Learning to Reweight Examples for Robust Deep Learning - Mengye Ren, Wenyuan Zeng, Bin Yang, Raquel Urtasun; 2018

Janaki Viswanathan, 2581499

Machine Learning for Natural Language Processing

Abstract

This report was written as part of the "Machine Learning for Natural Language Processing" seminar at Saarland University that took place in WS2019/2020. It covers the paper "Learning to Reweight Examples for Robust Deep Learning" by Ren et al. [1].

In the past, there have been many approaches on how to handle training set biases such as class imbalance and noisy labels separately. However, when the data set is both imbalanced and has noisy labels, we are left with a conundrum with the contradicting ideas of handling these problems. The authors propose a novel meta-learning approach on how to handle this. The interesting idea about this approach is that it requires no additional hyperparameter tuning and can easily be implemented on any type of deep network.

This comprehensive report details the proposed approach along with all the necessary concepts that are required to understand the aforementioned article.

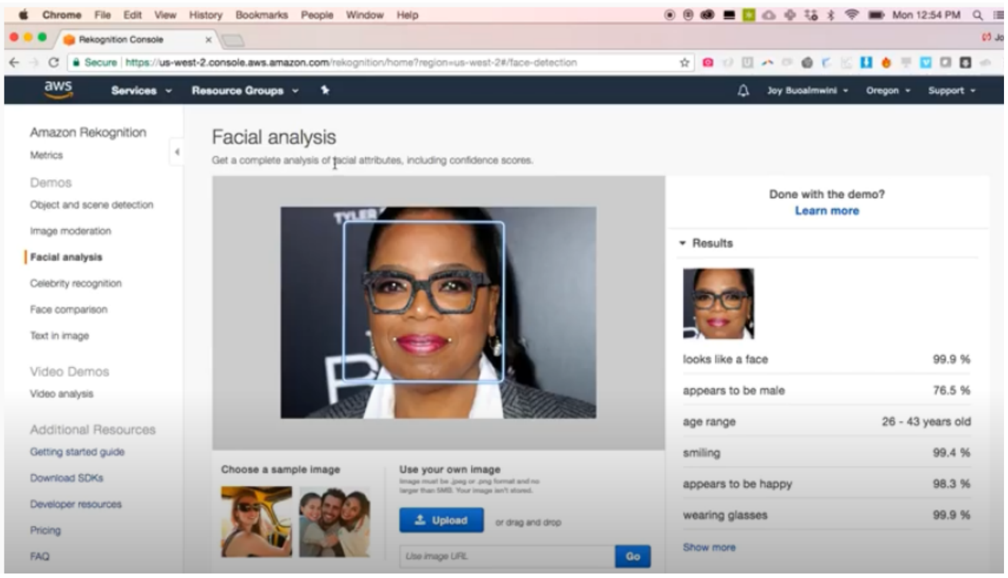
Contents

1 Introduction

Deep neural networks (DNNs) are being used in various disciplines which include computing, science, engineering, medicine, environmental, agriculture, mining, technology, climate, business, arts and nanotechnology, etc. [2] due to their powerful capacity for modeling complex input patterns. In spite of their powerful capacity of mapping input to output, there are several challenges or issues that are common. One among them is that DNNs can overfit to training set biases and label noises.

1.1 What is bias?

Bias can creep in at many stages of the Deep Learning process. Some of them include the following - while framing the problem statement, while collecting the data and while preparing the data [3]. Amazon received a lot of criticism for its internal recruiting tool which preferred men over women for technical roles and penalised resumes that included the word "women's" as in "women's chess club captain" [4]. In another study [5], 'Amazon Rekognition' which is currently used by several United States government agencies were criticised for its poor recognition of dark-skinned faces.



looks like a face	99.9 %
appears to be male	76.5 %
age range	26 - 43 years old
smiling	99.4 %
appears to be happy	98.3 %
wearing glasses	99.9 %

Figure 1: A screenshot showing Amazon Rekognition identifying Ms. Oprah as a man with 76.5% probability. (Figure taken from [6])

Some of the forms in which training set biases occur are:

- Class imbalance
- Noisy labels

The above mentioned criticisms faced by Amazon might be because of under representation of a particular sect of people in data. In general, a large dataset with high quality labels is desired to build a reasonable supervised deep model; presence of noisy labels severely degrades the performance of a model. However, to build a dataset with high quality labels, it requires a lot of human effort. Crowdsourced data with coarse labels are available at the expense of comparatively low performance.

1.2 Class imbalance

Class imbalance refers to the problem in machine learning when one of the classes have very less examples or samples when compared to the other classes. There is no specific threshold to define when to call a data imbalanced - it differs based on the use-case and the user. A very common example given to explain the case of class imbalance is fraud or anomaly detection. Consider a banking firm which wants to predict if a transaction is fraudulent or not. Generally, most of the transactions (almost 99.9%) are genuine, only a very little proportion (0.1%) of the transactions are fraudulent. When a machine learning model is built to predict fraudulent cases, we get 99.9% accuracy when the model blindly predicts all the transactions to be genuine. However, this defeats our purpose to predict fraudulent cases. Very less representation of a class in the data makes the model extremely difficult to learn to predict those classes. This makes the model extremely biased towards one of the classes.

