Introduction of Learning with Noisy Labels

Personal Reading Notes

Jianglin Lu

https://jianglin954.github.io/
jianglinlu@outlook.com

Outline

Outline

Preliminaries—Why Learning with Noisy Labels

The success of deep neural networks depends on access to high-quality labeled training data, as the presence of label errors (label noise) in training data can greatly reduce the accuracy of models on clean test data.

Unfortunately, large training datasets almost always contain examples with inaccurate or incorrect labels. This leads to a paradox: on one hand, large datasets are necessary to train better deep networks, while on the other hand, deep networks tend to memorize training label noise, resulting in poorer model performance in practice.

Preliminaries—Categories

Statistically Inconsistent Method

Employing heuristics to reduce the side-effect of noisy labels, e.g., select reliable examples, reweight examples, correct labels, employ side information, (implicitly) add regularization.

Note: the differences between the learned classifiers and the optimal ones for clean data are not guaranteed to vanish, i.e., no statistical consistency has been guaranteed.

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Preliminaries—Categories

The issue of statistically inconsistent method motivates researchers to explore algorithms in another category: risk-/classifier- consistent algorithms.

Statistically Consistent Method

Risk-consistent methods possess statistically consistent estimators to the clean risk (i.e., risk, w.r.t. the clean data), while classifier-consistent methods guarantee the classifier learned from the noisy data is consistent to the optimal classifier (i.e., the minimizer of the clean risk).

Utilizing noise transition matrix, denoting the probabilities that clean labels flip into noisy labels, to build consistent algorithms.

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Preliminaries—Categories

An estimator is risk-consistent if, by increasing the size of noisy samples, the *empirical risk* calculated by noisy samples and the modified loss function will converge to the *expected risk* calculated by clean examples and the original loss function.

An algorithm is classifier-consistent if, by increasing the size of noisy examples, the *learned classifier* will converge to the *optimal classifier* learned by clean examples.

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Preliminaries—Definitions

Problem Definition

Let $\mathcal{D}$ be the distribution of a pair of random variables $(X, Y) \in \mathcal{X} \times \{1, 2, ..., C\}$, where the feature space $\mathcal{X} \subseteq \mathbb{R}^d$ and C is the size of label classes. Our goal is to predict a label y for any given instance $x \in \mathcal{X}$. However, in many real-world classification problems, training examples drawn independently from distribution $\mathcal{D}$ are unavailable. Before being observed, their true labels are independently flipped and what we can obtain is a noisy training sample $\{X_i, \bar{Y}_i\}_{i=1}^n$, where $\bar{Y}$ denotes the noisy label. Let $\bar{\mathcal{D}}$ be the distribution of the noisy random variables $(X, \bar{Y}) \in \mathcal{X} \times \{1, 2, ..., C\}$.

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Transition Matrix

The random variables $\bar{Y}$ and Y are related through a noise transition matrix $T \in [0,1]^{C \times C}$, where the ij-th entry of the transition matrix $T_{ij}(x) = P(\bar{Y} = j | Y = i, X = x)$ represents the probability that the instance x with the clean label $Y = i$ will have a noisy label $\bar{Y} = j$.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Preliminaries—Definitions

Two Representative Transition Matrix T

Symmetry flipping:

$$T = \begin{bmatrix} 1-\epsilon & \frac{\epsilon}{C-1} & \cdots & \frac{\epsilon}{C-1} & \frac{\epsilon}{C-1} \\ \frac{\epsilon}{C-1} & 1-\epsilon & \frac{\epsilon}{C-1} & \cdots & \frac{\epsilon}{C-1} \\ \vdots & & \ddots & & \vdots \\ \frac{\epsilon}{C-1} & \cdots & \frac{\epsilon}{C-1} & 1-\epsilon & \frac{\epsilon}{C-1} \\ \frac{\epsilon}{C-1} & \frac{\epsilon}{C-1} & \cdots & \frac{\epsilon}{C-1} & 1-\epsilon \end{bmatrix}$$

Asymmetric pair flipping:

$$T = \begin{bmatrix} 1-\epsilon & \epsilon & 0 & \cdots & 0 \\ 0 & 1-\epsilon & \epsilon & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & 1-\epsilon & \epsilon \\ \epsilon & 0 & \cdots & 0 & 1-\epsilon \end{bmatrix}$$

where C is the number of class and ϵ is the noise rate.

Han et al. A Survey of Label-noise Representation Learning: Past, Present and Future. arXiv, 2021

Note that, the clean class posterior $P(Y|x) = [P(Y=1|X=x), ..., P(Y=C|X=x)]^T$ can be inferred by using the transition matrix and the noisy class posterior $P(\bar{Y}|x) = [P(\bar{Y}=1|X=x), ..., P(\bar{Y}=C|X=x)]^T$, i.e., we have

$$
\begin{aligned}
P(\bar{Y}|x) &= \begin{pmatrix} P(\bar{Y}=1|X=x) \\ \vdots \\ P(\bar{Y}=C|X=x) \end{pmatrix} && (1) \\
&= \begin{pmatrix} P(\bar{Y}=1, Y=1|X=x) + \cdots + P(\bar{Y}=1, Y=C|X=x) \\ \vdots \\ P(\bar{Y}=C, Y=1|X=x) + \cdots + P(\bar{Y}=C, Y=C|X=x) \end{pmatrix} && (2) \\
&= \begin{pmatrix} P(\bar{Y}=1|Y=1, X=x) & \cdots & P(\bar{Y}=C|Y=1, X=x) \\ \vdots & \vdots & \vdots \\ P(\bar{Y}=C|Y=1, X=x) & \cdots & P(\bar{Y}=C|Y=1, X=x) \end{pmatrix} \cdot \begin{pmatrix} P(Y=1|X=x) \\ \vdots \\ P(Y=C|X=x) \end{pmatrix} && (3) \\
&= T(x) \cdot P(Y|x) && (4)
\end{aligned}
$$

Note: $P(\bar{Y}=1, Y=1|X=x) = P(\bar{Y}=1|Y=1, X=x) \cdot P(Y=1|X=x)$.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Anchor Points

Anchor Points are widely used to estimate the transition matrix, which are defined in the clean data domain. An instance x is an anchor point for the class i if $P(Y = i|X = x)$ is equal to one or close to one. Given an anchor point x, we have $P(Y = k|X = x) = 0, \forall k \neq i$. Then, we have: $P(\bar{Y} = j|X = x) = \sum_{k=1}^{C} T_{kj} P(Y = k|X = x) = T_{ij}$.

That is to say, T can be obtained via estimating the noisy class posterior probabilities for anchor points.

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Outline

Statistically Inconsistent Classifiers

Categories

- Early Stopping
- Select Reliable Examples
- Correct Labels
- Add Regularization

Outline

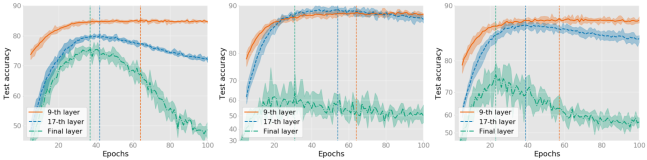
Progressive Early Stopping (PES)

Motivation: A DNN can be considered as a composition of a series of layers, and we find that *the latter layers in a DNN are much more sensitive to label noise, while their former counterparts are quite robust*. Selecting a stopping point for the whole network may make different DNN layers antagonistically affect each other, thus degrading the final performance.

Bai et al. Understanding and Improving Early Stopping for Learning with Noisy Labels. NIPS, 2021

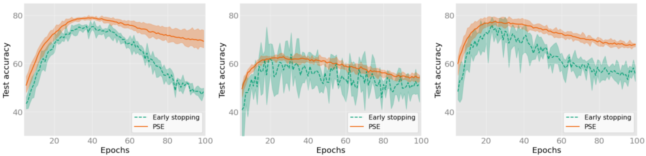
Early Stopping

Method: PES



(a) Symmetric 50% (b) Pairflip 45% (c) Instance 40%

Figure 1: We train a ResNet-18 model on CIFAR-10 with three types of noisy labels and evaluate the impact of noisy labels on the representations from the 9-th layer, the 17-th layer, and the final layer. The X-axis is the number of epochs for the first block of the network. The curves present the mean of five runs and the best performances are indicated with dotted vertical lines.



