SIGUA: Forgetting May Make Learning with Noisy Labels More Robust

Bo Han\textsuperscript{1,2}, Gang Niu\textsuperscript{2}, Xingrui Yu\textsuperscript{3}, Quanming Yao\textsuperscript{4}, Miao Xu\textsuperscript{2,5}, Ivor W. Tsang\textsuperscript{3}, Masashi Sugiyama\textsuperscript{2,6}

Abstract

Given data with noisy labels, over-parameterized deep networks can gradually memorize the data, and fit everything in the end. Although equipped with corrections for noisy labels, many learning methods in this area still suffer overfitting due to undesired memorization. In this paper, to relieve this issue, we propose stochastic integrated gradient underweighted ascent (SIGUA): in a mini-batch, we adopt gradient descent on good data as usual, and learning-rate-reduced gradient ascent on bad data; the proposal is a versatile approach where data goodness or badness is w.r.t. desired or undesired memorization given a base learning method. Technically, SIGUA pulls optimization back for generalization when their goals conflict with each other; philosophically, SIGUA shows forgetting undesired memorization can reinforce desired memorization. Experiments demonstrate that SIGUA successfully robustifies two typical base learning methods, so that their performance is often significantly improved.

1. Introduction

Data labeling may be heavily noisy in practice where label generation/corruption processes are usually agnostic (Xiao et al., 2015; Jiang et al., 2018; Wang et al., 2019; Welinder et al., 2010). As a result, learning with noisy labels seems inevitable. On the other hand, more complex data requires more expressive power, and then using over-parameterized deep networks as our models seems also inevitable (Goodfellow et al., 2016). This combination of noisy labels and deep networks is very pessimistic, since deep networks are able to fit anything given for training even if the labels are completely random (Zhang et al., 2017). Unfortunately, it is non-trivial to apply general-purpose regularization such as weight decay (Krogh & Hertz, 1991) and dropout (Srivastava et al., 2014) for controlling model complexities of deep networks. General-purpose regularization would hurt the capability of memorizing not only noisy labels but also complex data, which is never our desideratum.

Fortunately, even though deep networks can fit everything in the end, they learn patterns first (Arpit et al., 2017): this suggests deep networks can gradually memorize the data, moving from regular data to irregular data such as outliers and mislabeled data. As a consequence, the memorization events during training may be divided into two categories: desired memorization that helps generalization, and undesired memorization that hurts generalization. Our desideratum is to keep the former and avoid the latter, which may save generalization until the end of training hopefully.

However, it is hard to avoid undesired memorization from the beginning of training, since two categories can be relative to each other and cannot be distinguished without sufficient training. Consider sample selection, a correction for noisy labels where small-loss data are regarded as correct, and deep networks are trained only on selected data (Jiang et al., 2018; Han et al., 2018b; Yu et al., 2019). The losses could be enough informative after enough epochs, but then many mislabeled data have already been memorized many times. Consider backward correction, where the surrogate loss is corrected according to the label corruption process, and deep networks are trained based on this corrected loss (Natarajan et al., 2013; Patrini et al., 2017). The corrected loss is not necessarily a non-negative loss, and it might go fairly negative on a training data, which signifies this data being memorized too much. Thus, these learning methods equipped with corrections for noisy labels still suffer from undesired memorization which in turn leads to overfitting.

To relieve this issue of overfitting due to undesired memorization, we propose stochastic integrated gradient underweighted ascent (SIGUA). Specifically, SIGUA belongs to stochastic optimization and can be integrated into stochastic gradient descent (SGD) or its variants (e.g., Robbins & Monro, 1951; Kingma & Ba, 2015). SIGUA works in each mini-batch: it implements SGD on good data as usual, and if there are any bad data, it implements stochastic gradient ascent (SGA) on bad data with a reduced learning rate. It is a versatile approach, where data goodness or badness is
w.r.t. desired or undesired memorization arose in the base learning method. For instance, the good-data condition for sample selection is that the loss of the deep network being trained on a data is relatively small within the mini-batch; that for backward correction is the corrected loss on a data is still positive. We can see that a good-data condition can select desired memorization, and then a bad-data condition should be designed accordingly, so that it can select undesired memorization to be relieved by SGA.

SIGUA can be justified as follows. In machine learning, it is known that optimization shares the goal with generalization in the beginning of training, and we suffer underfitting if optimization is not well done. However, when optimization is well done, its goal will diverge from the goal of generalization, and we suffer overfitting if optimization is too much done. That is why we add regularizations (Goodfellow et al., 2016), including but not limited to the powerful early stopping (Morgan & Bourlard, 1990). Nevertheless, early stopping might not be a good choice because $X$ does not necessarily belong to a vector space and $x$ is not necessarily a vector.

Let $f : \mathcal{X} \to \mathbb{R}^k$ be the classifier to be trained, specifically, the score function. Let $\ell : \mathbb{R}^k \times \mathcal{Y} \to \mathbb{R}_+$ be the surrogate loss function for $k$-class classification, e.g., softmax cross-entropy loss. The classification risk of $f$ is defined as

$$R(f) = \mathbb{E}_{p(x,y)}[\ell(f(x),y)],$$

where $\mathbb{E}_{p(x,y)}$ denotes the expectation over $p(x,y)$. If it is supervised learning where $S$ is drawn from $p(x,y)$, we can approximate Eq. (1) by

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i),y_i),$$

which is the empirical risk and the objective of supervised classification before adding regularizations (Vapnik, 1998; Goodfellow et al., 2016). The empirical risk in Eq. (2) is an unbiased estimator of the risk in Eq. (1), and hence the minimizer of $\hat{R}(f)$ converges to the minimizer of $R(f)$ as $n$ goes to infinity (Vapnik, 1998).

