ProSelfLC: Progressive Self Label Correction for Training Robust Deep Neural Networks


Published in:

Document Version:
Peer reviewed version
ProSelfLC: Progressive Self Label Correction for Training Robust Deep Neural Networks

Xinshao Wang\textsuperscript{1,2}, Yang Hua\textsuperscript{3}, Elyor Kodirov\textsuperscript{1}, David A. Clifton\textsuperscript{2,3}; Neil M. Robertson\textsuperscript{1,3}

\textsuperscript{1}Zenith Ai, UK
\textsuperscript{2}Institute of Biomedical Engineering, University of Oxford, UK
\textsuperscript{3}Institute of Electronics, Communications and Information Technology, Queen’s University Belfast, UK
\{xinshao, elyor\}@zenithai.co.uk, \{y.hua, n.robertson\}@qub.ac.uk, \{xinshao.wang, david.clifton\}@eng.ox.ac.uk

Abstract

To train robust deep neural networks (DNNs), we systematically study several target modification approaches, which include output regularisation, self and non-self label correction (LC). Two key issues are discovered: (1) Self LC is the most appealing as it exploits its own knowledge and requires no extra models. However, how to automatically decide the trust degree of a learner as training goes is not well answered in the literature? (2) Some methods penalise while the others reward low-entropy predictions, prompting us to ask which one is better?

To resolve the first issue, taking two well-accepted propositions–deep neural networks learn meaningful patterns before fitting noise [3] and minimum entropy regularisation principle [10]–we propose a novel end-to-end method named ProSelfLC, which is designed according to learning time and entropy. Specifically, given a data point, we progressively increase trust in its predicted label distribution versus its annotated one if a model has been trained for enough time and the prediction is of low entropy (high confidence). For the second issue, according to ProSelfLC, we empirically prove that it is better to redefine a meaningful low-entropy status and optimise the learner toward it. This serves as a defence of entropy minimisation.

We demonstrate the effectiveness of ProSelfLC through extensive experiments in both clean and noisy settings. The source code is available at https://github.com/XinshaoAmosWang/ProSelfLC-CVPR2021.

1. Introduction

There exist many target (label) modification approaches. They can be roughly divided into two groups: (1) Output regularisation (OR), which is proposed to penalise over-confident predictions for regularising deep neural networks. It includes label smoothing (LS) [42, 29] and confidence penalty (CP) [33]; (2) Label correction (LC). On the one hand, LC regularises neural networks by adding the similarity structure information over training classes into one-hot label distributions so that the learning targets become structured and soft. On the other hand, it can correct the semantic classes of noisy label distributions. LC can be further divided into two subgroups: Non-self LC and Self LC. The former requires extra learners, while the latter relies on the model itself. A typical approach of Non-self LC is knowledge distillation (KD), which exploits the predictions of other model(s), usually termed teacher(s) [17]. Self LC methods include Pseudo-Label [23], bootstrapping (Boot-soft and Boot-hard) [35], Joint Optimisation (Jointsoft and Joint-hard) [43], and Tf-KD_{self} [55]. According to an overview in Figure 1 (detailed derivation is in Section 3 and Table 1), in label modification, the output target of a data point is defined by combining a one-hot label distribution and its corresponding prediction or a predefined label distribution.

Firstly, we present the drawbacks of existing approaches: (1) OR methods naively penalise confident outputs without leveraging easily accessible knowledge from other learners or itself (Figure 1a); (2) Non-self LC relies on accurate auxiliary models to generate predictions (Figure 1b). (3) Self LC is the most appealing because it exploits its own knowledge and requires no extra learners. However, there is a core question that is not well answered:

In Self LC, how much should we trust a learner to leverage its knowledge?

