7. Appendix

In the appendix, we first supplement the content mentioned in the Paper. According to the order of contents in the Paper, we introduce the following five parts:

- (1) Convergence of Our Difficulty Model;
- (2) Settings for Binary Long-Tailed MNIST;
- (3) Simulations for Learning Process of Instances;
- (4) Datasets in Generality Experiments;
- (5) Results on iNaturalist 2019;
- (6) Limitations and Future Works.

Besides, we present the implementation details of our experiments in Sec. 7.8.

7.1. Convergence of Our Difficulty Model

In this section, we prove the convergence of our difficulty model (*i.e.*, Eq. (4)). First, we assume that the model Net is converged when $t \to \infty$. Therefore, for any instance z_i ($z_i \in \mathcal{D}, i = 1, \ldots, N$), we have $du_{i,t} \to 0$ and $dl_{i,t} \to 0$.

$\therefore ||\vec{\boldsymbol{d}_{i,t}}|| \to 0$.

$\because ||\vec{\boldsymbol{D}_{i,t}}|| = ||\vec{\boldsymbol{D}_{i,t-1}} + \vec{\boldsymbol{d}_{i,t}}|| \leq ||\vec{\boldsymbol{D}_{i,t-1}}|| + ||\vec{\boldsymbol{d}_{i,t}}|| \leq ||\vec{\boldsymbol{D}_{i,t-1}}||$.

$\because \vec{\boldsymbol{D}_{i,t-1}} \cdot \vec{\boldsymbol{d}_{i,t-1}} \geq 0$,

$\therefore ||\vec{\boldsymbol{D}_{i,t}}|| = ||\vec{\boldsymbol{D}_{i,t-1}} + \vec{\boldsymbol{d}_{i,t}}|| \geq ||\vec{\boldsymbol{D}_{i,t-1}}||$.

$\therefore ||\vec{\boldsymbol{D}_{i,t-1}}|| = ||\vec{\boldsymbol{D}_{i,t}}||$, when $t \to \infty$.

In conclusion, the vector $\vec{\boldsymbol{D}_{i,t}}$ is converged by norm $||\cdot||$.

7.2. Settings for Binary Long-Tailed MNIST

The simulation experiments are carried out on the task of the binary classification for odd and even numbers in the long-tailed MNIST.

MNIST is a popular dataset of handwritten dight recognition. It has 60000 training cases and 10000 testing instances. each sample has 784 features to express a 28*28 picture. Labels 0-9 present the Arabic numbers 0-9. MNIST is a classic and simple dataset without class-level imbalance.

According to the construction method of Long-tailed CIFAR, minority classes are created with a given imbalance ratio by under-sampling. Specifically, the training instances per class are reduced, according to an exponential function $n = n_i \mu^i$, where i is the class index (indexed from 0), n_i is the original number of class i and $\mu \in (0, 1)$. And we define the imbalance ratio as $\frac{n_{\max}}{n_{\min}}$. The sampling rate of each digit with different imbalance ratios is illustrated in Fig. 5.

In particular, simulation experiments are done on the binary classification of evens(digits 0,2,4,6,8) and odds(digits 1,3,5,7,9) on the long-tailed MNIST. In the training process, classification models do not know the digit labels of pictures.

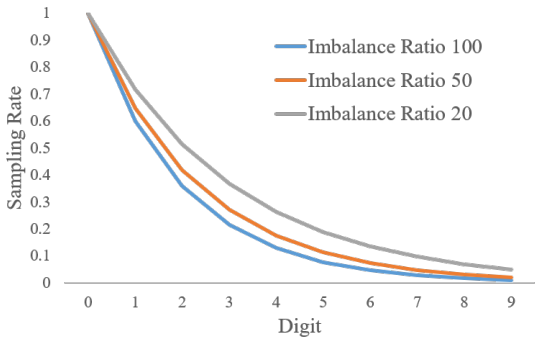


Figure 5. Sample Rate of Each Digit for Long-Tailed MNIST.

7.3. Simulations for Learning Process of Instances

In the Paper, we conducted simulations for instances with different unlearning probabilities. In this section, we introduce such simulations in detail. In order to observe the relationship between the unlearning frequency and our instance difficulty model more generally. We model the learning (*i.e.*, training) process of a certain instance as a random walk process. The instance walks toward the learning direction, but it sometimes walks toward the back (*i.e.*, unlearning direction). Moreover, the probability of backward walking is different for different instances.