However, in learning with noisy labels, we cannot replace the risk with the empirical risk as $S$ is actually drawn from $p(x,\tilde{y})$. We need some correction for noisy labels in order to approximately minimize the risk $R(f)$. Besides certain general-purpose regularizations that might work here such as virtual adversarial training (Miyato et al., 2019), there are three mainstreams—sample selection (e.g., Han et al., 2018b), label correction (e.g., Ma et al., 2018), as well as loss correction (e.g., Patrini et al., 2017):

- the first approach tries to select data with correct labels, while the second approach tries to recover correct labels for all data, so that both of them push the distribution of selected/corrected data towards $p(x,y)$;
- other than data manipulation, the third approach manipulates the loss, so that minimizing the expectation of the original loss over $p(x,y)$ can be rewritten into minimizing the expectation of the corrected loss over $p(x,\tilde{y})$.

For now, let us omit the technical details, and assume that we have a base learning method that is implemented as an algorithm with a forward pass and a backward pass given a mini-batch. The forward pass returns loss values for data in this mini-batch by feeding the data through $f$, and then the backward pass returns the gradient of the average loss by propagating the average loss through $f$. With this algorithmic abstraction, we can present SIGUA at a high level.

2. A Prototype of SIGUA

Let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output domains. Consider a $k$-class classification problem, $\mathcal{Y} = \{1, \ldots, k\}$. Let $(x, y)$ be the random variable pair of interest, and $p(x,y)$ be the underlying joint density from which test data will be sampled. In learning with noisy labels, the training data are all sampled from a corrupted joint density $p(x, \tilde{y})$ rather than $p(x,y)$, where $\tilde{y}$ denotes the random variable of the noisy label, $p(x)$ remains the same and $p(y | x)$ is corrupted into $p(\tilde{y} | x)$ (cf. Natarajan et al., 2013; Patrini et al., 2017):

$$S = \{(x_i, \tilde{y}_i)\}_{i=1}^{n} i.i.d. p(x, \tilde{y}) = p(\tilde{y} | x)p(x),$$

where $n$ denotes the sample size or the number of training data. We do not use bold $x$ because $X$ does not necessarily belong to a vector space and $x$ is not necessarily a vector.

$\hat{R}(f)$ is known that optimization shares the goal with generalization in the beginning of training, and we suffer underfitting if optimization is not well done. However, when optimization is well done, its goal will diverge from the goal of generalization, and we suffer overfitting if optimization is too much done. That is why we add regularizations (Goodfellow et al., 2016), including but not limited to the powerful early stopping (Morgan & Bourlard, 1990). Nevertheless, early stopping might not be a good choice because $X$ does not necessarily belong to a vector space and $x$ is not necessarily a vector.
After the for loop, the accumulated loss \( \ell_b \) is divide by the batch size \( n_b \), in Line 10 to make the average loss. Finally, the backward pass of \( \mathcal{B} \) is called in Line 11, and the optimizer \( \mathcal{O} \) comes to update the current model \( f_\theta \) in Line 12.

In order to integrate gradient ascent within an optimizer \( \mathcal{O} \) carrying out gradient descent, a loss accumulator suffices, and it is more efficient than a gradient accumulator. There is no difference between negating losses and negating gradients, while \( \gamma \) has the same effects in reducing losses and reducing the learning rate inside \( \mathcal{O} \). If using deep learning framework based on dynamic computational graph (Tokui et al., 2015) such as PyTorch and TensorFlow eager execution, we can modify losses in-place instead of accumulate them. In practice, we can also get rid of the for loop using a computationally more efficient implementation of Algorithm 1. Suppose the forward pass of \( \mathcal{B} \) returns \( \ell_b \in \mathbb{R}^{n_b} \), i.e., a vector but not a set of loss values, and \( \mathcal{C}_{\text{good}} \) and \( \mathcal{C}_{\text{bad}} \) directly map \( \mathcal{S}_b \) to \( \{0, 1\}^{n_b} \), i.e., two vectors of good- and bad-data masks. Then, Lines 2–10 may be replaced with a single line:

\[
\ell_b \leftarrow \ell_b \mathcal{T} (\mathcal{C}_{\text{good}}(\mathcal{S}_b) - \gamma \mathcal{C}_{\text{bad}}(\mathcal{S}_b)) / n_b, \quad (3)
\]

where \( \mathcal{T} \) denotes the transpose, and \( \mathcal{C}_{\text{good}}(\mathcal{S}_b) - \gamma \mathcal{C}_{\text{bad}}(\mathcal{S}_b) \) is a vector whose entries take 1 for good data, 0 for uncertain data, and \(-\gamma\) for bad data. Eq. (3) includes everything about SIGUA, and thus we will refer to either Algorithm 1 or Eq. (3) as SIGUA, interchangeably.

Last but not least, notice that SIGUA is extremely general. SIGUA becomes standard training, if \( \mathcal{C}_{\text{good}} \) always returns true. It becomes training on good data only, if \( \mathcal{C}_{\text{bad}} \) always returns false; we name it StopGrad because SIGUA is also named after how we handle non-good data. Moreover, the hyperparameter \( \gamma \) controls the strength of gradient ascent: when \( \gamma = 0 \), it becomes StopGrad again; when \( \gamma = 1 \), the bad data would be erased by gradient full ascent instead of gradient underweighted ascent. Consequently, \( \gamma \) should be carefully tuned on validation data in practice.

Motivations of design. The idea of SIGUA is motivated in the introduction, and its specific algorithm design is motivated here. At least four questions can be raised:

Q1 What \( \mathcal{B} \) can be used and what are its \( \mathcal{C}_{\text{good}} \) and \( \mathcal{C}_{\text{bad}} \)?
Q2 What is the technical implication of gradient ascent?
Q3 Why gradient ascent is necessary for training \( f_\theta \)?
Q4 Why underweight is necessary for gradient ascent?

