UNIVERSITY OF CALIFORNIA
SANTA CRUZ

SAMPLE REWEIGHTING USING EXPONENTIATED GRADIENT UPDATES FOR ROBUST TRAINING UNDER LABEL NOISE AND BEYOND

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Negin Majidi

September 2021

The Thesis of Negin Majidi
is approved:

Professor Roberto Manduchi, Chair

Professor Luca de Alfaro

Dr. David Harrison

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures **v**

List of Tables **vi**

Abstract **vii**

Dedication **viii**

Acknowledgments **ix**

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Outline 3

2 Previous Work **5**

3 Loss Minimization **8**
- 3.1 Common Loss Functions 12
- 3.2 Regression Loss Functions 13
  - 3.2.1 Squared Loss 13
  - 3.2.2 Mean Absolute Error 14
  - 3.2.3 Root Mean Squared Error 14
- 3.3 Classification Loss Functions 14
  - 3.3.1 Hinge Loss 15
  - 3.3.2 Relative Entropy / Kullback-Leibler Divergence 15
  - 3.3.3 Bi-tempered Loss 16
  - 3.3.4 Focal Loss 17
- 3.4 Sample Weight Assignment 18
  - 3.4.1 Weights Update 19

4 EG Updates **21**
- 4.1 Motivation 21

4.2 Derivation 22

5 Sample Pruning Using EG **24**

5.1 Mini-batch Setting 24

5.2 Regularized updates 25

5.3 Capped updates for PCA 27

5.4 Capturing Noisy Examples in Model Training 28

5.4.1 When to Apply the EG Updates? 29

5.4.2 When is the Noise Reflected in the Training Loss? 30

6 Experiments **34**

6.0.1 Noisy PCA 34

6.0.2 Noisy Imagenet Classification 37

7 Conclusion **41**

8 Contribution **43**

A Hyperparameter Tuning **46**

A.1 Noisy PCA Experiments 46

A.2 Noisy Imagenet Experiments 48

Bibliography **50**

List of Figures

5.1 **Fashion MNIST Classification – Label noise :** (a) The model overfits to noisy examples after ~ 15 epochs. (b) noisy examples incur relatively larger training loss values compared to the clean examples. (c) EG Reweighting successfully captures the noise and improves generalization. (d) A normalized subset of 20 weights shows how clean but hard to classify examples recover later on via the EG regularization. [57] 31

5.2 **Fashion MNIST Classification – Blur noise:** (a) The (clean) train and test accuracy does not reveal any major overfitting. (b) Average training loss values for the clean and noisy examples are roughly the same. (c) We apply EG Reweighting on a pseudo-loss, consisting of the negative variance of the Laplacian operator [11] applied to each image. EG Reweighting successfully improves the performance. (d) A normalized subset of 20 weights which indicates the progression of the weights for clean and noisy examples. [57] 32

6.1 **Noisy PCA with EG Reweighting:** (a) A subset of noisy images from AT&T dataset with increasing levels of Gaussian noise power σ (i.e. L_2-norm of the additive noise component) along with their corresponding weights w assigned by the EG Reweighting approach. (b) Weights of the examples vs. the noise power. [57] 38

List of Tables

6.1 **Results on the noisy PCA problems:** (Regularized) EG Reweighting outperforms the state-of-the-art robust PCA methods in all cases. [57] . 37
6.2 **Results on the noisy Imagenet classification problems:** EG Reweighting consistently outperforms other methods. For the case of label noise, naïvely combining EG Reweighting with bi-tempered loss using the best performing hyperparameters for each case improves the performance further. [57] 40

Abstract

Sample Reweighting Using Exponentiated Gradient Updates for Robust Training Under Label noise and Beyond

by

Negin Majidi

Learning tasks in machine learning usually involve taking a gradient step towards minimizing an objective. Most of the time, the objective is the average loss of the training batch. In many cases, the dataset is noisy, so treating the examples equally during the training can cause overfitting to the noise in the data and poor generalization. Noisy examples generally incur a larger loss in comparison to clean examples. Inspired by the expert setting in online learning, we propose an algorithm for learning from noisy examples. We take each example as an expert and maintain a probability distribution over all examples as their weights. We update the parameters of the model using gradient descent and example weights using exponentiated gradients, alternatingly. Unlike other methods, our method handles a general class of loss functions and noise types. Our experiments show that our approach outperforms the existing baseline methods in supervised settings such as classification problems and unsupervised settings such as principle component analysis.

To my parents,

who taught me to always strive to be the best.

Acknowledgments

Foremost, I would like to express my deepest appreciation to my thesis advisor, Prof. Manfred Warmuth, for his immense knowledge, expertise, and support. It was a great honor for me to have him as my advisor. I also wish to express my sincere gratitude to my co-advisor, Dr. Ehsan Amid, without whom completing this study could have never been possible. His knowledge and guidance were always a great assistance throughout my research work.

Besides my advisor and co-advisor, I would like to thank my thesis committee, Prof. Roberto Manduchi, Prof. Luca de Alfaro, and Dr. David Harrison. Thank you for taking the time to read my thesis, and I appreciate your insightful comments which helping me strengthen my thesis.

I want to thank my friends: Amirata, for being a consistent source of support and professional advice who kindly shared his experiences with me; my high school friends, Sahar and Parisa, for their endless and unconditional love and motivation; and many other fantastic friends.

Last but not least, I am thankful to my family. My parents, Jamal Majidi, and Tahereh Aliabadi, for their sacrifice, encouragement, and love throughout my entire life; and my sister, Nazanin Majidi, for being the best source of joy and hope in my life.

Chapter 1

Introduction

1.1 Motivation

Machine learning (ML) [2] models have shown to be powerful in learning the signal in data in many learning tasks such as classification, regression, etc. Due to the recent advancement of deep neural network architectures [29] and the availability of abundant data, the last decade has observed remarkable progress in fileds like computer vision [35, 39], natural language processing [17, 63], and speech recognition [98, 60] and their applications in healthcare [20, 21] and finance [18, 37]. The majority of this success can be attributed to supervised learning, typically leveraging a significant number of labeled training examples. However, the occurrence of noisy samples in datasets can significantly impact learning any meaningful information from data. One source of noise may be the sampled data itself which can happen due to the noisy nature of data gathering process (e.g. blurred images in satellite imagery [10]). The other common source is having noisy labels which happens due to human annotation error [19] or the intrinsic difficulty of the task [55]. Many experiments have shown that corrupted data dramatically leads to poor prediction results [1, 24]. For instance, although deep neural networks show an overall robustness against label noise in classification tasks [65], they tend to overfit to noise, which negatively impacts the model's performance for generalization [53]. Trying to solve this problem by focusing on gathering clean datasets is not always possible for several reasons, such as difficulty in labeling data correctly even for the experts [55], the expensiveness of the annotating process [71], or the disagreement among labelers with unknown levels of expertise [93]. Therefore, it is essential to account for the presence of sample or label noise and find methods to reduce their deleterious effects on ML models.

Importance reweighting is one of the popular methods to address the noisy data problem [12], which will be the main focus of this thesis. The idea is that noisy examples are more difficult to learn and therefore the goal is to guide the ML model to be trained using the easy examples and not to be distracted by the difficult examples. Curriculum learning represents a type of learning in which easy examples are learned first and then gradually the model exploits the difficult examples to learn better discriminating features. In this work, we introduce a general algorithm that works using sample importance reweighting to overcome the overfitting problem in the presence of noisy samples. In regular training, every example contributes equally to the training of the network. However, in the presence of noisy examples, the model's loss at each step of the training is dominated by the large loss of the noisy training examples. Since the noisy examples' loss is large, they can distract the network from being correctly trained. In order to reduce the effect of the noisy examples, we assign weights to each example and try to update the weights at each iteration of training based on the examples' loss. We need to assign large weights to clean data points and smaller weights to noisy data points. Inspired by exponentiated gradient (EG) method [45], we update the weights at each step of training as we update the model's parameters. We show that by using EG updates, we can assign larger weights to the easy examples and smaller weights to the hard examples. Unlike other methods, our approach handles a general class of supervised learning tasks such as classification and unsupervised learning tasks such as principle component analysis (PCA) [44].

1.2 Outline

In Chapter 2, we investigate the previous work. In Chapter 3, we discuss loss minimization in machine learning problems and various types of loss functions. We also explain how the sample weight assignment works. In Chapter 4, we give a common motivation for using EG updates along with the derivation of the update rules. In chapter 5, we introduce our method in which we use EG update to assign a weight to the samples as well as describe two extensions of this method, regularized updates, and capped updates. Chapter 6 reports the results of our experiments on ImageNet [16], Fashion MNIST [91] for the classification task, and on UMIST [88] and AT&T [67] face recognition datasets for the PCA task. We use label noise, blur noise, and JPEG compression in ImageNet classification; label noise and blur noise in Fashion MNIST classification; and random noise, occlusion noise, and blur noise in UMIST and AT&T.

Chapter 2

Previous Work

Several studies have been done on addressing the issues associated with noisy data in ML [29, 53, 28, 83, 99, 84, 92]. A group of methods are focused on introducing modified loss functions that are robust against noisy data. Examples are symmetric cross-entropy [84], two-temperature logistic loss [8], bi-tempered loss function [7], and focal loss function [52]. Symmetric cross-entropy augments the cross-entropy loss by adding a reverse cross-entropy term. Two-temperature logistic loss is motivated using the Tsallis divergence. Bi-tempered loss function enables learning from noisy labels, and Focal loss addresses the class imbalance problem. A similar approach is to regularization terms that prevent overfitting like early-learning regularization [53]. The idea is that the model only overfits on the noisy labels in the latter steps of the training process and therefore the idea is to prevent the training dynamics to deviate from early stages. These methods are only functional in supervised problems such as classification and fail to address the noisy data problem in unsupervised settings.

Our proposed method is related to Curriculum learning that has been a popular topic in machine learning recent research. This concept has been used in several approaches [15, 30, 40, 79, 68, 50, 75]. In the first works where the concept of curriculum learning was used such as [73] and [12], the curriculum was pre-defined and fixed before the training starts. The pre-defined curriculum favors the samples with smaller loss in the training [42]. Later on, [48] introduced Self Paced Learning (SPL), in which the curriculum is optimized simultaneously with the network parameters. In SPL, a weight is assigned to the training examples (similar to what we do in our algorithm). The weight variable is alternatively updated along with the model parameters at each step of training. More accurately, the examples with loss values larger than a threshold $\lambda > 0$ are completely ignored in practice, and their weight is set to zero. Harder examples are revealed to the network by increasing λ as training progresses, and the network learns from easy examples and switches onto hard examples later. This method has been used in various problems [49, 51, 61], and it has been used in some empirical problems such as computer vision [74, 14], natural language processing [82], and multitask learning [30]. In other works, several weighting methods has been introduced that improve SPL [22, 41, 43, 100]. Earlier works perform discrete sampling of data, which may cause the model to get stuck in local minima. In contrast, our method does not use a discrete selection of data (zero/one weights), and the weights are continuous real-valued numbers. In [69], data parameters are introduced which govern the importance of the samples in training. They use gradient descent to update their weights jointly with model parameters. On the other hand, we use EG updates instead of GD to update the training sample weights. Unlike [69], that the sample weights can be unboundedly negative, in EG update, the sample weights are a probability vector, and projecting the updated weights onto the probability simplex is relatively simpler. This projection corresponds to dividing the weights by the sum of the updated weights.

