_i\}$ and $\boldsymbol{x} \equiv \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$. The search algorithm to be introduced will then only need to search for a small set of hyperparameters $\boldsymbol{x}$.

Table 1. The four basis functions used to define the search space in the experiments. Here, a_i's are the hyperparameters.

f_1	$e^{-a_2 t^{a_1}} + a_3 (\frac{t}{T})^{a_4}$
f_2	$e^{-a_2 t^{a_1}} + a_3 \frac{\log(1+t^{a_4})}{\log(1+T^{a_4})}$
f_3	$\frac{1}{(1+a_2 t)^{a_1}} + a_3 (\frac{t}{T})^{a_4}$
f_4	$\frac{1}{(1+a_2 t)^{a_1}} + a_3 \frac{\log(1+t^{a_4})}{\log(1+T^{a_4})}$

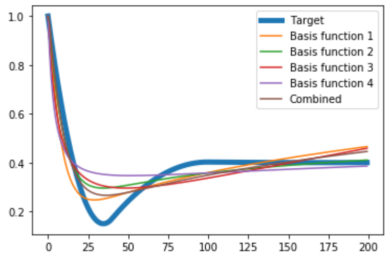


Figure 2. Plots of the basis functions in Table 1. An example $R(\cdot)$ to be learned is shown in blue.

With $\mathcal{F}$ in (4), the outer problem in (2) becomes

$$\{\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*\} = \arg\min_{R(\cdot)\in\mathcal{F}} \mathcal{L}_{\text{val}}(f(\boldsymbol{w}^*; R), \mathcal{D}_{\text{val}}), \quad (5)$$

and the optimal R^* in (2) is $\sum_{i=1}^{k} \alpha_i^* f_i(t; \beta_i^*)$.

3.3.3. Discussion

As will be shown in Section 4.2.1, the search space used in Co-teaching and Co-teaching+ is not large enough to ensure good performance. Besides, the design of $R(t)$ can be considered as a function learning problem, and general function approximators (such as radial basis function networks and multilayer perceptrons) can also be used. However, as will be demonstrated in Section 4.2.1, the resultant search space is too large for efficient search, while the prior on $\mathcal{F}$ in (4) can provide satisfactory performance. Note that the proposed search space can well approximate the space in Co-teaching (details are in Appendix **??**).

3.4. Search Algorithm Based on Relaxation

Gradient-based methods (Bengio, 2000; Liu et al., 2019; Yao et al., 2020) have been popularly used in NAS and hyperparameter optimization. Usually, the gradient w.r.t. hyperparameter $\boldsymbol{x}$ is computed via the chain rule as: $\nabla_{\boldsymbol{x}} \mathcal{L}_{\text{val}} = \nabla_{\boldsymbol{w}^*} \mathcal{L}_{\text{val}} \cdot \nabla_{\boldsymbol{x}} \boldsymbol{w}^*$. However, $\nabla_{\boldsymbol{x}} \boldsymbol{w}^*$ is hard to obtain here, as the hyperparameters in $R(\cdot)$ control the selection of samples in each mini-batch, a discrete operation.

3.4.1. Stochastic Relaxation with Newton's Method

To avoid a direct computation of the gradient w.r.t $\boldsymbol{x}$, we propose to transform problem (2) with stochastic relaxation (Geman & Geman, 1984). This has also been recently explored in AutoML (Baker et al., 2017; Pham et al., 2018; Akimoto et al., 2019). Specifically, instead of (2), we consider the following optimization problem:

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) \equiv \int_{\boldsymbol{x}\in\mathcal{F}} f(\boldsymbol{x}) p_{\boldsymbol{\theta}}(\boldsymbol{x}) \, d\boldsymbol{x}, \quad (6)$$

where $f(\boldsymbol{x}) \equiv \mathcal{L}_{\text{val}}(f(\boldsymbol{w}^*; R(\boldsymbol{x})), \mathcal{D}_{val})$ in (5), and $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a distribution (parametrized by $\boldsymbol{\theta}$) on the search space $\mathcal{F}$ in (4). As $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$, we use the Dirichlet distribution on $\boldsymbol{\alpha}$. We use the Beta distribution on $\boldsymbol{\beta}$, as each β_i lies in a bounded interval. Note that minimizing $\mathcal{J}(\boldsymbol{\theta})$ coincides with minimization of (2), i.e., $\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{x}} f(\boldsymbol{x})$ (Akimoto et al., 2019).