1.3 Noisy labels

In general, the performance of a model can be drastically improved in the presence of more data. However, having a training data with noisy labels drastically decreases the generalisation of the model which in turn results in an ill-fit model. Noisy label problem is much more expensive when compared to the class imbalance problem since it requires a lot of human intervention to assure the quality of the labels. Though there are services available to get coarse labels by crowd-sourcing the data to be labelled, the presence of noise affects the performance of the model.

2 Previous approaches

Training set biases are usually handled by one of the following approaches:

- Dataset resampling - by sampling the right proportion of the data.
- Reweighting the examples - by assigning a weight to each example and minimizing a weighted training loss.

2.1 Class imbalance

2.1.1 Dataset resampling

Sampling is a procedure of selecting a subset of the examples from a population as a representation of the population. Some of the widely used techniques to address the class imbalance issue are oversampling, undersampling, Synthetic Minority Oversampling Technique (SMOTE) and hard example mining.

Oversampling and Undersampling:
Oversampling and undersampling are opposite and roughly equivalent techniques [7]. Oversampling is a technique which involves sampling multiple copies of the examples belonging to the minority class which will enable the dataset to be much more balanced. On the other hand, undersampling is a technique which involves removing examples from the majority class, with or without replacement. There are studies proving that undersampling beats oversampling [8].

SMOTE:
In 2002, Chawla et al., proposed a new technique named SMOTE - Synthetic Minority Oversampling Technique [9]. It is a combination of both oversampling and undersampling which generates synthetic data for the minority class. SMOTE was inspired by a technique that was proved successful in handwritten character recognition [10] where they created synthetic data by perturbing the training data. The perturbation was application-specific where they applied operations such as rotation or skew to create more training examples. However, in SMOTE, the perturbation is not application-specific and is done in the

"feature-space" rather than in the "data-space". Depending upon the number of new examples or rather the amount of oversampling required to balance the data, k minority nearest neighbor examples are chosen and lines are drawn connecting them. New examples are then synthesised along those lines connecting the k minority nearest neighbor examples. For instance, if the amount of oversampling required is 200%, only two neighbors from the five nearest neighbors are chosen and one sample is generated in the direction of each.

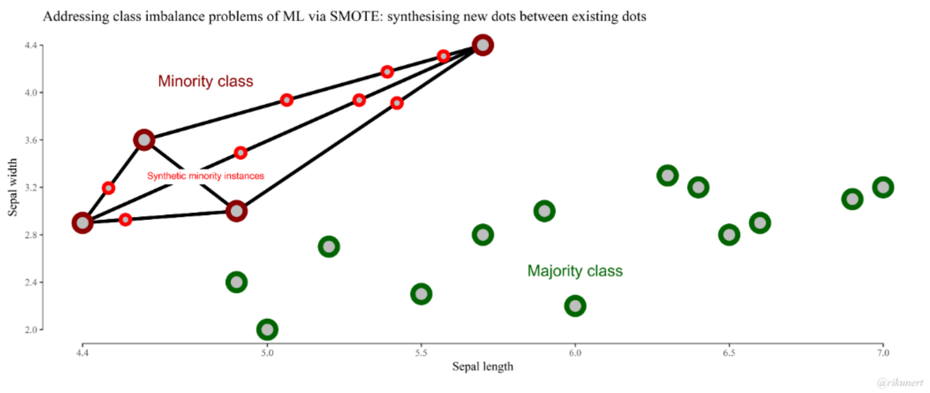


Figure 2: A sample from the famous Iris dataset is used to showcase class imbalance. The red class represents the species 'Setosa' and the green class represents the species 'Versicolor'. The small grey dots encircled in red shows the new synthesised samples which are generated from the line segments connecting k nearest neighbors in the minority class.(Figure taken from [11])

To generate those samples, Chawla et.al proposed the following: Take the difference between the sample under consideration (the feature vector) and its nearest neighbor. Multiply the difference by a random number between 0 and 1, and add it to the sample under consideration (the feature vector). This generalises the decision region of the minority class and helps to avoid biases.

2.1.2 Reweighting the examples

One of the widely used weighting approaches to deal with the case of class imbalance is AdaBoost - Adaptive Boosting which is an ensemble method. The main idea behind AdaBoost is that - 'the wisdom of the crowd is better than the wisdom of an individual'. AdaBoost builds multiple weak classifiers and the weighted sum of the output of all the weak classifiers are combined to make one final output. A classifier is termed as a weak classifier when it is slightly better than the baseline model. Typically, decision tree stumps are considered to represent a weak classifier. However, any classification algorithm can be used.

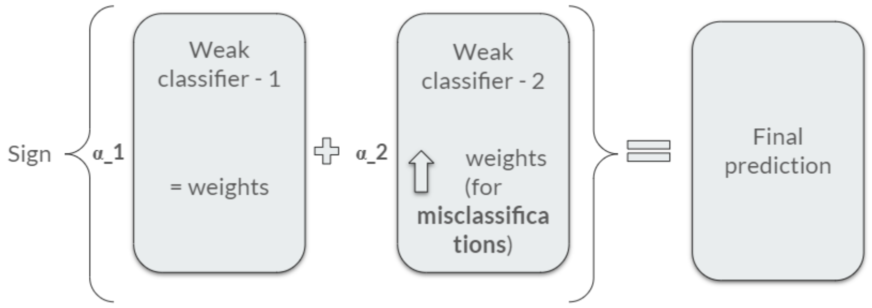


Figure 3: The pictorial representation explains how AdaBoost works: The weighted sum of multiple weak classifiers is considered to arrive at the final prediction. Note that in the first weak classifier, the weights are equal and in the subsequent weak classifiers, the weights are increased for the misclassified examples. (Figure: Self)