Among these questions, Q1 is most complicated—we will devote the entire Section 3 for answering it; the other three questions are answered below one by one.

The technical implication of gradient ascent depends on \( k \) and \( \ell \). Consider binary classification, and assume \( \ell : \mathbb{R} \times \mathcal{Y} \rightarrow \mathbb{R}_+ \) satisfies a symmetric condition (du Plessis et al., 2014; Niu et al., 2016): \( \ell(t, +1) + \ell(t, -1) = \text{Const.} \), for example ramp loss and sigmoid loss (cf. Kiryol et al., 2017). Then, we can obtain that

\[
-\nabla_\theta \ell(f_\theta(x_i), \hat{y}_i) = \nabla_\theta \ell(f_\theta(x_i), -\hat{y}_i),
\]
which indicates that ascent along such a gradient is equivalent to descent along another gradient where $\tilde{y}_i$ is flipped. For binary classification, $-\tilde{y}_i$ must be correct if $\tilde{y}_i$ is incorrect for $x_i$, which implies SIGUA is exactly same as label correction. When $\ell$ does not satisfy that symmetric condition, they become different but still conceptually similar.

That being said, for multi-class classification where $k \geq 3$, we cannot know which class is correct if $\tilde{y}_i$ is incorrect for $x_i$. Hence, we let the model forget the wrong information that $x_i$ is from class $\tilde{y}_i$. This forgetting behavior can return some capacity back to the model, and later the model may use this capacity in a better way. This forgetting behavior is the key of SIGUA and what we meant by forgetting may make learning with noisy labels more robust in the title.

Concerning why gradient ascent is necessary or why StopGrad is inadequate, we answer it with empirical evidence. We take the MNIST benchmark dataset and add 80% symmetric label noise: for each $x_i$, $Pr(\tilde{y}_i = y_i) = 0.2$ and for all $y \neq y_i$, $Pr(\tilde{y}_i = y) = 0.8/9$. Two neural networks are considered in this experiment:

- a wide network with an architecture $784$-[Lin$(10k)$-BN-ReLU]-[Lin$(100)$-BN-ReLU]-Lin$(10)$ which has $8,871k$ parameters coming from 5 trainable layers;
- a deep network with an architecture $784$-[Lin$(500)$-BN-ReLU]*5-Lin$(10)$ which has $1,405k$ parameters coming from 11 trainable layers.

Note that MNIST contains 60k training data, where $12k$ is intact and $48k$ is flipped, and thus the two neural networks are clearly over-parameterized. The base learning method $B$ is standard training that treats $S_0$ as drawn from $p(x, y)$ where $n_0$ is 1024. The optimizer $\mathcal{D}$ is the popular momentum SGD, where the learning rate and momentum are fixed to 0.01 and 0.9—there is no learning rate decay, otherwise StopGrad may be inadequate due to learning rate decay.

Since this experiment is for illustrating our motivation, we design $C_{\text{good}}$ and $C_{\text{bad}}$ using the true labels. The number of epochs is 300 and the shift is between 150 and 151: before the shift, we let $C_{\text{good}}(x_i, \tilde{y}_i) = 1$, and then StopGrad and SIGUA would fit all training data; after the shift, we let

$$C_{\text{good}}(x_i, \tilde{y}_i) = I(\tilde{y}_i = y_i), \quad C_{\text{bad}}(x_i, \tilde{y}_i) = I(\tilde{y}_i \neq y_i),$$

where $I$ denotes the indicator function and it tests a conditional expression, and then they would fit only intact data.

We let $\gamma = 0$ for StopGrad and $\gamma = 0.001$ for SIGUA, and as a result StopGrad would ignore flipped data but SIGUA would counter-fit flipped data during epochs 151–300. We repeat this random label flipping and training 5 times.

The experimental results are shown in Figure 1, where the means with standard deviations of the accuracy curves are plotted. In Figure 1, blue means training accuracy on intact data, red means that on flipped data, and purple means test accuracy on label-noise-free test data. We can see that:

- the wide and deep networks can both memorize all data, while perfect memorization occurred around epoch 120 for the wide one and epoch 60 for the deep one, namely the deep memorized faster than the wide;
- in fact the wide not only memorized slower but also forgot faster, implying that its model capacity is lower than the deep, even though it has 5 times more parameters;
- at the end of training, StopGrad made the wide and deep networks forget 5% and 0% flipped data, while their test accuracy was improved from 23% to 59% and 41%;
- SIGUA made both of them forget 99% flipped data and their test accuracy was improved from 23% to 95%.

Note that MNIST contains 10 classes, so that the accuracy on flipped data lower than 10% means that perfect memorization has already been erased. Indeed, SIGUA made the accuracy on flipped data lower than 1%, which means that the model memorized a fact that the labels of those flipped data are flipped. In other words, SIGUA achieved learning with complimentary labels on those flipped data implicitly (Ishida et al., 2017; Yu et al., 2018; Ishida et al., 2019; Feng et al., 2020; Chou et al., 2020).

Lastly, the necessity of the underweight parameter $\gamma$ is for the stability of optimization. Without underweight for gradient ascent, training may be completely destroyed, if $\mathcal{D}$ is sophisticated and uses adaptive learning rates for different parameters such as Adam (Kingma & Ba, 2015).

3. Two Realizations of SIGUA

In this section, we explain what $B$ can be used in SIGUA. We employ SIGUA to robustify self-teaching that belongs to the sample-selection approach and backward correction that belongs to the loss-correction approach. Self-teaching and backward correction are two representative and more

---

1This is because the softmax cross-entropy loss on intact data can still further approach to zero after perfect memorization.
importantly orthogonal methods in learning with noisy labels, which spotlights the great versatility of SIGUA.

