Training More Accurate Neural Network by Clustering Learning Trajectories

Zhangyu Wang, Zhiqi Huang
University of Massachusetts, Amherst
140 Governors Dr., Amherst, MA 01003
{zhangyuwang, zhiqihuang}@cs.umass.edu

Abstract

Sampling and re-weighting algorithms can help neural networks to overcome the overfitting on corrupted labels, lead to better learning accuracy and training efficiency. In this paper, we propose an novel data-driven algorithm that sample from training trajectories formed by past iterations in stochastic gradient descent (SGD), and tuning sample weight based on cluster purity estimation with respect to clean validation set. Comparing with existing methods, which assign weight to each training instance, we find that the cluster of training trajectories has the representation power of training data uncertainty. The experimental results show that our method improve the network performance on data set with different noise levels.

1 Introduction

In a general learning problem, materials usually have different learning curves for a particular learner. This phenomenon not only applies to human but also holds for some machine learning methodologies. *Curriculum learning* [3, 2] has shown that learning from easier instances first can improve the neural networks training. There could be a *priori* which samples are easy. Otherwise, examples with lower loss in early training can be inferred to be easier. By decreasing the weight of difficult examples in loss function, neural networks may become more robust to corrupted labels and outliers [15]. But such strategy often slows down the training process because easier examples barely contribute to the gradients, and the current model has already fully exploited the information on these samples.

On the contrary, focusing on harder instances, known as *hard example mining* [19], can accelerate and stabilize SGD on clean data. When training, difficulty of an example can be evaluated by its loss [8, 14, 19] or be corresponding to the magnitude of its gradient [1, 25, 6, 7]. However, without knowing how noisy our training set is, this strategy sometimes encounters the issue of overfitting because it keeps focusing on the noisy part of the data mistakenly [24].

In the literature, we can see that predefined curriculum cannot be used as a general strategy applying to all datasets. Especially, in the real world learning tasks, sometime it is hard to quantitatively evaluate the noise level of a dataset. Moreover, the "noise" might come from multiple sources [16]. For example, the distribution mismatch between training set and underlying true distribution. The common phenomena of such noise is the imbalance in dataset. Another type of noise is corrupted label, which means a proportion of the training label is incorrect. In practice, there could be a mix of these two type of noises which lead model with predefined curriculum to a sub-optimal solution. Motivated by this practical issue, this paper proposes a new data-driven method to sample and re-weight training examples that can improve neural networks learning accuracy with robustness and less computational burden.

COMPSCI 689 Project Report, University of Massachusetts Amherst, 2018

Intuitively, suppose training instances can be grouped by their type and degree of difficulty, then, by examining the learning procedure, a model should have distinct learning trajectory on each type and degree of difficulty. All starting from initialization, some examples will be quickly learned with high confidence, while the difficult/noisy examples will remain uncertain to the model. And with the training goes on, the learning trajectories between hard and noisy examples will begin to separate, because the hard points are learnable and noises can only degrade the model.

In some literature about data pruning, we see that by classifying dataset into categories and dropping uncertain parts of each category model learns more robustly, indicating that there are underlying features of data points that reflect its contribution to learning. Different from existing re-weighting approaches which try to re-weight each training instance based on either loss or gradient direction, our method first use the learning trajectory to cluster training data. Therefore, despite the size of dataset, there will only exist handful of groups clustered by their learning curves. This approach significantly reduce the computation complexity. Then leveraging an unbiased validation set, we are able to update the weights on training data clusters by estimating the overall cluster purity effectively. Because of the consistence of trajectory clustering, we could perform validation at every training iteration to dynamically update the weight of each group. We present experiments on MNIST and CIFAR 10 with different noise settings. The results show that our approach makes neural networks more robust without prior knowledge of noise, and significantly reduce the generalization error.

2 Related work

As (deep) neural networks become more widespread, noises in dataset and randomness in (mini-batch) SGD sometimes slow down the optimization. In addition to various regularizers, methods to sample and re-weight training set have become a popular solutions. Inspired by the idea of importance sampling from statistics, some studies accelerate SGD by weighting each class differently [7] or weighting each sample differently [8, 25, 14, 1, 6, 19], Many of these methods adopt importance sampling by focusing on hard examples. That is, optimization will focus on a predefined type of points during training. The criteria could based on the value of loss [8, 14, 19] or according to the magnitude of gradients [6].

Hard examples are not always preferred when there exists noisy in dataset. Since the importance sampling cannot tell the difference between noises and hard examples, for robustness, training instances with high loss are actually down sampled. Sampling and re-weighting strategies are closely related with *curriculum learning* (CL) [3], which showed that learning could be benefit when starting from easier instances. CL has been applied to a variety of problems, such as computer vision[21] and natural language processing [22, 20]. A common CL approach is to predefined a curriculum. In *self-paced learning* [12], examples are weighted by training loss, encouraging learning easier examples first. After that, many definition of curriculum are proposed. Fan et al [5]. proposed to use predefined sample weighting schemes as an implicit way to define a curriculum. Previous works have shown that predefined curriculum are useful in overcoming noisy labels. Nevertheless, these methods are concentrating to one or multiple type of noises, they more or less need clue on the dataset noise. Moreover, it is still an unsolved challenge to balance the easy and hard training examples to facilitate training while remaining robust to outliers.

In order to re-weight examples correctly without having prior knowledge of noise, some data-driven methods are proposed recently. *Active bias* [4] measures the variance of prediction probability [10, 23], comparing with decision threshold. And *MentorNet* [9] built a another neural network as mentor to guide the training of base neural networks as a student. It is worth mention that Ren et al. [17] discuss the idea of re-weighting instances against the gradient direction of an unbiased validation set, but their proposed method focus on re-weighting each instance at every training iteration, which according to their experiments, could be time-consuming on large dataset. Therefore, they rectify the output and choose 0-1 constraint on weight calculation to maintain efficiency. We think such approach is aggressive and inconsistent between iterations. Taking advantage of an unbiased validation set has been explored in recent meta-learning studies. Use the validation set as an oracle, our method assign equal weight to examples within a same group. The groups in training set are clustered by instance's historical predicted probability of the correct class. Only considering representative instances of training set, we could put more efforts on assigning a comprehensive weight from validation set.