The first weak classifier is built with examples of equal weights. In the subsequent weak classifiers that are built, the weights for the misclassified examples from the previous classifier are increased so that the model focuses more on the misclassified examples - which are misclassified most probably due to low representation. Hence, by increasing the weights for the misclassifications, AdaBoost improves the prediction on the less represented class making the final prediction more generalised. One of the key ideas in AdaBoost is that the models are also weighted. This is done to ensure that the predictions of the 'best weak classifier' among all the weak classifiers are given more importance.

2.2 Noisy labels

2.2.1 Dataset resampling

Bootstrapping is a resampling method which is commonly used when there is not much data available. The idea is to get inference about a population from sample data. It is used to get an estimate of the distribution by repeated random sampling with replacement. When there is a lack of data to obtain an estimate, bootstrapping is always chosen as a way to get a better estimate from the samples since it is easy and straight-forward [12].

Yarowsky [13] proposed bootstrapping or self-training a learning agent. An initial classifier is built using some seed examples and then new seed rules are generated iteratively when new classifiers are built in every iteration. This is repeated until convergence.

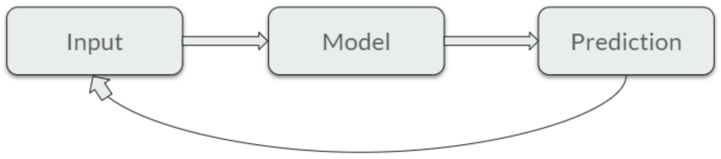


Figure 4: A model is built based on the input provided. Subsequently, the prediction is used to build new seed set for training in the next iteration. The steps are repeated until convergence. (Figure: Self)

2.2.2 Reweighting the examples

In 2014, Sukhbaatar et al. [14] proposed an approach to deal with noisy labels by adding a noise adaptation layer to an already existing classification model which is referred to as a 'base model'. This helps the network to adapt it's output to be robust towards noisy labels. The noise adaptation layer is a linear layer with weights equal to the noise distribution.

Two different approaches could be taken:

- Bottom-up noise model: A linear noise layer is added at the end of the base model. The weights given for the noise layer corresponds to the probabilities that a certain class being mislabelled to another class. The probabilities can be calculated as - $p(\tilde{y} = j|x, \theta) = \sum_i p(\tilde{y} = j|y^* = i)p(y^* = i|x)$

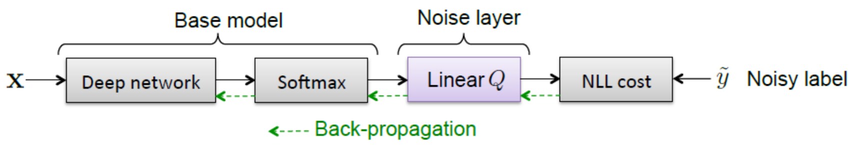


Figure 5: A linear noise layer is included between the already existing network and the cost layer. The weights for this layer tweaks the output from the base model to adapt to the noise distribution. (Figure taken from [14])

- Top-down noise model: In this approach, the noisy labels are changed instead of modifying the model to build an unbiased classification model.

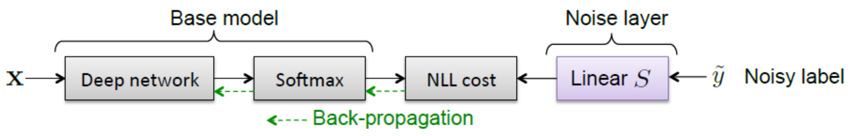


Figure 6: A linear noise layer is added after the cost layer. The noisy labels are converted before being used by the base model for training. (Figure taken from [14])

2.3 Shortcomings

In both class imbalance and noisy label cases, using a weighted training objective is a general approach:

$$\theta^*(w) = \arg\min_{\theta} \sum_{i=1}^{N} w_i \mathcal{L}_i^{\text{train}}(\theta)$$

where θ represent the model parameters, w represent the weights for the loss function $\mathcal{L}$ over N data points.

However, both the cases are handled with two different philosophies:

- For class imbalance problems: High loss examples are up-weighted or given more preference while training.
- For noisy label problems: Low loss examples are up-weighted or given more preference while training.

For class imbalance problems, the high loss examples are given larger weights since they are more likely to belong to the minority class. On the other hand, for noisy label problems the low loss examples are given larger weights since they are more likely to be clean.

When a dataset has only class imbalance problem or only noisy label problem, the above approaches works well. However, if the dataset has both class imbalance as well as noisy labels, then the existing methods would have wrong model assumptions. We arrive at a conundrum. Choosing one of the above methods requires us to assume that the data is either clean or balanced, this leads to a biased model.

When a model is built based on a biased training data, it results in an ill-fit model. One of the other common approaches to fix this is to use regularizers or example reweighting algorithms that we saw in Sections 2.1.2 and 2.2.2. Example reweighting algorithms use functions of the cost value of each example and in most cases, it is required to tune the additional hyperparameters. On the other hand, regularizers also require careful tuning of the regularization hyperparameters. This is a computational overhead. This raises an obvious question: when we use example reweighting approach, could be fix the problem with bias in the training data by bypassing the tuning of the hyperparameters which is a huge computational overhead? In the next section, the authors propose an algorithm which does not require any hyperparameter tuning and learns the weights for each of the examples from the data itself.