Let $\boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) \equiv \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$. As $\mathcal{J}(\boldsymbol{\theta})$ is smooth, it can be minimized by gradient descent, with

$$\nabla \mathcal{J}(\boldsymbol{\theta}) = \int_{\boldsymbol{x} \in \mathcal{F}} f(\boldsymbol{x}) \nabla p_{\boldsymbol{\theta}}(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{p_{\boldsymbol{\theta}}} \left[f(\boldsymbol{x}) \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) \right].$$

The expectation can be approximated by sampling K $\boldsymbol{x}_i$'s from $p_{\boldsymbol{\theta}}(\cdot)$, leading to

$$\nabla \mathcal{J}(\boldsymbol{\theta}) \simeq \frac{1}{K} \sum_{i=1}^{K} f(\boldsymbol{x}_i) \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x}_i). \tag{7}$$

The update at the mth iteration is then

$$\boldsymbol{\theta}^{m+1} = \boldsymbol{\theta}^m + \rho \boldsymbol{H}^{-1} \nabla \mathcal{J}(\boldsymbol{\theta}^m), \tag{8}$$

where ρ is the stepsize, $\boldsymbol{H} = \boldsymbol{I}$ for gradient descent and $\boldsymbol{H} = \mathbb{E}_{p_{\boldsymbol{\theta}^m}}[\boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x})^\top]$ (i.e., Fisher matrix) for natural gradient descent.

In general, natural gradient considers the geometrical structure of the underlying probability manifold, and is more efficient than simple gradient descent. However, here, the manifold is induced by a $p_{\boldsymbol{\theta}}$ that is artificially introduced for stochastic relaxation. Subsequently, the Fisher matrix is independent of the objective $\mathcal{J}$. In this paper, we instead propose to use the Newton's method and set $\boldsymbol{H} = \nabla^2 \mathcal{J}(\boldsymbol{\theta})$, which explicitly takes $\mathcal{J}$ into account. The following Proposition shows that the Hessian can be easily computed (proof is in Appendix **??**), and clearly incorporates more information than the Fisher matrix. Moreover, it can also be approximated with finite samples as in (7).

Proposition 1. $\nabla^2 \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}} \left[f(\boldsymbol{x}) \nabla^2 \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \right] + \mathbb{E}_{p_{\boldsymbol{\theta}}} \left[f(\boldsymbol{x}) \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \right]$.

The whole procedure, which will be called *Search to Exploit* (S2E), is shown in Algorithm 2.

3.4.2. Convergence Analysis

When $K = \infty$ in (7), classical analysis (Rockafellar, 1970) ensures that Algorithm 2 converges at a critical point of (6). When $\mathcal{J}$ is convex, a super-linear convergence rate is also guaranteed. However, when $K \neq \infty$, the approximation of $\nabla \mathcal{J}(\boldsymbol{\theta})$ in (7) and the analogous approximation of $\nabla^2 \mathcal{J}(\boldsymbol{\theta})$ introduce errors into the gradient. To make this explicit, we rewrite (8) as

$$\boldsymbol{\theta}^{m+1} = \boldsymbol{\theta}^m - (\boldsymbol{\Delta}^m)^{-1} (\nabla \mathcal{J}(\boldsymbol{\theta}^m) - \boldsymbol{e}^m), \tag{9}$$