(a) Symmetric 50% (b) Pairflip 45% (c) Instance 40%

Figure 2: Performance of the traditional early stopping trick and the proposed PES on CIFAR-10 with different types of label noise. The lines present the mean of five runs.

Bai et al. Understanding and Improving Early Stopping for Learning with Noisy Labels. NIPS, 2021

Method: PES

Assume the whole network $f(\cdot;\Theta)$ can be constituted with L DNN parts:

$$\begin{aligned} \boldsymbol{z}_1 &= f_1\left(\boldsymbol{x};\Theta_1\right), \\ \boldsymbol{z}_l &= f_l\left(\boldsymbol{z}_{l-1};\Theta_l\right), \quad l=2,\ldots,L \end{aligned}$$

where $f_l(\cdot;\Theta_l)$ is the l-th DNN part and $\boldsymbol{z}_l$ is the corresponding output. We initially optimize the parameter Θ_1 for the first part by training the whole network for T_1 epochs with the following objective:

$$\min_{\Theta_1 \ldots \Theta_k} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x_i;\Theta_1,\ldots,\Theta_L\right), \tilde{y}_i\right)$$

Bai et al. Understanding and Improving Early Stopping for Learning with Noisy Labels. NIPS, 2021

Early Stopping

Method: PES

Then, we keep the obtained parameter Θ_1^* fixed, reinitialize and progressively learn the l-th ($l = 2, \ldots, L$) DNN part with the parameters for preceding DNN parts fixed. The training procedure is conducted with T_l epochs by optimizing the following objective:

$$\min_{\Theta_l \ldots \Theta_k} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x_i; \Theta_1^*, \ldots, \Theta_{l-1}^*, \Theta_l, \ldots, \Theta_L\right), \tilde{y}_i\right), \quad l = 2 \ldots L$$

Since latter DNN parts are more sensitive to noisy labels than their former counterparts, we gradually reduce the training epochs (i.e., $T_1 \geq T_2 \geq \cdots \geq T_L$) to better exploit the memorization effect.

Bai et al. Understanding and Improving Early Stopping for Learning with Noisy Labels. NIPS, 2021

Method: PES

Algorithm 1: Progressive Early Stopping with Semi-Supervised Learning

Input: Neural network with trainable parameters $\Theta = \{\Theta_1, \ldots, \Theta_L\}$, Noisy training dataset $\{x_i, \tilde{y}_i)\}_{i=1}^n$, Number of training epochs for different part: $T_1, \ldots, T_L$, and training epochs T_c for refining with confident examples.

for $i = 1, \ldots, T_1$ **do**
  Optimize network parameter Θ with Eq. (3);

for $l = 2, \ldots, L$ **do**
  Froze $\{\Theta_1, \ldots, \Theta_{l-1}\}$ and re-initialize $\{\Theta_l, \ldots, \Theta_L\}$;
  for $i = 1, \ldots, T_l$ **do**
    Optimize network parameter $\{\Theta_l, \ldots, \Theta_L\}$ with Eq. (4);

Unfroze Θ;

for $i = 1, \ldots, T_c$ **do**
  Extract confident example set $\mathcal{D}_l$ and unlabeled set $\mathcal{D}_u$ with classifier $f(\cdot, \Theta)$ by Eq. (7);
  Training the classifier $f(\cdot, \Theta)$ with MixMatch loss on $\mathcal{D}_l$ and $\mathcal{D}_u$;

Evaluate the obtained classifier $f(\cdot, \Theta)$.

Bai et al. Understanding and Improving Early Stopping for Learning with Noisy Labels. NIPS, 2021

Outline

Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels (MentorNet)

Motivation 1: Deep CNNs are more prone to overfitting and memorizing corrupted labels. To address this issue, we focus on training very deep CNNs from scratch.

Motivation 2: Existing curriculums are usually predefined and remain fixed during training, ignoring the feedback from the student. Moreover, the alternating minimization requires alternative variable updates, which is difficult for training very deep CNNs via mini-batch stochastic gradient descent.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Select Reliable Examples

Preliminary on Curriculum Learning:

Curriculum Learning

Curriculum Learning (CL) is a learning paradigm inspired by the cognitive process of human and animals, in which a model is learned gradually using samples ordered in a meaningful sequence. A reasonable curriculum can help the student focus on the samples whose labels have a high chance of being correct.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Select Reliable Examples

Preliminary on Curriculum Learning:

Let $g_s(\mathbf{x}_i, \mathbf{w})$ denote the discriminative function of a neural network called StudentNet parameterized by $\mathbf{w} \in \mathbb{R}^d$, and $\mathbf{L}(\mathbf{y}_i, g_s(\mathbf{x}_i, \mathbf{w}))$, a m-dimensional column vector, denote the loss over m classes. Introduce the latent weight variable $\mathbf{v} \in [0,1]^{n\times m}$, and optimize the objective:

$$\min_{\mathbf{w}\in\mathbb{R}^d, \mathbf{v}\in[0,1]^{n\times m}} \mathbb{F}(\mathbf{w}, \mathbf{v}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_i^T \mathbf{L}(\mathbf{y}_i, g_s(\mathbf{x}_i, \mathbf{w})) + G(\mathbf{v}; \lambda) + \theta\|\mathbf{w}\|_2^2$$

where $\mathbf{v}_i \in [0,1]^{m\times 1}$ is a vector to represent the latent weight variable for the i-th sample, and the function G defines a **curriculum** parameterized by λ.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Select Reliable Examples

A predefined curriculum known as **self-paced learning** optimizes $\mathbf{v}$ by:

$$v_i^* = \mathbb{1}\left(\ell_i \leq \lambda\right), \forall i \in [1, n]$$

where we denote the loss $\mathbf{L}\left(\mathbf{y}_i, g_s\left(\mathbf{x}_i, \mathbf{w}\right)\right) = \ell_i$, $\mathbb{1}$ as the indicator function.

△ When updating $\mathbf{v}$ with fixed $\mathbf{w}$, a sample of smaller loss than the threshold λ is treated as an easy sample, and will be selected in training $\left(v_i^* = 1\right)$.

△ When updating $\mathbf{w}$ with fixed $\mathbf{v}$, the classifier is trained only on the selected "easy" samples. The hyperparameter λ controls the learning pace and corresponds to the "age" of the model. When λ is small, only samples of small loss will be considered. As λ grows, more samples of larger loss will be gradually added to train a more "mature" model.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Method: MentorNet

During training, MentorNet provides a curriculum (sample weighting scheme) for StudentNet to focus on the sample the label of which is probably correct. MentorNet can be learned to approximate an existing predefined curriculum or discover new data-driven curriculums from data.

The MentorNet g_m is learned to compute time-varying weights for each training sample. Let Θ denote the parameters in g_m. Given a fixed $\mathbf{w}$, our goal is to learn an Θ^* to compute the weight:

$$g_m(\mathbf{z}_i; \Theta^*) = \arg\min_{v_i \in [0,1]} \mathbb{F}(\mathbf{w}, \mathbf{v}), \forall i \in [1, n]$$

where $\mathbf{z}_i = \theta(\mathbf{x}_i, \mathbf{y}_i, \mathbf{w})$ indicates the input feature to MentorNet about the i-the sample.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Select Reliable Examples

Learning to Approximate Predefined Curriculums:

The first task is to learn a MentorNet to approximate a predefined curriculum. To do so, we minimize the following objective:

$$\arg\min_{\Theta} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} g_m(\mathbf{z}_i; \Theta)\, \ell_i + G\left(g_m(\mathbf{z}_i; \Theta); \lambda\right)$$

We employ the following predefined curriculum:

$$G(\mathbf{v}; \lambda) = \sum_{i=1}^{n} \frac{1}{2} \lambda_2 v_i^2 - (\lambda_1 + \lambda_1)\, v_i$$

where $\lambda_1, \lambda_1 \geq 0$ are hyper-parameters.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Learning to Approximate Predefined Curriculums:

Given a fixed **w**, we define $\mathbb{F}_{\mathbf{w}}(\mathbf{v}) = \sum_{i=1}^{n} f(v_i)$:

$$f(v_i) = v_i \ell_i + \frac{1}{2}\lambda_2 v_i^2 - (\lambda_1 + \lambda_2) v_i$$

By setting $\partial f / \partial v_i = 0$, we have:

$$g_m(\mathbf{z}_i; \Theta^*) = \begin{cases} \mathbb{1}(\ell_i \leq \lambda_1) & \lambda_2 = 0 \\ \min\left(\max\left(0, 1 - \frac{\ell_i - \lambda_1}{\lambda_2}\right), 1\right) & \lambda_2 \neq 0 \end{cases}$$

where Θ^* is the optimal MentorNet parameter obtained by SGD.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Select Reliable Examples

Learning Data-Driven Curriculums:

Θ can be learned on another dataset $\mathcal{D}' = \{(\phi(\mathbf{x}_i, y_i, \mathbf{w}), v_i^*)\}$ where $(\mathbf{x}_i, y_i)$ is sampled from $\mathcal{D}$ and $|\mathcal{D}'| \ll |\mathcal{D}|$. v_i^* is a given annotation and we assume it approximates the optimal weight, i.e., $v_i^* \simeq \arg\min_{v_i \in [0,1]} \mathbb{F}(\mathbf{v}, \mathbf{w})$. Specifically, we assign binary labels to v_i^*, where $v_i^* = 1$ iff y_i is a correct label. As v_i^* is binary, Θ is learned by minimizing the cross-entropy loss between v_i^* and $g(\mathbf{z}_i; \Theta)$.