As shown in Figure 1b, in Self LC, for a data point, we have two labels: a predefined one-hot q and a predicted structured p. Its learning target is \((1 - \epsilon)q + \epsilon p\), i.e., a trade-off between q and p, where \(\epsilon\) defines the trust score of a learner. In existing methods, \(\epsilon\) is fixed without considering that a model’s knowledge grows as the training progresses. For example, in bootstrapping, \(\epsilon\) is fixed throughout the training process. Joint Optimisation stage-wisely trains a model. It fully trusts predicted labels and uses them to replace old ones when a stage ends, i.e., \(\epsilon = 1\). Tf-KD_{self} trains a model by two stages: \(\epsilon = 0\) in the first one while \(\epsilon\) is tuned for the second stage. Note that p is gen-
The parameter $\epsilon$ defines how much a predicted label distribution is trusted. Accordingly, we reveal their drawbacks and propose ProSelfLC as a solution. ProSelfLC can: (1) enhance the similarity structure information over training classes; (2) correct the semantic classes of noisy label distributions. ProSelfLC is the first method to trust self knowledge progressively and adaptively.

Finally, we summarise our main contributions:

- We provide a theoretical study on popular target modification methods through entropy and KL divergence [21]. Accordingly, we reveal their drawbacks and propose ProSelfLC as a solution. ProSelfLC can: (1) enhance the similarity structure information over training classes; (2) correct the semantic classes of noisy label distributions. ProSelfLC is the first method to trust self knowledge progressively and adaptively.
- Our extensive experiments: (1) defend the entropy minimisation principle; (2) demonstrate the effectiveness of ProSelfLC in both clean and noisy settings.

**2. Related Work**

*Label noise and semi-supervised learning.* We test target modification approaches in the setting of label noise because it is generic and connected with semi-supervised learning, where only a subset of training examples are annotated, leading to *missing labels*. Then the key to semi-supervised training is to reliably fill them. *When these missing labels are incorrectly filled, the challenge of semi-supervised learning changes to noisy labels.* For a further comparison, in semi-supervised learning, the annotated set is clean and reliable, because the label noise only exists in the unannotated set. While in our experimental setting, we are not given information on whether an example is trusted or not, thus being even more challenging. We summarise...
existing approaches for solving label noise: (1) Loss correction, in which we are given or we need to estimate a noise-transition matrix, which defines the distribution of noise labels [26, 8, 41, 44, 52, 12, 32, 48]. A noise-transition matrix is difficult and complex to estimate in practice; (2) Exploiting an auxiliary trusted training set to differentiate examples [45, 24, 16]. This requires extra annotation cost; (3) Co-training strategies, which train two or more learners [28, 19, 13, 54, 47, 34, 25] and exploit their ‘disagreement’ information to differentiate data points; (4) Label engineering methods [40, 23, 35, 43, 52, 25], which relate to our focus in this work. Their strategy is to annotate unlabelled samples or correct noisy labels.

**LC and knowledge distillation (KD)** [5, 17]. Mathematically, we derive that some KD methods also modify labels. We use the term label correction instead of KD for two reasons: (1) label correction is more descriptive; (2) the scope of KD is not limited to label modification. For example, multiple networks are trained for KD [7]. When two models are trained, the consistency between their predictions of a data point is promoted in [4, 58], while the distance between their feature maps is reduced in [37]. Regarding self KD, two examples of the same class are constrained to have consistent output distributions [51, 56]. In another self KD [57], the deepest classifier provides knowledge for shallower classifiers. In a recent self KD method [55], TF-KD$_{rel}$ applies two-stage training. In the second stage, a model is trained by exploiting its learned knowledge in the first stage. Our focus is to improve the end-to-end self LC. First, self KD methods [51, 56, 57] maximise the consistency of intra-class images’ predictions or the consistency of different classifiers. In our view, they do not modify labels, thus being less relevant for comparison. Second, the two-stage self KD method [55] can be an add-on (i.e., an enhancement plugin) other than a competitor. E.g., in real-world practice, the first stage can be ProSelfLC instead of CCE with early stopping. Finally, we acknowledge that exploiting ProSelfLC to improve non-self KD and stage-wise approaches is an important area for future work, e.g., a better teacher model can be trained using ProSelfLC.