3 Method

In this section, we propose our method in two parts: first, the learning trajectory study based on sample uncertainty, then the re-weighting algorithm based on purity estimation on validation set. Finally, we will demonstrate an example algorithm combining with multilayer perceptron (MLP). From the model design perspective, the base neural network could be viewed as a core in our system, where the information in data are extracted and learned. And our method can quick evaluate the current model capacity and prepare materials for the next round by adjusting cluster weights in training set. With such flexible structure, our method could fit in different type of neural networks.

3.1 Pre-processing: Cluster learning trajectory by GMM

Let $\{(x_i, y_i), 1 \leq i \leq N\}$ be the training set, N is the number of pairs in it, and there is a small unbiased and clean validation set $\{(x_i^v, y_i^v), 1 \leq i \leq M\}$, where $M \ll N$. At training iteration t, H_{t-1} is the history of correct prediction probability which stores all $p(y_i|\mathbf{x}_i)$ when $\mathbf{x}_i$ is selected to train the network before current iteration t. For simplicity, assume $p_{ij} = p_j(y_i|\mathbf{x}_i)$ is the probability of classifying sample i into its correct class y_i at j's iteration, where $1 \leq j \leq t$. Then all samples at current iteration, we have a matrix of probability:

$$\mathbf{R}_t = \{p_{ij} \mid 1 \leq i \leq N,\ 1 \leq j \leq t-1\} \tag{1}$$

According to our setting, $\mathbf{R}_t$ is a $N \times (t-1)$ matrix. The jth column, $\mathbf{r}_{\cdot j}$, is the set of probabilities that model predicts each training instance to its correct class at jth iteration. Usually with the decision threshold, we could calculate the training accuracy from $\mathbf{r}_{\cdot j}$. Now consider ith row of this matrix, $\mathbf{r}_{i\cdot}$, it has all history probabilities of classifying sample i into its correct class since the beginning. Such row-wise data could be used to simulate the networks learning trajectory on each sample. The rest of this section will focus on how to use Gaussian Mixture Model (GMM) to cluster training set based on their learning trajectories.

Suppose at iteration t, there are K clusters in the training set, in mixture model framework, each learning trajectory, $\mathbf{r}_{i\cdot}$ belongs to cluster k with probability π_k, where

$$\sum_{k=1}^{K} \pi_k = 1 \text{ and } \pi_k \geq 0 \tag{2}$$

The entire set $\mathbf{R}_t$ is, therefore, modeled by a mixture of these distributions. Then the probability density function (pdf) of $\mathbf{R}_t$ in the finite mixture form is

$$P(\mathbf{r}_{i\cdot}|\Theta) = \sum_{k=1}^{K} p_k(\mathbf{r}_{i\cdot}|\theta_k)\pi_k \tag{3}$$

where the parameter $\Theta = \{\pi_1, \cdots \pi_K, \theta_1, \cdots \theta_K\}$, π is the mixture indicator and θ denotes the vector of all unknown parameters associated with the parametric forms adopted from mixture components $p_k(\cdot)$. For $\mathbf{R}_t = \{\mathbf{r}_{1\cdot}, \cdots, \mathbf{r}_{N\cdot}\}$, we wish to find the Θ that maximum the $P(\mathbf{R}_t|\Theta)$. This is know as the maximum likelihood estimates of Θ. In order to estimate Θ, it is typical to introduce the marginal log likelihood function defined as follows:

$$\mathcal{L}(\Theta) = \log P(\mathbf{R}_t|\Theta) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} p_k(\mathbf{r}_{n\cdot}|\theta_k)\pi_k \right) \tag{4}$$

The numerically optimizing this likelihood function is hard because of the logarithm of sum over K. The learning would decompose nicely over the mixture model if we could only take the sum over K outside the log. To form a better objective function, we introduce an auxiliary set of multinoulli probability distributions $q_n(K = k) = \phi_{kn}$ into the marginal log likelihood and apply the Jensen's inequality:

$$\mathcal{L}(\Theta) = \log P(\mathbf{R}_t|\Theta) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \frac{\phi_{kn}}{\phi_{kn}} p_k(\mathbf{r}_{n\cdot}|\theta_k)\pi_k \right) \geq \sum_{n=1}^{N} \sum_{k=1}^{K} \phi_{kn} \log \left(\frac{p_k(\mathbf{r}_{n\cdot}|\theta_k)\pi_k}{\phi_{kn}} \right) \tag{5}$$

The last term of Eq. 5 lower bound of Eq. 4. Sometime it called Q function of the mixture model. And it could be reformed in to the expected complete log likelihood and entropy of q_n.

$$\begin{aligned} Q(\Theta) &= \sum_{n=1}^{N}\sum_{k=1}^{K} \phi_{kn} \log\left(\frac{p_k(\mathbf{r}_{n\cdot}|\theta_k)\pi_k}{\phi_{kn}}\right) \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} \phi_{kn}(\log p_k(\mathbf{r}_{n\cdot}|\theta_k) + \log \pi_k - \log \phi_{kn}) \\ &= \sum_{n=1}^{N}\left(\mathbb{E}_{q_n}[\log p_k(\mathbf{r}_{n\cdot}|\theta_k)] + \mathbb{H}(q_n)\right) \end{aligned} \tag{6}$$

Remember we are using Gaussian distribution for mixture indicator which gives:

$$p_k(\mathbf{r}_{n\cdot}|\theta_n) = \mathcal{N}(\mathbf{r}_{n\cdot}, \theta_k) \text{ where } \theta_k = [\mu_k, \sigma_k] \tag{7}$$

The Eq. 6 could be optimized by Expectation-Maximization (EM) algorithm. At τ's round, E-step for updating ϕ_{kn} is:

$$\phi_{kn}^{\tau} \leftarrow \frac{\pi_k^{\tau-1}\mathcal{N}(\mathbf{r}_{n\cdot}, \mu_k, \sigma_k)}{\sum_{k'=1}^{K}\pi_{k'}^{\tau-1}\mathcal{N}(\mathbf{r}_{n\cdot}, \mu_{k'}, \sigma_{k'})} \tag{8}$$

M-Step update parameters:

$$\pi_k^{\tau} \leftarrow \frac{1}{N}\sum_{n=1}^{N}\phi_{kn}^{\tau} \quad \mu_k^{\tau} \leftarrow \frac{\frac{1}{N}\sum_{n=1}^{N}\phi_{kn}^{\tau}\mathbf{r}_{n\cdot}}{\sum_{n=1}^{N}\phi_{kn}^{\tau}} \quad \sigma_k^{\tau} \leftarrow \frac{\sum_{n=1}^{N}\phi_{kn}^{\tau}(\mathbf{r}_{n\cdot}-\mu_k^{\tau})^T(\mathbf{r}_{n\cdot}-\mu_k^{\tau})}{\sum_{n=1}^{N}\phi_{kn}^{\tau}} \tag{9}$$

Remark 1: At the end of tth iteration, we evaluate the validation set, then $\mathbf{R}_t$ append by 1 column and become $\mathbf{R}_{t+1}$. Instead of re-run the GMM algorithm completely, we could simply extend the $\theta_k = [\mu_k, \sigma_k]$ by 1 column using random initial and inherit the previous clustering results.