Meta-learning has also been investigated in some recent works to dynamically alter the loss functions to update the sample weights based on the label noise [22, 41, 43, 100]. Some recent works that discuss training deep learning models noisy data attempt to learn an optimal curriculum from data with a helper neural network [43, 31, 81], called MentorNet in [43]. The idea is that a Mentor network is trained on the original data and then the labels generated by this model are used to train a second network called StudentNet. This method involves training an extra network on a held-out clean dataset. In other words, to learn an effective curriculum, their method requires a clean validation set that is sometimes hard to achieve. In contrast, our approach learns the sample weights jointly with the network parameters and does not need to train another network. Moreover, we validate our network on a validation set with the same distribution as our training set which is a more realistic scenario in real-life datasets.

Chapter 3

Loss Minimization

Loss functions or error functions are measures for evaluating the performance of a machine learning model on a given set of data points. They define how the predictions of the model deviate from the ground truth, thus mapping a model prediction to an associated cost value. The ground truth can be a real value (regression), a category (classification), or a structured object (structured prediction). Training an ML models is to find the set of parameters that minimize the loss function over a set of training points:

$$\boldsymbol{\theta}^* = \underset{\{\boldsymbol{\theta} \in \mathbb{R}^m\}}{\arg\min}\, L(\boldsymbol{\theta}|\mathcal{X}) \quad \text{where} \quad L(\boldsymbol{\theta}|\mathcal{X}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\boldsymbol{\theta}|\boldsymbol{x}_i, \boldsymbol{y}_i) \tag{3.1}$$

where L is the total loss of the model, $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is a pair of input and target label in the training set with N examples, and $\ell(\boldsymbol{\theta}|\boldsymbol{x}_i, \boldsymbol{y}_i)$ is the loss of the example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ given model parameters $\boldsymbol{\theta}$.

In some machine learning problems it is possible to directly find $\boldsymbol{\theta}^*$. One example is linear regression where $L(\boldsymbol{\theta}|\mathcal{X}) = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{x}_i^T\boldsymbol{\theta} - \boldsymbol{y}_i||^2$ and the optimal parameters can be derived using a closed-form solution: $\boldsymbol{\theta}^* = (\boldsymbol{x}^T\boldsymbol{x})^{-1}\boldsymbol{x}^T\boldsymbol{y}$. In many cases, however, solving for $\boldsymbol{\theta}^*$ is not feasible in a single step. In practice, several iterative steps are applied where each step aims to improve the loss value compared to the previous step. Examples are the CART [56] algorithm for training decision trees, gradient boosting [27], and gradient descent. Our focus in this work will be on the family of problems that are solved using gradient descent. In gradient descent, we maintain an estimate of the parameters $\boldsymbol{\theta}^t$ at each step t using an update rule. The update rule uses the parameters of the previous step and the gradient of the loss function with respect to those parameters.

In the update rule, two main things should be considered: First, at each step, the network should update its knowledge of the examples that it has observed. That is, given the same set of example for the second time, the loss function of those examples should be no larger than the first time. This tendency of the model to improve its predictions is called *correctiveness*. Secondly, training steps should not be independent and at each step of the training the network should remember part of its knowledge in the previous steps. Thus, new parameters should be close to the old ones. This closeness can be measured by a distance function $D(\boldsymbol{\theta}^{t+1}, \boldsymbol{\theta}^t)$ which is called the *inertia* term. This tendency of network to keep the current solution to the old solution is called *conservativeness*. In gradient descent, we use Squared Loss as a distance function.

There is always a trade-off between correctiveness and conservativeness. Therefore, we define the update rule as minimizing a linear loss function $L(\boldsymbol{\theta}|\mathcal{X})$ that accounts for correctiveness along with the distance of the current parameters with the parameters of the previous step that accounts for conservativeness. The two loss terms are aggregated with an importance coefficient $\eta > 0$ given to correctiveness relative to conservativeness which is called the *learning rate.* All in all, the gradient descent update at step $t+1$ can be written as a regularized loss minimization problem w.r.t. the parameters $\boldsymbol{\theta}$ at step t,

$$\boldsymbol{\theta}^{t+1} = \arg\min_{\theta} \left\{ {}^{1}\!/\!{}_{2\eta} \, \|\boldsymbol{\theta} - \boldsymbol{\theta}^{t}\|^2 + L(\boldsymbol{\theta}|\mathcal{X}) \right\}. \tag{3.2}$$

To solve the equation, we set the derivative of the right side of (3.2) w.r.t $\boldsymbol{\theta}$ to zero which results in the following equation:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta \nabla L(\boldsymbol{\theta}^{t+1}|\mathcal{X}). \tag{3.3}$$

The above equation is called implicit gradient descent update for parameter $\boldsymbol{\theta}$ [33, 46, 6, 4]. Since calculating the future loss of the model is not feasible, the future loss is approximated by the current loss of the network, and yields

$$\boldsymbol{\theta}^{t+1} \approx \boldsymbol{\theta}^{t} - \eta \nabla L(\boldsymbol{\theta}^{t}|\mathcal{X}) = \boldsymbol{\theta}^{t} - \eta \, \frac{1}{N} \sum_{i} \nabla \ell(\boldsymbol{\theta}^{t}|\boldsymbol{x}_i, \boldsymbol{y}_i) \tag{3.4}$$

This equation is called explicit gradient descent update for parameter $\boldsymbol{\theta}$. The last term can be seen as an expectation over the loss of individual examples w.r.t. the uniform distribution (each example is treated equally). We will generalize this notion by maintaining a probability distribution over the examples and instead of a uniform distribution, we assign a probability weight to each example that shows the importance of the example in the training of the model.

Choosing a loss function to solve any learning problem is an important task as the loss function is the evaluation metric for the training which determines the model parameters. If we choose an inappropriate loss function, then the model may not generalize well on unforeseen data.

Several factors should be considered in choosing a loss function for a particular machine learning problem, such as the type of machine learning task at hand, the configuration of the output, difficulty of computing its derivatives with respect to model parameters , and the problem's constraints.

Loss functions are broadly categorized into two classes: Classification loss functions for classification tasks and regression loss functions for regression tasks. Loss functions used in structured prediction tasks are one of the classification (e.g. object segmentation [66]) or regression (e.g. object detection [64]) loss functions. In the classification task, we intend to predict the output from a set of finite categories. For example, in the ILSVRC2012 dataset usually referred to as the ImageNet [16], the goal is to classify 1.2 million images into one of the 1000 different categories. On the other hand, in regression, we want to predict an output value, which is a continuous quantity. This output is then compared with the actual value to get a measure of loss. For instance, in the famous Boston Housing dataset [32], the task is to predict the median price of a house prices based on some of its features like its size and neighborhood. Additionally, triplet losses are used for image recognition [70] and dimensionality reduction [5]. Therefore, the type of problem that we need to solve is a factor in the type of loss function we want to use. Nevertheless, it is not the only factors as other factors like outliers, class imbalance, and label noise are important. The following section will discuss the loss functions' properties and introduce some standard loss function used in the learning tasks.

3.1 Common Loss Functions

The loss function has an important impact on the training task and it should consider all of the model's specs and the task's goals. Therefore, it is crucial to choose a proper loss function for a particular task.

Loss of the model is usually measured by a distance function D applied to the ground truth value $\boldsymbol{y}$ and the model prediction $\hat{\boldsymbol{y}}$. For instance, in classification task, our goal is to learn a prediction function f which maps an example $\boldsymbol{x}_i$ to its predicted label probability $\hat{\boldsymbol{y}} = f(\boldsymbol{\theta}|\boldsymbol{x}_i)$ and the goal is to make this probability as close as possible to the true label $\boldsymbol{y}$. Therefore, the loss function of the model usually has the form $\ell(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}) = D(\boldsymbol{y}, \hat{\boldsymbol{y}})$ where the true label is $\boldsymbol{y} \in \mathbb{R}^d$, and the predicted label is $\hat{\boldsymbol{y}} = f(\boldsymbol{\theta}|\boldsymbol{x}) \in \mathbb{R}^d$. This distance function can be calculated using the squared Euclidean distance, or relative entropy, also known as Kullback-Leiber divergence. The loss function which uses squared Euclidean distance is called Squared Error, and the loss function which uses Kullback-Leiber divergence is called cross-entropy. The EG algorithm results from using Kullback-Leiber divergence as a distance function for conservativeness. The EG update has been used in boosting [25], online PCA [85], and learning rate adaptation [3].

Although Euclidean distance and cross-entropy are widely used in many machine learning problems, they can be susceptible to noisy data. In what follows, in addition to common loss functions, we will discuss other loss functions that are shown to be robust to label noise, such as Bi-tempered loss function, and Focal loss.

3.2 Regression Loss Functions

Regression problems involve predicting a quantity, which is usually a real number. In this section, we discuss loss functions that are appropriate for regression tasks.

3.2.1 Squared Loss

Squared loss is one of the fundamental loss functions and the workhorse of the regression problems. As the function's name suggests, the function calculates the mean of the squared errors.

$$D_{\mathrm{Sq}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = {}^1\!/_2 \, \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 \tag{3.5}$$

i.e., ℓ is the loss function, N is the number of data points in the training set, $\hat{\boldsymbol{y}}_i$ is the prediction of the network for the i-th training sample, and $\boldsymbol{y}_i$ is the label of i-th training sample.

Squared error is only concerned with the magnitude of deviation from the actual value irrespective of their deviation direction. Due to squaring the errors, predictions that are distant from the actual values penalize the network notably compared to less deviated examples.

3.2.2 Mean Absolute Error

As we mentioned before, MSE penalizes outliers (examples with large errors) more strongly. Therefore, to avoid this problem we need to use a loss function which is more robust to outliers. In MAE, instead of calulating the average squared error over all of the examples, we calculate the average absolute value of the distance between the real value and the predicted value over all of the data points.

$$D_{\text{MAE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) == \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2} \tag{3.6}$$

Like MSE, MAE works with the magnitude of error and does not consider the direction of error in the samples as well.

3.2.3 Root Mean Squared Error

Although MAE is more robust to examples with large errors, the main issue with this method is that it is not differentiable at its minimum. This issue can cause convergence problems when training the model. To avoid the problem of differentiability and robustness to outliers, we usually use the squared version of MSE which is RMSE.

$$D_{\text{RMSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N}|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i| \tag{3.7}$$

3.3 Classification Loss Functions

Classification problems are those predictive problems where samples are assigned to a class out of a finite number of classes. In classification task, we desire to minimize zero-one loss which is $D_{0/1}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \mathbb{I}\{\boldsymbol{y} \neq \hat{\boldsymbol{y}}\}$. However, minimizing this loss function is an NP-hard problem [23]. This section investigates common surrogate loss functions that are mainly used in classification problems instead of the zero-one loss.

3.3.1 Hinge Loss

Hinge loss is used for "maximum-margin" classification and is commonly used for training support vector machines (SVM). In binary classification task, where $y \in \{+1, -1\}$, the hinge loss is

$$D_{\text{HG}}(y, \hat{y}) = \max(0, 1 - y\hat{y}) \tag{3.8}$$

When the data points are farther away from the correct side of the desicion boundry, the loss increases and the model penalizes those points.

3.3.2 Relative Entropy / Kullback-Leibler Divergence

Relative Entropy or Kullback-Leibler Divergence (KL divergence) is commonly used in classification tasks. KL divergence measures how much a probability distribution differs from another probability distribution.

For two probability distributions P and Q, KL divergence can be calculated by the negative sum of each event in P multiplied by the log of the probability of the event in P over the probability of the event in Q.