The information on the correct label may not always be available on the target dataset D. In this case, we learn the curriculum on a different small dataset where the correct labels are available.

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Select Reliable Examples

SPADE (Scholastic gradient PArtial DEscent)

The partial gradient update on weight parameters is performed when G is used (Step 9). Otherwise, we directly apply the weights computed by the learned MentorNet (Step 11). The curriculum can change during training. In Step 6, the MentorNet parameter Θ is updated to adapt to the most recent model parameters of StudentNet. In experiments, we update Θ twice after the learning rate is changed.

Algorithm 1 SPADE for minimizing Eq. (1)

Input : Dataset $\mathcal{D}$, a predefined G or a learned $g_m(\cdot;\Theta)$
Output : The model parameter $\mathbf{w}$ of StudentNet.

1 Initialize $\mathbf{w}^0, \mathbf{v}^0, t = 0$
2 **while** *Not Converged* **do**
3 Fetch a mini-batch Ξ_t uniformly at random
4 For every $(\mathbf{x}_i, y_i)$ in Ξ_t compute $\phi(\mathbf{x}_i, y_i, \mathbf{w}^t)$
5 **if** *update curriculum* **then**
6 $\Theta \leftarrow \Theta^*$, where Θ^* is learned in Sec. 3.1
7 **end**
8 **if** *G is used* **then**
9 $\mathbf{v}_\Xi^t \leftarrow \mathbf{v}_\Xi^{t-1} - \alpha_t \nabla_\mathbf{v} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^{t-1})|_{\Xi_t}$
10 **end**
11 **else** $\mathbf{v}_\Xi^t \leftarrow g_m(\phi(\Xi_t, \mathbf{w}^{t-1}); \Theta)$;
12 $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \alpha_t \nabla_\mathbf{w} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^t)|_{\Xi_t}$
13 $t \leftarrow t + 1$
14 **end**
15 **return** $\mathbf{w}^t$

Jiang et al. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. ICML, 2018

Robust Training of Deep Neural Networks (Co-Teaching)

Motivation: recent studies on the memorization effects of deep neural networks show that they would first memorize training data of clean labels and then those of noisy labels.

Han et al. Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels. NeurIPS, 2018

Method: Co-Teaching

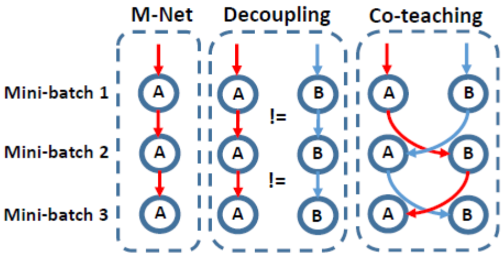


Figure 1: Comparison of error flow among MentorNet (M-Net) [17], Decoupling [26] and Co-teaching. Assume that the error flow comes from the biased selection of training instances, and error flow from network A or B is denoted by red arrows or blue arrows, respectively. **Left panel:** M-Net maintains only one network (A). **Middle panel:** Decoupling maintains two networks (A & B). The parameters of two networks are updated, when the predictions of them disagree (!=). **Right panel:** Co-teaching maintains two networks (A & B) simultaneously. In each mini-batch data, each network samples its small-loss instances as the useful knowledge, and teaches such useful instances to its peer network for the further training. Thus, the error flow in Co-teaching displays the zigzag shape.

Han et al. Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels. NeurIPS, 2018

Method: Co-Teaching

Algorithm 1 Co-teaching Algorithm.

1: **Input** w_f and w_g, learning rate η, fixed τ, epoch T_k and $T_{\max}$, iteration $N_{\max}$;
for $T = 1, 2, \ldots, T_{\max}$ **do**
  2: **Shuffle** training set $\mathcal{D}$; //noisy dataset
  for $N = 1, \ldots, N_{\max}$ **do**
    3: **Fetch** mini-batch $\bar{\mathcal{D}}$ from $\mathcal{D}$;
    4: **Obtain** $\bar{\mathcal{D}}_f = \arg\min_{\mathcal{D}':|\mathcal{D}'| \geq R(T)|\bar{\mathcal{D}}|} \ell(f, \mathcal{D}')$; //sample $R(T)\%$ small-loss instances
    5: **Obtain** $\bar{\mathcal{D}}_g = \arg\min_{\mathcal{D}':|\mathcal{D}'| \geq R(T)|\bar{\mathcal{D}}|} \ell(g, \mathcal{D}')$; //sample $R(T)\%$ small-loss instances
    6: **Update** $w_f = w_f - \eta \nabla \ell(f, \bar{\mathcal{D}}_g)$; //update w_f by $\bar{\mathcal{D}}_g$;
    7: **Update** $w_g = w_g - \eta \nabla \ell(g, \bar{\mathcal{D}}_f)$; //update w_g by $\bar{\mathcal{D}}_f$;
  end
  8: **Update** $R(T) = 1 - \min\left\{\frac{T}{T_k}\tau, \tau\right\}$;
end
9: **Output** w_f and w_g.

In each mini-batch of data, each network views its small-loss instances as the useful knowledge, and teaches such useful instances to its peer network for updating the parameters.

Han et al. Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels. NeurIPS, 2018

Robust Training of Deep Neural Networks (Co-Teaching)

Q 1: Why can sampling small-loss instances help us find clean instances?

Answer: Intuitively, when labels are correct, small-loss instances are more likely to be the ones which are correctly labeled. *Deep networks will learn clean and easy pattern in the initial epochs.* So, they have the ability to filter out noisy instances using their loss values at the beginning of training.

Q 2: Why do we need two networks and cross-update the parameters?

Answer: Intuitively, *different classifiers can generate different decision boundaries and then have different abilities to filter out the label noise*. This motivates us to exchange the selected small-loss instances, so that these two networks can adaptively correct the training error by the peer network if the selected instances are not fully clean.

Han et al. Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels. NeurIPS, 2018

Pytorch Code: Co-Teaching

```
Loss functions
f loss_coteaching(y_1, y_2, t, forget_rate, ind, noise_or_not):
  loss_1 = F.cross_entropy(y_1, t, reduce = False)
  ind_1_sorted = np.argsort(loss_1.data).cuda()
  loss_1_sorted = loss_1[ind_1_sorted]

  loss_2 = F.cross_entropy(y_2, t, reduce = False)
  ind_2_sorted = np.argsort(loss_2.data).cuda()
  loss_2_sorted = loss_2[ind_2_sorted]

  remember_rate = 1 - forget_rate
  num_remember = int(remember_rate * len(loss_1_sorted))

  pure_ratio_1 = np.sum(noise_or_not[ind[ind_1_sorted[:num_remember]]])/float(num_remember)
  pure_ratio_2 = np.sum(noise_or_not[ind[ind_2_sorted[:num_remember]]])/float(num_remember)

  ind_1_update=ind_1_sorted[:num_remember]
  ind_2_update=ind_2_sorted[:num_remember]
  # exchange
  loss_1_update = F.cross_entropy(y_1[ind_2_update], t[ind_2_update])
  loss_2_update = F.cross_entropy(y_2[ind_1_update], t[ind_1_update])

  return torch.sum(loss_1_update)/num_remember, torch.sum(loss_2_update)/num_remember, pure_ratio_1, pure_ratio_2
```

Disagreement Help Generalization against Label Corruption (Co-Teaching+)

Motivation 1: With the increase of epochs, two networks converge to a consensus and Co-teaching reduces to the self-training MentorNet.