Specifically, follow the formulation in Sec. 3.1, we further set $p_i = Net(\boldsymbol{x_i}) = softmax(h_i)$, where $h_i = (h_{i,1}, \ldots, h_{i,k})$ represents the preference(or called logits) of different k classes for instance z_i. In the simulation training process, h_{i,y_i} is decreased by α with a probability of β, h_{i,y_i} is increased by α with a probability of $1 - \beta$, where α denotes the walking pace length, β denotes the unlearning probability.

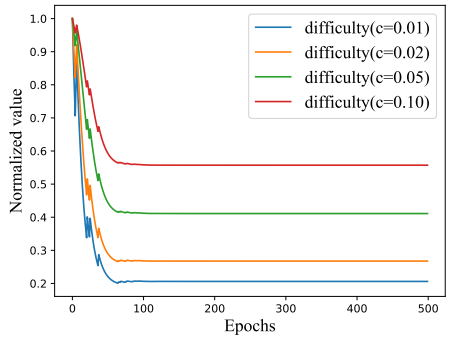
For "Easy" case, $\beta = 10\%$. For "Normal" case, $\beta = 20\%$. For "Hard" case, $\beta = 40\%$. Then set k to 10 and α to 0.1. Our simulation results in Fig. 4 will be reproduced.

By the way, we present the effect of parameter c in Eq. (4) by Fig. 6. We can see that c controls the sensitivity of our difficulty model for the variation of predictions. A larger c leads to smaller fluctuation of instance difficulty.

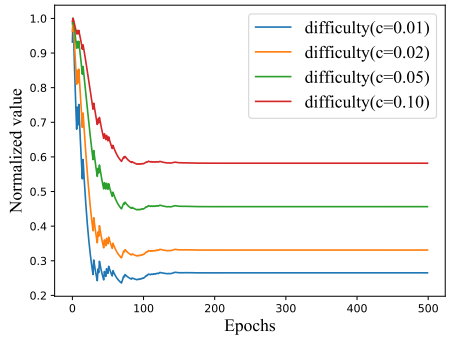
7.4. Datasets in Generality Experiments

We adopt 10 tiny classification datasets in the UCI machine learning repository and 1 large-scale dataset.

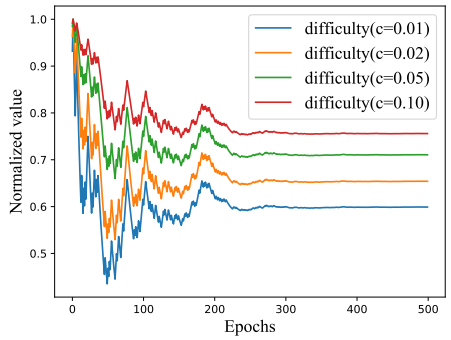
The tiny datasets are Wine, Statlog.(Heart), Wall-Following Robot Navigation Data, Ecoli, Glass Identification, Balance Scale, Iris, Seeds, Contraceptive Method Choice, Connectionist Bench(Sonar, Mines vs. Rocks). In the Paper, we call Statlog.(Heart) as Heart, call Wall-Following Robot Navigation Data as Robot, call Glass Iden-



(a) Easy



(b) Normal



(c) Hard

Figure 6. Difficulty with Different c in Training.

tification as Glass, call Balance Scale as Balance, call Contraceptive Method Choice as CMC and call Connectionist Bench(Sonar, Mines vs. Rocks) as Sonar. We randomly choose 80% instances of each dataset as the training data and the rest part as the test data.

The large-scale dataset is iNaturalist 2019. It is a real-world long-tailed dataset that contains 265,213 training images from 1010 classes.

Table 6. Accuracy % on iNaturalist 2019. ResNet-m denotes the ResNet model with m layers.

Networks	ResNet-50	ResNet-101
Base Model	70.19	72.84
Focal Loss	70.24	73.13
Class-Balance Loss	70.83	73.17
Our Method	**71.08**	**73.32**

7.5. Results on iNaturalist 2019

We compare our method with baselines, which adjusts the data distribution on the class level, on a large-scale dataset named iNaturalist 2019, whose results are shown in Tab. 6.

7.6. Limitations and Future Works

In this section, we discuss the limitations of our method in two aspects: (1) Training speed; (2) Influences of Noises.

Training Speed. In the actual training process, our method needs to infer the entire training set when updating the sampling weights. If the number of weight updates is greatly reduced, the performance of our method will decrease. Therefore, our method is not applicable in some scenes with high requirements for training speed. However, based on the analysis of the training process in Sec. 3.3, there are many variants of difficulty modeling. How to estimate difficulty faster and more accurately is an interesting and challenging problem in future works.

