Explicit Regularization for Overparameterized Models

by

Tiffany Y. Huang

B.S. Computer Science and Engineering, Physics
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by:	Tiffany Y. Huang Department of Electrical Engineering and Computer Science May 19, 2023
Certified by:	Navid Azizan Edgerton Assistant Professor of Mechanical Engineering Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Explicit Regularization for Overparameterized Models

by

Tiffany Y. Huang

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In many learning problems, it is desirable to incorporate explicit regularization in the objective to avoid overfitting the data. Typically, the regularized objective is solved via weight decay. However, optimizing with weight decay can be challenging because we cannot tell if the solution has reached a global minimum. Further, weight decay can have large run-to-run variations and is sensitive to the choice of regularization hyperparameter. To this end, we propose a new approach to optimize objectives with explicit regularization, called Regularizer Mirror Descent (RMD). In the overparameterized regime, where the number of model parameters exceeds the size of data, RMD provably converges to a point "close" to a minimizer of the regularized objective. Additionally, RMD is computationally efficient and imposes virtually no overhead to standard gradient descent. We observe that RMD is remarkably robust and consistent compared to gradient descent with weight decay despite solving for the same objective. We also illustrate the practical utility of RMD by applying it to learning problems with corrupted labels, where it can match or outperform the state-of-the-art methods without requiring additional hyperparameter tuning or ad-hoc heuristics tailored for this task.

Thesis Supervisor: Navid Azizan
Title: Edgerton Assistant Professor of Mechanical Engineering

Acknowledgments

I would like express my deepest thanks to the following people and groups, without which the completion of this thesis would have never been possible:

First, I extend gratitude toward Professor Navid Azizan, who welcomed me into his lab at the beginning of 2022. In this past year and a half, Navid's boundless kindness, patience, intelligence, and expertise have enabled me to carry out this research to the best of my ability. I've learned so much working under him, from the basics of mirror descent to how to frame the story around a research project, and I cannot thank him enough.

I also thank Haoyuan Sun for his guidance over this year and a half. Without him, I would have lacked the theoretical and practical foundations needed to tackle this project. I must also mention Hao's vast experience in writing and revising technical papers, which have been deeply appreciated as I put this thesis together.

I would also like to thank Sahin Lale. Though we met only recently, the insights Sahin provided about how to best present this work, from the experiments to the structure of the paper, have been invaluable to pulling this thesis together.

I also give thanks to the rest of the Azizan Lab. It has been so great meeting each person in the lab, each of whom is so sharp and talented. Everyone has welcomed me with kindness, and their weekly presentations have introduced me to so many topics that I otherwise would have never gained familiarity with.

I also acknowlege the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing computing resources that have contributed to the experiments performed for this thesis. I similarly acknowledge the MIT Physics Department and the SubMIT computing cluster – an extra source of computing resources is always very much appreciated.

Last but not least, I thank the loved ones in my life: my parents, brother, and aunt for the constant support and encouragement. The friends I met through MIT, whether that was recently or early on in college, through McCormick Hall or Site 4 or other means. Thank you for these exciting few years and for always being there as a source of support. I have truly grown so much having known you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A key problem in machine learning is to impose some sort of capacity control on the learned models so that they do not *overfit* to the training data and perform poorly at test time. To this end, regularization is a common way to find "simpler" solutions and avoid overfitting. One straightforward way to achieve this is through *explicit regularization*, i.e., modifying the objective of the optimization problem by adding an explicit penalty term to it [22, 15, 26].

Over the past few years, regularization has received renewed attention, particularly in the context of highly *overparameterized* models in deep learning, i.e., models that have a large enough capacity to allow them to perfectly (over)fit the training data. Interestingly, such models sometimes perform well in the absence of any explicit regularization, and it has been postulated that this is due, at least in part, to the *implicit regularization* effects induced by other components such as the learning algorithm (also referred to as the optimization algorithm) [52, 6, 9, 8, 38, 7]. In particular, it has been argued that many first-order methods, such as gradient descent and mirror descent, tend to exhibit implicit bias towards converging to solutions that minimize a penalty while interpolating the training data [16, 3, 40, 10, 45, 44].