One another shortcoming is that we use a biased (imbalanced or noisy) test set to evaluate a model. By evaluating a model on a biased test set, we fail to understand the actual performance of the model. Hence, solving the training set bias problem is inherently ill-defined without an unbiased test set since the evaluation is based on the test set. In the following section, we will look in detail about the approach that has been proposed by Ren et al. [1] to handle these shortcomings.

3 Learning to reweight examples for robust deep learning

This section describes the main article this report is focused on: "Learning to Reweight Examples for Robust Deep Learning" written by Mengye Ren, Wenyuan Zeng, Bin Yang and Raquel Urtasun from Uber Advanced Technologies Group, Toronto ON, Canada and Department of Computer Science, University of Toronto, Toronto ON, Canada.

This is to avoid confusion while reading the report: From here on, the mention of 'weights' represents the weights for each training example. The usual usage of 'weights' to refer to the weights of each parameter obtained after training is invalid and is referred to as 'model parameters'.

3.1 Intuition

Using a biased test set to solve the training bias problem is inherently ill-defined. To overcome this, the authors propose to have a small unbiased validation set to guide the training.

The best example weighting should minimize the loss of a set of unbiased clean validation examples

It is also practical to create a small clean validation set and a large dataset with coarse labels for training. The human effort required to create a small validation set is necessary and possible and yields much better results. The large training dataset with coarse labels could be crowdsourced or could be constructed using weakly supervised models.

In general, the model is validated and tested at the end of the training. But by using a small clean validation set to guide the training, it helps in generalising the model and is consistent with the evaluation procedure. In that regard, the authors propose to perform validation using a small clean validation set at every iteration during training so that the example weights could be learnt dynamically for the current batch.

3.2 Meta-learning objective

Suppose that (x, y) is the input-target pair and $\{(x_i, y_i), 1 \leq i \leq N\}$ is the training set and $\{(x_i^v, y_i^v), 1 \leq i \leq M\}$ is a small unbiased and clean validation set where $M \ll N$. It is assumed that the training set contains the validation set. Let $\Phi(x, \theta)$ be our neural network model and θ be the model parameters.

In general, for a supervised algorithm to learn from data, we try to minimize an objective function. The objective is to minimize the loss function which expresses how far the prediction is from the truth. Let $C(\hat{y}, y)$ be the loss function that has to be minimized during training, where $\hat{y} = \Phi(x, \theta)$. Then, the expected loss for the training set is minimized $\frac{1}{N}\sum_{i=1}^{N} C(\hat{y_i}, y_i) = \frac{1}{N}\sum_{i=1}^{N} f_i(\theta)$, where $f_i(\theta)$ is the loss function of the sample x_i.

However, the authors propose to minimize a weighted loss where each example is assigned a weight. The above mentioned objective function: $\frac{1}{N}\sum_{i=1}^{N} C(\hat{y_i}, y_i)$ can be viewed as the one without weights or with same weights. Whereas, the authors' proposed weights are chosen based on the corresponding example's performance in the validation set. Then, the objective function here can be defined as follows:

$$\theta^*(w) = \arg\min_{\theta} \sum_{i=1}^{N} w_i f_i(\theta)$$

where w_i represents the weights for each example and is unknown upon beginning. The weights $\{w_i\}_{i=1}^{N}$ are learnt based on it's performance in the validation set $\{(x_i^v, y_i^v), 1 \leq i \leq M\}$:

$$w^* = \arg\min_{w, w \geq 0} \frac{1}{M} \sum_{i=1}^{M} f_i^v(\theta^*(w))$$

It is noteworthy that the weights w_i should always be non-negative since minimizing a negative training loss can usually result in unstable behavior.

With the proposed method, weights can be learnt dynamically. Considering the meta-learning perspective, to calculate the optimal weights w^*, it is necessary to have two nested loops: [15]

- The inner loop: to optimize the model parameters θ^*

  The model parameters are optimized according to some weighted training objective: $\theta^*(w) = \arg\min_\theta \sum_{i=1}^{N} w_i \mathcal{L}_i^{\text{train}}(\theta)$ where it is assumed that the weights w_i are already given for $i = 1, ..., N$ training examples

- The outer loop: to choose the optimal weights w^*

  The optimal model parameters θ^* obtained from the inner loop is used to evaluate its performance in the validation set. The weights which gives the best performance in the validation set are then chosen as the optimal weights w^*

This approach of calculating the optimal parameters using two nested loops can be very expensive. The inner loop to obtain the parameters itself is very time consuming making it impractical to learn the weights. So, an obvious question would be - could the weights be learnt using one single optimization loop instead? This motivated the authors to propose a solution which will help in efficiently calculating both the parameters in the inner loop as well as the weights in the outer loop.

3.3 Online approximation

The authors propose an approach of online approximation to use one single optimization loop to adapt the weights. This is achieved by selecting the weights according to the similarity between the training and the validation loss surface.