### 3. Mathematical Analysis and Theory

Let $X = \{(x_i, y_i)\}_{i=1}^{N}$ represent $N$ training examples, where $(x_i, y_i)$ denotes $i$-th sample with input $x_i \in \mathbb{R}^{D}$ and label $y_i \in \{1, 2, ..., C\}$. $C$ is the number of classes. A deep neural network $z$ consists of an embedding network $f(\cdot) : \mathbb{R}^{D} \rightarrow \mathbb{R}^{K}$ and a linear classifier $g(\cdot) : \mathbb{R}^{K} \rightarrow \mathbb{R}^{C}$, i.e., $z(x) = g(f(x)) : \mathbb{R}^{D} \rightarrow \mathbb{R}^{C}$. For the brevity of analysis, we take a data point and omit its subscript so that it is denoted by $(x, y)$. The linear classifier is usually the last fully-connected layer. Its output is named logit vector $z \in \mathbb{R}^{C}$. We produce its classification probabilities $p$ by normalising the logits using a softmax function:

$$p(j \mid x) = \frac{\exp(z_j)}{\sum_{m=1}^{C} \exp(z_m)},$$

(1)

where $p(j \mid x)$ is the probability of $x$ belonging to class $j$. Its corresponding ground-truth is usually denoted by a one-hot representation $q$: $q(j \mid x) = 1$ if $j = y$, $q(j \mid x) = 0$ otherwise.

### 3.1. Semantic class and similarity structure in a label distribution

A probability vector $p \in \mathbb{R}^{C}$ can also be interpreted as an instance-to-classes similarity vector, i.e., $p(j \mid x)$ measures how much a data point $x$ is similar with (analogously, likely to be) $j$-th class. Consequently, $p$ should not be exactly one-hot, and is proposed to be corrected at training, so that it can define a more informative and structured learning target. For better clarity, we first present two definitions:

**Definition 1 (Semantic Class).** Given a target label distribution $q(x) \in \mathbb{R}^{C}$, the semantic class is defined by $\arg\max_{j} \tilde{q}(j \mid x)$, i.e., the class whose probability is the largest.

**Definition 2 (Similarity Structure).** In $\tilde{q}(x)$, $x$ has $C$ probabilities of being predicted to $C$ classes. The similarity structure of $x$ versus $C$ classes is defined by these probabilities and their differences.

### 3.2. Revisit of CCE, LS, CP and LC

**Standard CCE.** For any input $(x, y)$, the minimisation objective of standard CCE is:

$$L_{\text{CCE}}(q, p) = \mathbb{H}(q, p) = E_{q}(- \log p),$$

(2)

where $\mathbb{H}(\cdot, \cdot)$ represents the cross entropy. $E_{q}(- \log p)$ denotes the expectation of negative log-likelihood, and $q$ serves as the probability mass function.

**Label smoothing.** In LS [42, 17], we soften one-hot targets by adding a uniform distribution: $q_{\text{LS}}(i) = (1 - \epsilon)q + \epsilon u$, $u \in \mathbb{R}^{C}$, and $\forall i, u_{i} = \frac{1}{C}$. Consequently:

$$L_{\text{CCE+LS}}(q, p; \epsilon) = \mathbb{H}(q_{\text{LS}}, p) = E_{q_{\text{LS}}}(- \log p) = (1 - \epsilon)\mathbb{H}(q, p) + \epsilon \mathbb{H}(u, p).$$

(3)

**Confidence penalty.** CP [33] penalises highly confident predictions:

$$L_{\text{CCE+CP}}(q, p; \epsilon) = (1 - \epsilon)\mathbb{H}(q, p) - \epsilon \mathbb{H}(p, p).$$

(4)