Algorithm 2 *Search to Exploit* (S2E) algorithm for the minimization of the relaxed objective $\mathcal{J}$ in (6).

```
1: Initialize $\boldsymbol{\theta}^1 = \mathbf{1}$ so that $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is uniform distribution.
2: for $m = 1, \dots, M$ do
3:   for $k = 1, \dots, K$ do
4:     draw hyperparameter $\boldsymbol{x}$ from distribution $p_{\boldsymbol{\theta}^m}(\boldsymbol{x})$;
5:     using $\boldsymbol{x}$, run Algorithm 1 with $R(\cdot)$ in (4);
6:   end for
7:   use the $K$ samples in steps 3-6 to approximate $\nabla \mathcal{J}(\boldsymbol{\theta}^m)$ in (7) and $\nabla^2 \mathcal{J}(\boldsymbol{\theta}^m)$ in Proposition 1;
8:   update $\boldsymbol{\theta}^m$ by (8);
9: end for
```

where $\boldsymbol{\Delta}^m$ and $\boldsymbol{e}^m$ are the approximated Hessian and gradient errors, respectively, at the mth iteration.

We make the following Assumption on $\mathcal{J}$, which requires $\mathcal{J}$ to be smooth and bounded from below.

Assumption 2. *(i) $\mathcal{J}$ is L-Lipschitz smooth, i.e., $\|\nabla \mathcal{J}(\boldsymbol{x}) - \nabla \mathcal{J}(\boldsymbol{y})\| \leq L \|\boldsymbol{x} - \boldsymbol{y}\|$ for some positive L; (ii) $\mathcal{J}$ is coercive, i.e., $\inf_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) > -\infty$ and $\lim_{\|\boldsymbol{\theta}\| \to \infty} \mathcal{J}(\boldsymbol{\theta}) = \infty$.*

We make the following Assumption 3 on (9). Note that $\varepsilon = 0$ when $K \to \infty$. However, since $K \neq \infty$ in practice, the errors in $\boldsymbol{\Delta}^m$ and $\boldsymbol{e}^m$ do not vanish, i.e., $\lim_{m \to \infty} \left[\boldsymbol{\Delta}^m - \nabla^2 \mathcal{J}(\boldsymbol{\theta}^m) \right] \neq \mathbf{0}$ and $\lim_{m \to \infty} \boldsymbol{e}^m \neq \mathbf{0}$, Assumption 3 is more relaxed than the typical vanishing error assumptions used in classical analysis of first-order optimization algorithms (Schmidt et al., 2011; Bolte et al., 2014; Yao et al., 2017).