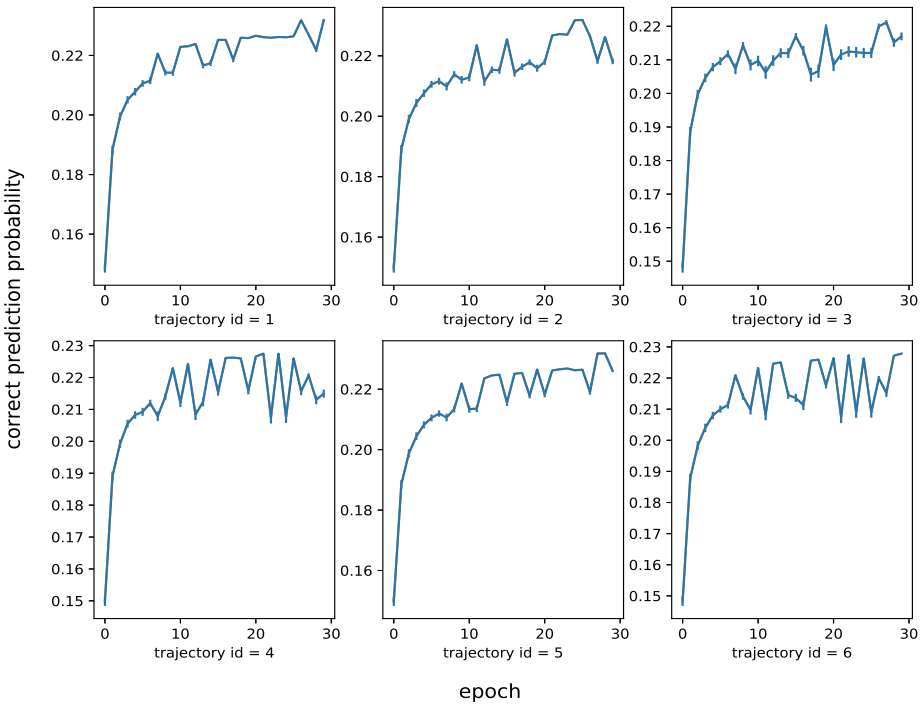

Figure 1: GMM with cluster = 6 trained on MNIST data after 30 epochs

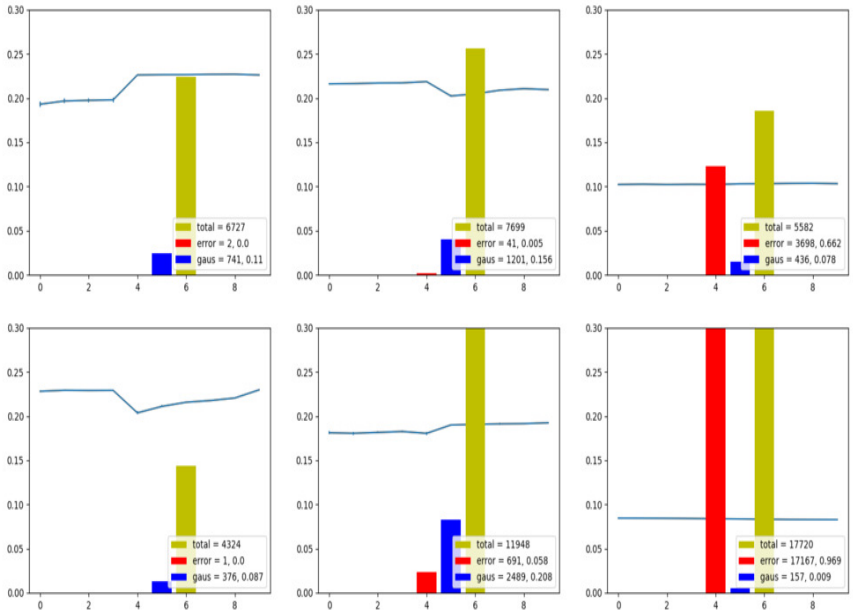

Figure 2: Data point types within each cluster

Remark 2: Fig 1 shows the clustering results on MNIST data after 30 iterations with number of cluster = 6. Visually, we can find the patterns are distinct from each other, whcih means the instances they representing has the different contributions to the model.

3.2 Post-processing: Re-weight by purity estimation

In order to strengthen the influence of cleaner data points and weaken the influence of dirtier data points, we choose to over-sample the cleaner ones and under-sample the dirtier ones. Unlike in [17], where the author kept close surveillance on the SGD behavior of every single data point in the training set at a cost of high computational complexity (with 2 extra forward passes and 2 extra backward passes and 2 nested loops of optimization), we find that due to the stably high quality of clustering results (see Fig 2), we don't have to evaluate individual data points but the entire cluster to decide whether to over-sample or under-sample certain data points. Here we assume that points within each cluster have similar contribution to the learning process, and experiments solidly support our assumption.

Remark 3: Fig 2 shows the clustering performance of our learned GMM in Fig 1. In each sub-graph, bars denote the amount of data points classified into this cluster, and the curve denotes the learned feature trajectory of this cluster. The red bar shows the amount of label error points, the blue bar shows the amount of Gaussian noised points, and the yellow bar shows the total amount of data points in this cluster. We can see clearly that most error labels are cleanly classified into 2 distinct clusters, while Gaussian noised points are not separable from clean points.