**SIGUA robustifies self-teaching.** The sample-selection approach regards small-loss data as “correct”, and it trains the model \( f_0 \) only on selected small-loss data (Jiang et al., 2018; Han et al., 2018b; Yu et al., 2019). Self-teaching, or equivalently self-paced MentorNet (Jiang et al., 2018),\(^2\) is the most primitive method in this direction. It maintains a single model, selects small-loss data as useful knowledge, and teaches this knowledge to itself.

In order to select small-loss data, a parameter of the label corruption process is needed—the noise level \( \epsilon \) measuring how many labels are corrupted. Note that \( \epsilon \) is a scalar and can be easily estimated in practice (e.g., Liu & Tao, 2016; Patrini et al., 2017). Then, the rate of data to be selected is

\[
\rho(t) = 1 - \epsilon \cdot \min(t/T_k, 1),
\]

where \( t \) is the current epoch number, and \( T_k \) is a hyperparameter denoting the number of epochs for warm-up. This means we gradually select more and more data in the first \( T_k \) epochs, and then we select a fixed amount of data from epoch \( T_k \), as the losses are unreliable after \( f_0 \) is randomly initialized and they become more and more reliable during training (Han et al., 2018b). Subsequently, let us denote by

\[
\ell_i = \ell(f_0(x_i), \hat{y}_i) \text{ for } (x_i, \hat{y}_i) \in \mathcal{S}_0
\]

where \( \mathcal{S}_0 \) and define

\[
\mathcal{E}_{\text{good}}(x_i, \hat{y}_i) = \mathbb{I}\left(\sum_{j=1}^{n_i} \mathbb{I}(\ell_i > \ell_j) \leq n_i \rho(t)\right),
\]

where \( \mathbb{I}(\ell_i > \ell_j) \) tests if the loss on \( (x_i, \hat{y}_i) \) is greater than the loss on \( (x_j, \hat{y}_j) \), and then \( \sum_{j=1}^{n_i} \mathbb{I}(\ell_i > \ell_j) \) counts how many data in \( \mathcal{S}_0 \) have smaller losses than \( \ell_i \). Equation (5) is true if the count is smaller than or equal to \( n_i \rho(t) \), i.e., \( (x_i, \hat{y}_i) \) is a small-loss data in \( \mathcal{S}_0 \). Self-teaching can be realized by plugging Eq. (5) and \( \mathcal{E}_{\text{bad}}(x_i, \hat{y}_i) = 0 \) into Eq. (3).

Eq. (4) claims loss values can be enough informative after \( T_k \) epochs, at which time many mislabeled data have been memorized many times. Moreover, small-loss data are just likely to be correct but not certainly, so that incorrect sample selection will mislead the model training which will in turn mislead the selection next time. As a consequence, we should employ SIGUA to robustify self-teaching. Let \( \delta(t) \) be the rate of data to be forgotten, then \( \mathcal{E}_{\text{bad}}(x_i, \hat{y}_i) \) can be defined similarly as

\[
\mathcal{E}_{\text{bad}}(x_i, \hat{y}_i) = -\mathcal{E}_{\text{good}}(x_i, \hat{y}_i) \wedge \mathbb{I}\left(\sum_{j=1}^{n_i} \mathbb{I}(\ell_i > \ell_j) \leq n_i \rho(t) + n_i \delta(t)\right),
\]

where \( -\mathcal{E}_{\text{good}}(x_i, \hat{y}_i) \) is necessary for Eq. (3) but not Algorithm 1. We refer to this self-teaching enhanced by Eq. (6) as SIGUA\(_{SL} \), where SL stands for small loss.

It seems counter-intuitive to select middle-loss data rather than large-loss data as our bad data. In fact, large-loss data are not memorized very well—no hope to confuse \( \mathcal{E}_{\text{good}} \) in Eq. (5), and then no need to be selected by \( \mathcal{E}_{\text{bad}} \) in Eq. (6). On the other hand, similarly to large-loss data, middle-loss data might be mislabeled with high probability, while they are memorized relatively well. To this end, we would like the model to slightly forget these middle-loss data, and let \( \mathcal{E}_{\text{good}} \) be less confused by them. This motivates the design of the bad-data condition \( \mathcal{E}_{\text{bad}} \) in Eq. (6).

**SIGUA robustifies backward correction.** On the other hand, the loss-correction approach creates a corrected loss from \( \ell \) and then trains the model \( f_0 \) based on the corrected loss (Patrini et al., 2017). Backward correction, for binary classification (Natarajan et al., 2013) or multi-class classification (Patrini et al., 2017), is the most primitive method in this direction. It builds \( \ell^* : \mathbb{R}^k \times \mathcal{Y} \rightarrow \mathbb{R} \) to reverse the label corruption process, and minimizes

\[
\hat{R}^-(f) = \frac{1}{n} \sum_{i=1}^{n} \ell^-(f(x_i), \hat{y}_i),
\]

which is the corrected empirical risk.

In order to reverse the label corruption process, a model of it is needed. Note that \( p(\hat{y} | x) = \sum_y p(\hat{y} | x, y)p(y | x) \). A model for \( p(\hat{y} | x, y) \) is called instance-dependent noise (cf. Menon et al., 2018; Cheng et al., 2020; Berthon et al., 2020), which is unfortunately unidentifiable without some extra assumption/information. Thus, a common practice is to assume that \( p(\hat{y} | x, y) = p(\hat{y} | y) \), i.e., the corruption is instance-independent and class-conditional. This model is called class-conditional noise (CCN). Let \( T \in \mathbb{R}_+^{k \times k} \) be a transition matrix such that \( [T]_{i,j} = p(\hat{y} = j | y = i) \). This \( T \) is much more difficult to estimate than \( \epsilon \), which is a hot topic in learning with noisy labels and there are still many methods (Liu & Tao, 2016; Patrini et al., 2017; Han et al., 2018a; Hendrycks et al., 2018; Xia et al., 2019; Yao et al., 2020; Xia et al., 2020). Using \( T \), \( \ell^* \) is defined as

\[
\ell^-(f(x), \hat{y}) = [T^{-1} \ell_{p(f(x))}\hat{y}],
\]

where \( \ell_{p(f(x))}\hat{y} = (\ell(f(x), 1), \ldots, \ell(f(x), k)) \in \mathbb{R}_+^k \), and it holds that (Patrini et al., 2017, Theorem 1)

\[
\mathbb{E}_{\mathcal{S}}[\hat{R}^-(f)] = \mathbb{E}_{p(x, \hat{y})}[^\ell^-(f(x), \hat{y})] = R(f),
\]

i.e., \( \hat{R}^-(f) \) is an unbiased estimator of \( R(f) \) or backward correction is risk-consistent.