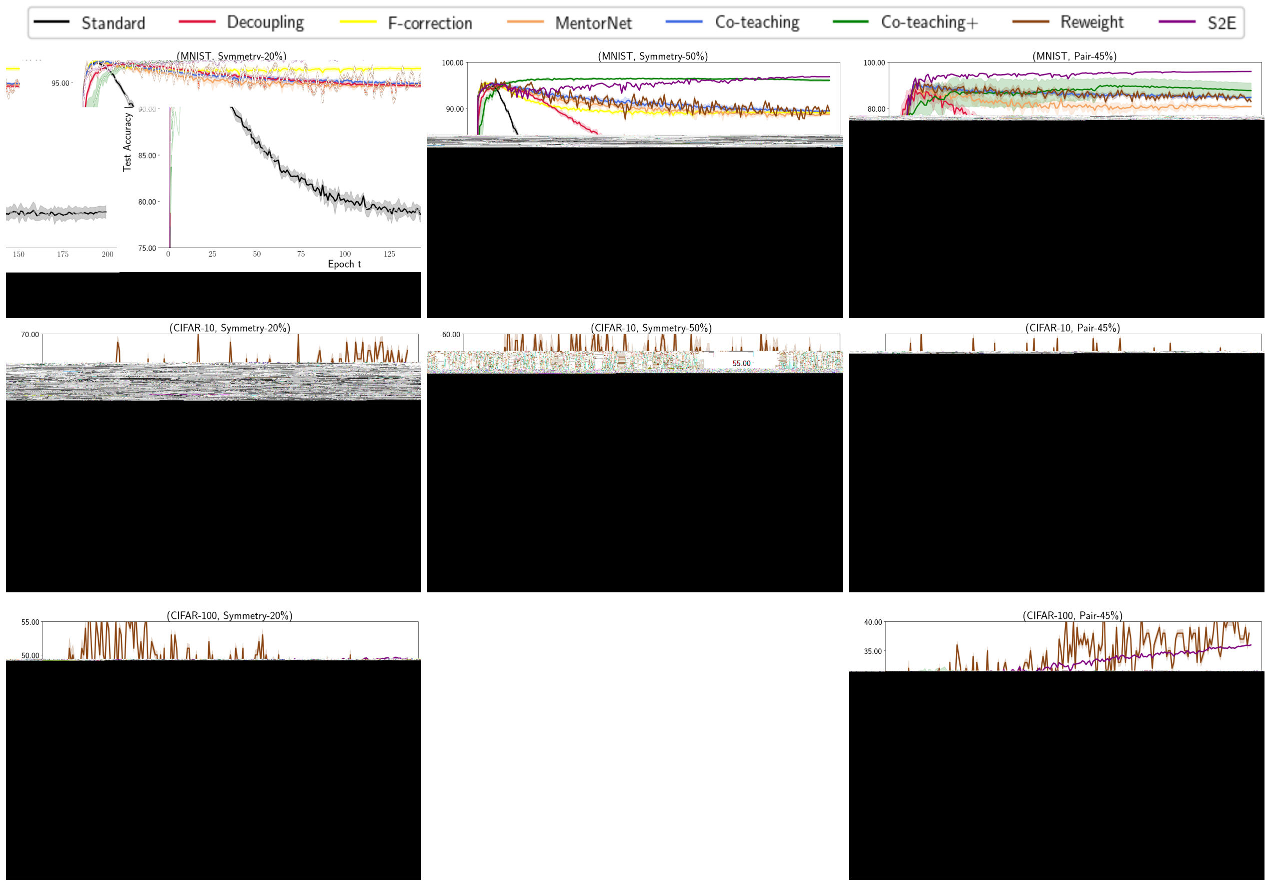
Assumption 3. *(i) $\eta \leq \sigma(\boldsymbol{\Delta}^m) \leq L$, where $\sigma(\cdot)$ denotes eigenvalues of the matrix argument, and η is a positive constant; (ii) Gradient errors are bounded: $\forall m$, $\|\boldsymbol{e}^m\| \leq \varepsilon$.*

Using Assumptions 2 and 3, the following Proposition bounds the difference in objective values at two consecutive iterations. Note that the RHS below may not be positive, and so $\mathcal{J}$ may not be non-increasing.

Proposition 2. $\mathcal{J}(\boldsymbol{\theta}^m) - \mathcal{J}(\boldsymbol{\theta}^{m+1}) \geq \frac{2 - L\eta}{2\eta} \|\boldsymbol{\gamma}^m\|^2 - \|\boldsymbol{e}^m\| \|\boldsymbol{\gamma}^m\|$*, where* $\boldsymbol{\gamma}^m = \boldsymbol{\theta}^{m+1} - \boldsymbol{\theta}^m$.

The following Theorem shows that we can obtain an approximate critical point for which the gradient norm is bounded by a constant factor of the gradient error. As $\varepsilon = 0$ when $K \to \infty$, Theorem 1 ensures that a limit point can be obtained.

Theorem 1. *Assume that $2 - L\eta + \eta^2$ and $2\eta^2 + L\eta - 2$ are non-negative. Then, (i) For every bounded sequence $\{\boldsymbol{\theta}^m\}$ generated by Algorithm 2, there exists a limit point $\boldsymbol{\theta}$ such that $\|\nabla \mathcal{J}(\boldsymbol{\theta})\| \leq c_1 \varepsilon$, where c_1 is a positive constant. (ii) If $\{\boldsymbol{\theta}^m\}$ converges, then $\lim_{m \to \infty} \|\boldsymbol{e}^m\| \leq c_2 \varepsilon$, where c_2 is a positive constant.*



(a) symmetry flipping (20%). (b) symmetry flipping (50%). (c) pair flipping (45%).