**Label correction.** As illustrated in Figure 1. LC is a family of algorithms, where a one-hot label distribution is modified to a convex combination of itself and a predicted distribution:

$$q_{\text{LC}} = (1 - \epsilon)q + \epsilon p \Rightarrow L_{\text{CCE+LC}}(q, p; \epsilon) = \mathbb{H}(q_{\text{LC}}, p) = (1 - \epsilon)\mathbb{H}(q, p) - \epsilon \mathbb{H}(p, p).$$

(5)

We remark: (1) $p$ provides meaningful information about an example’s relative probabilities of being different training classes; (2) If $\epsilon$ is large, and $p$ is confident in predicting a different class, i.e., $\arg\max_{j} p(j \mid x) \neq \arg\max_{j} q(j \mid x)$, $q_{\text{LC}}$ defines a different semantic class from $q$. 

754
3.3. Theory

**Proposition 1.** LS, CP and LC modify the learning targets of standard CCE.

Proof. \( L_{\text{CCE}+\text{CP}}(q, p; e) = (1 - e)H(q, p) - eH(p, p) = E_{(1-e)q-eP}(-\log p) \). Therefore, \( \tilde{q}_{\text{CP}} = (1 - e)q - eP \). Additionally, \( \tilde{q}_{\text{LS}} = (1 - e)q + eP \).

**Proposition 2.** Some KD methods, which aim to minimise the KL divergence between predictions of a teacher and a student, belong to the family of label correction.

Proof. In general, a loss function of such methods can be defined to be \( L_{\text{KD}}(q, p_t, p) = (1 - e)H(q, p) + eH(p_t, p) \) [55]. \( KL(\cdot \| \cdot) \) denotes the KL divergence. As \( KL(p_t \| p) = H(p_t, p) - H(p_t, p_t) \), \( p_t \) is from a teacher and fixed when training a student. We can omit \( H(p_t, p_t) \):

\[
L_{\text{KD}}(q, p_t, p) = (1 - e)H(q, p) + eH(p_t, p) = E_{(1-e)q+eP}(-\log p) \quad (6)
\]

Consistent with LC in Eq (5), \( L_{\text{KD}}(q, p_t, p) \) revises a label using \( p_t \). □

**Proposition 3.** Compared with CCE, LS and CP penalise entropy minimisation while LC reward it.

**Proposition 4.** In CCE, LS and CP, a data point \( x \) has the same semantic class. In addition, \( x \) has an identical probability of belonging to other classes except for its semantic class.

The proof of propositions 3 and 4 is presented in the Appendix A. Only LC exploits informative information and has the ability to correct labels, while LS and CP only relax the hard targets. We summarise CCE, LS, CP and LC in Table 1. Constant terms are ignored for concision.

### 4. ProSelfLC: Progressive and Adaptive Label Correction

In standard CCE, a semantic class is considered while the similarity structure is ignored. It is mainly due to the difficulty of annotating the similarity structure for every data point, especially when \( C \) is large [50]. Fortunately, recent progress demonstrates that there are some effective approaches to define the similarity structure of data points without annotation: (1) In KD, an auxiliary teacher model can provide a student model the similarity structure information [17, 29]; (2) In Self LC, e.g., Boot-soft, a model helps itself by exploiting the knowledge it has learned so far. We focus on studying the end-to-end Self LC.

In Self LC, \( \epsilon \) indicates how much a predicted label distribution is trusted. In ProSelfLC, we propose to set it automatically according to learning time \( t \) and prediction entropy \( H(p) \), i.e., ProSelfLC trusts self knowledge according to training time and confidence. For any \( x \), we summarise:

\[
\begin{align*}
\text{Loss:} & \quad L(\tilde{q}_{\text{ProSelfLC}}, p; \epsilon_{\text{ProSelfLC}}) = H(\tilde{q}_{\text{ProSelfLC}}, p) = E_{\text{ProSelfLC}}(-\log p) \\
\text{Label:} & \quad \tilde{q}_{\text{ProSelfLC}} = (1 - \epsilon_{\text{ProSelfLC}})q + \epsilon_{\text{ProSelfLC}}p \\
\end{align*}
\]

\( t \) and \( \Gamma \) are the iteration counter and the number of total iterations, respectively. \( h(\eta, B) = 1/(1 + \exp(-\eta \times B)) \). Here, \( \eta = t/\Gamma - 0.5 \). \( B, \Gamma \) are task-dependent and searched on a validation set.