In case of classification problem, this loss function calculates the relative entropy between the true labels and the predictions. It is assumed that y and $\hat{\boldsymbol{y}}$ are probability vectors: all the components of $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are positive and the constraints $\sum_{i=1}^{d} y_i = 1$ and $\sum_{i=1}^{d} \hat{y}_i = 1$ are maintained.

$$D_{\mathrm{KL}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{d} y_i \log \frac{y_i}{\hat{y}_i} = \sum_{i=1}^{d} (y_i \log y_i - y_i \log \hat{y}_i) \tag{3.9}$$

where D_{KL} is the relative entropy entropy loss function, $\boldsymbol{y} \in \mathbb{R}^d$ is the vector of true labels, $\hat{\boldsymbol{y}} = f(\boldsymbol{\theta}|\boldsymbol{x}) \in \mathbb{R}^d$ is the vector of predicted labels, and N is the number of examples in the training set.

Note that the first term in the distance is independent of the model parameters and therefore only the second term is optimized during training. This second term is referred to as the cross-entropy loss function can be used in both binary classification and multi-class classification tasks.

3.3.3 Bi-tempered Loss

Bi-tempered loss function [7] is in the family of tunable loss functions which provides robustness to noise during training. Bi-tempered loss introduces a temperature t_1, and the second temperature t_2 to generalize the cross-entropy loss and the softmax function, respectively. The standard softmax function with cross-entropy is recovered when $t_1 = t_2 = 1$. However, introducing a mismatch between the two temperatures makes the loss function non-convex and more robust to noise.

3.3.4 Focal Loss

In some classification problems, the distribution of samples across the classes is imbalanced or skewed i.e., the examples from one or more classes are over-represented compared to other classes. Training on an imbalanced dataset makes the model biased towards learning more from the over-represented class and ignore the under-represented class [34] (as the second term in Eq. 3.4 is dominated by the examples in the majority class). The Focal loss function has been introduced to solve this problem by adding weighting to Cross-Entropy loss [52]. In case of binary classification the Focal loss function is defined by

$$D_{\mathrm{FL}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\alpha(1-\hat{\boldsymbol{y}})^{\gamma}\boldsymbol{y}\log\hat{\boldsymbol{y}} + (1-\alpha)\hat{\boldsymbol{y}}^{\gamma}(1-\boldsymbol{y})\log(1-\hat{\boldsymbol{y}}) \tag{3.10}$$

where $\hat{\boldsymbol{y}}$ is the prediction probability of the positive class. The parameter α is a hyperparameter that handles the class imbalance problem through increasing the weight of the minotrity class in the loss function and γ is the other hyper-parameter that controls the loss of misclassified examples. Both hyper-parameters are tuned using cross-validation. Using Focal Loss with $\gamma > 1$ reduces the loss for correctly predicted examples or examples when the model predicts the right right class (probability of right class more than 0.5). In contrast, it increases loss for wrongly predicted samples (when the model predicts the right class with probability less than 0.5). As the minority class examples are the ones that are predicted incorrectly and have a small correct prediction probability, the gamma parameter helps minority examples to dominate the loss and therefore draw the model's attention towards the under-represented class.

3.4 Sample Weight Assignment

As discussed earlier, in the training stage of the model, the optimizer intends to minimize the average loss over the entire training set. Therefore, the training examples contribute equally, regardless of whether they have a large loss value (wrongly predicted) or they have a small loss value (correctly predicted) at the current state of the model. However, some of the examples are noisy and they can distract the network from learning the general pattern in the data. It is shown [53] that throughout the iterations of training, the network learns to classify the "easy" examples, the examples that are well-represented in the training set and have a small loss, and memorizes the hard examples, the examples with large loss (outliers). In order to reduce the effect of the noisy examples in the training and avoid overfitting, we assign a weight to each example in the loss function that shows the importance of the example. Therefore, the overall loss of the model would be a weighted average loss over the training examples. The optimization problem is now to find both the model parameters and the sample weights

$$\boldsymbol{\theta}^*, \boldsymbol{w}^* = \underset{\{\boldsymbol{\theta}\in\mathbb{R}^m,\, \boldsymbol{w}\in\Delta^{n-1}\}}{\arg\min} \; L(\boldsymbol{\theta}, \boldsymbol{w}|\mathcal{X}) \quad \text{where} \quad L(\boldsymbol{\theta}, \boldsymbol{w}|\mathcal{X}) = \sum_{n=1}^{N} w_i \ell(\boldsymbol{\theta}|\boldsymbol{x}_i, \boldsymbol{y}_i) \tag{3.11}$$

where L is the total loss of the model, $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is a pair of input and target label in the training set with N examples, $\boldsymbol{w} \in \mathbb{R}^N$ is the sample weight vector, and $\ell(\boldsymbol{\theta}|\boldsymbol{x}_i, \boldsymbol{y}_i)$ is the loss of the example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ given model parameters $\boldsymbol{\theta}$.

The constraint $\sum_{i=1}^{N} w_i = 1$ is always maintained to have a unique solution and so that the common unweighted sum of loss over the examples would be an especial case of the weighted sum where $\boldsymbol{w} = \mathbb{1}_N/N$ where $\mathbb{1}_N$ is a vector of size N where all elements are equal 1.

To confront the effect of noisy examples on the training dynamics, we preferably want to assign larger weights to the easier data points than the harder ones to prevent them from dominating our objective. At each step of training, as the parameters of the network are being updated to minimize the cost function, we update the weights of the example as well. The most important question is how can we update the sample weights? We present a flexible approach using exponentiated gradient descent that can be used to reweight the examples. The goal is to reduce the effect of noisy examples during training. Unlike other methods of reweighting, our method can be applied to a broader set of loss functions and can be used in supervised problems such as classification, regression, as well as unsupervised problems such as PCA, and model averaging in federated learning.

3.4.1 Weights Update

As mentioned above, we take a vector of weights $\boldsymbol{w} \in \mathbb{R}^N$, where the i-th element w_i in this vector indicates the weight of the i-th example in the training data. At the first step of training, we initialize the weight vector and dynamically update it as the training proceeds. At first, we assume that the importance of each example in the training are the same, so that every example should equally contribute in the training. Therefore, we initialize the weight vector with $\boldsymbol{w} = \mathbb{1}_N$. We denote the weight vector at training time t with $\boldsymbol{w}^t$. At training step t, we minimize the weighted loss w.r.t. $\boldsymbol{\theta}^t$ as well as $\boldsymbol{w}^t$. We update the parameters $\boldsymbol{\theta}$ of the network using weighted gradient descent and we update the sample weights $\boldsymbol{w}$, alternatively.

In the update procedure for $\boldsymbol{w}$, we would like to reduce the loss, but we do not want to move away from the current estimate $\boldsymbol{w}^t$ (conservativeness). Therefore, the objective in the weight update would be minimizing the weighted loss of the network along with minimizing the divergence on $\boldsymbol{w}$. For a fixed $\boldsymbol{\theta}$, the weighted the loss function is linear w.r.t. $\boldsymbol{w}$. However, the loss is constrained to vectors of $\boldsymbol{w}$ with sum equal to one. To satisfy this constraint, using squared regularizer as the inertia term is not a good choice. In the next section, we expand this and motivate using EG updates for the weight updating task.

Chapter 4

EG Updates

Consider a setting that we have a vector of size N for sample weights and we need to update the weight along with the network parameters to minimize a loss function and achieve our final goal which is getting a better prediction on the unseen data. Here, we present an approach inspired by the idea of learning in the expert setting [47] and updating sample weights using exponentiated gradient update rule [45] at each step of training.

4.1 Motivation

Consider a model with parameters $\boldsymbol{\theta}^t$ at iteration t that observes an example $\boldsymbol{x}_i$. The weight of the example is w_i^t. Then, using the prediction function f the model predicts $\hat{\boldsymbol{y}}_i = f(\boldsymbol{\theta}^t|\boldsymbol{x}_i) \in \mathbb{R}^d$ as the output for this example, and the true label y_i is revealed afterwards. Then the model updates its parameters and the weight of the example to $\boldsymbol{\theta}^{t+1}$ and w_i^{t+1}, respectively.

Similar to the network's parameters' update, correctness and conservativeness are both important for updating sample weights. The network should learn something at each step of training, which means that it should move towards a set of weights where the loss of the network is reduced. In the meanwhile, we also need to keep the current solution of the sample weights close to the weights from the previous step. As we mentioned before, to maintain the condition $\sum_{i=1}^{N} w_i = 1$, we use relative entropy or KL divergence to measure conservativeness of the model in EG update. Therefore, we define the update rule as minimizing a linear loss function $\boldsymbol{w} \cdot \boldsymbol{\ell}^t$ (accounts for correctiveness) along with the distance of the current weights with the weights of the previous step (accounts for conservativeness) with an importance coefficient $\eta > 0$ given to correctiveness relative to conservativeness which is the learning rate,

$$\boldsymbol{w}^{t+1} = \underset{\boldsymbol{w} \in \Delta^{n-1}}{\arg\min} \left\{ {}^1\!/\!_\eta \, D_{\mathrm{KL}}(\boldsymbol{w}, \boldsymbol{w}^t) + \boldsymbol{w} \cdot \boldsymbol{\ell}^{t+1} \right\}. \tag{4.1}$$

4.2 Derivation

In order to guarantee $\sum_{i=1}^{N} w_i = 1$, we introduce a Lagrangian multiplier λ,

$$\boldsymbol{w}^{t+1} = \underset{\boldsymbol{w} \in \mathbb{R}_+^n}{\arg\min} \left\{ {}^1\!/\!_\eta \, D_{\mathrm{KL}}(\boldsymbol{w}, \boldsymbol{w}^t) + \boldsymbol{w} \cdot \boldsymbol{\ell}^{t+1} + \lambda(\sum_{i=1}^{N} w_i - 1) \right\}. \tag{4.2}$$

To minimize function (4.2), we need to take a partial derivative of the function w.r.t. w_i's, for all $i = 1, ..., N$, and set it to zero.

Solving for $\frac{\partial \boldsymbol{w} \cdot \boldsymbol{\ell}^{t+1}}{\partial w_i}$ is difficult, however, we can replace it with $\frac{\partial \boldsymbol{w} \cdot \boldsymbol{\ell}^{t}}{\partial w_i}$ which is reasonable approximation. Thus, we get the following equation:

$$^1\!/\eta \, \frac{\partial D_{\mathrm{KL}}(\boldsymbol{w}, \boldsymbol{w}^t)}{\partial w_i} + \frac{\partial \boldsymbol{w} \cdot \boldsymbol{\ell}^t}{\partial w_i} + \frac{\partial \lambda (\sum_{i=1}^{N} w_i - 1)}{\partial w_i} = 0 \tag{4.3}$$

Solving (4.3) for $\boldsymbol{w}_i$ and combining the derivatives to make a vector $\boldsymbol{w}$ yields

$$^1\!/\eta \, (\log \boldsymbol{w} - \log \boldsymbol{w}^t) + \boldsymbol{\ell}^t + \lambda = 0 \tag{4.4}$$

$$\Rightarrow \exp(\log \boldsymbol{w} - \log \boldsymbol{w}^t) = \exp(-\eta \boldsymbol{\ell}^t + \lambda)$$

$$\Rightarrow \boldsymbol{w}^{t+1} = \boldsymbol{w}^t \exp(-\eta \boldsymbol{\ell}^t)$$

where $w_i \geq 0$ for all $i = 1, ..., N$, and $\sum_{i=1}^{N} w_i = 1$.