Figure 3. Testing accuracies (mean and standard deviation) on MNIST (top), CIFAR-10 (middle) and CIFAR-100 (bottom).

Proofs are in Appendix **??**, and are inspired by (Sra, 2012; Schmidt et al., 2011; Yao et al., 2017). However, they do not consider stochastic relaxation and the use of Hessian.

4. Experiments

In this section, we demonstrate the superiority of the proposed *Search to Exploit* (S2E) algorithm over the state-of-the-art in combating noisy labels. In step 5 of Algorithm 2, we use Co-teaching as Algorithm 1. Experiments are performed on standard benchmark data sets. All the codes are implemented in PyTorch 0.4.1, and run on a GTX 1080 Ti GPU.

4.1. Benchmark Comparison

In this experiment, we use three popular benchmark data sets: MNIST, CIFAR-10 and CIFAR-100. Following (Patrini et al., 2017; Han et al., 2018), we add two types of label noise: (i) symmetric flipping, which flips the label to other incorrect labels with equal probabilities; and (ii) pair flipping, which flips a pair of similar labels. We use the same network architectures as in (Yu et al., 2019). The detailed experimental setup is in Appendix **??**.

4.1.1. Learning Performance

We compare the proposed *S2E* with the following state-of-the-art methods: (i) *Decoupling* (Malach & Shalev-Shwartz, 2017); (ii) *F-correction* (Patrini et al., 2017); (iii) *MentorNet* (Jiang et al., 2018); (iv) *Co-teaching* (Han et al., 2018); (v) *Co-teaching+* (Yu et al., 2019); and (vi) *Reweight* (Ren et al., 2018). As a simple baseline, we also compare with a standard deep network (denoted *Standard*) that trains directly on the full noisy data set. All experiments are repeated five times, and we report the averaged results.

As in (Patrini et al., 2017; Han et al., 2018), Figure 3 shows convergence of the testing accuracies. As can be seen, *S2E* significantly outperforms the other methods and is much more stable.

Acknowledgment

This work is performed when Hansi Yang was an intern in 4Paradigm Inc supervised by Quanming Yao. Dr. Bo Han was partially supported by the Early Career Scheme (ECS) through the Research Grants Council of Hong Kong under Grant No.22200720, HKBU Tier-1 Start-up Grant and HKBU CSD Start-up Grant.

References

Akimoto, Y., Shirakawa, S., Yoshinari, N., Uchida, K., Saito, S., and Nishida, K. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*, pp. 171–180, 2019.

Amari, S. Natural gradient works efficiently in learning. *NeuComp*, 10(2):251–276, 1998.

Arpit, D., Jastrzkbski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M., Maharaj, T., Fischer, A., Courville, A., and Bengio, Y. A closer look at memorization in deep networks. In *ICML*, pp. 233–242, 2017.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

Bengio, Y. Gradient-based optimization of hyperparameters. *NeuComp*, 12(8):1889–1900, 2000.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *JMLR*, 13(Feb):281–305, 2012.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *NIPS*, pp. 2546–2554, 2011.

Bolte, J., Sabach, S., and Teboulle, M. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *MathProg*, 146(1-2):459–494, 2014.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *ICCS*, pp. 177–186, 2010.

Cheng, J., Liu, T., Ramamohanarao, K., and Tao, D. Learning with bounded instance-and label-dependent label noise. *ICML*, 2020.

Colson, B., Marcotte, P., and Savard, G. An overview of bilevel optimization. *Ann. Oper. Res.*, 153(1):235–256, 2007.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In *NIPS*, pp. 2962–2970, 2015.