#### 4.1. Self trust scores

**Global trust score** \( g(t) \) denotes how much we trust a learner. It is independent of data points, thus being global. \( g(t) \) grows as \( t \) rises. \( B \) adjusts the exponentiation’s base and growth speed of \( g(t) \).

**Local trust score** \( l(p) \) indicates how much we trust an output distribution \( p \), which is data-dependent. \( l(p) \) rises as \( H(p) \) becomes lower, rewarding a confident distribution.

#### 4.2. Design reasons

Regarding \( g(t) \), in the earlier learning phase, i.e., \( t < \Gamma/2 \), \( g(t) < 0.5 \Rightarrow \epsilon_{\text{ProSelfLC}} < 0.5 \forall p \), so that the human annotations dominate and ProSelfLC only modifies the similarity structure. When a learner has not seen the training data for enough time at the earlier stage, its knowledge is less reliable and a wrong confident prediction may occur. Our design assuages the bad impact of such unexpected cases. When it comes to the later training phase, i.e., \( t > \Gamma/2 \), we have \( g(t) > 0.5 \) as it has been trained for more than half of entire iterations.
Regarding \( l(p) \), it affects the later learning phase. If \( p \) is less confident, \( l(p) \) will be smaller, then \( \epsilon_{\text{ProSelfLC}} \) will be smaller, hence we trust \( p \) less when it is of higher uncertainty. If \( p \) is highly confident, we trust its confident knowledge. Ablation study of our design is in Figure 2, where three variants of \( \epsilon \) are presented. In our experiments, note that when \( \epsilon \) is fixed, we try three values (0.125, 0.25, 0.50) and display the best instantiation, i.e., \( \epsilon = 0.50 \).

### 4.3. Case analysis

Due to the potential memorisation in the earlier phase (which is more rare), we may get unexpected confident wrong predictions for noisy labels, but their trust scores are small as \( g(t) \) is small. We conduct the case analysis of ProSelfLC in Table 2 and summarise its core tactics as follows:

1. Correct the similarity structure for every data point in all cases, thanks to exploiting the self knowledge of a learner, i.e., \( p \).

2. Revise the semantic class when \( t \) is large enough and \( p \) is confidently inconsistent. As highlighted in Table 2, when both conditions are met, we have \( \epsilon_{\text{ProSelfLC}} > 0.5 \) and \( \arg \max_j p(j|x) \neq \arg \max_j q(j|x) \), then \( p \) redefines the semantic class. For example, if \( p = [0.95, 0.01, 0.04], q = [0, 0.1], \epsilon_{\text{ProSelfLC}} = 0.8 \Rightarrow \tilde{d}_{\text{ProSelfLC}} = (1 - \epsilon_{\text{ProSelfLC}})q + \epsilon_{\text{ProSelfLC}}p = [0.76, 0.008, 0.232] \). Note that ProSelfLC also becomes robust against lengthy exposure to the training data, as demonstrated in Figures 2, 3.