Typically, Stochastic Gradient Descent (SGD) or its variants is used to optimize loss functions. At every iteration or step of training, for a variant of SGD - the mini-batch gradient descent, a mini-batch of training examples is sampled and the parameters are adjusted according to the descent direction of the expected loss on the mini-batch. This can still be time-consuming.

The authors hence propose to take a single gradient descent step on a mini-batch of validation samples wrt. ϵ_t before tweaking the output to get the weights for the examples:

$$u_{i,t} = -\eta \frac{\partial}{\partial \epsilon_{i,t}} \frac{1}{m} \sum_{j=1}^{m} f_j^v(\theta_{t+1}(\epsilon))|_{\epsilon_{i,t}=0}$$

where $\epsilon_{i,t}$ is the initial weight for the example i at step t, m is the number of examples in the validation set, $f_j^v(\theta_{t+1}(\epsilon))$ is the loss on the validation set v and η is the descent step size on ϵ

It is important to have a non negative weight for the examples since having a negative weight will cause instability in the optimization problem. Typically, we choose the parameters which minimize the loss functions. In this case, since we use a weighted loss function $f_{i,\epsilon}(\theta) = \epsilon_i f_i(\theta)$, it has to be ensured that the calculated weights are not negative. If the weights are negative, the optimizer will end up maximising the loss function instead of minimizing it. By maximizing the loss function, the model will favor misclassifications instead of penalizing them. Hence, a (post) processing of the weights is necessary:

$$\tilde{w}_{i,t} = max(u_{i,t}, 0)$$

where $\tilde{w}_{i,t}$ are the non-negative weights obtained by capping the lower bound to be zero.

One additional step of normalising the weights $\tilde{w}_{i,t}$ has to be done so that the weights sum up to one:

$$w_{i,t} = \frac{\tilde{w}_{i,t}}{(\sum_j \tilde{w}_{j,t}) + \delta(\sum_j \tilde{w}_{j,t})}$$

where $\delta(.)$ is the Kronecker delta where $\delta(a) = 1$ if $a = 0$ and $\delta(a) = 0$ otherwise. The Kronecker delta has been introduced to account for the weights which are zero.

3.4 The algorithm

We will first look at the proposed algorithm in plain English:

- **Step-1:** Sample a mini batch of the training data and validation data
- **Step-2:** Do a forward pass on the training data with the initial parameters
- **Step-3:** Calculate the weighted training loss with initial weights being zero
- **Step-4:** Backpropagate on the training data and get the updated parameters (There will be no update in this step since the weights are zero)
- **Step-5:** Do a forward pass on the validation set with the updated parameters
- **Step-6:** Calculate the loss on the validation set
- **Step-7:** Backpropagate on the validation set and get the updated weights
- **Step-8:** Calculate the weighted training loss with the updated weights
- **Step-9:** Backpropagate on the training data and get the updated parameters

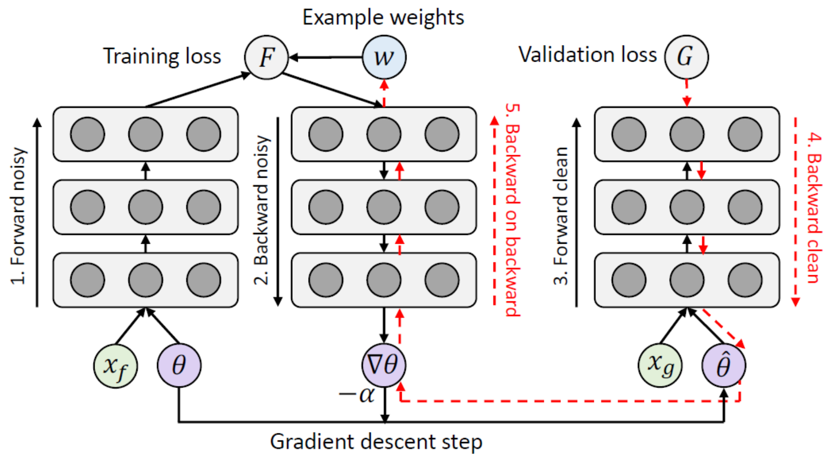


Figure 7: Computational graph of the algorithm. (Figure taken from [1])