Nonetheless, we should be careful of the above theoretical guarantee. Eq. (9) is about the asymptotic case rather than the finite-sample case. Note that \( T^{-1} \in \mathbb{R}_+^{k \times k} \) though \( T \in \mathbb{R}_+^{k \times k} \), and then \( \ell^* \) is no longer a non-negative loss. Since we are minimizing \( \ell^-(f_0(x_i), \hat{y}_i) \) where \( (x_i, \hat{y}_i) \) is from a finite sample \( \mathcal{S} \) and \( f_0 \) is an over-parameterized model, the
loss must go negative sooner or later whenever it could go negative (Kiryo et al., 2017; Ishida et al., 2019; Lu et al., 2020). If \( \ell^r (f(x), \tilde{y}_i) \) is fairly negative, it signifies that \((x_i, \tilde{y}_i)\) has been memorized too much and it suggests that the optimizer \( \mathcal{O} \) should focus on the data whose losses are still positive. Consequently, we should employ SIGUA to robustify backward correction.

More specifically, we have two choices, as there is no specific good-data condition yet: one focuses on \( \ell^r (f(x), \tilde{y}) \), and the other focuses on \( T^{-1} \ell_{g|f}(x) \) as a whole. We adopt the latter one, since requiring \( \ell^r (f(x), \tilde{y}) \geq 0 \) may be too strict and aggressive. Taking a closer look at the derivation of backward correction, we can see that for any \( x \),

\[
p_{g|x} \equiv \frac{\hat{\mathcal{L}}_{g|f}(x)}{p(y = 1 \mid x), \ldots, p(y = k \mid x)} ,
\]

where \( p_{g|x} = (p(y = 1 \mid x), \ldots, p(y = k \mid x)) \), \( p_{g|x} \) is as \( p_{g|x} \), and \( \hat{\mathcal{L}}_{g|f}(x) \) is \( T^{-1} \ell_{g|f}(x) \). In Eq. (10), the right-hand side is always non-negative, and so should be the left-hand side. However, \( p_{g|x} \) is unknown to us, and thus we replace it with the uninformative uniform distribution. Finally, denote by \( \mathbf{1} \) the all-one vector in \( \mathbb{R}^k \), and then the good- and bad-data conditions can be defined as

\[
\mathcal{C}_{\text{good}}(x_i, \tilde{y}_i) = \mathbf{1}^T \hat{\mathcal{L}}_{g|f}(x) \geq 0 ,
\]

\[
\mathcal{C}_{\text{bad}}(x_i, \tilde{y}_i) = -\mathcal{C}_{\text{good}}(x_i, \tilde{y}_i) .
\]

We refer to this backward correction enhanced by Eq. (11) and Eq. (12) as SIGUA\(_{BC}\).

4. Experiments

We verify the effectiveness of SIGUA\(_{SL}\) and SIGUA\(_{BC}\) on noisy MNIST, CIFAR-10, CIFAR-100 and NEWS following Han et al. (2018b). Three noises are considered:

- under symmetry-20\%, \( [T]_{i,i} = 0.8 \) and \( \forall j \neq i, [T]_{i,j} = 0.2/9 \), where the intact-vs-flipped margin is 0.78;
- under symmetry-50\%, \( [T]_{i,i} = 0.5 \) and \( \forall j \neq i, [T]_{i,j} = 0.5/9 \), where the intact-vs-flipped margin is 0.44;
- under pair-45\%, \( [T]_{i,i} = 0.55 \), \( [T]_{i,i \text{ mod } 10 + 1} = 0.45 \), and other entries are 0, where the margin is 0.10.

Thus, the noises move from easy to harder until very hard. Additionally, we test SIGUA\(_{SL}\) and SIGUA\(_{BC}\) on the more challenging open-set setting (Wang et al., 2018; Lee et al., 2019) by replacing CIFAR-10 images with SVHN images while keeping the labels of those “mislabeled” data intact.

According to the learning methods being involved, the experiments can be divided into two sets. SET1 involves

- standard training with \( \mathcal{C}_{\text{good}} \equiv 1, \mathcal{C}_{\text{bad}} \equiv 0 \),
- self-teaching with \( \mathcal{C}_{\text{good}} \) as Eq. (5) and \( \mathcal{C}_{\text{bad}} \equiv 0 \),
- SIGUA\(_{SL}\) with \( \mathcal{C}_{\text{good}} \) as Eq. (5) and \( \mathcal{C}_{\text{bad}} \) as Eq. (6).

The first method is \( \mathbb{B} \) in this set in which the surrogate loss \( \ell \) is softmax cross-entropy loss. For \( \rho(t) \) in Eq. (4), \( T_k = 10 \) and \( \epsilon \) is given as its true value; \( \delta(t) \) is a constant independent of \( t \) and depends only on the dataset. SET2 involves

- backward correction (BC) with \( \mathcal{C}_{\text{good}} \equiv 1, \mathcal{C}_{\text{bad}} \equiv 0 \),
- non-negative backward correction (nnBC) with \( \mathcal{C}_{\text{good}} \) as Eq. (11) and \( \mathcal{C}_{\text{bad}} \equiv 0 \),
- SIGUA\(_{BC}\) with \( \mathcal{C}_{\text{good}} \) as Eq. (11) and \( \mathcal{C}_{\text{bad}} \) as Eq. (12).