From the above equation and the additional equation $\sum_{i=1}^{N} w_i = 1$ we have

$$w_i^{t+1} = \frac{w_i^t \, \exp(-\eta \, \ell_i^t)}{\sum_j w_j^t \, \exp(-\eta \, \ell_j^t)} \, . \tag{4.5}$$

Chapter 5

Sample Pruning Using EG

In EG, sample weights update is usually followed by network parameter $\boldsymbol{\theta}$ update by computing the gradient of the objective function w.r.t. $\boldsymbol{\theta}$ and performing a gradient descent step

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta_{\boldsymbol{\theta}} \sum_{i=1}^{N} w_i^{t+1} \nabla \boldsymbol{\ell}(\boldsymbol{\theta}^{t} | \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{5.1}$$

where $\eta_{\boldsymbol{\theta}} > 0$ indicates the network parameter learning rate. Note that the sample gradient $\nabla \boldsymbol{\ell}(\boldsymbol{\theta}^{t} | \boldsymbol{x}_i, \boldsymbol{y}_i)$ is scaled by sample weights w_i^{t+1}.

5.1 Mini-batch Setting

As discussed in [57], a common practice for training is to use a mini-batch of examples at each step instead using the full-batch of examples. That is, a subset of indices $\mathcal{B}^t \subseteq [n]$ corresponding to the subset of examples $\mathcal{X}^t \subseteq \mathcal{X}$ is used at iteration t. Thus, we consider a more general approach that includes the full-batch setting as a special case. Specifically, instead of maintaining a normalized weight $\boldsymbol{w}^t \in \Delta^{n-1}$, we can consider the unnormalized weights $\widetilde{\boldsymbol{w}}^t \in \mathbb{R}^n_+$ throughout the training and apply the EGU weight updates on the coordinates (i.e. weights) that are present in the mini-batch $\mathcal{X}^t$,

$$\widetilde{w}_i^{t+1} = \widetilde{w}_i^t \, \exp(-\eta_w \, \boldsymbol{\ell}(\boldsymbol{\theta}^t |\, \boldsymbol{x}_i, \boldsymbol{y}_i)) \quad \text{for } \; i \in \mathcal{B}^t \, . \tag{5.2}$$

The EGU update is then followed by a projection onto the simplex to form the following weighted loss for updating the model parameters $\boldsymbol{\theta} \in \mathbb{R}^m$,

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^m} \Big\{ \frac{1}{W_{\mathcal{B}^t}} \sum_{i \in \mathcal{B}^t} \widetilde{w}_i^{t+1} \, \ell(\boldsymbol{\theta} | \boldsymbol{x}_i, \boldsymbol{y}_i) \Big\} \, , \text{ where } W_{\mathcal{B}^t} := \sum_{i \in \mathcal{B}^t} w_i^{t+1} . \tag{5.3}$$

The EG weight update is followed by a parameter update using the gradient of the objective w.r.t. $\boldsymbol{\theta}$ and performing a gradient descent step as before,

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta_\theta \, \frac{1}{W_{\mathcal{B}^t}} \sum_{i \in \mathcal{B}^t} \widetilde{w}_i^{t+1} \nabla \ell(\boldsymbol{\theta}^t | \boldsymbol{x}_i, \boldsymbol{y}_i) \, ,$$

where $\eta_\theta > 0$ denotes the parameter learning rate. Note that now the sample gradient $\nabla \ell(\boldsymbol{\theta}^t | \boldsymbol{x}_i, \boldsymbol{y}_i)$ is scaled by the factor $\frac{\widetilde{w}_i^{t+1}}{W_{\mathcal{B}^t}}$.[1]

5.2 Regularized updates

The main goal of example reweighting is to reduce the effect of noisy examples by penalizing the example with large loss values. However, this reweighting procedure also affects the harder examples which may initially have large loss values, but are

[1]Our mini-batch updates are closely related to the specialist approach developed for on-line learning [26], which would use the (normalized) EG to update the example weights of the current mini-batch, always renormalizing so that total of weight of the mini-batch examples remains unchanged. Our method of maintaining unnormalized weights combined with GD updates on the reweighted gradients performs better experimentally.

eventually learned by the model later in the training. This side effect is desirable initially to allow the model to learn from the easier examples to form a better hypothesis. A similar approach is considered in curriculum learning where the level of difficulty of the training examples is increased gradually over the course of training [12]. However, in the case of EG updates, there is no mechanism to recover an example which incurs large loss values earlier in the training. In this section, we propose a regularized version of the updates that alleviates this problem [57].

Consider the following modified formulation of the EG update in Eq. (4.1),

$$\boldsymbol{w}^{t+1} = \underset{\boldsymbol{w}\in\Delta^{n-1}}{\arg\min} \Big\{ {}^1\!/\!_\eta \, D_{\mathrm{KL}}(\boldsymbol{w}, \boldsymbol{w}^t) + \boldsymbol{w}\cdot\boldsymbol{\ell}^t \underbrace{- {}^1\!/\!_\gamma \, H(\boldsymbol{w})}_{\text{maximize entropy}} \Big\}, \tag{5.4}$$

where the last term $H(\boldsymbol{w}) := -\sum_i \big(w_i \log w_i - w_i\big)$ is the entropy of the distribution $\boldsymbol{w}$ and $\gamma > \eta$ is a regularization constant. The effect of adding the negative entropy of $\boldsymbol{w}$ to the objective function is to bring the weights closer towards a uniform distribution after the update. By introducing a Lagrangian multiplier λ into Eq. (5.4) to enforce the constraint $\sum_i^n w_i = 1$, we have

$$\boldsymbol{w}^{t+1} = \underset{\boldsymbol{w}\in\mathbb{R}^n}{\arg\min} \big\{ {}^1\!/\!_\eta \, D_{\mathrm{KL}}(\boldsymbol{w}, \boldsymbol{w}^t) + \boldsymbol{w}\cdot\boldsymbol{\ell}^t - {}^1\!/\!_\gamma \, H(\boldsymbol{w}) + \lambda(\sum_{i=1}^n w_i - 1) \big\}, \tag{5.5}$$

Setting the derivatives to zero yields

$$({}^1\!/\!_\eta + {}^1\!/\!_\gamma) \log \boldsymbol{w} - {}^1\!/\!_\eta \log \boldsymbol{w}^t + \boldsymbol{\ell}^t + \lambda = 0 \tag{5.6}$$

Let $\frac{1}{\eta'} := \frac{1}{\eta} + \frac{1}{\gamma}$. For $\eta > 0$ and $\gamma > 0$, we have $\eta' \leq \eta$. Let $0 \leq r := \frac{\eta}{\eta'} \leq 1$. Refactoring Eq. (5.6) in term of r and enforcing the constraint yields,

Algorithm 1 Example Reweighting Using the Exponentiated Gradient Update

```
input: training examples $\mathcal{X}$ of size $n$, model parameters $\boldsymbol{\theta}$, model parameters and weights learning rates $(\eta_\theta, \eta_w)$, regularizer factor $0 \le r \le 1$
initialize $\widetilde{\boldsymbol{w}}^0 = \mathbf{1}_n$
for $t = 0$ to $T-1$ do
    for a given data batch $\mathcal{X}^t \subseteq \mathcal{X}$ indexed by $\mathcal{B}^t \subseteq [n]$ do
    • update the weights: $\widetilde{w}_i^{t+1} = \big(\widetilde{w}_i^t \exp(-\eta_w\, \ell(\boldsymbol{\theta}^t|\boldsymbol{x}_i, \boldsymbol{y}_i))\big)^r$ for $i \in \mathcal{B}^t$
    • calculate the sum: $W_{\mathcal{B}}^t = \sum_{i\in\mathcal{B}^t} \widetilde{w}_i^{t+1}$
    • update the model parameters: $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta_\theta \frac{1}{W_{\mathcal{B}^t}} \sum_i \widetilde{w}_i^{t+1} \nabla\ell(\boldsymbol{\theta}^t|\boldsymbol{x}_i, \boldsymbol{y}_i)$
end for
```

$$\boldsymbol{w}^{t+1} = \frac{1}{W^t}\big(\boldsymbol{w}^t \exp(-\eta\,\boldsymbol{\ell}^t)\big)^r \quad \text{for} \quad 0 \le r \le 1\,, \qquad \text{(Regularized EG Update)}$$

where $W^t := \sum_i \big(w_i^t \exp(-\eta\,\ell_i^t)\big)^r$. Note that $r = 1$ recovers the vanilla EG update whereas $r = 0$ sets the updated weights to a uniform distribution, i.e. $\boldsymbol{w}^{t+1} = {}^1\!/\!_n\,\mathbf{1}$. Model training using regularized EG update for example reweighting is summarized in Algorithm 1 [57].

5.3 Capped updates for PCA

We use *capping* algorithm, introduced in [86], to regularize the updated weight vector. We would like to prevent to updated weight vector to be larger than the capping threshold. After updating the vector $\boldsymbol{w}^t$ at training step t, the resulting weight vector might lie outside of the capped probability simplex. Therefore, we use the Capping algorithm to rescale the weights and prorate the excess amount of weight among all of the examples in the training batch. In the experiments, we observe that using Capping algorithm helps to get better reconstruction error in PCA problem.

5.4 Capturing Noisy Examples in Model Training

In this section, we discuss the practical considerations for applying EG Reweighting for noise-robust training. First, we discuss the problem of overfitting to noise and develop a learning rate schedule for the EG update that we found beneficial in practice. We also show the effect of the regularization on the updates. Next, we discuss the limitations of the EG Reweighting approach using the training loss for the weight updates, especially when applied to deep neural networks. We provide practical solutions in this direction and validate our approach in the experimental section. As our motivating example, we consider a simple convolutional neural network (two convolutional layers of size 32 and 64, followed by two dense layers of size 1024 and 10) trained on different noisy versions of the Fashion MNIST dataset [91]. We use a SGD with heavy-ball momentum (0.9) optimizer with a fixed learning rate of 0.1 and train the model for 80 epochs with a batch size of 1000. The baseline model achieves $91.35 \pm 0.38\%$ test accuracy [57].

5.4.1 When to Apply the EG Updates?

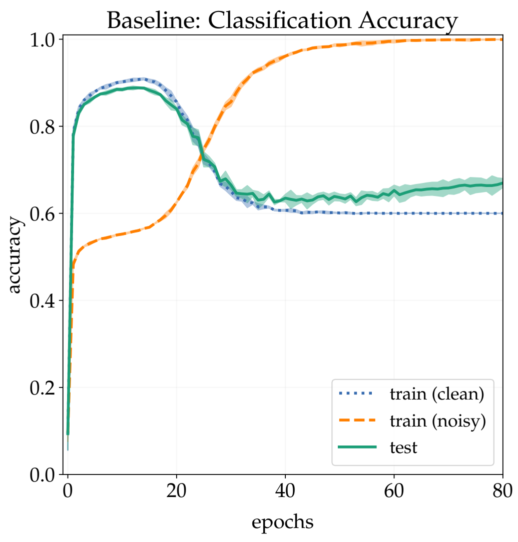
As we mentioned in [57], the update in Eq. (5.2) adjusts the weight of an example based on the value of the training loss of the example. This update entails the assumption that the noise may move the examples far away from the decision boundary, thus causing the example to induce larger loss values. We will discuss the validity of such assumption more thoroughly for different cases in the next section. The main question that we aim to address in this section is at which stage of training the model, the loss of the example may be indicative of noise?