Motivation 2: To address the consensus issue in Co-teaching, we should consider how to always keep two networks diverged within the training epochs, or how to slow down the speed that two networks will reach a consensus with the increase of epochs.

Yu et al. How does Disagreement Help Generalization against Label Corruption? ICML, 2019

Method: Co-teaching+

Figure 1. Comparison of divergence (evaluated by Total Variation) between two networks trained by the "Disagreement" strategy, Co-teaching and Co-teaching+, respectively. Co-teaching+ naturally bridges the "Disagreement" strategy with Co-teaching.

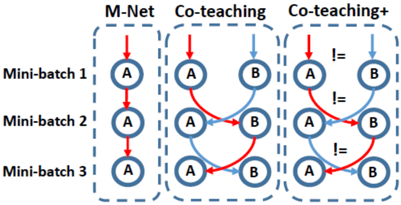


Figure 2. Comparison of error flow among MentorNet (M-Net), Co-teaching and Co-teaching+. Assume that the error flow comes from the selection of training instances, and the error flow from network A or B is denoted by red arrows or blue arrows, respectively. **Left panel:** M-Net maintains only one network (A). **Middle panel:** Co-teaching maintains two networks (A & B) simultaneously. In each mini-batch data, each network selects its small-loss data to teach its peer network for the further training. **Right panel:** Co-teaching+ also maintains two networks (A & B). However, two networks feed forward and predict each mini-batch data first, and keep prediction disagreement data (!=) only. Based on such disagreement data, each network selects its small-loss data to teach its peer network for the further training.

Yu et al. How does Disagreement Help Generalization against Label Corruption? ICML, 2019

Method: Co-teaching+

Disagreement-Update: prediction disagreement data $\overline{\mathcal{D}}'$

$$\overline{\mathcal{D}}' = \left\{(x_i, y_i) : \bar{y}_i^{(1)} \neq \bar{y}_i^{(2)}\right\}$$

Cross-Update: $\lambda(e)$ controls how many small loss data should be selected in each training epoch

$$\lambda(e) = 1 - \min\left\{\frac{e}{E_k}\tau, \left(1 + \frac{e - E_k}{E_{\max} - E_k}\right)\tau\right\}$$

where $E_k = 10$ and $E_{\max} = 200$.

Algorithm 1 Co-teaching+. Step 4: disagreement-update; Step 5-8: cross-update.

1: **Input** $w^{(1)}$ and $w^{(2)}$, training set $\mathcal{D}$, batch size B, learning rate η, estimated noise rate τ, epoch E_k and $E_{\max}$;
for $e = 1, 2, \ldots, E_{\max}$ **do**
  2: **Shuffle** $\mathcal{D}$ into $\frac{|\mathcal{D}|}{B}$ mini-batches; //noisy dataset
  for $n = 1, \ldots, \frac{|\mathcal{D}|}{B}$ **do**
    3: **Fetch** n-th mini-batch $\bar{\mathcal{D}}$ from $\mathcal{D}$;
    4: **Select** prediction disagreement $\bar{\mathcal{D}}'$ by Eq. (1);
    5: **Get** $\bar{\mathcal{D}}'^{(1)} = \arg\min_{\mathcal{D}':|\mathcal{D}'|\geq\lambda(e)|\bar{\mathcal{D}}'|} \ell(\mathcal{D}'; w^{(1)})$; //sample $\lambda(e)\%$ small-loss instances
    6: **Get** $\bar{\mathcal{D}}'^{(2)} = \arg\min_{\mathcal{D}':|\mathcal{D}'|\geq\lambda(e)|\bar{\mathcal{D}}'|} \ell(\mathcal{D}'; w^{(2)})$; //sample $\lambda(e)\%$ small-loss instances
    7: **Update** $w^{(1)} = w^{(1)} - \eta\nabla\ell(\bar{\mathcal{D}}'^{(2)}; w^{(1)})$; //update $w^{(1)}$ by $\bar{\mathcal{D}}'^{(2)}$;
    8: **Update** $w^{(2)} = w^{(2)} - \eta\nabla\ell(\bar{\mathcal{D}}'^{(1)}; w^{(2)})$; //update $w^{(2)}$ by $\bar{\mathcal{D}}'^{(1)}$;
  end
  9: **Update** $\lambda(e) = 1 - \min\{\frac{e}{E_k}\tau, \tau\}$ or $1 - \min\{\frac{e}{E_k}\tau, (1 + \frac{e-E_k}{E_{\max}-E_k})\tau\}$;
end
10: **Output** $w^{(1)}$ and $w^{(2)}$.

Yu et al. How does Disagreement Help Generalization against Label Corruption? ICML, 2019

Combating Noisy Labels by Agreement: A Joint Training Method with Co-Regularization (JoCoR)

Motivation: Co-teaching+ and Decoupling introduce the Disagreement strategy, where "when to update" depends on a disagreement between two different networks. However, *there are only a part of training examples that can be selected by the Disagreement strategy, and these examples cannot be guaranteed to have ground-truth labels*. Therefore, there arises a question to be answered: *Is Disagreement necessary for training two networks to deal with noisy labels?*

Wei et al. Combating Noisy Labels by Agreement: A Joint Training Method with Co-Regularization. CVPR, 2020

Select Reliable Examples

Method: JoCoR

$$\ell\left(x_i\right)=(1-\lambda) * \ell_{\text{sup}}\left(\boldsymbol{x}_i, y_i\right)+\lambda * \ell_{\text{con}}\left(\boldsymbol{x}_i\right)$$

where we use Cross-Entropy Loss as the supervised part to minimize the distance between predictions and labels:

$$\begin{aligned}\ell_{\text{sup}}\left(\boldsymbol{x}_i, y_i\right) &= \ell_{\mathrm{C}1}\left(\boldsymbol{x}_i, y_i\right)+\ell_{\mathrm{C}2}\left(\boldsymbol{x}_i, y_i\right) \\ &= -\sum_{i=1}^{N} \sum_{m=1}^{M} y_i \log\left(p_1^m\left(\boldsymbol{x}_i\right)\right)-\sum_{i=1}^{N} \sum_{m=1}^{M} y_i \log\left(p_2^m\left(\boldsymbol{x}_i\right)\right)\end{aligned}$$

and utilize the contrastive term as Co-Regularization (which maximizes the agreement between two classifiers) to make the networks guide each other:

$$\ell_{\text{con}}=D_{\mathrm{KL}}\left(\boldsymbol{p}_1 \| \boldsymbol{p}_2\right)+D_{\mathrm{KL}}\left(\boldsymbol{p}_2 \| \boldsymbol{p}_1\right)$$

where $D_{\mathrm{KL}}\left(\boldsymbol{p}_1 \| \boldsymbol{p}_2\right)=\sum_{i=1}^{N} \sum_{m=1}^{M} p_1^m\left(\boldsymbol{x}_i\right) \log \frac{p_1^m\left(\boldsymbol{x}_i\right)}{p_2^m\left(\boldsymbol{x}_i\right)}$.