Every epoch before training starts, there is a sampling step. We sample a fixed-size subset of training data proportional to the sampling weights, i.e., if cluster 1 has a sampling weight of 0.5, then 50% data points in the subset will come from cluster 1. Then the subset will be passed to the training step to do ordinary mini-batch SGD. We can also do such sampling every batch, but it's of course more computationally expensive.

The key strategy we adopt to update the sampling weights is cluster purity estimation. Every epoch before sampling, we need to first estimate how clean each cluster is (with the help of validation

set), assign them scores, normalize the scores, and finally use the normalized scores to update the sampling weights. Therefore, our re-weighting algorithm consists of two supersteps: purity estimation and weight update. By selecting different purity estimation and weight update strategies we can construct different re-weighting algorithms. In this section we will analyze 3 re-weighting algorithms theoretically and evaluate their performance on noisy datasets later in Section 4.

Naive Screening Re-weighting Algorithm: This algorithm is very intuitive. At the beginning of every epoch, we run a forward pass on the small validation set to obtain the **screening loss**. Then we randomly sample batches from each cluster and run forward passes to obtain their losses. We set the unnormalized scores of each cluster as 1 if its loss is smaller than or equal to the screening loss, or 0 if greater. Finally we normalize the scores and set the sampling weights to be the normalized cluster scores.

Algorithm 1: Naive Screening Re-weighting

Input: Training data clusters $C_1, C_2, \cdots, C_K$ containing training pairs (X_K, y_K), validation set $V = (X_v, y_v)$, network NN, batch size M.
Output: Updated sampling weights $\mathbf{w_s} = (w_{s,1}, \cdots w_{s,K})$

```
Scores = (s_1, s_2, ··· s_K) ;
ScreenLoss ← Loss(NN(X_V), y_V) ;
for k ∈ K do
    X_kM, y_kM ← Sample(C_k, M) ;
    Loss_k ← Loss(NN(X_kM), y_kM) ;
    if Loss_k ≤ ScreenLoss then
        s_k ← 1 ;
    else
        s_k ← 0 ;
    end
end
w_s ← Normalize(Scores)
```

This algorithm is no doubt very computationally efficient. While it requires only minimal extra computation, it works not bad when clustering quality is guaranteed, that is, when there are distinctly clean and dirty clusters. Its performance may deteriorate if there are clusters that contain a moderate proportion (for example, 30%) of error label points, because such clusters may either be classified as good if the model is too poorly learned (that is, it behaves very badly on validation set and makes the screening loss very high) so that an non-negligible amount of error points will leak into our training subset, or be completely excluded from training process if the model is too well learned (that is, it has a very low screening loss that only very clean clusters may yield a lower loss) so that the clean data points in this cluster will never been seen by the model. In conclusion, this algorithm relies heavily on the clustering results and doesn't have any self-adaption capability.

Gaussian Confidence Screening Re-weighting Algorithm: This algorithm is based on the knowledge that during training, neither too easy nor too hard examples are good for SGD. In our setting, easy means clean and hard means with Gaussian noise (because they may serve as model regularization), while label error data points are proven to be always harmful to learning. Therefore, what we want to do is to over-sample the clusters with mostly clean data points and a moderate portion of Gaussian noised data points, while screening error label data points as clearly as possible.

As is with Naive Screening Algorithm, at the beginning of every epoch we again obtain the validation loss and the losses of each cluster batch. Then, we assume that there is a normal distribution on cluster losses. Its mean μ_n is the validation loss after finishing the current epoch's SGD, and its standard variance σ_n the standard variance of the losses of each cluster also after the current epoch. We estimate the mean and standard variance by computing the mean and variance of the validation loss and cluster losses that are smaller than or equal to the validation loss. Then we assign scores to each cluster with the following formula:

$$P(x) \sim \mathcal{N}(\mu_n, \sigma_n^2) \tag{10}$$

$$S(loss) = max(0, \frac{1}{2} - P(x < loss - \epsilon\sigma_n)) \tag{11}$$

where ϵ is a hyperparameter, deciding how tolerant the screening algorithm is to clusters with losses greater than μ_n. Finally we normalize the scores and set the sampling weights to be the normalized cluster scores.

Algorithm 2: Gaussian Confidence Screening Re-weighting

Input: Training data clusters $C_1, C_2, \cdots, C_K$ containing training pairs (X_K, y_K), validation set $V = (X_v, y_v)$, network NN, batch size M.
Output: Updated sampling weights $\mathbf{w_s} = (w_{s,1}, \cdots w_{s,K})$

$ValidationLoss \leftarrow Loss(NN(X_V), y_V)$;
$CandidateSet \leftarrow \emptyset$;
$Losses = (l_1, l_2, \cdots l_K)$;
$Scores = (s_1, s_2, \cdots s_K)$;
for $k \in K$ **do**
  $X_{kM}, y_{kM} \leftarrow Sample(C_k, M)$;
  $l_k \leftarrow Loss(NN(X_{kM}), y_{kM})$;
  if $l_k \leq ValidationLoss$ **then**
    $CandidateSet \leftarrow CandidateSet \cup \{l_k\}$
  end
end
$\mu_n \leftarrow Mean(CandidateSet)$;
$\sigma_n \leftarrow StdVar(CandidateSet)$;
$P(x) \leftarrow \mathcal{N}(\mu_n, \sigma_n^2)$;
for $k \in K$ **do**
  $s_k \leftarrow max(0, \frac{1}{2} - P(x < l_k - \epsilon\sigma_n))$
end
$\mathbf{w_s} \leftarrow Normalize(Scores)$

This algorithm is much more stable and independent from model states and clustering results, because it dynamically estimates the expected mean and variance of the cluster losses. The use of Gaussian distribution reduces the chance of overfitting or stagnant learning, because clusters with too small losses (which means they contain too clean and too easy data points) will be relatively under-sampled and some hard clusters, i.e. clusters with mostly clean data points, a moderate proportion of Gaussian noised data points, and minimal proportion of error data points, will be relatively over-sampled, while dirty clusters will still be strictly screened out.

Gradient Cosine Similarity Re-weighting Algorithm: The key part of this algorithm is its gradient-based estimation step. Unlike in Naive Screening Algorithm and Gaussian Confidence Screening Algorithm, here we set the gradients obtained by training model on validation set to be the benchmark gradients. We assume that the cleaner the batch is, the more similar its gradients will be to the benchmark. Under this assumption, we flatten and concatenate the gradient matrices and measure the similarity between benchmark and batch gradients with cosine similarity (of course we can use other measurements). Then we multiply the similarity (notice its components can be positive, zero, or negative) with a hyperparameter η to obtain the weight change δ. Finally, we update the sampling weights by adding δ to the previous sampling weights.