Since implicit regularization would still lead to a solution that perfectly fits the training data, it may not be sufficient to avoid overfitting in many scenarios, such as when the data contains a lot of noise or corruption. Therefore, it is often desirable to use explicit regularization to control the strength and properties of the desired regularization directly. For

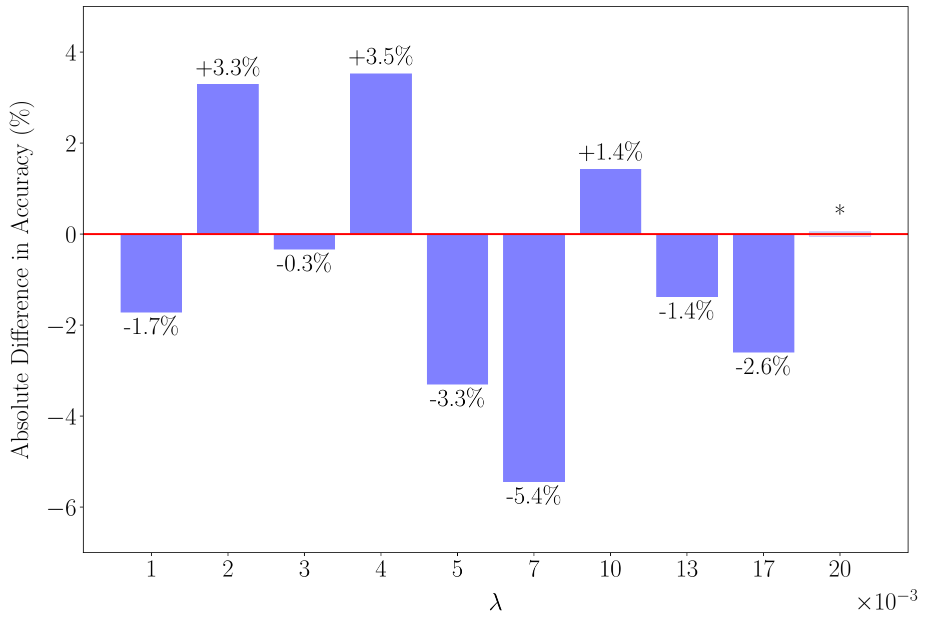


Figure 1-1: Comparison of the test accuracies achieved by training with two different initializations of the ResNet-18 model using SGD with weight decay on the CIFAR-10 dataset with 40% corrupted labels for a range of regularization strengths denoted by the parameter λ. Both initializations are centered around the all-zero weight vector and the absolute difference between the test accuracies achieved are reported for each λ. The asterisk (*) indicates a case where weight decay failed to converge with one initialization but not the other. These results highlight the difficulty and inconsistency of training with weight decay.

example, one common form of explicit regularization is ridge (Tikhonov) regularization, whose objective penalizes the ℓ_2-norm of the parameters. For gradient descent and its variants, ridge regularization is typically implemented via *weight decay*. However, optimizing with weight decay can be challenging. For overparameterized models, while achieving zero loss in the unregularized problem implies convergence to a global minimum, adding explicit regularization creates uncertainty around the value of the minimum, making it difficult to verify the convergence of the algorithm. The following case study illustrates the difficulty of deploying weight decay in practical scenarios.

1.1 Case Study: Weight Decay

Consider training ResNet-18 [20] with weight decay on the CIFAR-10 dataset [25] with 40% of the data points corrupted, i.e., the labels are randomly flipped (see Chapter 6 for details). Fig. 1-1 presents the difference in test accuracy when ResNet-18 is initialized at two different sets of weight vectors around the all-zero weight vector. Further details about the experimental setup are discussed in Chapter 6. The results demonstrate that the training is unstable and highly sensitive to initialization. There are significant discrepancies in the test accuracy achieved by the two initializations, and weight decay fails to converge with the second initialization for regularization strength $\lambda = 0.02$, despite successfully converging with the first.

This experiment highlights the inconsistency of weight decay and its potential to hinder the search for the best-performing model. To address this issue, in this work, we develop a novel method for optimization with explicit regularization that comes with guarantees and more reliable convergence behavior.

1.2 Contributions

1. We introduce a novel approach for training overparameterized models with explicit regularization, called *Regularizer Mirror Descent (RMD)*. By leveraging the properties of overparameterization and the implicit regularization of mirror descent (MD) algorithms, RMD provably converges to a point "close" to the minimizer of the explicitly regularized cost, i.e., the sum of the empirical loss and the explicit regularization penalty. This makes RMD the first explicit regularization method for highly overparameterized nonlinear models with theoretical convergence guarantees.

2. Besides the desirable theoretical guarantees, RMD is both computationally and memory-wise efficient. It incurs virtually no additional overhead compared to gradient descent and can easily scale to large problem instances. We define a mini-batch version of RMD that can be used in practice and provide a sample implementation of RMD that is easy to integrate with little modification to existing training procedures.

3. Compared to traditional methods such as weight decay, RMD is more versatile since it can accommodate any strictly convex function of the weights as an explicit

regularizer. Furthermore, in the case of ℓ_2 (ridge) regularization, we can observe that RMD is significantly more consistent than weight decay. In particular, RMD is less sensitive to random initialization and exhibits consistent and predictable changes as the regularization strength varies.