Wei et al. Combating Noisy Labels by Agreement: A Joint Training Method with Co-Regularization. CVPR, 2020

Select Reliable Examples

Method: JoCoR

Small-loss Selection:

$$\tilde{D}_n = \arg \min_{D_n' : |D_n'| \geq R(t)|D_n|} \ell\left(D_n'\right)$$

After obtaining the small-loss instances, we calculate the average loss on these examples for further backpropagation:

$$L = \frac{1}{|\tilde{D}|} \sum_{\boldsymbol{x} \in \tilde{D}} \ell(\boldsymbol{x})$$

Algorithm 1 JoCoR

Input: Network f with $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2\}$, learning rate η, fixed τ, epoch T_k and $T_{\max}$, iteration $I_{\max}$;

- **for** t = 1,2,...,$T_{\max}$ **do**
  - **Shuffle** training set D;
  - **for** $n = 1, \ldots, I_{\max}$ **do**
    - **Fetch** mini-batch D_n from D;
    - $\boldsymbol{p}_1 = f(\boldsymbol{x}, \boldsymbol{\Theta}_1), \forall \boldsymbol{x} \in D_n$;
    - $\boldsymbol{p}_2 = f(\boldsymbol{x}, \boldsymbol{\Theta}_2), \forall \boldsymbol{x} \in D_n$;
    - **Calculate** the joint loss ℓ by (1) using $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$;
    - **Obtain** small-loss sets $\tilde{D}_n$ by (4) from D_n;
    - **Obtain** L by (5) on $\tilde{D}_n$;
    - **Update** $\boldsymbol{\Theta} = \boldsymbol{\Theta} - \eta \nabla L$;
  - **end for**
  - **Update** $R(t) = 1 - \min\left\{\frac{t}{T_k}\tau, \tau\right\}$
- **end for**

Output: $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$

Wei et al. Combating Noisy Labels by Agreement: A Joint Training Method with Co-Regularization. CVPR, 2020

Method: JoCoR

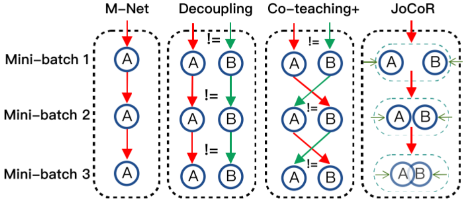


Figure 1. Comparison of error flow among MentorNet (M-Net) [16], Decoupling [23], Co-teaching+ [41] and JoCoR. Assume that the error flow comes from the biased selection of training instances, and error flow from network A or B is denoted by red arrows or green arrows, respectively. **First panel**: M-Net maintains only one network (A). **Second panel**: Decoupling maintains two networks (A&B). The parameters of two networks are updated, when the predictions of them disagree (!=). **Third panel**: In Co-teaching+, each network teaches its small-loss instances with prediction disagreement (!=) to its peer network. **Fourth panel**: JoCoR also maintains two networks (A&B) but trains them as a whole with a joint loss, which makes predictions of each network closer to ground true labels and peer network's.

Wei et al. Combating Noisy Labels by Agreement: A Joint Training Method with Co-Regularization. CVPR, 2020

Outline

Outline

Outline

Statistically Consistent Classifiers

Categories of Label Noise

- Random Classification Noise (RCN): each label is flipped independently with a constant probability ρ.
- Class-Conditional Random Label Noise (CCN): the flip probabilities (noise rates) ρ_y are the same for all labels from one certain class y.
- Instance- and Label-Dependent Noise (ILN): the flip rate $\rho_y(x)$ is dependent on both the instance x and the corresponding true label y.

Cheng et al. Learning with Bounded Instance- and Label-Dependent Label Noise. ICML, 2020

Outline

Outline

Class-Conditional Random Label Noise (CCN)

Learning Without Anchor Points (T Revision)

Motivation1: when there are no anchor points in datasets, how to maintain the efficacy of those consistent algorithms?

Motivation2: existing risk-consistent estimators involve the inverse of transition matrix, which degenerates classification performances and makes tuning the transition matrix ineffectively. How to design a risk-consistent estimator that does not involve the inverse of the transition matrix?

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Class-Conditional Random Label Noise (CCN)

Method: Risk-Consistent Estimator Using Importance Reweighting

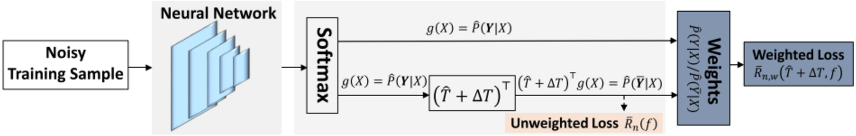
$$
\begin{aligned}
R(f) &= \mathbb{E}_{(X,Y)\sim D}[\ell(f(X),Y)] = \int_x \sum_i P_D(X=x, Y=i)\ell(f(x),i)dx \\
&= \int_x \sum_i P_{\bar{D}}(X=x, \bar{Y}=i)\frac{P_D(X=x,Y=i)}{P_{\bar{D}}(X=x,\bar{Y}=i)}\ell(f(x),i)dx \\
&= \int_x \sum_i P_{\bar{D}}(X=x, \bar{Y}=i)\frac{P_D(Y=i|X=x)}{P_{\bar{D}}(\bar{Y}=i|X=x)}\ell(f(x),i)dx \\
&= \mathbb{E}_{(X,Y)\sim \bar{D}}[\bar{\ell}(f(X),Y)],
\end{aligned}
$$

where $\bar{\ell}(f(X),i) = \frac{P_D(Y=i|X=x)}{P_{\bar{D}}(\bar{Y}=i|X=x)}\ell(f(X),i)$.

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Class-Conditional Random Label Noise (CCN)

Method: Reweight T-Revision



$$\bar{R}_{n,w}(T, f) = \frac{1}{n} \sum_{i=1}^{n} \frac{g_{\bar{Y}_i}(X_i)}{(T^{\top} g)_{\bar{Y}_i}(X_i)} \ell\left(f(X_i), \bar{Y}_i\right)$$

where $g(x) = \hat{P}(Y|X = x) \approx P(Y|X = x)$, $T^T g(x) = (\hat{T} + \Delta T)^T g(x) = \hat{P}(\bar{Y}|X = x) \approx P(\bar{Y}|X = x)$

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Class-Conditional Random Label Noise (CCN)

Pytorch Code:

Step 1: Train data with unweighted loss to learn an initial transition matrix $\hat{T}$, $epoch = 20$.

```
with torch.no_grad():
    model.eval()
    for index,(batch_x,batch_y) in enumerate(estimate_loader)
        batch_x = batch_x.cuda()
        out = model(batch_x, revision=False)
        out = F.softmax(out,dim=1)
        out = out.cpu()
        if index <= index_num:
            A[epoch][index*args.batch_size:(index+1)*args.bat
        else:
            A[epoch][index_num*args.batch_size, len(train_da
```

```
for epoch in range(args.n_epoch_estimate):

    print('epoch {}'.format(epoch + 1))
    model.train()
    train_loss = 0.
    train_acc = 0.
    val_loss = 0.
    val_acc = 0.

    for batch_x, batch_y in train_loader:
        batch_x = batch_x.cuda()
        batch_y = batch_y.cuda()
        optimizer_es.zero_grad()
        out = model(batch_x, revision=False)
        loss = loss_func_ce(out, batch_y)
        train_loss += loss.item()
        pred = torch.max(out, 1)[1]
        train_correct = (pred == batch_y).sum()
        train_acc += train_correct.item()
        loss.backward()
        optimizer_es.step()
```

Xia et al. Are anchor points really indispensable in label-noise learning. NeurIPS, 2019

Class-Conditional Random Label Noise (CCN)

Pytorch Code:

Step 2: Update network using weighted loss and the learned transition matrix $\hat{T}$, $epoch = 200$.

```
for epoch in range(args.n_epoch):
    print('epoch {}'.format(epoch + 1))
    # training-----------------------------
    train_loss = 0.
    train_acc = 0.
    val_loss = 0.
    val_acc = 0.
    eval_loss = 0.
    eval_acc = 0.
    scheduler.step()
    model.train()
    for batch_x, batch_y in train_loader:
        batch_x = batch_x.cuda()
        batch_y = batch_y.cuda()
        optimizer.zero_grad()
        out = model(batch_x, revision=False)
        prob = F.softmax(out, dim=1)
        prob = prob.t()
        loss = loss_func_reweight(out, T, batch_y)
        out_forward = torch.matmul(T.t(), prob)
        out_forward = out_forward.t()
        train_loss += loss.item()
        pred = torch.max(out_forward, 1)[1]
        train_correct = (pred == batch_y).sum()
        train_acc += train_correct.item()
        loss.backward()
        optimizer.step()
```

Class-Conditional Random Label Noise (CCN)

Pytorch Code:

Step 3: Train data using weighted loss with the learned transition matrix $\hat{T}$ to learn ΔT, $epoch = 200$.

```
for epoch in range(args.n_epoch_revision):

    print('epoch {}'.format(epoch + 1))
    # training-----------------------------
    train_loss = 0.
    train_acc = 0.
    val_loss = 0.
    val_acc = 0.
    eval_loss = 0.
    eval_acc = 0.
    model.train()
    for batch_x, batch_y in train_loader:
        batch_x = batch_x.cuda()
        batch_y = batch_y.cuda()
        optimizer_revision.zero_grad()
        out, correction = model(batch_x, revision=True)
        prob = F.softmax(out, dim=1)
        prob = prob.t()
        loss = loss_func_revision(out, T, correction, batch_y)
        out_forward = torch.matmul((T+correction).t(), prob)
        out_forward = out_forward.t()
        train_loss += loss.item()
        pred = torch.max(out_forward, 1)[1]
        train_correct = (pred == batch_y).sum()
        train_acc += train_correct.item()
        loss.backward()
        optimizer_revision.step()
```

Class-Conditional Random Label Noise (CCN)

Reducing Estimation Error for Transition Matrix (Dual T)

Motivation: Anchor points are hard to identify, but can be learned from noisy data by $x^i = \arg\max_x P(\bar{Y} = i|x)$, which means that learning anchor points relies heavily on the estimation of the noisy class posterior. However, the estimation error for noisy class posterior could be large due to the randomness of label noise, which would lead the transition matrix to be poorly estimated.