Algorithm 3: Gradient Cosine Similarity Re-weighting

Input: Training data clusters $C_1, C_2, \cdots, C_K$ containing training pairs (X_K, y_K), validation set $V = (X_v, y_v)$, network NN, batch size M.
Output: Updated sampling weights $\mathbf{w_s} = (w_{s,1}, \cdots w_{s,K})$

$ValidationGradient \leftarrow Loss(NN(X_V), y_V).backward()$;
$Scores = (s_1, s_2, \cdots s_K)$;
for $k \in K$ **do**
  $X_{kM}, y_{kM} \leftarrow Sample(C_k, M)$;
  $g_k \leftarrow Loss(NN(X_{kM}), y_{kM}).backward()$;
  $s_k \leftarrow \eta\, Cosine(ValidationGradient, g_k)$
end
$\mathbf{w_s} \leftarrow \mathbf{w_s} + Scores$

This algorithm is a modified version of the algorithm proposed in [17]. Unlike its method, however, we don't evaluate the gradient of every data point and re-weight the loss function, but evaluate the entire cluster with randomly sampled batches and re-weight the training set. This modification greatly reduced the computation needed. The negative side is, of course, it introduces more randomness to final performance, because 1) estimating cluster gradients with batches is random and 2) before model state stablizes, the validation gradients also have great randomness (the benchmark method in [17] has exactly the same problem). We will discuss this algorithm's performance in more detail in Section 4.

3.3 Complete Trajectory-Clustering-Based Re-weighted SGD Algorithm

The complete Trajectory-Clustering-Based Re-weighting (TCBR) SGD algorithm is listed below. As is clearly seen, our method doesn't require any hacking into user network code, thus can be easily injected into existent models. Furthermore, by tuning the GMM parameters, replaceing GMM with other clustering models, or using different re-weighting/sampling algorithms, one can easily experiment with custom settings to find the best TCBR SGD algorithm for their specific model and dataset without any change to the overall framework.

Algorithm 4: Trajectory-Clustering-Based Re-weighted Stochastic Gradient Descent (SGD)

Input: Training data $S = (X, y)$, validation set $V = (X_v, y_v)$, network NN, network weights $\mathbf{w}$ Gaussian Mixture Model GMM, batch size M, learning rate η, max pre-processing iteration T_p, max iteration T.

Output: Optimized network parameters $\mathbf{w}_*$

$\mathbf{w} \leftarrow \mathbf{w}_0$;
$\mathbf{R_0} \leftarrow \emptyset$;
for t_p *in* T_p **do**
  $scores \leftarrow NN(X)$;
  $\mathbf{R_{t_p}} \leftarrow Concatenate(\mathbf{R_{t_{p-1}}}, Normalize(scores))$;
  $grads \leftarrow Loss(scores, y).backward()$;
  $\mathbf{w} \leftarrow \mathbf{w} + Update(\eta, grads)$
end
$Cluster \leftarrow GMM(\mathbf{R_{t_p}})$;
$C_1, C_2, \cdots C_K \leftarrow Cluster(S)$;
for $t \in T$ **do**
  $\mathbf{w_s} \leftarrow Reweight(V, C_1, C_2, \cdots C_K, M)$;
  $X_{train}, y_{train} \leftarrow Sample(S, \mathbf{w_s})$;
  $grads \leftarrow Loss(NN(X_{train}), y_{train}).backward()$;
  $\mathbf{w} \leftarrow \mathbf{w} + Update(\eta, grads)$
end
$\mathbf{w}_* \leftarrow \mathbf{w}$

4 Experiments

4.1 Data and configuration

In experiments, we use common benchmark datasets for image recognition task: MNIST and CIFAR [13, 11]. MNIST has handwritten digits data with 60,000 28x28 images. The CIFAR-10 dataset consists of 60,000 32x32 colour images in 10 classes, with 6000 images per class. CIFAR-100 has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. A brief dataset information is in Table 1.

Table 1: Dataset and optimization settings

Name	Description	# Instances	Cases	Optimizer	Learning rate	# Epoch
MNIST	Handwritten digits	60000	10	SGD	0.001	80
CIFAR-10	Image recognition data	60000	10	SGD	1e-6	30
CIFAR-100	Image recognition data	60000	100	ADAM	0.1	50

4.2 MNIST data imbalance experiments

We use the standard MNIST handwritten digit classification dataset and subsample the dataset to generate a class imbalance binary classification task. We select a total of 5,000 image of size 28×28 on class 4 and 9 which is a benchmark problem. And let class 9 dominates the training data distribution. We train a standard LeNet on this task and we compare our method with 1) PROPORTION weights each example by inverse frequency 2) RESAMPLE sample a class-balanced mini-batch for each iteration.

To make sure that our method does not have the privilege of training on more data, we split the balanced validation set directly from training set. The network is trained with SGD with learning rate of 1e-3 and mini-batch size of 100 for a total of 8,000 steps.

Fig 3 plots the test error rate across various imbalance ratios averaged from 10 runs with random splits. Note that our method outperformed commonly used methods for class imbalance. With class imbalance ratio 1:200, our method only reports a small increase of error rate around. Comparing with RESAMPLE and PROPORTION, our approach takes advantage of every training instance and assigns more accurate weights.