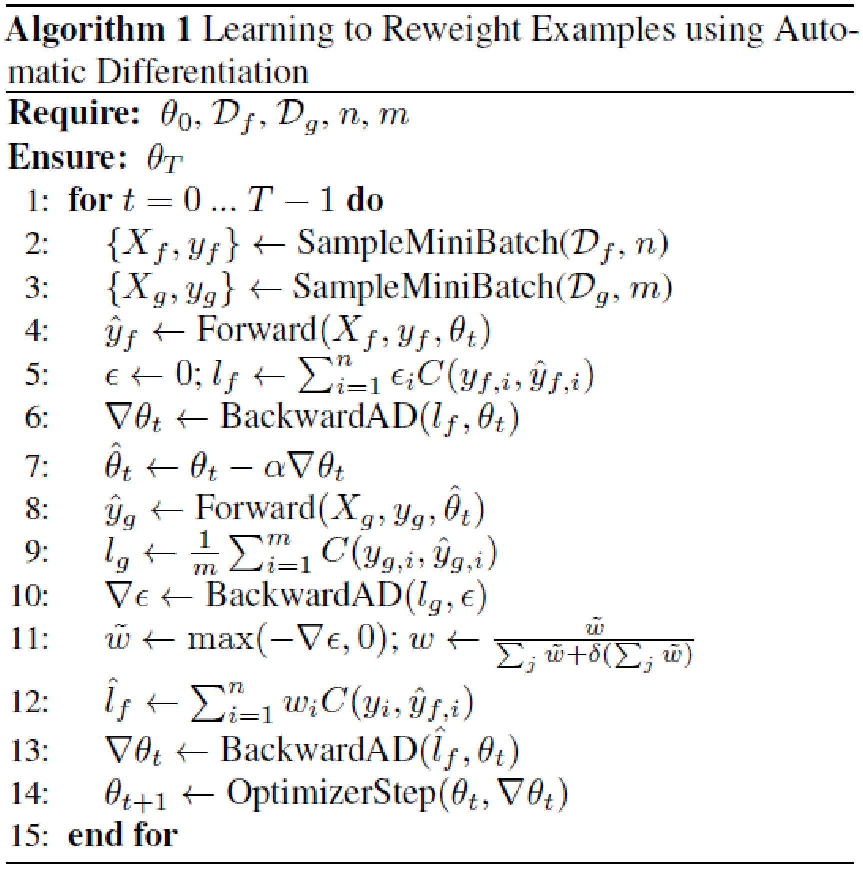
The first four steps comprise building a normal neural network model with an exception that weighted loss function is used instead of a normal or equal weighted loss function. Since the weights are assigned to be zero for all the examples, the weighted loss will be zero. This implies that there is nothing to update (i.e) the initial parameters are unchanged.

In the steps five to seven, a forward and backward pass with respect to the weights is done using the parameters from the previous step (it is equal to the initial parameters since the weights were unchanged) to learn the weights for the examples.

Now that the weights are learnt, the training loss is calculated again with the new updated weights. A second round of backpropagation is done on the training data to obtain new optimized parameters. This completes one iteration.

The algorithm first does two full forward and backward passes - one on the training set and one on the validation set. Additionally, it does a backward on backward pass to obtain the gradients to the example weights and another final backward pass to change the parameters based on the new reweighted loss. In general, a backward on backward pass usually takes about the same time as a forward pass. Hence, the authors specify that the algorithm takes 3 $\times$ training time when compared to a normal neural network training. However, this is one trade-off which could be made instead of spending a lot more of time and energy to create an unbiased training data.

A complete pseudo-code of the algorithm written by the authors is provided below for reference (Figure taken from [1]):



```
**Algorithm 1** Learning to Reweight Examples using Automatic Differentiation
**Require:** $\theta_0, \mathcal{D}_f, \mathcal{D}_g, n, m$
**Ensure:** $\theta_T$
 1: for $t = 0 \ldots T-1$ do
 2:     $\{X_f, y_f\} \leftarrow$ SampleMiniBatch($\mathcal{D}_f, n$)
 3:     $\{X_g, y_g\} \leftarrow$ SampleMiniBatch($\mathcal{D}_g, m$)
 4:     $\hat{y}_f \leftarrow$ Forward($X_f, y_f, \theta_t$)
 5:     $\epsilon \leftarrow 0; l_f \leftarrow \sum_{i=1}^{n} \epsilon_i C(y_{f,i}, \hat{y}_{f,i})$
 6:     $\nabla\theta_t \leftarrow$ BackwardAD($l_f, \theta_t$)
 7:     $\hat{\theta}_t \leftarrow \theta_t - \alpha\nabla\theta_t$
 8:     $\hat{y}_g \leftarrow$ Forward($X_g, y_g, \hat{\theta}_t$)
 9:     $l_g \leftarrow \frac{1}{m}\sum_{i=1}^{m} C(y_{g,i}, \hat{y}_{g,i})$
10:     $\nabla\epsilon \leftarrow$ BackwardAD($l_g, \epsilon$)
11:     $\tilde{w} \leftarrow \max(-\nabla\epsilon, 0); w \leftarrow \frac{\tilde{w}}{\sum_j \tilde{w} + \delta(\sum_j \tilde{w})}$
12:     $\hat{l}_f \leftarrow \sum_{i=1}^{n} w_i C(y_i, \hat{y}_{f,i})$
13:     $\nabla\theta_t \leftarrow$ BackwardAD($\hat{l}_f, \theta_t$)
14:     $\theta_{t+1} \leftarrow$ OptimizerStep($\theta_t, \nabla\theta_t$)
15: end for
```

3.5 Results

This section consists of the results published by the authors as part of their work.

3.5.1 Class imbalance

- Data: MNIST handwritten digit classification dataset - 5000 images of size 28×28
- Class imbalance: The minority class is '4' and the majority class is '9'
- Model: LeNet
- Validation set: 10 images - a balanced set
- Mini-batch size: 100 images

The LeNet model has been compared with some of the methods used to handle class imbalance problems such as - Proportion, Resample, Hard mining and Random.

Proportion: This method weights each example by the inverse frequency.
Resample: A well balanced sample is sampled from the data.
Hard mining: Selects the highest loss examples from the majority class.
Random: A random example weight baseline that assigns weights based on a rectified Gaussian distribution.

The image below compares the 'learning to reweight examples' approach with the other methods mentioned above. Clearly, it performs much better than the other methods by having a very low test error percentage across different proportion of the majority class.