4. We illustrate the effectiveness of RMD through a case study in learning with corrupted/noisy labels. Our experiments on the CIFAR-10 dataset with varying levels of corruption show that RMD outperforms or matches the state-of-the-art methods in the literature. Notably, this achievement is primarily due to the general and principled approach of RMD for explicit regularization. Unlike prior methods, RMD does not require tuning of additional hyperparameters or ad-hoc heuristics tailored to this specific problem. The simplicity and effectiveness of RMD highlight its potential for training other overparameterized machine learning models.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 surveys related works that characterize the current understanding of regularization in machine learning and describes the state-of-the-art for learning with corrupted labels. Chapters 3 and 4 provide important background, with general terminology defined in Chapter 3 and a deep dive into mirror descent and its convergence properties in Chapter 4.

We present our novel algorithm RMD and its various special cases and implementation in Chapter 5. In Chapter 6, we evaluate the empirical performance of RMD by applying it to deep neural network (DNN) training, where we show that RMD is a superior alternative to weight decay and demonstrate its utility in practical applications. Finally, in Chapter 7, we prove the convergence guarantees of RMD, building on the implicit regularization properties of mirror descent detailed earlier in Chapter 4.

Chapter 8 concludes this thesis, describing key takeaways and proposing open questions to be addressed by future work.

Chapter 2

Related Work

There is an array of regularization techniques that are deployed in conjunction with the training procedures of models in machine learning [54, 21, 35, 27, 51, 37, 23, 43, 28]. See, e.g., [15, 26] for a survey. Since our work is focused on explicit regularization, which involves adding a regularization term to the objective of the optimization problem, the most closely related method to our setting is weight decay [53], which adds an ℓ_2-norm regularizer to the objective. However, RMD is more general, as it can handle any desired strictly convex regularizer.

There is also the so-called implicit regularization property of various learning algorithms themselves, which have been extensively studied over the years, e.g., [44, 18, 47, 24, 45, 10]. Even though RMD builds on the recent literature on the implicit regularization behavior of a family of optimization algorithms called stochastic mirror descent [5], it significantly differs from the prior work due to its goal. RMD aims to provide explicit regularization, thus it tackles a fundamentally different problem. To achieve this goal, it reformulates the explicit regularization objective and turns it into a mirror descent algorithm which is theoretically guaranteed to converge to model parameters that are near a global minimizer of the explicitly regularized optimization problem. In Section 3.3 and Chapter 5, we further provide the relevant background on implicit regularization and how RMD utilizes it, respectively.

One of the key applications of explicit regularization is in the problem of learning with corrupted labels. There are several methods that have been proposed for tackling this problem. In particular, Mixup [54] and label smoothing [46, 31] have been shown to provide effective denoising against label corruption. Mixup constructs convex combinations of pairs of training samples, while label smoothing uses a weighted average of dataset labels and a uniform distribution over labels. Other methods that have been developed to address the problem of learning with noisy labels are robust early learning (CDR) [49], the Partially Huberised loss (PHuber) [34], and the T-revision method [50]. The CDR technique avoids overfitting by pruning parameters that are deemed uncritical, whereas PHuber is derived from a variant of gradient clipping that improves robustness to label noise. The T-revision method, on the other hand, builds on importance reweighting [30] by simultaneously learning the classifier and revising the transition matrix defining the probabilities of the occurrence of noisy labels. Compared to these methods, RMD does not require the use of ad-hoc heuristics and additional hyperparameters to tune, and instead, it achieves robustness via strictly solving an optimization problem designed for general explicit regularization.

Chapter 3

Background

In this chapter, we introduce concepts utilized in the remainder of this paper: gradient descent, explicit regularization, implicit regularization, and the Bregman divergence. Gradient descent is a standard method for optimization that serves as the foundation for many training procedures in machine learning. We define explicit regularization, a form of regularization that avoids overfitting to the training data by adding a penalty to the objective of the optimization. We also describe the implicitly regularizing properties that some optimization algorithms display, in particular, highlighting the implicit regularization of gradient descent and its generalization, mirror descent. The Bregman divergence is a notion of distance generalized to geometries beyond the standard Euclidean geometry that is important when discussing the convergence properties of learning algorithms.

3.1 Gradient Descent

Suppose we have the training dataset $\{(x_i, y_i) : i = 1, \dots, n\}$ where the inputs are $x_i \in \mathbb{R}^d$ and the labels are $y_i \in \mathbb{R}$. Let us define a model by the function $f(x, w)$ with weights $w \in \mathbb{R}^p$. Denote the output of the model on data point i as $f_i(w) := f(x_i, w)$. We can express the total loss on the training set as $L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w)$, where $L_i(w) = \ell(y_i, f_i(w))$ is the nonnegative loss on the ith data point for a weight vector $w \in \mathbb{R}^p$.