The first method is \( \mathbb{B} \) in this set in which \( \ell \) is again softmax cross-entropy loss. \( T \) is given as its true value for constructing \( \ell^r \) in Eq. (8). Note that our experiments are proof-of-concept, and the baselines are just chosen for this purpose. In principle, SIGUA can robustify other methods (e.g., Reed et al., 2015; Goldberger & Ben-Reuven, 2017; Hendrycks et al., 2019) provided that we could distinguish desired and undesired memorization conceptually.

The six learning methods are implemented using PyTorch. In SET1, \( \mathcal{O} \) is Adam (Kingma & Ba, 2015) in its default, and the number of epochs is 200 with batch size \( n_b \) as 128; the learning rate is linearly decayed to 0 from epoch 80 to 200. We set \( \gamma = 0.01 \) for all cases, except that \( \gamma = 0.001 \) for pair-45\% of MNIST. SET2 is a bit complicated:

- for MNIST, \( \mathcal{O} \) is Adam with betas as (0.9, 0.1), and \( \mathrm{lr} \) is divided by 10 every 10 epochs;
- for CIFAR-10, \( \mathcal{O} \) is SGD with momentum as 0.9, and \( \mathrm{lr} \) is divided by 10 every 20 epochs;
- other hyperparameters have the same values as in SET1.

We simply set \( \gamma = 1.0 \) for all cases. Neural network architectures for benchmark datasets are given in Appendix A. Data augmentation is excluded from consideration (cf. Ma et al., 2018; Zhang & Sabuncu, 2018), as our experiments are proof-of-concept. Due to the limited space, the experiments on CIFAR-100 and NEWS are completely deferred to Appendix B. All the experiments are repeated five times and the mean accuracy with standard deviation is recorded for each method in SET1 or SET2.

SET1 results. Figure 2 shows the accuracy curves of the three methods in SET1 (MNIST in the top and CIFAR-10 in the bottom). We can clearly see in Figure 2 that models learned patterns first, and hence a robust learning method should be able to stop (or alleviate) the accuracy decrease. On this point, SIGUA\(_{SL}\) stopped the decrease in Standard and Self-Teach under two symmetry cases, and alleviated the decrease under pair-45\%, on MNIST. SIGUA\(_{SL}\) did a particularly good job on CIFAR-10, where it successfully made the accuracy continue to increase without a remarkable decrease, which indicates that SIGUA\(_{SL}\) is superior to early stopping. Table 1 shows the average accuracy under the open-set noise and we can see SIGUA\(_{SL}\) outperformed

---

3 With learning rate \( \mathrm{lr} \) as 0.001 and coefficients for computing running averages of gradient and its square \( \text{betas} \) as (0.9, 0.999).

4 The model on MNIST is exactly same as CIFAR-10/100—a 9-layer CNN—which is more than needed. This makes the accuracy very high in the beginning, while the overfitting is owing to not only noisy labels but also excess expressive power.
Self-Teach significantly. In summary, SIGUA consistently improved Self-Teach under the easy, harder, and very hard noises (note that the scales of y-axis are different), and the improvements were always significant.

**SET2 results.** Figure 3 shows the accuracy curves of the three methods in SET2. Surprisingly, models trained with BC still learned patterns first, even though \( \ell \) was corrected into \( \ell^\circ \). This implies there should be some other cause of overfitting (since Eq. (10) holds for any \( x \)), and indeed the cause is negative \( \ell^\circ \) on certain \((x_i, \tilde{y}_i)\). In Figure 3, nnBC was sometimes good enough but sometimes not enough to stop or alleviate the accuracy decrease, since nnBC tries to ignore rather than fix any negative loss. SIGUA\(_{BC}\) stopped the decrease in 5 cases and alleviated it in 1 case by fixing negative losses. In Table 1, we can also see that SIGUA\(_{BC}\) outperformed BC and nnBC significantly. In order to sum up, SIGUA consistently improved BC/nnBC under different noises, and the improvements were often significant.\(^5\)

**Comparing results from SET1 and SET2.** Note that \( \epsilon \) or \( T \) is given to SET1 or SET2 methods, and thus there is no error in estimating the label corruption process. We can roughly see on MNIST, the performance of SIGUA\(_{SL}\) and SIGUA\(_{BC}\) were very close under two symmetry cases, but SIGUA\(_{BC}\) was superior under pair-45%; on CIFAR-10, the performance of SIGUA\(_{BC}\) was inferior to SIGUA\(_{SL}\) under symmetry cases and again superior under pair-45%.\(^6\)

This is because the small-loss criterion is more reliable under symmetry noises than pair noises, where the reliability is determined by the intact-vs-flipped margin more than by the noise level \( \epsilon \). On the other hand, Eq. (8) for constructing \( \ell^\circ \) is equally reliable under different noises as long as the noise belongs to CCN. This explains why SET2 methods were remarkably outperformed by SET1 methods under open-set noise—this noise does not even belong to the noise belongs to CCN. This explains why SET2 methods were remarkably outperformed by SET1 methods under open-set noise—this noise does not even belong to the label noise, let alone CCN. Actually, sample selection is a bit more general than loss correction and label correction, in a sense that it serves as corrections for not only flipping \( y \) in label noise but also replacing \( x \) in open-set noise.

Last but not least, comparing SET1 and SET2, we can find that desired memorization is not a concept definitely associated with intact data, and then neither is undesired memorization definitely associated with flipped data.