In general, model training with noise can be split into two phases: i) initial warm-up and ii) overfitting to noise [90, 54]. In the early phase of training, the model starts from an initial solution and gradually forms a hypothesis based on the given set of noisy examples. In this phase, the model parameters are still close to the initial solution, thus the model behaviour is mainly dominated by a regularization effect similar to early stopping. At this point, the solution may not be highly descriptive of the (clean) data, but at the same time has not overfit to the noisy examples. This is shown in Fig. 5.2(a) where we train the model on the Fashion MNIST dataset with 40% symmetric label noise. The classification accuracy on both the clean train set (not available to the model) as well the held-out test set improves up to around 15 epochs. However at this stage, the large loss (thus, the gradient) of noisy examples starts to dominate the model training (Fig. 5.2(b)). Thus, the model starts to overfit to the noise and thus, sacrificing generalization.

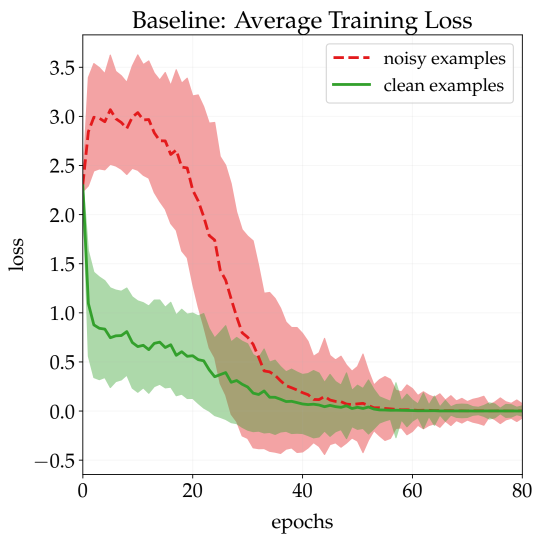
Based on this observation, the effect of example reweighting should gradually increase during the warm-up phase and then decrease over time once the overfitting is prevented. Thus, we propose using the following learning rate schedule for the EG Reweighting procedure, which we found effective in our experiments: *a linear warm-up in the early stage followed by a decay over time.* In this example, we apply a linear warm-up, up to 0.1, in the first 20 epochs followed by an exponential decay by a factor of 0.95 per epoch. We use $r = 0.98$. Fig. 5.2(c) shows that EG Reweighting with the proposed schedule successfully captures the noise and improves generalization. Fig. 5.2(d) shows a subset of 20 weights (normalized by their sum). As can be seen, the weights for some of the clean examples drop initially. These clean examples are inherently harder for the model to classify initially, thus incurring comparatively larger training losses. However, the regularization in the EG updates allows these examples to recover later on.

5.4.2 When is the Noise Reflected in the Training Loss?

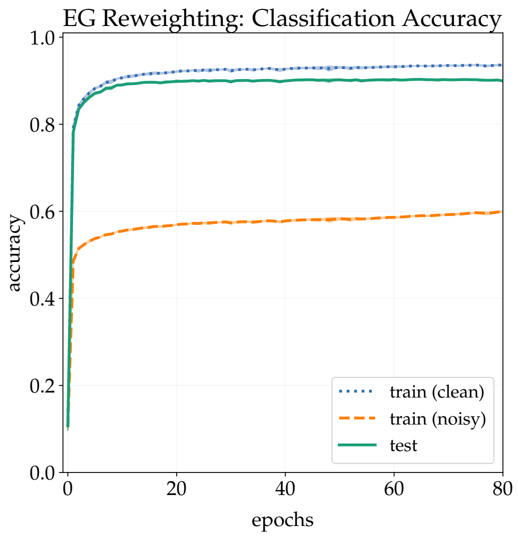
The underlying assumption for example reweighting in Algorithm 1 is that the noisy examples (after the initial warm-up stage) incur much higher loss values than the clean examples. Here, we show a case where such an assumption does not hold. For this, we consider the same model as in the previous section for classifying the Fashion MNIST dataset [91]. However, instead of adding label noise, we consider corrupting a subset of examples by adding blur noise to the input image. Specifically, we sample 40% of the images in the training set randomly and convolve these images with a Gaussian filter with a radius of 2. As can be seen in Fig. 5.2(a), although the generalization



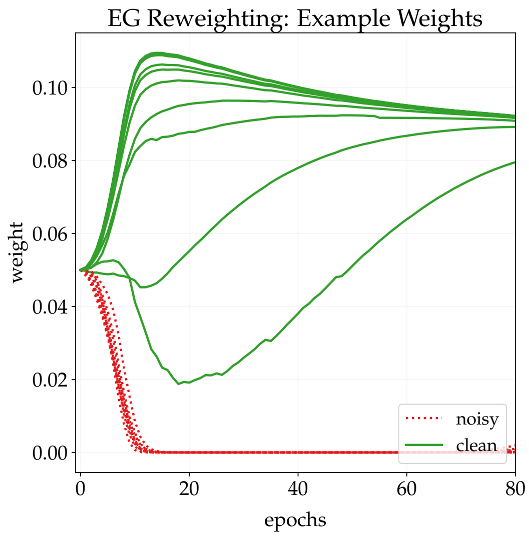
(a)



(b)

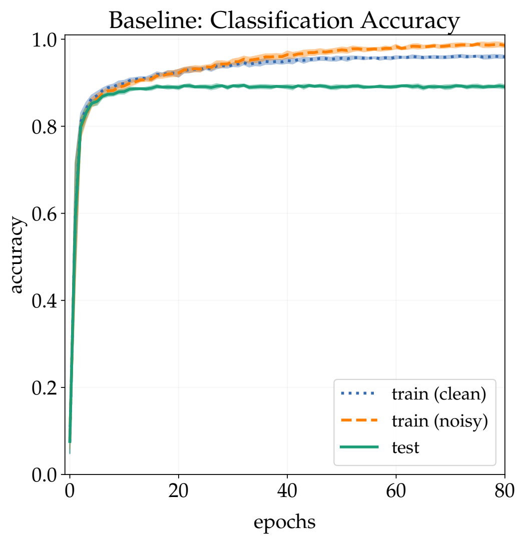


(c)

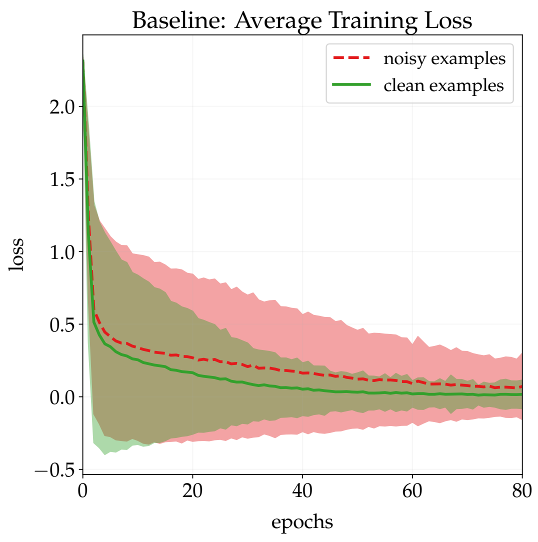


(d)

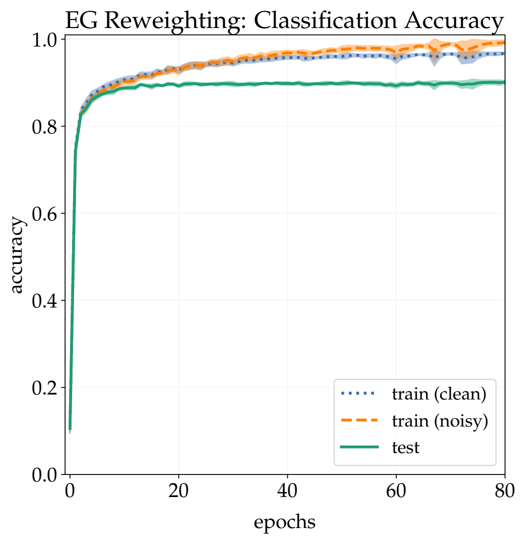
Figure 5.1: **Fashion MNIST Classification – Label noise :** (a) The model overfits to noisy examples after ∼15 epochs. (b) noisy examples incur relatively larger training loss values compared to the clean examples. (c) EG Reweighting successfully captures the noise and improves generalization. (d) A normalized subset of 20 weights shows how clean but hard to classify examples recover later on via the EG regularization. [57]



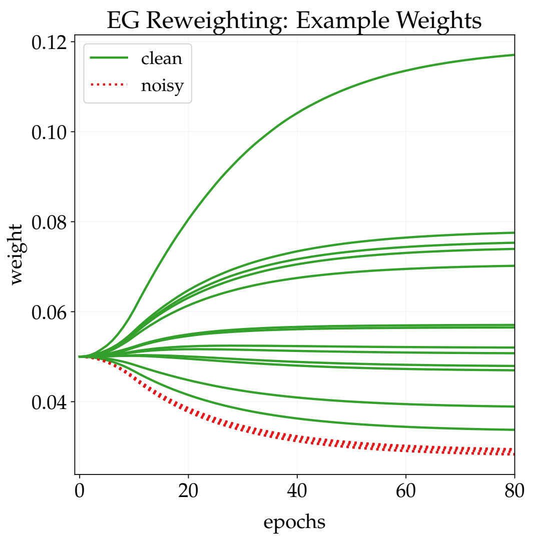
(a)



(b)



(c)



(d)

Figure 5.2: **Fashion MNIST Classification – Blur noise:** (a) The (clean) train and test accuracy does not reveal any major overfitting. (b) Average training loss values for the clean and noisy examples are roughly the same. (c) We apply EG Reweighting on a pseudo-loss, consisting of the negative variance of the Laplacian operator [11] applied to each image. EG Reweighting successfully improves the performance. (d) A normalized subset of 20 weights which indicates the progression of the weights for clean and noisy examples. [57]

performance of the model degrades compared to the noise-free baseline model, no clear overfitting manifests during training and the noisy and clean examples incur roughly the same average training loss values throughout (Fig. 5.2(b)). Thus, any method, including vanilla EG Reweighting, which aims to reduce the effect of noisy examples based on the training loss may fail to improve the performance.

In such settings, having access to a method that can reflect the noise in the data can be highly beneficial. Consequently, we can leverage this information for reweighting the examples using the EG update. In particular, we consider the signal from such a model as a *pseudo-loss* for EG, replacing the training loss in the weight update in Algorithm 1. In the case of the Gaussian blur in our toy example, we calculate the variance of the Laplacian operator [11]. A lower value of variance indicates smooth edges, thus suggesting a higher rate of blurring. Therefore, we consider the negative of the variance of the Laplacian operator as the pseudo-loss. We use the same hyperparameters and learning rate schedule as the previous section, but set $r = 1$. Fig. 5.2(c) shows that using this pseudo-loss improves the generalization performance of the model (from $89.07 \pm 0.50\%$ test accuracy to $90.13 \pm 0.38\%$ test accuracy). Also, Fig. 5.2(d) illustrates a subset of 20 weights (normalized). As the final remark, note that in many cases the pseudo-loss does not depend on the model parameters, thus does not vary during training. In such cases (as is in this example), the pseudo-loss needs to be calculated only once for each example (when visited for the first time). We will consider a more realistic setup in the next section.