Motivation2: the estimation error of the noisy class posterior is significantly larger than that of the clean class posterior. How to find an alternative estimator that avoids directly using the estimated noisy class posterior to approximate the transition matrix.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Class-Conditional Random Label Noise (CCN)

Method: Introduce An Intermediate Class And Factorize T As:

$$
\begin{aligned}
T_{ij} &= P(\bar{Y} = j | Y = i) \\
&= \sum_{l \in \{1,\ldots,C\}} P\left(\bar{Y} = j, Y' = l | Y = i\right) \\
&= \sum_{l \in \{1,\ldots,C\}} P\left(\bar{Y} = j | Y' = l, Y = i\right) P\left(Y' = l | Y = i\right) \\
&\triangleq \sum_{l \in \{1,\ldots,C\}} T^{\spadesuit}_{lj}(Y = i) T^{\clubsuit}_{il}
\end{aligned}
$$

where Y' represent the random variable for the introduced intermediate class, $T^{\spadesuit}_{lj}(Y = i) = P\left(\bar{Y} = j | Y' = l, Y = i\right)$ represents the transition from the clean and intermediate class labels to the noisy class labels, and $T^{\clubsuit}_{il} = P\left(Y' = l | Y = i\right)$ represents the transition from the clean labels to the intermediate class labels.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Class-Conditional Random Label Noise (CCN)

Estimate $T_{il}^{\clubsuit}$: $T_{il}^{\clubsuit} = P\left(Y' = l | Y = i\right)$

We can design the intermediate class Y' in such a way that $P\left(Y'|x\right) \triangleq \hat{P}\left(\bar{Y}|x\right)$, where $\hat{P}\left(\bar{Y}|x\right)$ represents an estimated noisy class posterior, and can be obtained by exploiting the noisy data at hand. If anchor points are given, the estimation error for $T_{il}^{\clubsuit}$ is zero, since we have access to $P\left(Y'|x\right) \triangleq \hat{P}\left(\bar{Y}|x\right)$ directly.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Estimate $T^{\spadesuit}_{lj}$: $T^{\spadesuit}_{lj}(Y=i) = P\left(\bar{Y}=j|Y'=l, Y=i\right)$

Since the clean class labels are not available, we aim to eliminate the dependence on clean class for $T^{\spadesuit}_{lj}$. Specifically, if the clean class Y is less informative for the noisy class $\bar{Y}$ than the intermediate class Y', in other words, given Y', Y contains no more information for predicting $\bar{Y}$, then Y is independent of $\bar{Y}$ conditioned on Y', i.e.,

$$T^{\spadesuit}_{lj}(Y=i) = P\left(\bar{Y}=j|Y'=l, Y=i\right) = P\left(\bar{Y}=j|Y'=l\right)$$

An sufficient condition for holding the above equalities is to let the intermediate class labels be identical to noisy labels. Since it is hard to find an intermediate class whose labels are identical to noisy labels, the mismatch will be the main factor that contributes to the estimation error for $T^{\spadesuit}_{lj}$.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Class-Conditional Random Label Noise (CCN)

Estimate $T^{\spadesuit}_{lj}$: $T^{\spadesuit}_{lj}(Y=i) = P\left(\bar{Y}=j|Y'=l\right)$

Since the labels for the noisy class and intermediate class are available, $P\left(\bar{Y}=j|Y'=l\right)$ is easy to estimate by just counting the discrete labels as:

$$\hat{T}^{\spadesuit}_{lj} = \hat{P}\left(\bar{Y}=j|Y'=l\right) = \frac{\sum_i \mathbb{1}_{\{(\arg\max_k P(Y'=k|\boldsymbol{x}_i)=l)\wedge \bar{y}_i=j\}}}{\sum_i \mathbb{1}_{\{\arg\max_k P(Y'=k|\boldsymbol{x}_i)=l\}}}$$

where $\mathbb{1}_A$ is an indicator function which equals one when A holds true and zero otherwise. As we can see, we change the problem of estimating the noisy class posterior into the problem of fitting the noisy labels. The noisy class posterior is in the range of $[0,1]$ while the noisy class labels are in the set $\{1,\ldots,C\}$. Intuitively, learning the class labels are much easier than learning the class posteriors.

Yao et al. Dual T: reducing estimation error for transition matrix in label-noise Learning. NeurIPS, 2020

Class-Conditional Random Label Noise (CCN)

Pytorch Code: Generating Two Transition Matrices:

```
def get_transition_matrices(est_loader, model):
    model.eval()
    est_loader.eval()
    p = []
    T_spadesuit = np.zeros((args.num_classes,args.num_classes))
    with torch.no_grad():
        for i, (images, n_target,_) in enumerate(est_loader):
            images = images.cuda()
            n_target = n_target.cuda()
            pred = model(images)
            probs = F.softmax(pred, dim=1).cpu().data.numpy()
            _, pred = pred.topk(1, 1, True, True)
            pred = pred.view(-1).cpu().data
            n_target = n_target.view(-1).cpu().data
            for i in range(len(n_target)):
                T_spadesuit[int(pred[i])][int(n_target[i])]+=1
            p += probs[:].tolist()
    T_spadesuit = np.array(T_spadesuit)
    sum_matrix = np.tile(T_spadesuit.sum(axis = 1),(args.num_classes,1)).transpose()
    T_spadesuit = T_spadesuit/sum_matrix
    p = np.array(p)
    T_clubsuit = est_t_matrix(p,filter_outlier=True)
    T_spadesuit = np.nan_to_num(T_spadesuit)
    return T_spadesuit, T_clubsuit
```

Outline

Part-dependent Label Noise (PTD)

Motivation1: Humans perceive instances by decomposing them into parts. Annotators are therefore more likely to annotate instances based on the parts rather than the whole instances, where a wrong mapping from parts to classes may cause the instance-dependent label noise.

Motivation2: The noise of an instance depends only on its parts. We term this kind of noise as part-dependent label noise.

Xia et al. Part-dependent Label Noise: Towards Instance-dependent Label Noise. NeurIPS, 2020

Method: PTD

Since instances can be approximately reconstructed by a combination of parts, *we approximate the instance-dependent transition matrix for an instance by a combination of the transition matrices for the parts of the instance*.

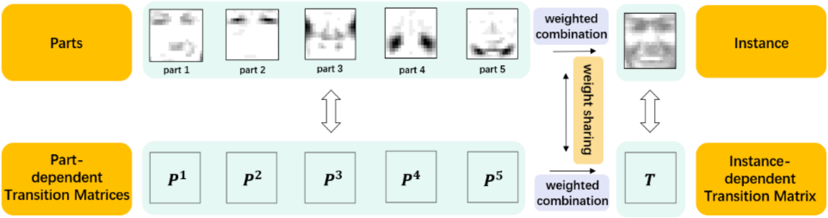


Figure 1: The proposed method will learn the transition matrices for parts of instances. The instance-dependent transition matrix for each instance can be approximated by a weighted combination of the part-dependent transition matrices.

Xia et al. Part-dependent Label Noise: Towards Instance-dependent Label Noise. NeurIPS, 2020

The parts-based representation learning:

$$\min_{W \in \mathbb{R}^{d \times r}, \boldsymbol{h}(\boldsymbol{x}_i) \in \mathbb{R}_+^r, \|\boldsymbol{h}(\boldsymbol{x}_i)\|_1 = 1, i=1,\ldots,n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - W\boldsymbol{h}(\boldsymbol{x}_i)\|_2^2$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ is data matrix, W is the matrix of parts (each column of W denotes a part of the instances) and the $\boldsymbol{h}(\boldsymbol{x}_i)$ denotes the combination parameters to reconstruct the instance $\boldsymbol{x}_i$.