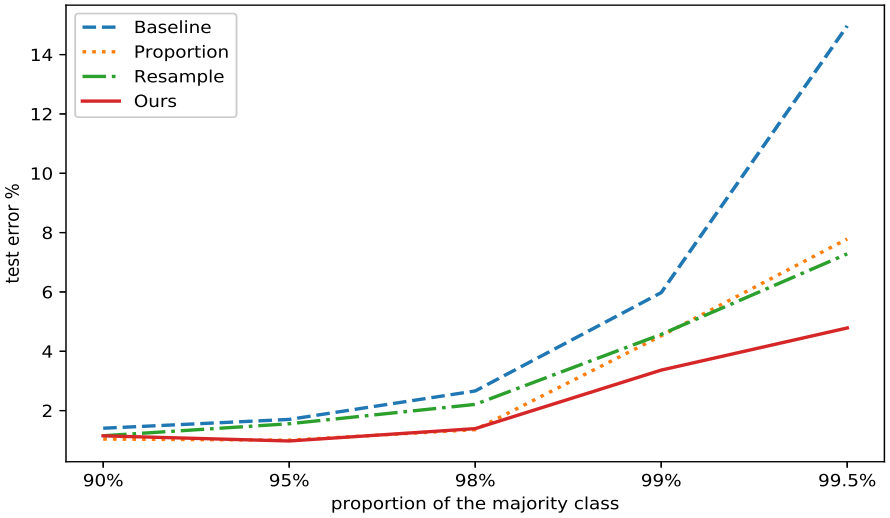

Figure 3: MNIST 4-9 binary classification error using LeNet on imbalanced classes.

4.3 CIFAR corrupted label experiments

We choose CIFAR-10 as the base data set for our corrupted label experiments because we observed the same phenomena mentioned in [18]: deep neural networks, especially convolutional networks, are very robust to noise labels. Even training on a dataset of as many as 80% label error data points, a very naive 3-layer ConvNet can still steadily achieve over 90% recall accuracy. Under this setting, using re-weighting strategies may even lower recall performance, because any re-weighting algorithm will definitely screening out some clean data points by fault, reduce the amount of information the model sees, and deteriorate its generalization capability. This is shown in Fig 4.

Remark 4: Fig 4 shows that when applying re-weighting to a 3-layer ConvNet model on 50% noise rate complete MNIST dataset, though using our method does improve training accuracy (that means our method is correctly sampling clean data points), it performs even worse than the non-weighting model on test set.

Remark 5: Fig 5 shows the recall accuracy over epochs with different noise rate. We can see clearly that straightforward training on corrupted dataset without any re-weighting strategy easily achieves over 90% recall accuracy until noise rate goes as high as 80%.

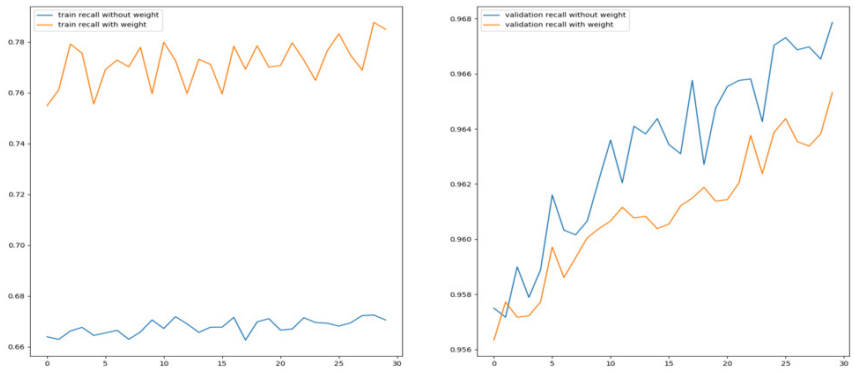

Figure 4: Training and recall accuracy over epochs on complete MNIST (60000), 50% label error noise rate, using a 3-layer ConvNet, Adam SGD, with and without re-weighting algorithm.

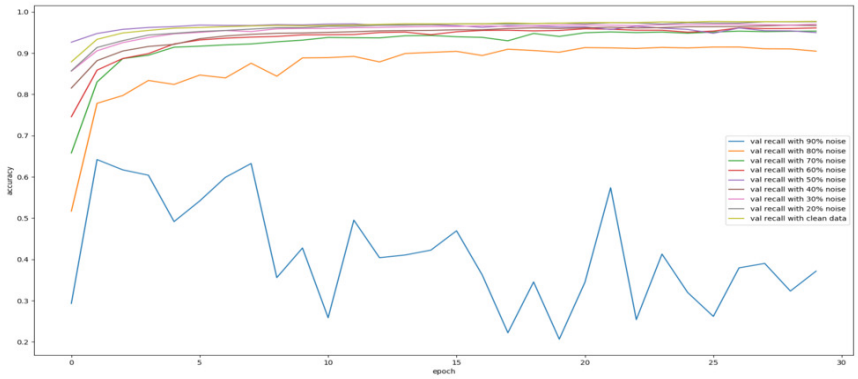

Figure 5: Recall accuracy over training epochs with different label error noise rate on complete MNIST (60000), using a 3-layer ConvNet, Adam SGD and no re-weighting algorithm.

With extensive experiments, we find that noise rate is not the determining factor of whether the model will fail to learn. In fact, the behavior of a model on noisy dataset is a result of three correlated variables: task difficulty, model capacity, and absolute number (instead of relative proportion) of clean data points. As is shown in Fig 6, when we reduce the total amount of data points from 60000 to 6000, while recall accuracy on clean dataset remains relatively uneffected (around 90%), the performance of the model on noisy dataset becomes visibly worse compared to the complete MNIST dataset counterpart with the same noise rate. The logical inference is that the learning robustness is bounded by the absolute number of clean data points our model has access to. When the model has already seen enough clean data points, it will remain stable over epochs. Of course, the necessary amount of clean data points needed to stablize a network differs according to the learning task difficulty (how much information it requires to extract enough features) and the model capacity (deeper models have more parameters and more complicated structures, and reasonably require more data inputs).

Remark 6: Fig 6 shows the recall accuracy over epochs with different noise rate when the model is trained on a uniformly sampled subset of MNIST (6000 data points). We can see clearly that the recall accuracy becomes much more sensitive to noise rates.