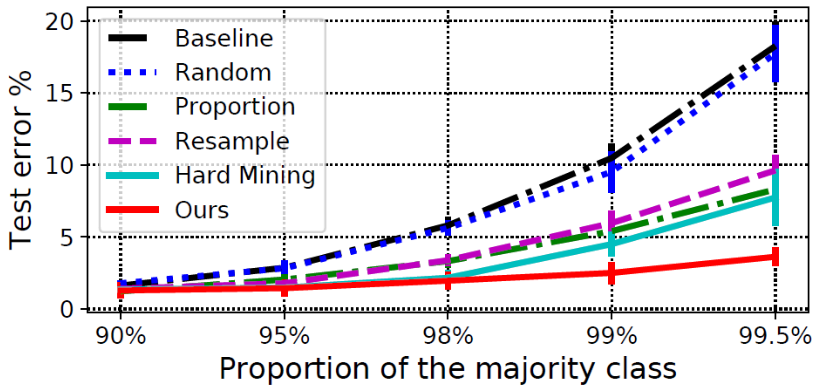


Figure 8: Test error percentage across different proportion of the majority class for different methods. MNIST 4-9 binary classification using LeNet on imbalanced dataset.(Figure taken from [1])

3.5.2 Noisy labels

The authors have tried two different tricks to make the data noisy: Unifrom flip and Background flip and the results for both the methods are available below:

- Data: CIFAR-10 which contains 10 classes
- Noisy labels: Two types of tricks to obtain noisy labels have been tried:

  Uniform flip: The label classes are uniformly flipped to any other label classes

  Background flip: The label classes are flipped to a single background class

- Model:

  Uniform flip: MentorNet with Wide ResNet-28-10 instead of Wide ResNet-101-10

  Background flip: ResNet-32

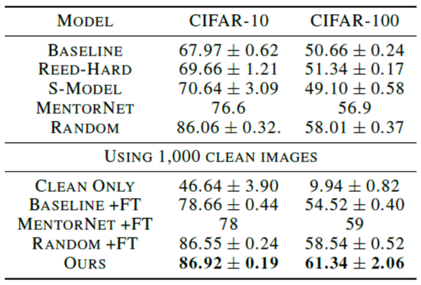
- Validation set:

  Uniform flip: 1000 clean images per class

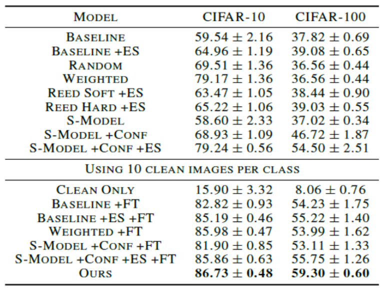
  Background flip: 10 clean images per class

- Mini-batch size: 100 clean images

Some of the other methods used to compare the performance of the method are as follows:
S-Model: Adds a fully connected softmax layer after the regular classification output layer to model the noise transition matrix.
S-Model + Conf + ES: S-Model with the transition matrix initialised to be the confusion matrix obtained from a baseline model. ES represents early stopping.
S-Model + Conf + ES + FT: S-Model with transition matrix initialised as confusion matrix with early stopping and fine tuning.
Reed: Bootstrapping technique where the training target is a convex combination of the model prediction and the model with $\beta = 0.8$ for hard bootstrapping and $\beta = 0.95$ for soft bootstrapping.
MentorNet: An RNN based meta-learning model.



Model	CIFAR-10	CIFAR-100
Baseline	67.97 ± 0.62	50.66 ± 0.24
Reed-Hard	69.66 ± 1.21	51.34 ± 0.17
S-Model	70.64 ± 3.09	49.10 ± 0.58
MentorNet	76.6	56.9
Random	$86.06 \pm 0.32.$	58.01 ± 0.37
Using 1,000 clean images		
Clean Only	46.64 ± 3.90	9.94 ± 0.82
Baseline +FT	78.66 ± 0.44	54.52 ± 0.40
MentorNet +FT	78	59
Random +FT	86.55 ± 0.24	58.54 ± 0.52
Ours	$\mathbf{86.92 \pm 0.19}$	$\mathbf{61.34 \pm 2.06}$



Model	CIFAR-10	CIFAR-100
Baseline	59.54 ± 2.16	37.82 ± 0.69
Baseline +ES	64.96 ± 1.19	39.08 ± 0.65
Random	69.51 ± 1.36	36.56 ± 0.44
Weighted	79.17 ± 1.36	36.56 ± 0.44
Reed Soft +ES	63.47 ± 1.05	38.44 ± 0.90
Reed Hard +ES	65.22 ± 1.06	39.03 ± 0.55
S-Model	58.60 ± 2.33	37.02 ± 0.34
S-Model +Conf	68.93 ± 1.09	46.72 ± 1.87
S-Model +Conf +ES	79.24 ± 0.56	54.50 ± 2.51
Using 10 clean images per class		
Clean Only	15.90 ± 3.32	8.06 ± 0.76
Baseline +FT	82.82 ± 0.93	54.23 ± 1.75
Baseline +ES +FT	85.19 ± 0.46	55.22 ± 1.40
Weighted +FT	85.98 ± 0.47	53.99 ± 1.62
S-Model +Conf +FT	81.90 ± 0.85	53.11 ± 1.33
S-Model +Conf +ES +FT	85.86 ± 0.63	55.75 ± 1.26
Ours	$\mathbf{86.73 \pm 0.48}$	$\mathbf{59.30 \pm 0.60}$