We could identify the part-dependent transition matrices by assuming that the parameters for reconstructing the instance-dependent transition matrix are identical to those for reconstructing an instance:

$$T(\boldsymbol{x}) \approx \sum_{j=1}^{r} \boldsymbol{h}_j(\boldsymbol{x}) P^j$$

Xia et al. Part-dependent Label Noise: Towards Instance-dependent Label Noise. NeurIPS, 2020

Instance- and Label-Dependent Noise (ILN)

Let $\boldsymbol{x}^i$ be an anchor point of the i-th class. We have:

$$\Pr\left(\bar{Y}=j \mid X=\boldsymbol{x}^i\right)=\sum_{k=1}^{c} \Pr\left(\bar{Y}=j \mid Y=k, X=\boldsymbol{x}^i\right) \Pr\left(Y=k \mid X=\boldsymbol{x}^i\right)=T_{ij}\left(\boldsymbol{x}^i\right)$$

If the instance-dependent transition matrix and combination parameters are given, learning the part-dependent transition matrices is a convex problem:

$$\min_{P^1,\ldots,P^r \in [0,1]^{c\times c}} \sum_{i=1}^{c} \sum_{l=1}^{k} \left\| T_{i:}\left(\boldsymbol{x}_l^i\right) - \sum_{j=1}^{r} \boldsymbol{h}_j\left(\boldsymbol{x}_l^i\right) P_{i:}^j \right\|_2^2,$$
$$\text{s.t. } \left\| P_{i:}^j \right\|_1 = 1, i \in \{1,\ldots,c\}, j \in \{1,\ldots,r\},$$

where $\left(\boldsymbol{x}_1^i,\ldots,\boldsymbol{x}_k^i\right)$ are k anchor points of i-class.

Xia et al. Part-dependent Label Noise: Towards Instance-dependent Label Noise. NeurIPS, 2020

Outline

Appendix—PAC Learning Framework

Definition

- c: A concept $c : \mathcal{X} \to \mathcal{Y}$ is a mapping from $\mathcal{X}$ to $\mathcal{Y}$.
- $\mathcal{C}$: A concept class $\mathcal{C}$ is a set of concepts we may wish to learn.
- $\mathcal{D}$: The unknown distribution $\mathcal{D}$ where the examples are independently and identically distributed (i.i.d).
- $S = (x_1, \ldots, x_m)$: a sample drawn i.i.d. from $\mathcal{D}$ with the labels $(c(x_1), \ldots, c(x_m))$ that are based on a specific target concept $c \in \mathcal{C}$ to learn.
- $\mathcal{H}$: a fixed set of possible concepts, called hypothesis set, that a learner wants to consider.
- h: a hypothesis $h \in \mathcal{H}$, where h_S means the hypothesis h_S is selected from $\mathcal{H}$ by using the labeled sample S.

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Appendix—PAC Learning Framework

Generalization Error

Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and an underlying distribution $\mathcal{D}$, the generalization error or risk of h is defined by:

$$R(h) = \mathop{\mathbb{P}}_{x \sim \mathcal{D}}[h(x) \neq c(x)] = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}\left[1_{h(x) \neq c(x)}\right]$$

where 1_{ω} is the indicator function of the event ω.

The generalization error of a hypothesis is not directly accessible to the learner since both the distribution $\mathcal{D}$ and the target concept c are unknown.

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Empirical Error

Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and a sample $S = (x_1, \ldots, x_m)$, the empirical error or empirical risk of h is defined by:

$$\widehat{R}_S(h \,|) = \frac{1}{m} \sum_{i=1}^{m} 1_{h(x_i) \neq c(x_i)}$$

The empirical error of $h \in \mathcal{H}$ is its average error over the sample S, while the generalization error is its expected error based on the distribution $\mathcal{D}$.

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

For a fixed $h \in \mathcal{H}$, *the expectation of the empirical error based on an i.i.d. sample S is equal to the generalization error:*

$$\begin{aligned}
\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\widehat{R}_S(h)\right] &= \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[1_{h(x_i)\neq c(x_i)}\right] \text{ (linearity of the expectation)} \\
&= \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[1_{h(x)\neq c(x)}\right] \text{ (sample is drawn i.i.d.)} \\
&= \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[1_{h(x)\neq c(x)}\right] \\
&= \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[1_{h(x)\neq c(x)}\right] \\
&= R(h)
\end{aligned}$$

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Appendix—PAC Learning Framework

PAC-Learning

A concept class $\mathcal{C}$ is said to be PAC-learnable if there exists an algorithm A:

$$\underset{S\sim\mathcal{D}^m}{\mathbb{P}}\left[R\left(h_S\right) \leq \epsilon\right] \geq 1-\delta$$

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Appendix—Markov's Inequality

Markov's Inequality

Let X be a random variable that takes only nonnegative values. Then, for any $a > 0$,

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a} \tag{5}$$

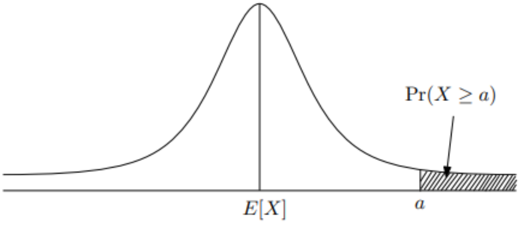


Figure 1: Markov's Inequality bounds the probability of the shaded region.

Lecture Notes CS:5360 Randomized Algorithms

Proof: We define a new random variable I by:

$$I = \begin{cases} 1, & \textit{if } X \geq a \\ 0, & \textit{otherwise.} \end{cases} \tag{6}$$

This is called an indicator variable for the event $X \geq a$.

When $X \geq a$, we have $I = 1$. Thus, $\frac{X}{a} \geq 1 = I$. If, on the other hand, $X < a$, then as both X and a are non-negative, we have $\frac{X}{a} \geq 0 = I$.

Therefore, in either case, we have the inequality $\frac{X}{a} \geq I$.

This implies the inequality of their expected values: $\mathbb{E}\left[\frac{X}{a}\right] \geq \mathbb{E}[I]$, i.e.,

$$\mathbb{E}\left[\frac{X}{a}\right] = \frac{\mathbb{E}[X]}{a} \geq \mathbb{E}[I] = 0 \cdot \mathbb{P}(0) + 1 \cdot \mathbb{P}(1) = \mathbb{P}(1) = \mathbb{P}(X > a) \tag{7}$$

Here, we complete the proof. □

Renjie Liao, Notes on Rademacher Complexity

Appendix—Chebyshev's Inequality

Chebyshev's Inequality

For any $a > 0$,

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq a) \leq \frac{Var[X]}{a^2} \tag{8}$$

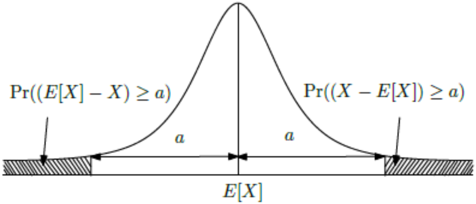


Figure 2: Chebyshev's Inequality bounds the probability of the shaded regions.

Lecture Notes CS:5360 Randomized Algorithms

Proof:

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq a) = \mathbb{P}((X - \mathbb{E}[X])^2 \geq a^2) = \mathbb{P}(Y \geq a^2) \tag{9}$$

where $Y = (X - \mathbb{E}[X])^2$. Note that Y is a non-negative random variable. Therefore, using Markov's Inequality, we have:

$$\mathbb{P}(Y \geq a^2) \leq \frac{\mathbb{E}[Y]}{a^2} = \frac{\mathbb{E}((X - \mathbb{E}[X])^2)}{a^2} = \frac{Var[X]}{a^2} \tag{10}$$

Here, we complete the proof. □

Lecture Notes CS:5360 Randomized Algorithms

Appendix—Hoeffding's Lemma

Hoeffding's Lemma

Let X be a random variable with $\mathbb{E}[X] = 0$ and $a \leq X \leq b$ with $b > a$. Then, for any $t > 0$, the following inequality holds:

$$\mathbb{E}\left[e^{tX}\right] \leq e^{\frac{t^2(b-a)^2}{8}} \tag{11}$$

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Renjie Liao, Notes on Rademacher Complexity