If we increase the task difficulty, the model will also be more unstable. We replace MNIST with CIFAR-10, and it can be clearly seen in Fig 7 that the model is more sensitive to noise, even though we have 60000 data points. On the basis of this result, we try to replace the naive 3-layer ConvNet

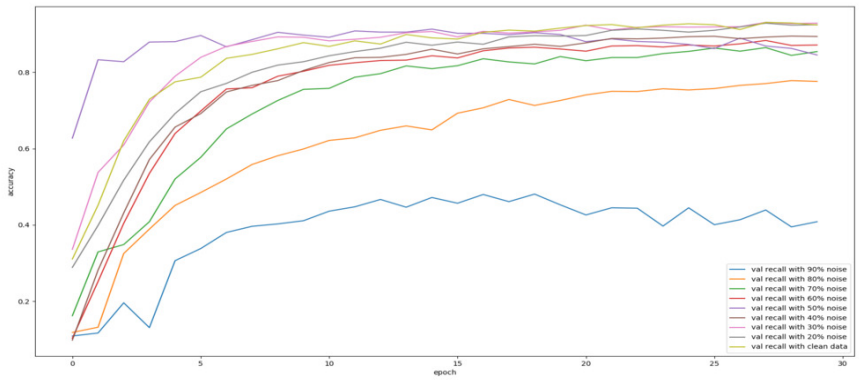

Figure 6: Recall accuracy over training epochs with different label error noise rate on MNIST subset (6000), using a 3-layer ConvNet, Adam SGD and no re-weighting algorithm.

model to be a more complex ResNet28 model. The latter has a bigger parameter capacity, and Fig 8 demonstrates that under this setting, the model is even more sensitive to noise.

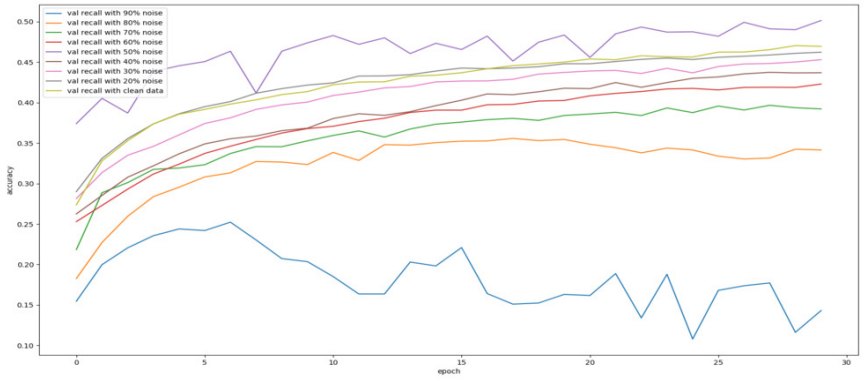

Figure 7: Recall accuracy over training epochs with different label error noise rate on CIFAR-10, using a 3-layer ConvNet, Adam SGD and no re-weighting algorithm.

Interestingly, noise types also matter in model performance. Experiments show that Gaussian noise doesn't influence learning process that significantly as label error noise (actually, a moderate proportion of Gaussian noise serves as global regularization), and changing its rate will not cause much difference. This justifies our clustering strategy: though our GMM model cannot split clean data points from Gaussian noised data points, the final re-weighting performance will not be severely influenced. Therefore, as long as our clustering algorithm can accurately differentiate between clean data points and label error data points, the effectiveness of our re-weighting algorithms will be guaranteed.

With these discoveries, we design the CIFAR-10 corrupted label experiment in such setting: we will train a ResNet28 model on the complete CIFAR-10 dataset of 40% error label noise and 10% Gaussian noise (two types of noise will not co-occur on the same data point).We first pre-train our model on the entire dirty dataset without any re-weighting for 10 epochs, collecting each data point's classification uncertainty trajectory, and then we train the GMM on trajectories. After that, we fix the weights of GMM, using it as a classifier that decides which cluster a certain data point belongs to, and iteratively apply the TCBR SGD algorithm as discussed in **3.3** until the max iteration is achieved. This setting ensures the model is sensitive to noise rate while still learnable.

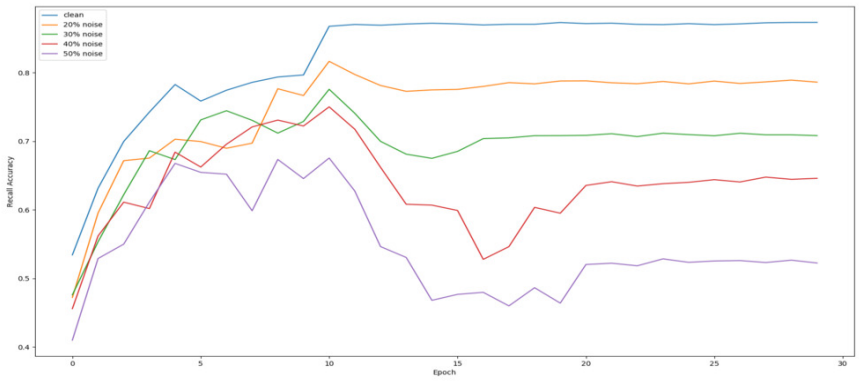

Figure 8: Recall accuracy over training epochs with different label error noise rate on CIFAR-10, using ResNet28, Adam SGD and no re-weighting algorithm.

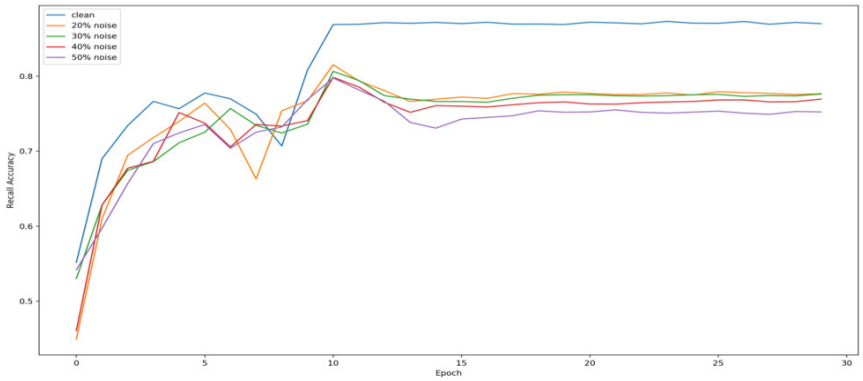

Figure 9: Recall accuracy over training epochs with different Gaussian noise rate on CIFAR-10, using ResNet28, Adam SGD and no re-weighting algorithm.

Experiments show that all three re-weighting algorithms we propose in **3.2** significantly increase and stablize recall accuracy. As is depicted in Fig 9, training without re-weighting deteriorates very fast down to near 50%, which matches the results in [17], and training with re-weighting remains at around 75% and higher. The problem is, however, the re-weighted training doesn't go up to 90% recall accuracy, nor does it go up to the 85% plus recall accuracy obtained on clean dataset with identical settings. There are several reasons for such phenomenon.