**Comparison with supervised learning.** Next, we investigate the performance gap between SIGUA and the oracle supervised learning with clean labels. In supervised learning, we train the same model using the same optimizer but on label-noise-free training data, and thus its performance upper bounds the performance of any learning method, no

\(^5\) Notice that nnBC is also a method proposed in this paper.

\(^6\) The comparison is slightly unfair, since SET1 methods only knew \( \epsilon \) whereas SET2 methods fully knew \( T \).
Table 2. Average test accuracy w. std dev (in %) over the last ten epochs of supervised learning, SIGUA sl and SIGUA nc.

<table>
<thead>
<tr>
<th></th>
<th>Symmetry-20%</th>
<th>Symmetry-50%</th>
<th>Pair-45%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>99.61 (0.02)</td>
<td>99.61 (0.02)</td>
<td>99.61 (0.02)</td>
</tr>
<tr>
<td>SIGUA sl</td>
<td>98.91 (0.19)</td>
<td>98.10 (0.30)</td>
<td>89.37 (0.82)</td>
</tr>
<tr>
<td>SIGUA nc</td>
<td>99.42 (0.10)</td>
<td>97.73 (0.05)</td>
<td>99.47 (0.02)</td>
</tr>
</tbody>
</table>

Table 3. Average test accuracy w. std dev (in %) over the last ten epochs of SIGUA bc and robust-loss-based learning methods.

<table>
<thead>
<tr>
<th></th>
<th>Symmetry-20%</th>
<th>Symmetry-50%</th>
<th>Pair-45%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGUA bc</td>
<td>99.42 (0.10)</td>
<td>97.73 (0.05)</td>
<td>99.47 (0.02)</td>
</tr>
<tr>
<td>Huber</td>
<td>93.61 (0.25)</td>
<td>65.38 (0.33)</td>
<td>56.48 (0.67)</td>
</tr>
<tr>
<td>Log-sum</td>
<td>94.35 (0.12)</td>
<td>67.46 (0.40)</td>
<td>57.38 (0.33)</td>
</tr>
</tbody>
</table>

Comparison with robust losses. In the end, \( \ell^\epsilon \) is compared with Huber loss and log-sum loss for robust regression (Candes et al., 2008). To make use of them, let \( \mathbf{y} \) and \( \mathbf{y}' \) be the one-hot vectors of \( y \) and \( \tilde{y} \), and \( \ell(f(x), \mathbf{y}') \) be the sum of Huber/log-sum losses from \( k \) dimensions (\( f \) itself is vector-valued). The assumed noise model is additive by these losses: \( \mathbf{y} \) and \( \mathbf{y}' \) are continuous and their difference \( \epsilon \) is sampled from \((1 - \epsilon)N(0, \sigma I) + \epsilon N(0, \sigma' I)\) where \( N \) denotes the multivariate normal distribution, \( 0 \) denotes the all-zero vector in \( \mathbb{R}^k \), and \( \sigma \ll \sigma' \) for covariance matrices. The results are shown in Table 3, where two robust losses notably failed under symmetry-50% and pair-45%. It is as expected, since they are specially designed to be robust to outliers or similar additive noises, whereas \( \ell^\epsilon \) is specially designed to be robust to flipping noises.

5. Conclusions

We presented in this paper a versatile approach to learning with noisy labels called SIGUA. By carefully distinguishing desired and undesired memorization, SIGUA was successful in robustifying two typical base learning methods: self-teaching from sample selection, and backward correction from loss correction. We demonstrated through experiments that two enhanced methods can result in significant improvements. In general, SIGUA can be applied to other methods or even other problem settings like learning from similarity-unlabeled data (e.g., Bao et al., 2018) and pairwise comparison data (e.g., Xu et al., 2019).

SIGUA exhibits pulling optimization back for generalization in learning with noisy labels. There should be a trade-off between them implemented as an equilibrium between gradient descent and ascent, and then the model will travel between (uncountably infinite) reasonably good solutions, where the goodness is in the sense of optimization instead of generalization. See Ishida et al. (2020) for a dedicated study of this phenomenon in supervised learning.
Acknowledgments

BH was 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, HKBU CSD Start-up Grant and a RIKEN BAIHO Award. IWT was supported by Australian Research Council under Grants DP180100106 and DP200101328. MS was supported by the International Research Center for Neurointelligence (WPI-IRCN) at The University of Tokyo Institutes for Advanced Study.

References


Table 4. CNN on MNIST and CIFAR-10/100.

<table>
<thead>
<tr>
<th>Input</th>
<th>28×28 Gray Image</th>
<th>32×32 Color Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>Conv(3×3, 128)-BN-LReLU</td>
<td>Conv(3×3, 128)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 128)-BN-LReLU</td>
<td>Conv(3×3, 128)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>MaxPool(2×2, stride = 2)</td>
<td>MaxPool(2×2, stride = 2)</td>
</tr>
<tr>
<td></td>
<td>Dropout(p = 0.25)</td>
<td>Dropout(p = 0.25)</td>
</tr>
<tr>
<td>Block 2</td>
<td>Conv(3×3, 256)-BN-LReLU</td>
<td>Conv(3×3, 256)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 256)-BN-LReLU</td>
<td>Conv(3×3, 256)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>MaxPool(2×2, stride = 2)</td>
<td>MaxPool(2×2, stride = 2)</td>
</tr>
<tr>
<td></td>
<td>Dropout(p = 0.25)</td>
<td>Dropout(p = 0.25)</td>
</tr>
<tr>
<td>Block 3</td>
<td>Conv(3×3, 512)-BN-LReLU</td>
<td>Conv(3×3, 512)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 256)-BN-LReLU</td>
<td>Conv(3×3, 256)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 128)-BN-LReLU</td>
<td>Conv(3×3, 128)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>GlobalAvgPool(128)</td>
<td>GlobalAvgPool(128)</td>
</tr>
<tr>
<td>Score</td>
<td>Linear(128, 10 or 100)</td>
<td>Linear(128, 10 or 100)</td>
</tr>
</tbody>
</table>

Table 5. CNN for open-set noise.