### 5. Experiments

In deep learning, small differences (e.g., random accelerators like cudnn and different frameworks like Caffe [18], Tensorflow [1] and PyTorch [31]) may lead to a large gap of final performance. Therefore, to compare more properly, we re-implement CCE, LS and CP. Regarding Self LC methods, we re-implement Boot-soft [35], where \( \epsilon \) is fixed throughout training. We do not re-implement stage-wise Self LC and KD methods, e.g., Joint Optimisation and Tf-KD respectively, because time-consuming tuning is required. We fix the random seed and do not use any random accelerator. In standard and synthetic cases, we train on 80% training data (corrupted in synthetic cases) and use 20% trusted training data as a validation set to search all hyperparameters, e.g., \( \epsilon, \Gamma, B \) and settings of an optimiser. Note that \( \Gamma \) and an optimiser’s settings are searched first and then fixed for all methods. Finally, we retrain a model on the entire training data (corrupted in synthetic cases) and report its accuracy on the test data to fairly compare with prior results. In real-world label noise, the used dataset has a separate clean validation set. Here, a clean dataset is used only for validation, which is generally necessary for any method and differs from the methods [44, 45, 19, 36, 26, 16, 53, 60] which use a clean dataset to train a network’s learning parameters.

#### 5.1. Standard image classification

**Datasets and training details.** (1) CIFAR-100 [20] has 20 coarse classes, each containing 5 fine classes. There are 500 and 100 images per class in the training and testing sets, respectively. The image size is 32 × 32. We apply simple data augmentation [15], i.e., we pad 4 pixels on every side of the image, and then randomly crop it with a size of 32 × 32. Finally, this crop is horizontally flipped with a
Table 3: Test accuracy (%) in the standard setting. We report three settings of hyperparameters.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CCE (0.125)</th>
<th>LS (0.25)</th>
<th>CP (0.50)</th>
<th>Boot-soft (0.125)</th>
<th>ProSelfLC (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-100</td>
<td>69.0</td>
<td>69.9</td>
<td>69.6</td>
<td>68.4</td>
<td>69.5</td>
</tr>
<tr>
<td>ImageNet 2012</td>
<td>75.5</td>
<td>75.3</td>
<td>75.2</td>
<td>74.9</td>
<td>75.2</td>
</tr>
</tbody>
</table>

Figure 3: Comprehensive learning dynamics on CIFAR-100 with 40% asymmetric label noise. Vertical axes denote evaluation metrics. Their mean results are displayed. At training, a learner is NOT GIVEN whether a label is trusted or not. We store intermediate models and analyse them when the training ends.

5.2. Synthetic label noise

Noise generation. (1) Symmetric label noise: the original label of an image is uniformly changed to one of the other classes with a probability of \( r \); (2) Asymmetric label noise: we follow [46] to generate asymmetric label noise to fairly compare with their reported results. Within each coarse class, we randomly select two fine classes \( A \) and \( B \). Then we flip \( r \times 100\% \) labels of \( A \) to \( B \), and \( r \times 100\% \) labels of \( B \) to \( A \). We remark that the overall label noise rate is smaller than \( r \).
Table 4: Accuracy (%) on the CIFAR-100 clean test set. All compared methods use ResNet-44.

<table>
<thead>
<tr>
<th>Method</th>
<th>Asymmetric Noisy Labels</th>
<th>Symmetric Noisy Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r=0.2</td>
<td>r=0.3</td>
</tr>
<tr>
<td>Results From</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SL [46]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boot-hard</td>
<td>63.4</td>
<td>63.2</td>
</tr>
<tr>
<td>Forward</td>
<td>64.1</td>
<td>64.0</td>
</tr>
<tr>
<td>D2L</td>
<td>62.4</td>
<td>63.2</td>
</tr>
<tr>
<td>GCE</td>
<td>63.0</td>
<td>63.2</td>
</tr>
<tr>
<td>SL</td>
<td>65.6</td>
<td>65.1</td>
</tr>
<tr>
<td>Our Trained Results</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCE</td>
<td>66.6</td>
<td>63.4</td>
</tr>
<tr>
<td>LS</td>
<td>67.9</td>
<td>66.4</td>
</tr>
<tr>
<td>CP</td>
<td>67.7</td>
<td>66.0</td>
</tr>
<tr>
<td>Boot-soft</td>
<td>66.9</td>
<td>65.3</td>
</tr>
<tr>
<td>ProSelfLC</td>
<td>68.7</td>
<td>68.5</td>
</tr>
</tbody>
</table>

Table 5: The results of different hyperparameters on CIFAR-100 using ResNet-44. Under different noise rates, the best instantiation of each approach is bolded except for CCE.