Appendix—Hoeffding's Lemma

Proof: Since $f(X) = e^{tX}$ is a convex function, for any $\alpha \in (0,1)$, we have $f(\alpha a + (1-\alpha)b) \le \alpha f(a) + (1-\alpha)f(b)$. Therefore, for $a \le X \le b$, let $\alpha = \frac{b-X}{b-a}$, then $X = b - \alpha b + \alpha a = \alpha a + (1-\alpha)b$, and we have:

$$\begin{aligned} e^{tX} &= f(X) \\ &= f(\alpha a + (1-\alpha)b) \\ &\le \alpha f(a) + (1-\alpha)f(b) \\ &= \frac{b-X}{b-a}e^{ta} + \frac{X-a}{b-a}e^{tb} \end{aligned} \tag{12}$$

Thus, using $\mathbb{E}[X] = 0$:

$$\mathbb{E}\left[e^{tX}\right] \le \mathbb{E}\left[\frac{b-X}{b-a}e^{ta} + \frac{X-a}{b-a}e^{tb}\right] = \frac{b}{b-a}e^{ta} - \frac{a}{b-a}e^{tb} \tag{13}$$

Stratos: Hoeffding, Azuma, McDiarmid

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Bt setting $e^{\phi(t)} = \frac{b}{b-a}e^{ta} - \frac{a}{b-a}e^{tb} = e^{ta}\left(\frac{b}{b-a} - \frac{a}{b-a}e^{t(b-a)}\right)$, we have:

$$\begin{aligned}\phi(t) &= \ln\left(e^{ta}\left(\frac{b}{b-a} - \frac{a}{b-a}e^{t(b-a)}\right)\right) \\ &= ta + \ln\left(\frac{b}{b-a} - \frac{a}{b-a}e^{t(b-a)}\right)\end{aligned} \tag{14}$$

For any $t > 0$, the first and second derivative of $\phi(t)$ are given below:

$$\phi'(t) = a - \frac{ae^{t(b-a)}}{\left(\frac{b}{b-a} - \frac{a}{b-a}e^{t(b-a)}\right)} = a - \frac{a}{\left(\frac{b}{b-a}e^{-t(b-a)} - \frac{a}{b-a}\right)} \tag{15}$$

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Stratos: Hoeffding, Azuma, McDiarmid

Appendix—Hoeffding's Lemma

$$\begin{aligned}
\phi''(t) &= \frac{-abe^{-t(b-a)}}{\left(\frac{b}{b-a}e^{-t(b-a)} - \frac{a}{b-a}\right)^2} \\
&= \frac{\beta(1-\beta)e^{-t(b-a)}(b-a)^2}{\left((1-\beta)e^{-t(b-a)} + \beta\right)^2} \\
&= \frac{\beta}{\left((1-\beta)e^{-t(b-a)} + \beta\right)} \frac{(1-\beta)e^{-t(b-a)}}{\left((1-\beta)e^{-t(b-a)} + \beta\right)}(b-a)^2 \\
&= \mu(1-\mu)(b-a)^2
\end{aligned} \tag{16}$$

where $\beta = \frac{-a}{b-a}$, and $\mu = \frac{\beta}{\left((1-\beta)e^{-t(b-a)}+\beta\right)}$. Note that, $\phi(0) = \phi'(0) = 0$ and $\phi''(0) = \mu(1-\mu)(b-a)^2$. Since $\mu(1-\mu)$ is upper bounded by $1/4$, we have $\phi''(0) = \mu(1-\mu)(b-a)^2 \leq \frac{(b-a)^2}{4}$.

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Stratos: Hoeffding, Azuma, McDiarmid

Thus, by the second order expansion of function $\phi(t)$, there exists $\theta \in [0, t]$, such that:

$$\phi(t) = \phi(0) + t\phi'(0) + \frac{t^2}{2}\phi''(\theta) \leq t^2 \frac{(b-a)^2}{8} \tag{17}$$

Therefore, we have

$$\mathbb{E}\left[e^{tX}\right] \leq e^{\phi(t)} \leq e^{\frac{t^2(b-a)^2}{8}} \tag{18}$$

Here, we complete the proof. □

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Stratos: Hoeffding, Azuma, McDiarmid

Appendix—Hoeffding's Inequality

Hoeffding's Inequality

Let $X_1, \cdots, X_m$ be independent random variables with X_i taking values in $[a_i, b_i]$ for all $i \in [m]$. Then, for any $\epsilon > 0$, the following inequalities hold for $S_m = \sum_{i=1}^{m} X_i$:

$$\begin{aligned} \mathbb{P}\left(S_m - \mathbb{E}[S_m] \geq \epsilon\right) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2}} \\ \mathbb{P}\left(\mathbb{E}[S_m] - S_m \geq \epsilon\right) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2}} \end{aligned} \tag{19}$$

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Renjie Liao, Notes on Rademacher Complexity

Proof: For any $t > 0$, we have

$$\begin{aligned}
\mathbb{P}\left(S_m - \mathbb{E}[S_m] \geq \epsilon\right) &= \mathbb{P}\left(e^{t(S_m - \mathbb{E}[S_m])} \geq e^{t\epsilon}\right) \\
&\leq \frac{\mathbb{E}\left[e^{t(S_m - \mathbb{E}[S_m])}\right]}{e^{t\epsilon}} \qquad \text{(Markov's Inequality)} \\
&= \frac{\mathbb{E}\left[e^{t\sum_{i=1}^{m}(X_i - \mathbb{E}[X_i])}\right]}{e^{t\epsilon}} \\
&\leq \frac{e^{\frac{\sum_{i=1}^{m} t^2(b_i - a_i)^2}{8}}}{e^{t\epsilon}} \qquad \text{(Hoeffding's Lemma)} \\
&= e^{\frac{\sum_{i=1}^{m} t^2(b_i - a_i)^2}{8} - t\epsilon}
\end{aligned} \tag{20}$$

In the last inequality, we apply Hoeffding's Lemma to each $X_i - \mathbb{E}[X_i]$ individually since $\mathbb{E}[X_i - \mathbb{E}[X_i]] = 0$. (Question? $X_i - \mathbb{E}[X_i] \in [a_i - b_i, b_i - a_i]$.)

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Renjie Liao, Notes on Rademacher Complexity

Since the above inequality holds for any $t > 0$, we can find the tightest bound as:

$$\begin{aligned}\mathbb{P}\left(S_m - \mathbb{E}[S_m] \geq \epsilon\right) &\leq \inf_{t>0} e^{\frac{\sum_{i=1}^{m} t^2 (b_i - a_i)^2}{8} - t\epsilon} \\ &= e^{\frac{-2\epsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2}}\end{aligned} \tag{21}$$

where the optimal $t^* = \frac{4\epsilon}{\sum_{i=1}^{m}(b_i - a_i)^2}$. Here, we complete the proof. □

Mehryar Mohri et al, Foundations of Machine Learning, Second Edition.

Renjie Liao, Notes on Rademacher Complexity

Appendix—McDiarmid's Inequality

Hoeffding's Inequality applies to sums of independent random variables. We will now develop its generalization to arbitrary real-valued functions of independent random variables that satisfy a certain condition.

Let X be some set, and consider a function $g : X^n \to \mathbb{R}$. We say that g has bounded differences if there exist nonnegative numbers $c_1, \cdots, c_n$, such that:

$$\sup_{x \in X} g(x_1, \cdots, x_{i-1}, x, x_{i+1}, \cdots, x_n) - \inf_{x \in X} g(x_1, \cdots, x_{i-1}, x, x_{i+1}, \cdots, x_n) \leq c_i \tag{22}$$

for all $i = 1, \cdots, n$ and all $x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_n \in X$. In words, if we change the i-th variable while keeping all the others fixed, the value of g will not change by more than c_i.

Maxim Raginsky: Concentration inequalities.

McDiarmid's Inequality

Let $X^n = (X_1, \cdots, X_m) \in \mathcal{X}^n$ be an n-tuple of independent X-valued random variables. If a function $g : \mathcal{X}^n \to \mathbb{R}$ has bounded differences, as in (22), then, for all $\epsilon > 0$,

$$\begin{aligned} \mathbb{P}\left(g(X^n) - \mathbb{E}[g(X^n)] \geq \epsilon\right) &\leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^m c_i^2}} \\ \mathbb{P}\left(\mathbb{E}[g(X^n)] - g(X^n) \geq \epsilon\right) &\leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^m c_i^2}} \end{aligned} \tag{23}$$

The End!