<table>
<thead>
<tr>
<th>Input</th>
<th>32×32 Color Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>Conv(3×3, 64)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 64)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>MaxPool(2×2)</td>
</tr>
<tr>
<td>Block 2</td>
<td>Conv(3×3, 128)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 128)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>MaxPool(2×2)</td>
</tr>
<tr>
<td>Block 3</td>
<td>Conv(3×3, 196)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>Conv(3×3, 196)-BN-LReLU</td>
</tr>
<tr>
<td></td>
<td>MaxPool(2×2)</td>
</tr>
<tr>
<td>Score</td>
<td>Linear(256, 10)</td>
</tr>
</tbody>
</table>

Table 6. 1D CNN on NEWS.

<table>
<thead>
<tr>
<th>Input</th>
<th>Sequence of Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embed</td>
<td>300D GloVe</td>
</tr>
<tr>
<td>Block 1</td>
<td>Conv(3, 100)-ReLU</td>
</tr>
<tr>
<td>Score</td>
<td>Linear(100, 7)</td>
</tr>
</tbody>
</table>

Table 7. MLP on NEWS.

<table>
<thead>
<tr>
<th>Input</th>
<th>Sequence of Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embed</td>
<td>300D GloVe</td>
</tr>
<tr>
<td>Block 1</td>
<td>Linear(300, 300)-Softsign</td>
</tr>
<tr>
<td>Score</td>
<td>Linear(300, 2)</td>
</tr>
</tbody>
</table>

A. Neural Network Architectures for Benchmark Datasets

Table 4 describes the 9-layer CNN (Laine & Aila, 2017; Miyato et al., 2019) used on MNIST and CIFAR-10/100. In fact, it has 9 convolutional layers but 19 trainable layers. Table 5 describes the CNN used on CIFAR-10 under open-set noise. It has 6 convolutional layers but 13 trainable layers. Furthermore, the 1D CNN on NEWS for SET1 methods and MLP on NEWS for SET2 methods are given in Tables 6 and 7 respectively. Here, BN stands for batch normalization layers (Ioffe & Szegedy, 2015); LReLU stands for Leaky ReLU (Xu et al., 2015), a special case of Parametric ReLU (He et al., 2015); GloVe stands for global vectors for word representation (Pennington et al., 2014); Softsign is an activation function which looks very similar to Tanh (Glorot & Bengio, 2010).

Note that the 9-layer CNN is a standard and common practice in weakly supervised learning, including but not limited to semi-supervised learning (e.g., Laine & Aila, 2017; Miyato et al., 2019) and noisy-label learning (e.g., Han et al., 2018b). Actually, this CNN was not born in those areas—it came from Salimans et al. (2016) where it served as the discriminator of GANs on CIFAR-10. We decided to use this CNN, because then the experimental results are directly comparable with previous papers in the same area, and it would be crystal clear where the proposed methods stand in the area.

That being said, SIGUA can definitely achieve better performance if given better models. In order to demonstrate this, let us take ResNet-18 (He et al., 2016), the smallest ResNet in torch vision model zoo for example. The experimental results are shown in Table 8. For each noise, we selected the better one among SIGUA_{SL} and SIGUA_{BC}, and replaced the model with ResNet-18. We can see from Table 8 that the improvements were very great by training bigger ResNet-18.

Table 8. Average test accuracy (in %) over the last ten epochs on CIFAR-10.

<table>
<thead>
<tr>
<th></th>
<th>SIGUA_{SL} under symmetry-20%</th>
<th>SIGUA_{SL} under symmetry-50%</th>
<th>SIGUA_{BC} under pair-45%</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-layer CNN</td>
<td>84.05</td>
<td>77.12</td>
<td>81.82</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>89.41</td>
<td>81.96</td>
<td>89.56</td>
</tr>
<tr>
<td>Absolute Acc Increase</td>
<td>5.36</td>
<td>4.84</td>
<td>7.74</td>
</tr>
<tr>
<td>Relative Err Reduction</td>
<td>33.61</td>
<td>21.15</td>
<td>42.57</td>
</tr>
</tbody>
</table>

B. More Experiments

Due to the limited space, the experiments on CIFAR-100 and NEWS are moved here. The setup of CIFAR-100 is similar to CIFAR-10, but the momentum is 0.5 and lr is divided by 10 every 30 epochs for SET2 methods. The setup of NEWS is similar to other three datasets, except O is AdaGrad (Duchi et al., 2011) that automatically decays lr every mini-batch.

Figure 4 shows the accuracy curves of the three methods in SET1 (CIFAR-100 in the top and NEWS in the bottom). The trend in Figure 4 is similar to the trend in Figure 2 that SIGUA_{SL} either stopped or alleviated the decrease in Standard and
Self-Teach. Especially on CIFAR-100, after a remarkable decrease in the first half, the accuracy in the second half started to increase once more, and it eventually surpassed the best accuracy that can be obtained by early stopping. If we plot the test error rather than the test accuracy, this phenomenon is exactly an epoch-wise double descent (Nakkiran et al., 2020). Figure 5 shows the accuracy curves of the three methods in SET2 where SIGUA_BC still stopped or alleviated the decrease in BC and/or nnBC. The reason why BC suffered more under pair-45% may be explained by the maximum and minimum elements of \( T^{-1} \): when \( k = 10 \), they are 2.101 and -1.719 under pair-45% but 2.125 and -0.125 under symmetry-50%. It is interesting on CIFAR-10, nnBC and SIGUA_BC under pair-45% outperformed themselves under symmetry-20%, which provides an evidence that the issue of negative losses can be fixed at least empirically.