Chapter 6

Experiments

In this chapter, as stated in [57], we evaluate the performance of our EG Reweighting algorithm through extensive unsupervised and supervised learning experiments. Our first set of experiments involve the unsupervised problem of learning the principal component, i.e. PCA, from noisy examples. Next, we focus on a significantly more challenging classification problem compared to the commonly used baselines in the previous work, namely classifying noisy versions of the Imagenet dataset. We show the utility of our approach on three different types of noise. The details of the hyperparameter tuning for different methods are given in Appendix A and the code for EG Reweighting will be made available online.

6.0.1 Noisy PCA

In this experiment, we examine our EG Reweighting method on the PCA problem and show improvements when the examples are noisy. Given a set of unlabeled training examples $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, the goal of the PCA algorithm [44] is to learn a subspace of dimension $k \ll d$ which minimizes the reconstruction loss of all examples. More formally, given a mean vector $\boldsymbol{m} \in \mathbb{R}^d$ and an orthonormal matrix $\boldsymbol{U} \in \mathbb{R}^{d\times k}$ s.t. $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}_k$, the *reconstruction loss* of the example $\boldsymbol{x}_i \in \mathcal{X}$ is defined as

$$\ell(\boldsymbol{m}, \boldsymbol{U} |\, \boldsymbol{x}_i) = \|(\boldsymbol{x}_i - \boldsymbol{m}) - \boldsymbol{U}\boldsymbol{U}^\top (\boldsymbol{x}_i - \boldsymbol{m})\|^2 .$$

The goal of the PCA algorithm is to find parameters $\{\boldsymbol{m}, \boldsymbol{U}\}$ that minimize the average reconstruction loss over all examples. Instead, we consider a reweighted version of the problem as in Eq. (3.11) by maintaining a distribution $\boldsymbol{w} \in \Delta^{n-1}$ on all examples. The algorithm proceeds by alternatingly solving for $\{\boldsymbol{m}, \boldsymbol{U}\}$ and updating the weights $\boldsymbol{w}$ using EG. For a fixed $\boldsymbol{w}$, the parameter updates follow similar to [96]: the mean vector is replaced with a weighted average $\boldsymbol{m} = \sum_i w_i\, \boldsymbol{x}_i$ and $\boldsymbol{U}$ is set to the top-k eigenvectors of the matrix $\sum_i w_i (\boldsymbol{x}_i - \boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{m})^\top$.

To evaluate the performance of different algorithms, after computing the mean $\boldsymbol{m}$ and the subspace matrix $\boldsymbol{U}$, we report the average reconstruction loss on a set of clean test examples. More specifically, we split the datasets into training and test sets with a 90%/10% ratio, respectively and then add noise only to the training examples. We run experiments for three types of noise: 1) *Gaussian random noise*, 2) *occlusion noise*, and 3) *Gaussian blur noise.* In occlusion noise, we corrupt 50% of training images by placing a rectangular occlusion with a size chosen uniformly at random between 25-100% the size of the image. The location of the occlusion is chosen randomly on the image and its pixels are randomly drawn from a uniform distribution. In Gaussian random noise, we add a random Gaussian noise to the entire image where the noise std is randomly drawn from a beta distribution $\beta(a = 2, b = 5)$. For blur noise, we convolve each image with a Gaussian filter with standard deviation drawn from the same beta distribution multiplied by 10.

To evaluate the effect of our proposed regularization on EG, we also consider an alternative regularization based on capping [87] which upper-bounds the value of each weight by a constant value. Thus, we report the results for three versions of the EG Reweighting approach: (1) PCA with EG Reweighting (EGR-PCA), but no regularization, (2) PCA with capped EG (Capped EGR-PCA), and (3) PCA with regularized EG (Regularized EGR-PCA). We compare our results against vanilla PCA [44], robust 2DPCA with optimal mean (R2DPCA) [94], capped robust 2DPCA with optimal mean (Capped R2DPCA) [94], and robust PCA with RWL-AN [96]. We consider two benchmark face image datasets in our experiments, (i) AT&T [67] which consists of 400 images of size 64×64 from 40 classes, and UMIST [88] which contains 575 images of size 112×92 from 20 classes. Table 6.1 summarizes all of the experimental results. To account for the variance in results due to the random nature of the added noise, we run each algorithm 50 times with different random seeds and reset the train-test at each run. It can be shown that for all three noises, EG, especially the regularized EGR-PCA, results in lower reconstruction losses on the test set compared to other algorithms. Fig. 6.1(a) shows a subset of the noisy training examples from the AT&T dataset that are corrupted with additive Gaussian noise. Each image is captioned by its noise power σ, i.e. norm of the additive component, and its weight w. It can be seen

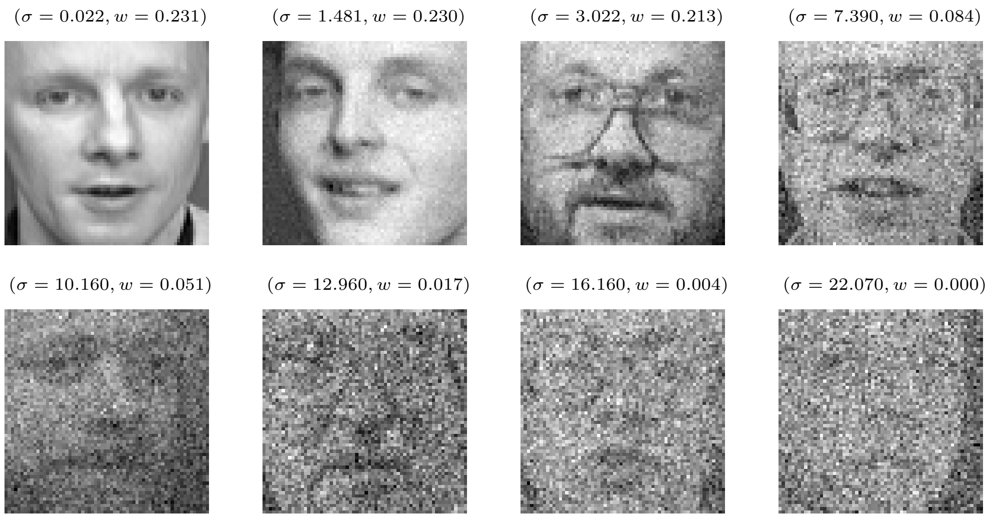
Table 6.1: **Results on the noisy PCA problems:** (Regularized) EG Reweighting outperforms the state-of-the-art robust PCA methods in all cases. [57]

	Vanilla PCA	RWL-AN	R2DPCA	Capped R2DPCA	EGR-PCA	Capped EGR-PCA	Regularized EGR-PCA
AT&T (random)	37.48 ± 1.37	29.96 ± 1.90	30.67 ± 0.22	28.51 ± 0.17	28.47 ± 1.71	28.36 ± 1.63	$\mathbf{25.92 \pm 1.42}$
AT&T (occlusion)	23.06 ± 0.19	20.01 ± 0.36	28.85 ± 0.06	28.39 ± 0.15	19.85 ± 0.28	19.86 ± 0.33	$\mathbf{19.82 \pm 0.36}$
AT&T (blur)	19.01 ± 0.20	19.41 ± 0.32	28.41 ± 0.07	28.40 ± 0.08	18.93 ± 0.21	18.99 ± 0.15	$\mathbf{18.91 \pm 0.19}$
UMIST (random)	100.48 ± 2.94	85.25 ± 4.66	115.47 ± 0.45	111.73 ± 0.27	79.63 ± 3.75	79.61 ± 4.00	$\mathbf{73.77 \pm 3.61}$
UMIST (occlusion)	65.26 ± 0.48	58.75 ± 1.36	114.25 ± 0.19	114.22 ± 0.23	58.69 ± 1.42	58.54 ± 1.41	$\mathbf{58.38 \pm 1.42}$
UMIST (blur)	55.97 ± 0.57	56.98 ± 0.85	110.99 ± 0.11	110.97 ± 0.10	56.02 ± 0.50	55.97 ± 0.50	$\mathbf{55.78 \pm 0.50}$

that as the examples become more corrupted, EG algorithm reduces their effect on the overall loss function by assigning smaller weights. Fig. 6.1(b) shows the same result for all images in the dataset where we see that the weight of examples decreases as they become more noisy.

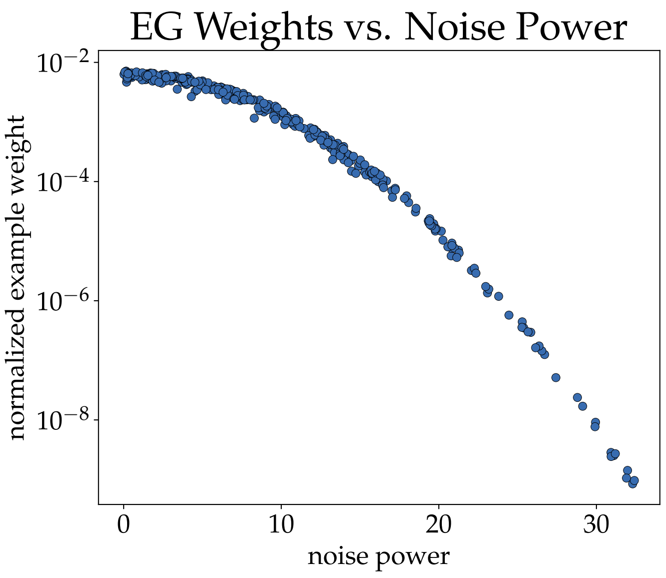
6.0.2 Noisy Imagenet Classification

It is noted that we have done Noisy Imagenet classification experiments in collaboration with Google AI and we used Google resources to run the experiments. As stated in [57], we consider classifying noisy versions of the Imagenet dataset [16] which contains ~ 1.28M examples from 1000 classes. We use a ResNet-18 architecture [35] trained with SGD with heavy-ball momentum (0.9) optimizer with staircase learning rate decay and batch size of 1024 for 240 epochs. The baseline model achieves $72.58 \pm 0.09\%$ top-1 accuracy on the test set. We consider three types of noise: 1) *Symmetric label noise*: the labels of a random 40% subset of the training examples are flipped, 2) *JPEG compression noise* [62]: where the JPEG quality factors are sampled from a beta distribution $\beta(a = 0.9,\, b = 0.8)$ and multiplied by 100. (a higher quality factor induces less noise.) The JPEG compression noise is highly non-uniform since for a fixed quality



($\sigma = 0.022, w = 0.231$)
($\sigma = 1.481, w = 0.230$)
($\sigma = 3.022, w = 0.213$)
($\sigma = 7.390, w = 0.084$)
($\sigma = 10.160, w = 0.051$)
($\sigma = 12.960, w = 0.017$)
($\sigma = 16.160, w = 0.004$)
($\sigma = 22.070, w = 0.000$)

(a)



(b)

Figure 6.1: **Noisy PCA with EG Reweighting:** (a) A subset of noisy images from AT&T dataset with increasing levels of Gaussian noise power σ (i.e. L_2-norm of the additive noise component) along with their corresponding weights w assigned by the EG Reweighting approach. (b) Weights of the examples vs. the noise power. [57]

factor, the amount of perceivable noise largely depends on the content of the image [58]. 3) *Blur noise*: where we apply a Gaussian filter with a kernel size of 5 to each image (noise std $\sim\beta(a = 1, b = 0.2)$).

For comparison, we consider the following methods: 1) baseline trained with cross entropy loss, 2) baseline trained with the bi-tempered cross entropy with softmax loss [7] which provides a robust generalization of the standard cross entropy loss, 3) DCL [69] which has similar complexity to our algorithm and has shown promising performance in settings with label noise, 4) our EG Reweighting method trained with cross entropy loss, 5) our EG Reweighting method combined with bi-tempered cross entropy loss. For each method, we tune the hyperparameters. For the combined EG additional Reweighting + bi-tempered loss, we do not perform any further tuning and simply combine the best performing hyperparameters for each case. Each result is averaged over 5 runs.