Geman, S. and Geman, D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *TPAMI*, (6):721–741, 1984.

Ghosh, A., Kumar, H., and Sastry, P. Robust loss functions under label noise for deep neural networks. In *AAAI*, pp. 1919–1925, 2017.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT, 2016.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Hinton, G., Srivastava, N., and Swersky, K. An overview of mini-batch gradient descent. Technical report, Neural networks for machine learning: Lecture 6, 2012.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, pp. 4700–4708, 2017.

Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.). *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.

Jiang, L., Zhou, Z., Leung, T., Li, J., and Li, F.-F. MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *ICML*, pp. 2309–2318, 2018.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2014.

Laine, S. and Aila, T. Temporal ensembling for semi-supervised learning. In *ICLR*, 2017.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18(1):6765–6816, 2017.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *ICLR*, 2019.

Liu, T. and Tao, D. Classification with noisy labels by importance reweighting. *TPAMI*, 38(3):447–461, 2015.

Malach, E. and Shalev-Shwartz, S. Decoupling "when to update" from "how to update". In *NIPS*, pp. 960–970, 2017.

Miyato, T., Maeda, S., Koyama, M., and Ishii, S. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. In *ICLR*, 2016.

Patrini, G., Rozza, A., Menon, A., Nock, R., and Qu, L. Making deep neural networks robust to label noise: A loss correction approach. In *CVPR*, pp. 2233–2241, 2017.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameter sharing. In *ICML*, pp. 4092–4101, 2018.

Reed, S., Lee, H., Anguelov, D., Szegedy, C., Erhan, D., and Rabinovich, A. Training deep neural networks on noisy labels with bootstrapping. In *ICLR Workshop*, 2015.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. Learning to reweight examples for robust deep learning. In *ICML*, pp. 4331–4340, 2018.

Rockafellar, R. T. *Convex Analysis*. Princeton University Press, 1970.

Schmidt, M., Roux, N. L., and Bach, F. R. Convergence rates of inexact proximal-gradient methods for convex optimization. In *NIPS*, pp. 1458–1466, 2011.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. In *ICLR*, 2020.

Snoek, J., Larochelle, H., and Adams, R. Practical Bayesian optimization of machine learning algorithms. In *NIPS*, pp. 2951–2959, 2012.

Sra, S. Scalable nonconvex inexact proximal splitting. In *NIPS*, pp. 530–538, 2012.

Sukhbaatar, S., Bruna, J., Paluri, M., Bourdev, L., and Fergus, R. Training convolutional networks with noisy labels. In *ICLR Workshop*, 2015.

Tarvainen, A. and Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NIPS*, 2017.

Wang, X., Wang, S., Wang, J., Shi, H., and Mei, T. Co-Mining: Deep face recognition with noisy labels. In *ICCV*, pp. 9358–9367, 2019.

Xie, L. and Yuille, A. Genetic CNN. In *ICCV*, pp. 1388–1397, 2017.

Yao, Q. and Wang, M. Taking human out of learning applications: A survey on automated machine learning. Technical report, Arxiv: 1810.13306, 2018.

Yao, Q., Kwok, J., Gao, F., Chen, W., and Liu, T.-Y. Efficient inexact proximal gradient algorithm for nonconvex problems. In *IJCAI*, 2017.

Yao, Q., Xu, J., Tu, W.-W., and Zhu, Z. Efficient neural architecture search via proximal iterations. In *AAAI*, 2020.

Yu, X., Han, B., Yao, J., Niu, G., Tsang, I., and Sugiyama, M. How does disagreement help generalization against label corruption? In *ICML*, pp. 7164–7173, 2019.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *ICLR*, 2016.

Zhang, H., Yao, Q., Yang, M., Xu, Y., and Bai, X. Efficient backbone search for scene text recognition. In *ECCV*, 2020.

Zoph, B. and Le, Q. Neural architecture search with reinforcement learning. In *ICLR*, 2017.