Figure 9: The results in the table on the left represents test accuracy in percentage when Uniform flip is used on the CIFAR dataset with 40% noise ratio using a WideResNet-28-10 model. The results in the table on the right represents test accuracy in percentage when Background flip is used on the CIFAR dataset with 40% noise ratio using a ResNet-32 model. FT represents fine tuning.(Figure taken from [1])

3.5.3 Class imbalance and noisy labels

- Data: CIFAR-100 with 40% flipped labels using Background flipping.
- Model: ResNet-32
- Validation set: 100 clean and unbiased images
- Mini-batch size: 100 images

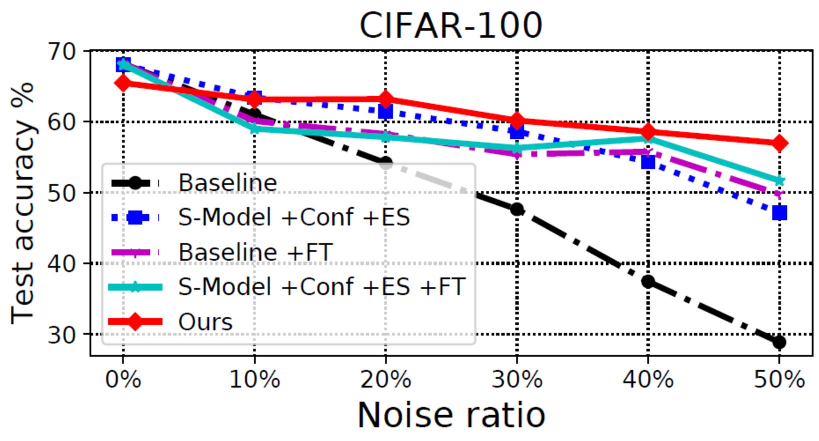


Figure 10: Test accuracy percentage on a noisy and imbalanced CIFAR-100 dataset using a ResNet-32 model.(Figure taken from [1])

The proposed approach clearly performs comparatively better than the other considered methods.

4 Takeaway

- With the opposing ideas or philosophies to handle class imbalance and noisy label problems, we arrive at a conundrum when the data is both imbalanced and noisy. The authors provide a neat approach on how to handle data when it is both imbalanced and noisy.
- Solving the training set bias problem is inherently ill-defined without an unbiased test set. In that regard, the authors suggest learning the weights for the training examples by evaluating the model on a small unbiased test set. The weights are chosen based on the similarity between the training and the validation loss surface.
- Most of the example reweighting approaches require hyperparameter tuning. However, the proposed approach does not require any hyperparameter tuning and learns the weights from the data while learning.
- The example reweighting approach is very time consuming since it has to learn weights for each of the examples. The authors propose an online approximation approach to avoid the computational over-head thereby a time and computationally efficient algorithm.
- Finally, as a cherry on top, the proposed method can be directly used with any kind of deep learning architecture and it does not require any additional hyperparameter search.

5 Future work and criticism

One obvious question which arises when we implement the algorithm is whether learning the weights for each of the examples will overfit to the training set? The authors mention that validation on every training step has links with model regularization which could be further explored.

From the experiment results that has been published in the article, we can see that the authors have not compared their method with some of the sophisticated methods that they have mentioned in the beginning of the article like SMOTE or any variant of boosting algorithms like AdaBoost which makes one wonder how good the performance would be in comparison with the more prevalent and widely used methods to handle biased data. Nevertheless, the proposed method provides a neat and robust approach which can be implemented in any deep learning architecture without any hyperparameter tuning.

References

[1] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. 03 2018.

[2] Oludare Abiodun, Aman Jantan, Oludare Omolara, Kemi Dada, Nachaat Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. volume 4, page e00938, 11 2018.

[3] Mit technology review: This is how ai bias really happens—and why it's so hard to fix. `https://www.technologyreview.com/2019/02/04/137602/this-is-how-ai-bias-really-happensand-why-its-so-hard-to-fix/`.

[4] Reuters: Amazon scraps secret ai recruiting tool that showed bias against women. `https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G`.

[5] Mit media lab: Gender shades. `https://www.media.mit.edu/projects/gender-shades/overview/`.

[6] Youtube: Amazon rekognition labels oprah winfrey male by joy buolamwini. `https://youtu.be/zNWyvuVCv50`.

[7] Wiki: Oversampling and undersampling in data analysis. `https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis`.

[8] Chris Drummond, Robert C Holte, et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. Citeseer.

[9] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. volume 16, pages 321–357, 2002.

[10] T. M. Ha and H. Bunke. Off-line, handwritten numeral recognition by perturbation method. volume 19, pages 535–539, 1997.

[11] Smote explained for noobs - synthetic minority over-sampling technique line by line. `http://rikunert.com/SMOTE_explained`.

[12] Wiki: Bootstrapping (statistics). `https://en.wikipedia.org/wiki/Bootstrapping_(statistics)`.

[13] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.

[14] Sainbayar Sukhbaatar and Rob Fergus. Learning from noisy labels with deep neural networks. *arXiv preprint arXiv:1406.2080*, 2(3):4, 2014.

[15] Talk by Mengye Ren: Learning to reweight examples for robust deep learning. `https://vimeo.com/287808016`.