Firstly, as is discussed in **3.2**, our batch-based estimation inevitably introduces randomness. Records show that during training, sometimes the model will randomly move backwards to higher loss state, causing dirty clusters sampled by fault. We can see this mechanism directly from Fig 9: as naive screening (red line) and gradient cosine similarity re-weighting (yellow line) are both sensitive to model state, once in certain epoch the model accidentally samples excessive dirty data points, the model will corrupt; then in the next epoch, the corrupted model will favor the dirty data points as it has learned from dirty ones in the previous epoch. Such vicious circle will continue to worsen the performance of re-weighting. Contrarily, the Gaussian confidence screening algorithm (blue line) uses dynamic estimation, and it's significantly more stable than other methods, especially considering the drop-and-rise around epoch 22. It shows that this algorithm is self-adaptive.

Secondly, because our re-weighting algorithm is essentially screening error label data points out, that means the amount of available data points to our model is almost half of the complete clean dataset.

For example, if we have 60000 clean data points, then with 50% noise rate, the best re-weighting scenario is that during every epoch, we successfully eliminate all error points and only feed clean points to the model. But that means every epoch, only 30000 instead of 60000 data points can be seen. In practice, it's impossible to extract all 30000 clean data points neatly from dirty data points, and it's either accidentally sampling error points in, dropping clean data out, or both. Theoretically, we are unable to achieve close or better accuracy as on clean dataset.

Finally, because of limited time and computation resources, we only run 30 epochs of our re-weighted model for each setting. As the training of ResNet on CIFAR-10 usually takes over 100 epochs to achieve stably high recall accuracy (around 92%), our results at hand don't necessarily reflect the full capacity of our re-weighting model. In the near future, we will re-tune the settings and run full epochs on our algorithm to determine its final performance. But we can conclude now that, qualitatively, our re-weighting model does help significantly with learning and stablizing on corrupted label dataset.

Additionally, we notice that learning rate also matters. Though we are still not very clear how this factor influences re-weighting performance, we do observe that when model learning rate is set higher, gradient-based re-weighting algorithms become much more unstable, and the recall accuracy without re-weighting dramatically increases. This still requires further research, and we will follow up with it.

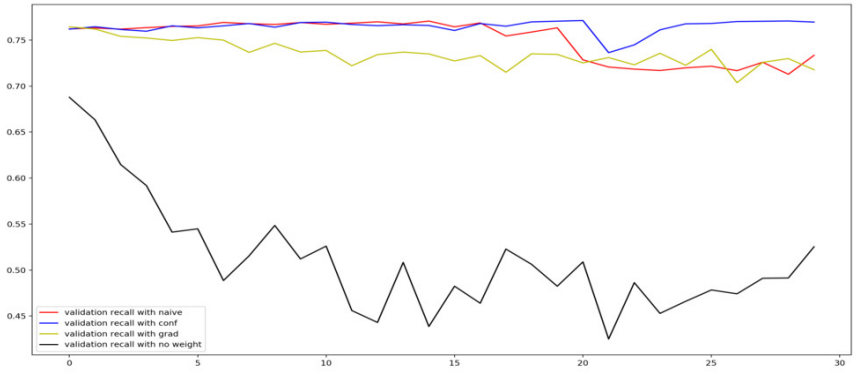

Figure 10: Recall accuracy over training epochs with different re-weighting algorithms on a 50% noise rate CIFAR-10 dataset, using ResNet28, Adam SGD

5 Conclusion

In this paper, we proposed algorithms of re-weighting neural networks training instances based on learning trajectory clustering. Leveraging the idea that different learning materials will have distinct learning curves towards neural networks, we form the prediction probability of each iteration to a set of learning trajectory. Our empirical study shows that such learning trajectory clustering significantly reflects the training instances' type and degree of difficulties. With clustering, we proposed three re-weighting methods on training set with different type and level of noise. For deep neural networks, sensitivity of noise is decided by the learning task, neural network capacity and number of clean data points. We show that when noise has impact on training, our methods could improve the learning accuracy with small computation cost. Moreover, our re-weighting structure can be directly applied to any deep learning architecture without any additional hyper-parameter tuning. To show more robust results, in the near future, we will comparing our approach with other meta-learning algorithms.

Acknowledgments

The authors would like to thank Professor Benjamin M. Marlin for giving nice machine learning lectures. And appreciate all TAs for they hard work. For this final project, we also would like to thank Haw-Shuian Chang for helpful discussions and kind support.

References

[1] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

[2] V. Avramova. Curriculum learning with deep convolutional neural networks, 2015.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[4] H.-S. Chang, E. Learned-Miller, and A. McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, pages 1002–1012, 2017.

[5] Y. Fan, R. He, J. Liang, and B.-G. Hu. Self-paced learning: An implicit regularization perspective. In *AAAI*, volume 3, page 4, 2017.

[6] J. Gao, H. Jagadish, and B. C. Ooi. Active sampler: Light-weight accelerator for complex data analytics at scale. *arXiv preprint arXiv:1512.03880*, 2015.

[7] S. Gopal. Adaptive sampling for sgd by exploiting side information. In *International Conference on Machine Learning*, pages 364–372, 2016.

[8] G. E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[9] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. *arXiv preprint arXiv:1712.05055*, 2017.

[10] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[11] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[12] M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.

[13] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[14] I. Loshchilov and F. Hutter. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

[15] D. Meng, Q. Zhao, and L. Jiang. What objective does self-paced learning indeed optimize? *arXiv preprint arXiv:1511.06049*, 2015.

[16] C. G. Northcutt, T. Wu, and I. L. Chuang. Learning with confident examples: Rank pruning for robust classification with noisy labels. *arXiv preprint arXiv:1705.01936*, 2017.

[17] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050*, 2018.

[18] D. Rolnick, A. Veit, S. J. Belongie, and N. Shavit. Deep learning is robust to massive label noise. *CoRR*, abs/1705.10694, 2017.

[19] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 761–769, 2016.

[20] E. Strubell, P. Verga, D. Belanger, and A. McCallum. Fast and accurate entity recognition with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098*, 2017.

[21] J. S. Supancic and D. Ramanan. Self-paced learning for long-term tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2379–2386, 2013.

[22] J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[23] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

[24] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[25] P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*, pages 1–9, 2015.