For the label noise experiment, we apply the EG Reweighting algorithm on the training loss. For the JPEG compression and blur noise experiments, we rely on the negative of the *visual quality score* from the NIMA model as the pseudo-loss. NIMA [78] is a deep convolutional neural network trained with images rated for aesthetic and technical perceptual quality. The baseline model used in NIMA is Inception-v2 [76]. In [78], the last layer of Inception-v2 is modified to 10 neurons to match the 10 human ratings bins in the AVA dataset [59]. The original learning loss used in NIMA is the Earth Mover's Distance, yet, in the current framework we compute the mean score and use it in our hybrid loss. This approach is similar to the method used in [77] to employ

Table 6.2: **Results on the noisy Imagenet classification problems:** EG Reweighting consistently outperforms other methods. For the case of label noise, naïvely combining EG Reweighting with bi-tempered loss using the best performing hyperparameters for each case improves the performance further. [57]

Noise type	Baseline	Bi-tempered loss	DCL	EG Reweighting	Bi-tempered loss & EG Reweighting
40% label noise	65.36 ± 0.19	66.18 ± 0.21	67.21 ± 0.09	67.44 ± 0.11	$\mathbf{67.63 \pm 0.19}$
JPEG Compression noise	70.88 ± 0.18	70.95 ± 0.07	70.63 ± 0.16	$\mathbf{71.05 \pm 0.02}$	70.91 ± 0.20
Blur noise	71.01 ± 0.16	71.06 ± 0.05	70.83 ± 0.10	$\mathbf{71.18 \pm 0.11}$	70.87 ± 0.14

NIMA as a training loss. As shown in [78, 77], the NIMA loss is sensitive to image degradation such as blur, imbalanced exposure, compression artifacts and noise.

The results are shown in Table 6.2. EG Reweighting consistently outperforms the comparator methods in all cases. Additionally, combining EG Reweighting and the bi-tempered loss naïvely provides further improvement in the label noise case. This shows the potential of combining the EG Reweighting method with a variety of loss functions. Also, we believe such improvements are possible for the JPEG compression and blur noise as well, however, this may require additional tuning of the hyperparameters. Note that methods such as DCL which reply on the training loss for handling noise yield even worse performance than the baseline in the case of JPEG compression and blur noise [57].

Chapter 7

Conclusion

The Exponentiated Gradient update [45] is one of the main updates developed mainly in the on-line learning context. It is based on trading off the loss with a relative entropy to the last weight vector instead of the squared Euclidean distance used for motivating Gradient Descent. A large number of techniques have been theoretically analyzed for enhancing the EG update such as specialist experts [26], capping the weights from above [87] and lower bounding the weights for handling shifting and sleeping experts [38, 13]. We explore some of these techniques experimentally for the purpose of reweighting examples with the goal of increasing noise robustness. Surprisingly, these methods work well experimentally for both supervised and unsupervised settings under a large variety of noise methods even though we introduce one additional weight per example. We found alternates to the specialist update and capping serving the same function as the original that work better experimentally. The modifications still clearly belong to the family of relative entropy motivated updates. In future work, we are also planning to explore techniques for shifting and sleeping experts.

Chapter 8

Contribution

This work has been done in collaboration with Ehsan Amid, and Hossein Talebi from Google AI, under the supervision of my advisor, Prof. Manfred Warmuth. A summary of our results appears as a preprint manuscript in [57].

Exponentiated Gradient is a method which was created by Prof. Manfred Warmuth in [45] and we adapted the algorithm for settings involving large number of examples. My contributions to this project are as follows:

1) I contributed to the main idea of how to use Exponentiated Gradient updates for both supervised and unsupervised settings. Specifically, I contributed to how to use EG for the mini-batch setting, specifically how to normalize the weights of the examples when there is a large number of examples in the training set and we only observe a small subset of the examples in a mini-batch.

2) I contributed to using regularization for the example weights, and I implemented the code for this method.

3) I implemented all of the experiments for PCA, including the implementation of our method and six other baseline methods.

4) I implemented and conducted experiments on small convolutional neural networks for MNIST/Fashion MNIST classification using Tensorflow. My collaborators, Ehsan Amid and Hossein Talebi, later used my code for running the experiments on large neural networks for ImageNet classification using Google resources since I did not have access to these resources as they are internal to Google.

5) I implemented the code for applying various types of noises on the data, such as feature noise and symmetric label noise.

6) Prof. Warmuth suggested using capping for the EG updates in the experiments. I implemented the function for capping the weights. Ehsan helped me debug the code.

7) I contributed to the idea of using a linear warm-up in the early stage, followed by a decay over time which improves the performance of EG.

8) I contributed to writing most parts of the paper [57]. I had written this thesis before we started writing the paper.

Ehsan Amid contributed to the main idea and theoretical parts, such as using capping and regularization for EG. He conducted the experiments for ImageNet classification with large neural networks. He advised me on writing the paper.

Hossein Talebi joined the effort around January 2021. He helped Ehsan set up the large-scale experiments and debug the code to run them using Google resources.

Prof. Warmuth, as my advisor, helped me with framing the overall problem

and providing feedback on the write-up.

Appendix A

Hyperparameter Tuning

In this section, we provide more details about the hyperparameter tuning for the experiments.

A.1 Noisy PCA Experiments

In R2DPCA and capped R2DPCA the reduced dimensionalities are set to $d_1 = 5$ and $d_2 = 5$, and in all other methods the reduced dimensionality is $k = d_1 \times d_2 = 25$. In capped R2DPCA, for parameter ϵ, we perform a grid-search on values $\{10, 20, \ldots, 50\}$. The integer parameter in RWL-AN is set to $r_s \times n$, where r_s is the ratio parameter searched in a grid of $\{0.05, 0.1, ..., 0.95\}$. As there is no warm-up phase for the PCA problem, we consider a learning rate of the form η_0/t^{α} where η_0 is the initial learning rate and α is tuned in the range $\{0.6, 0.65, ..., 0.95\}$. The initial earning rate for EG is tuned in a grid of $\{10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 1\}$. For capped EGR-PCA and regularized EGR-PCA, in addition to the learning rate and the decay factor α, we tune the capping ratio in a set of $\{0.2, 0.25, ..., 0.9\}$ and regularization factor in a set of $\{0.1, 0.15, ..., 0.95\}$, respectively.

The hyperparameters chosen for methods are as follows:

1. For RWL-AN, $r_s = 0.65$ for UMIST(blur), $r_s = 0.6$ for UMIST(random), $r_s = 0.8$ for UMIST(occlusion), $r_s = 0.25$ for AT&T(blur), $r_s = 0.95$ for AT&T(occlusion), and $r_s = 0.45$ for AT&T(random)

2. For capped R2DPCA, $\epsilon = 10$ for all kinds of noise on AT&T, $\epsilon = 50$ for UMIST(occlusion), $\epsilon = 20$ for UMIST(blur) and UMIST(random)

3. For EGR-PCA, $(\eta_0, \alpha) = (0.01, 0.7)$ for UMIST(blur), $(\eta_0, \alpha) = (0.01, 0.95)$ for UMIST(random), $(\eta_0, \alpha) = (0.01, 0.6)$ for UMIST(occlusion), $(\eta_0, \alpha) = (0.01, 0.6)$ for AT&T(blur), $(\eta_0, \alpha) = (0.032, 0.8)$ for AT&T(random), $(\eta_0, \alpha) = (0.032, 0.6)$ for AT&T(occlusion).

4. For Capped EGR-PCA, $(\text{capping-ratio}, \eta_0, \alpha) = (0.25, 0.01, 0.75)$ for UMIST (blur), $(\text{capping-ratio}, \eta_0, \alpha) = (0.45, 0.32, 0.75)$ for UMIST(random), (capping-ratio, $\eta_0, \alpha) = (0.45, 0.01, 0.8)$ for UMIST(occlusion), (capping-ratio, $\eta_0, \alpha) = (0.45, 0.01, 0.6)$ for AT&T(blur), $(\text{capping-ratio}, \eta_0, \alpha) = (0.35, 0.1, 0.85)$ for AT&T(random), $(\text{capping-ratio}, \eta_0, \alpha) = (0.4, 0.032, 0.6)$ for AT&T (occlusion).

5. For regularized EGR-PCA, $(\text{reg-factor}, \eta_0, \alpha) = (0.15, 0.1, 0.68)$ for UMIST (blur), $(\text{reg-factor}, \eta_0, \alpha) = (0.5, 0.1, 0.95)$ for UMIST(random), (reg-factor, η_0,

$\alpha) = (0.35, 0.032, 0.65)$ for UMIST(occlusion), (reg-factor, $\eta_0, \alpha) = (0.5, 0.32,$ $0.9)$ for AT&T(blur), (reg-factor, $\eta_0, \alpha) = (0.45, 0.32, 0.95)$ for AT&T(random), (reg-factor, $\eta_0, \alpha) = (0.35, 0.1, 0.6)$ for AT&T(occlusion).

A.2 Noisy Imagenet Experiments

In noisy Imagenet experiments we tune hyperparameters in the suggested range for all methods: for bi-tempered loss, we tune t_1 and t_2 in the range $[0.8, 0.99]$, $[1.05, 1.2]$, respectively. For DCL, we set set the regularizer to $5e$-4 and tune the scale learning rate for the examples in the range $[0.01, 1.0]$ and set the momentum parameter to 0.9. We tune the learning rate and the regularizer factor r as in the previous section and use a linear ramp-up until 20 epochs followed by a decay of 0.9 every 30 epochs. For the combined EG additional Reweighting + bi-tempered loss, we do not perform any tuning and simply combine the best performing hyperparameters for each case. The list of hyperparameters for different methods are as follows: (1) Bi-tempered loss: $(t_1, t_2) = (0.92, 1.1)$ for label noise and $(t_1, t_2) = (0.95, 1.05)$ for JPEG compression and blur noise experiments. We start all temperatures from 1.0 (i.e. cross entropy with softmax) and linearly interpolate to the final value in the first over the first 10 epochs. (2) DCL: we set the example scale learning rate to 0.05 in all experiments (all the non-trivial values of the learning rate deteriorated the performance compared to the baseline for the JPEG compression and blur noise). We do not apply any scaling based on the class. (3) EG Reweighting: we set the learning rate and the regularizer to 0.1 and 0.9, respectively, for label noise, 0.02 and 1.0 for JPEG compression noise, and 0.05 and 1.0 for blur noise.

Bibliography

[1] Görkem Algan and Ilkay Ulusoy. Label noise types and their effects on deep learning. *arXiv preprint arXiv:2003.10471*, 2020.

[2] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[3] Ehsan Amid, Rohan Anil, Christopher Fifty, and Manfred K. Warmuth. Step-size adaptation using exponentiated gradient updates. In *Workshop on "Beyond first-order methods in ML systems" at the 37th International Conference on Machine Learning (ICML)*, 2020.

[4] Ehsan Amid, Rohan Anil, and Manfred K. Warmuth. LocoProp: Enhancing BackProp via local loss optimization. *arXiv preprint arXiv:2106.06199*, 2021.

[5] Ehsan Amid and Manfred K. Warmuth. TriMap: Large-scale Dimensionality Reduction Using Triplets. *arXiv preprint arXiv:1910.00204*, 2019.