<table>
<thead>
<tr>
<th>Method (hyperparameter)</th>
<th>Value of hyperparameter</th>
<th>Asymmetric label noise</th>
<th>Symmetric label noise</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>20% 30% 40%</td>
<td>20% 40% 60%</td>
</tr>
<tr>
<td>CCE</td>
<td>None or $\epsilon = 0$</td>
<td>66.6 63.4 59.5</td>
<td>58.0 50.1 37.9</td>
</tr>
<tr>
<td>LS ($\epsilon$)</td>
<td>0.125</td>
<td>66.4 65.6 63.1</td>
<td>61.7 52.5 39.1</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>67.9 66.4 65.0</td>
<td>62.8 55.4 40.9</td>
</tr>
<tr>
<td></td>
<td>0.50</td>
<td>66.8 65.8 64.6</td>
<td>63.8 57.2 46.5</td>
</tr>
<tr>
<td>CP ($\epsilon$)</td>
<td>0.125</td>
<td>65.7 64.2 60.3</td>
<td>59.8 52.3 39.6</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>66.8 65.1 61.6</td>
<td>61.0 53.3 40.9</td>
</tr>
<tr>
<td></td>
<td>0.50</td>
<td>67.7 66.0 64.4</td>
<td>64.0 56.8 44.1</td>
</tr>
<tr>
<td>Boot-soft ($\epsilon$)</td>
<td>0.125</td>
<td>65.8 64.1 60.7</td>
<td>59.7 51.2 40.6</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>66.2 64.1 60.3</td>
<td>61.1 54.4 43.3</td>
</tr>
<tr>
<td></td>
<td>0.50</td>
<td>66.9 65.3 61.0</td>
<td>63.2 59.0 44.8</td>
</tr>
<tr>
<td>ProSelfLC ($B$)</td>
<td>8</td>
<td>67.8 67.4 67.9</td>
<td>64.7 57.7 47.7</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>68.5 68.5 66.8</td>
<td>63.9 59.0 47.5</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>68.6 67.9 67.4</td>
<td>64.0 59.3 47.5</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>68.7 68.0 67.8</td>
<td>64.8 59.0 47.4</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>68.4 67.2 67.3</td>
<td>63.7 59.0 32.3</td>
</tr>
</tbody>
</table>

**Baselines.** We compare with the results reported recently in SL [46]. Forward is a loss correction approach that uses a noise-transition matrix [32]. D2L monitors the subspace dimensionality change at training [27]. GCE denotes generalised cross entropy [59] and SL is symmetric cross entropy [46]. They are robust losses designed for solving label noise. Training details are the same as Section 5.1.

**Result analysis.** For all methods, we directly report their final results when training terminates. Therefore, we test the robustness of a model against not only label noise, but also a long time being exposed to the data. In Table 4, we observe that: (1) ProSelfLC outperforms all baselines, which is significant in most cases; (2) In both implementations, Boot-hard and Boot-soft perform worse than the others. However, our ProSelfLC makes Self LC the best solution. Furthermore, learning dynamics are visualised in Figure 3, which helps to understand why ProSelfLC works better. Figure 4 shows the learning dynamics when $r$ changes. Results of different $B$, $\epsilon$ are in Table 5. We note that when the noise rate is higher, a smaller $B$ performs better.

We further discuss their differences on model calibration [11] in Appendix B, the changes of entropy and $\epsilon_{\text{ProSelfLC}}$ during training in Appendix C.