Influences of Noises. Our method is easily affected by noises that denote the instances with wrong labels. Noises are extremely difficult to be learned by the model. If the weights of noises are set too high, the learning of the model will be terribly affected. In fact, our method already has a certain ability to recognize noises since noises usually are the most difficult instances. We can simply use truncation to mitigate the effects of noises. Some details and experiments are shown in the next section. However, distinguishing right-labeled difficult instances from noises is still a great challenge. We can incorporate noise detection technology into the definition of difficulty in future works.

7.7. Discussion for Noise Influences

For a classification dataset, the class labels of instances are usually manually annotated. Therefore, the labels of some instances may be wrong. We call such wrong labeled instances as noises. Obviously, noises are difficult to learn by the models. Empirically, our method tends to give such noises higher sampling weights, which is harmful to the training process. So when the noise problem in the dataset is serious, the performance of our method may decrease.

In fact, our method has the ability to detect noise, because noises usually are assigned the highest weights by

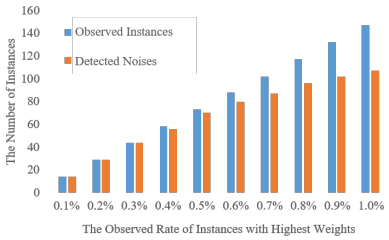


Figure 7. Noise Detection by The Weights of Our Method

Algorithm 2 Noise Resistance Strategy

Input: weight ω, noise rate γ, data size N
$n \leftarrow Integer(\gamma * N)$
if $n > 0$ **then**
  $I \leftarrow$ indices of Top-m values in ω
  $\omega_I \leftarrow 0$
end if
$\omega_i \leftarrow \frac{\omega_i}{\sum_{j=1}^{N} \omega_j}$ for $i = 1, \ldots, N$
Return ω

our method. Specifically, we discuss this phenomenon in Sec. 7.7.1. In light of this, in actual training, we adopt a simple technique to help the model resist noises. We introduce such a technique in Sec. 7.7.2.

7.7.1 Noise Detection Experiments

To test the ability of our method for detecting the noises, we randomly choose 1% instances from long-tailed MNIST and change their labels to make some noise instances. After training, we sort the instances in descending order of weight and then observe the top part of the rank. As shown in Fig. 7, we can see that noises usually are assigned the highest weights by our method. In particular, in the top-0.01% to top-0.03%, almost all instances are noises.

7.7.2 Noise Resistance Strategy

Assume that the noise rate is γ, for the entire dataset, the total number of noises is n ($n = N * \gamma$). Since noises usually have higher sampling weights, we treat the top-n instances with the largest weight as noises. Specifically, we call the top-n part of the rank as the abandoned area. In the training process, the sampling weights of instances in the abandoned area are reset as 0. The algorithm is presented in Algorithm 2.

In the early stage of training, it is difficult for our method to directly capture the noises in the abandoned area. However, with the generalization ability of the model, even though normal instances in the abandoned area are not trained, they can gradually be learned and escape from the abandoned area due to the training of the outside instances from the same class. While the noises are often isolated, and they are incompatible with other normal instances in the training of the model. When the noises fall into the abandoned area, they are difficult to be rescued by the learning of other instances.

However, there may be correctly labeled instances whose learning difficulties are similar to the noises. Our method can not distinguish them from noise and may abandon them. Therefore, our current noise resistance strategy can not completely solve the problem of noises. In future works, we hope to combine some more effective noise recognition methods with our methods to further improve the generality of our methods.

7.8. Reproducibility

In this section, we introduce the implementation details of our experiments.

7.8.1 Environments

Experiments are set in the python environment, and the main information of environment is:

- Python version: 3.6,
- Pytorch version: 1.7.1,
- torchvision version: 0.8.2,
- GPU: Tesla K80,
- Cuda version: 9.2.
- Random seed: 0

7.8.2 Parameters

The models are trained by using SGD optimizer with momentum 0.9. Specially,

- For experiments on long-tailed CIFAR: $c = 10$, $\gamma = 0.001$.
- For simulation experiments: $c = 1$, $\gamma = 0$.
- For experiments on iNaturalist: $c = 10$, $\gamma = 0.001$.
- For experiments on UCI datasets: $\gamma = 0$. In particular, we set $c = 0.1$ for ecoli and heart, set $c = 1$ for sonar, balance, glass, and iris, set set $c = 10$ for cmc, robot, seeds and wine.

Here, γ denotes the parameter for the noise rate which is introduced in Sec. 7.7. Usually, we do not know how many noise instances there are, so we optimize the final result with adjusting γ. In the actual tuning, we generally set $0 \leq \gamma \leq 0.001$.