[6] Ehsan Amid and Manfred K Warmuth. An implicit form of Krasulina's k-PCA

update without the orthonormality constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3179–3186, 2020.

[7] Ehsan Amid, Manfred K. Warmuth, Rohan Anil, and Tomer Koren. Robust bi-tempered logistic loss based on Bregman divergences. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS, 2019.

[8] Ehsan Amid, Manfred K. Warmuth, and Sriram Srinivasan. Two-temperature logistic regression based on the tsallis divergence. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 2388–2396. PMLR, 2019.

[9] Ehsan Amid and Manfred K. K Warmuth. Reparameterizing mirror descent as gradient descent. In *Advances in Neural Information Processing Systems*, volume 33, pages 8430–8439, 2020.

[10] Rizwan Ahmed Ansari and Krishna Mohan Buddhiraju. Noise filtering in high-resolution satellite images using composite multiresolution transforms. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 86(5):249–261, 2018.

[11] Raghav Bansal, G. Raj, and T. Choudhury. Blur image detection using lapla-

cian operator and open-cv. *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*, pages 63–67, 2016.

[12] Yoshua Bengio, J. Louradour, Ronan Collobert, and J. Weston. Curriculum learning. In *ICML '09*, 2009.

[13] O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. *J. of Machine Learning Research*, 3(Nov):363–396, 2002.

[14] Xinlei Chen and Abhinav Gupta. Webly supervised learning of convolutional networks. *CoRR*, abs/1505.01554, 2015.

[15] Monojit Choudhury, Kalika Bali, Sunayana Sitaram, and Ashutosh Baheti. Curriculum design for code-switching: Experiments with language identification and language modeling with deep neural networks. In *Proceedings of the 14th International Conference on Natural Language Processing (ICON-2017)*, pages 65–74, Kolkata, India, December 2017. NLP Association of India.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine Learning in Finance*. Springer, 2020.

[19] Rajmadhan Ekambaram, Dmitry B Goldgof, and Lawrence O Hall. Finding label noise examples in large scale datasets. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2420–2424. IEEE, 2017.

[20] Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. Deep learning-enabled medical computer vision. *NPJ Digital Medicine*, 4(1):1–9, 2021.

[21] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017.

[22] Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. *CoRR*, abs/1805.03643, 2018.

[23] Vitaly Feldman, Venkatesan Guruswami, Prasad Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. *SIAM Journal on Computing*, 41(6):1558–1590, 2012.

[24] David Flatow and Daniel Penner. On the robustness of convnets to training on noisy labels, 2017.

[25] Y. Freund and R. E. Schapire. A Decision Theoretic Generalization of On-Line

Learning and an Application to Boosting. In *Second European Conference on Computational Learning Theory (EuroCOLT-95)*, pages 23–37, 1995.

[26] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and combining predictors that specialize. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 334–343. ACM, 1997.

[27] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[28] Aritra Ghosh, Himanshu Kumar, and PS Sastry. Robust loss functions under label noise for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[29] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[30] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *CoRR*, abs/1704.03003, 2017.

[31] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R. Scott, and Dinglong Huang. Curriculumnet: Weakly supervised learning from large-scale web images. *CoRR*, abs/1808.01097, 2018.

[32] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand

for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[33] Babak Hassibi, Ali H Sayed, and Thomas Kailath. H^{∞} optimality of the LMS algorithm. *IEEE Transactions on Signal Processing*, 44(2):267–280, 1996.

[34] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[37] James B Heaton, Nick G Polson, and Jan Hendrik Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.

[38] M. Herbster and M. K. Warmuth. Tracking the best expert. *Journal of Machine Learning*, 32(2):151–178, August 1998.

[39] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[40] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016.

[41] Lu Jiang, Deyu Meng, Teruko Mitamura, and Alexander G. Hauptmann. Easy samples first: Self-paced reranking for zero-example multimedia search. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, page 547–556, New York, NY, USA, 2014. Association for Computing Machinery.

[42] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[43] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Regularizing very deep neural networks on corrupted labels. *CoRR*, abs/1712.05055, 2017.

[44] Ian Jolliffe. *Principal Component Analysis*. 2011.

[45] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[46] Jyrki Kivinen, Manfred K Warmuth, and Babak Hassibi. The p-norm generalization of the LMS algorithm for adaptive filtering. *IEEE Transactions on Signal Processing*, 54(5):1782–1793, 2006.

[47] Wouter M Koolen, Dmitry Adamskiy, and Manfred K Warmuth. Putting bayes to sleep. In *NIPS*, pages 135–143, 2012.

[48] M. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[49] Y. J. Lee and K. Grauman. Learning the easy things first: Self-paced visual category discovery. In *CVPR 2011*, pages 1721–1728, 2011.

[50] Siyang Li, Xiangxin Zhu, Qin Huang, Hao Xu, and C.-C. Jay Kuo. Multiple instance curriculum learning for weakly supervised object detection. *CoRR*, abs/1711.09191, 2017.

[51] Jian Liang, Zhihang Li, Dong Cao, Ran He, and Jingdong Wang. Self-paced cross-modal subspace matching. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 569–578, New York, NY, USA, 2016. Association for Computing Machinery.

[52] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[53] Sheng Liu, Jonathan Niles-Weed, Narges Razavian, and Carlos Fernandez-

Granda. Early-learning regularization prevents memorization of noisy labels. *arXiv preprint arXiv:2007.00151*, 2020.

[54] Sheng Liu, Jonathan Niles-Weed, Narges Razavian, and Carlos Fernandez-Granda. Early-learning regularization prevents memorization of noisy labels, 2020.

[55] Yuan Liu, Ayush Jain, Clara Eng, David H Way, Kang Lee, Peggy Bui, Kimberly Kanada, Guilherme de Oliveira Marinho, Jessica Gallegos, Sara Gabriele, et al. A deep learning system for differential diagnosis of skin diseases. *Nature Medicine*, 26(6):900–908, 2020.

[56] Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

[57] Negin Majidi, Ehsan Amid, Hossein Talebi, and Manfred K. Warmuth. Exponentiated gradient reweighting for robust training under label noise and beyond. *arXiv preprint arXiv:2104.01493*, 2021.

[58] Juan Carlos Mier, Eddie Huang, Hossein Talebi, Feng Yang, and Peyman Milanfar. Deep perceptual image quality assessment for compression. *arXiv preprint arXiv:2103.01114*, 2021.

[59] Naila Murray, Luca Marchesotti, and Florent Perronnin. AVA: A large-scale database for aesthetic visual analysis. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2408–2415. IEEE, 2012.

[60] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol

Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[61] Te Pi, Xi Li, Zhongfei Zhang, Deyu Meng, Fei Wu, Jun Xiao, and Yueting Zhuang. Self-paced boost learning for classification. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 1932–1938. AAAI Press, 2016.

[62] Matt Poyser, Amir Atapour-Abarghouei, and Toby P Breckon. On the impact of lossy image and video compression on the performance of deep convolutional neural network architectures. *arXiv preprint arXiv:2007.14314*, 2020.

[63] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[64] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[65] David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medi-*

cal image computing and computer-assisted intervention, pages 234–241. Springer, 2015.

[67] F. S. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*, pages 138–142, 1994.

[68] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, 2016.

[69] Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. Data parameters: A new family of parameters for learning a differentiable curriculum. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[70] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[71] Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS workshop on cost-sensitive learning*, volume 1. Vancouver, CA:, 2008.

[72] Jonathon Shlens. A tutorial on principal component analysis, 2005.

[73] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Baby steps: How

"less is more" in unsupervised dependency parsing. In *NIPS 2009 Workshop on Grammar Induction, Representation of Language and Language Learning*, 2009.

[74] James S. Supancic, III and Deva Ramanan. Self-paced learning for long-term tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[75] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. Automatic curriculum graph generation for reinforcement learning agents. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017.

[76] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[77] Hossein Talebi and Peyman Milanfar. Learned perceptual image enhancement. In *2018 IEEE international conference on computational photography (ICCP)*, pages 1–13. IEEE, 2018.

[78] Hossein Talebi and Peyman Milanfar. NIMA: Neural image assessment. *IEEE Transactions on Image Processing*, 27(8):3998–4011, 2018.

[79] Kevin Tang, Vignesh Ramanathan, Li Fei-fei, and Daphne Koller. Shifting weights: Adapting object detectors from image to video. In F. Pereira, C. J. C.

Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[80] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. *arXiv preprint arXiv:2007.00644*, 2020.

[81] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2691–2699, 2015.

[82] Joseph Turian, Lev Ratinov, and Yoshua 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*, ACL '10, page 384–394, USA, 2010. Association for Computational Linguistics.

[83] Xinshao Wang, Yang Hua, Elyor Kodirov, and Neil M Robertson. Imae for noise-robust learning: Mean absolute error does not treat examples equally and gradient magnitude's variance matters. *arXiv preprint arXiv:1903.12141*, 2019.

[84] Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi, and James Bailey. Symmetric cross entropy for robust learning with noisy labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 322–330, 2019.

[85] Manfred K. Warmuth and Dima Kuzmin. Randomized online PCA algorithms

with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9(75):2287–2320, 2008.

[86] Manfred K. Warmuth and Dima Kuzmin. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9(75):2287–2320, 2008.

[87] Manfred K Warmuth and Dima Kuzmin. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9(Oct):2287–2320, 2008.

[88] Harry Wechsler, Jonathon P Phillips, Vicki Bruce, Francoise Fogelman Soulie, and Thomas S Huang. *Face recognition: From theory to applications*, volume 163. Springer Science & Business Media, 2012.

[89] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Learning to teach with dynamic loss functions, 2018.

[90] Xiaobo Xia, Tongliang Liu, Bo Han, Chen Gong, Nannan Wang, Zongyuan Ge, and Yi Chang. Robust early-learning: Hindering the memorization of noisy labels. In *International Conference on Learning Representations*, 2021.

[91] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[92] Yilun Xu, Peng Cao, Yuqing Kong, and Yizhou Wang. L_dmi: A novel

information-theoretic loss function for training deep nets robust to label noise. In *NeurIPS*, pages 6222–6233, 2019.

[93] Yan Yan, Rómer Rosales, Glenn Fung, Ramanathan Subramanian, and Jennifer Dy. Learning from multiple annotators with varying expertise. *Machine learning*, 95(3):291–327, 2014.

[94] Rui Zhang, Feiping Nie, and Xuelong Li. Auto-weighted two-dimensional principal component analysis with robust outliers. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6065–6069. IEEE, 2017.

[95] Rui Zhang, Feiping Nie, and Xuelong Li. Auto-weighted two-dimensional principal component analysis with robust outliers. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pages 6065–6069. IEEE, 2017.

[96] Rui Zhang and Hanghang Tong. Robust principal component analysis with adaptive neighbors. *NeuIPS*, 2019.

[97] Rui Zhang and Hanghang Tong. Robust principal component analysis with adaptive neighbors. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[98] Yu Zhang, James Qin, Daniel S Park, Wei Han, Chung-Cheng Chiu, Ruoming

Pang, Quoc V Le, and Yonghui Wu. Pushing the limits of semi-supervised learning for automatic speech recognition. *arXiv preprint arXiv:2010.10504*, 2020.

[99] Zhilu Zhang and Mert R Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv preprint arXiv:1805.07836*, 2018.

[100] Qian Zhao, Deyu Meng, Lu Jiang, Qi Xie, Zongben Xu, and Alexander G. Hauptmann. Self-paced learning for matrix factorization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 3196–3202. AAAI Press, 2015.