**Revising the semantic class and similarity structure.** In Figures 3b and 3c, we show dynamic statistics of different approaches on fitting wrong labels and correcting them. ProSelfLC is much better than its counterparts. Semantic class correction reflects the change of similarity structure.

**To redefine and reward a low-entropy status.** On the one hand, we observe that LS and CP work well, being consistent with prior claims. In Figures 3d and 3e, the entropies of both clean and noisy subsets are much higher in LS and CP, correspondingly their generalisation is the best except for ProSelfLC in Figure 3f. On the other hand, ProSelfLC has the lowest entropy while performs the best, which proves that a learner’s confidence does not necessarily weaken its generalisation performance. Instead, a model needs to be careful with what to be confident in. As shown

---

1. We do not consider DisturbLabel [49], which flips labels randomly and is counter-intuitive. It weakens the generalisation because generally the accuracy drops as the uniform label noise increases.
Table 6: Test accuracy (%) on the real-world noisy dataset Clothing 1M.

<table>
<thead>
<tr>
<th>Bootstrap</th>
<th>Forward</th>
<th>D2L</th>
<th>GCE</th>
<th>SL</th>
<th>S-adaptation</th>
<th>Masking</th>
<th>MD-DYR-SH</th>
<th>Joint-soft</th>
<th>Our Trained Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CCE</td>
<td>LS</td>
<td>CP</td>
<td>Boot-soft</td>
<td>ProSelfLC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>68.9</td>
<td>69.8</td>
<td>69.5</td>
<td>69.8</td>
<td>70.3</td>
<td>71.0</td>
<td>71.0</td>
<td>72.2</td>
<td>71.8</td>
</tr>
<tr>
<td></td>
<td>72.6</td>
<td>72.4</td>
<td>72.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72.3</td>
</tr>
<tr>
<td></td>
<td>73.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Learning dynamics on CIFAR-100 under asymmetric noisy labels. We show all iterations only in (a) and (d). In the others, we show the second half iterations, which are of higher interest. As the noise rate increases, the superiority of ProSelfLC becomes more significant, i.e., avoiding fitting noise in the 2nd row and better generalisation in the 1st row.

by Figures 3b and 3c, ProSelfLC has the least wrong fitting and most semantic class correction, which indicates that a meaningful low-entropy status is redefined.

5.3. Real-world label noise

Clothing 1M [48] has around 38.46% label noise in the training data and about 1 million images of 14 classes from shopping websites. Its internal noise structure is agnostic.

Baselines. For loss correction and estimating the noise-transition matrix, S-adaptation [8] uses an extra softmax layer, while Masking [12] exploits human cognition. MD-DYR-SH [2] is a combination of three techniques: dynamic mixup (MD), dynamic bootstrapping together with label regularisation (DYR) and soft to hard (SH). The other baselines have been introduced heretofore.

Training details. We follow [43] to train ResNet-50 and initialise it by a trained model on ImageNet. We follow Section 5.1 with small changes: the initial learning rate is 0.01 and we train 10k iterations. They are searched on the separate clean validation set.

Result analysis. In Table 6, analogously to CIFAR-100, we report our trained results of CCE, LS, CP, Boot-soft and ProSelfLC for an entirely fair comparison. ProSelfLC has the highest accuracy, which demonstrates its effectiveness again.

6. Conclusion

We present a thorough mathematical study on several target modification techniques. Through analysis of entropy and KL divergence, we reveal their relationships and limitations. To improve and endorse self label correction, we propose ProSelfLC. Extensive experiments prove its superiority over existing methods under standard and noisy settings. ProSelfLC enhances the similarity structure information over classes, and rectifies the semantic classes of noisy label distributions. ProSelfLC is the first approach to trust self knowledge progressively and adaptively.

ProSelfLC redirects and promotes entropy minimisation, which is in marked contrast to recent practices of confidence penalty [42, 33, 6].
[54] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W Tsang, and Masashi Sugiyma. How does disagreement help generalization against label corruption? In ICML, 2019.