This total loss is typically minimized through gradient descent (GD) or one of its variants. Denoting the model parameters at the t-th time step by $w_t \in \mathbb{R}^p$, the update rule of GD can be simply written as

$$w_t = w_{t-1} - \eta \nabla L(w_{t-1}), \quad t \geq 1, \tag{3.1}$$

where η is the so-called step size or learning rate, w_0 is the initialization of the weights, and $\nabla L(\cdot)$ is the gradient of the total loss $L(w)$. In the overparameterized regime where there are many more parameters than the number of data points ($p \gg n$), applying GD often can achieve (near) zero training error and "interpolate" the training data [52, 33].

In practice, stochastic gradient descent (SGD) [42], in which only a subset of data points contribute to the gradient at each iteration, is most often used for its computational efficiency, and much of what we know about GD also translates well to SGD. Specifically, we define the "mini-batch" version of GD, whose update is written

$$w_t = w_{t-1} - \frac{\eta}{|B|} \sum_{i \in B} \nabla L(w_{t-1}), \quad t \geq 1, \tag{3.2}$$

where B denotes the subset of data points selected.

3.2 Explicit Regularization

It is often preferable to find a "simpler" solution instead of (over)fitting the training data to zero error. One direct way to achieve this goal is explicit regularization, where we augment the loss function with a strictly convex and differentiable regularizer $\psi : \mathbb{R}^p \to \mathbb{R}$ that penalizes a solution that is deemed too "complex," and consider

$$\boxed{\min_w \; L(w) + \lambda \psi(w),} \tag{3.3}$$

where $\lambda \geq 0$ is a hyperparameter that controls the strength of regularization relative to the loss function. A simple and common choice of regularizer is $\psi(w) = \frac{1}{2}\|w\|^2$. In this case, when GD is applied to (3.3), it is commonly referred to as weight decay.

Note that the smaller λ is, the more effort in the optimization is spent on minimizing

$L(w) = \frac{1}{n}\sum_{i=1}^{n} L_i(w)$. Since the losses $L_i(\cdot)$ are non-negative, the lowest these terms can get is zero. This is attainable in the overparameterized regime. As a result, for highly overparameterized models and in the limit where $\lambda \to 0$, the problem would be equivalent to the following:

$$\min_{w} \ \psi(w) \qquad \text{s.t.} \quad L_i(w) = 0, \forall\, i = 1, \dots, n. \tag{3.4}$$

3.3 Implicit Regularization

In recent analysis of overparameterization, it has been noted in several papers that, even *without* imposing any explicit regularization in the objective, i.e., by optimizing only the loss function $L(w)$ over a highly overparameterized model, many optimization algorithms would converge to a minimizer with certain properties [16, 17, 3, 32, 41]. This is referred to as the implicit regularization of optimization algorithms, as the choice of algorithm constrains the solution according to its convergence properties, without the use of a modified objective.

For example, when initialized at the origin, GD with sufficiently small step size tends to converge to interpolating solutions with minimum ℓ_2-norm [14, 16], i.e.,

$$\min_{w} \ \|w\|_2 \qquad \text{s.t.} \ L_i(w) = 0, \ \forall\, i = 1, \dots, n.$$

More generally, it has been shown [16, 5] that mirror descent (MD), whose update rule is defined for a differentiable strictly convex "potential function" $\psi(\cdot)$ as

$$\nabla\psi(w_t) = \nabla\psi(w_{t-1}) - \eta\nabla L(w_{t-1}),$$

with proper initialization and a sufficiently small learning rate tends to converge to the solution of

$$\boxed{\min_{w} \ \psi(w) \qquad \text{s.t.} \ L_i(w) = 0, \ \forall\, i = 1, \dots, n.} \tag{3.5}$$

Note that this is equivalent to the case of explicit regularization with $\lambda \to 0$, i.e., problem (3.4). As our proposed method leverages the properties of mirror descent in the overparameterized regime, we describe both the mirror descent algorithm and this result more precisely in Chapter 4.

3.4 Bregman Divergence

Before we enter a detailed introduction of mirror descent and its convergence properties, we define the Bregman divergence [12]. For a strictly convex, differentiable function $\psi(\cdot)$, the Bregman divergence is defined as

$$D_\psi(x, y) := \psi(x) - \psi(y) - \nabla\psi(y)^T(x - y), \quad \forall\, x, y \in \mathbb{R}^p. \tag{3.6}$$

$D_\psi(x, y)$ defines a notion of distance with respect to $\psi(\cdot)$.

Chapter 4

Mirror Descent

This chapter introduces mirror descent, a generalization of the popularly-used gradient descent (GD) algorithm, first introduced by Nemirovski and Yudin [39]. Through the choice of a potential function, which we discuss below, MD is able to exploit different geometries of the problem to arrive at different solutions. As discussed in Section 3.3, mirror descent has convergence guarantees that define how it implicitly regularizes overparameterized models.

4.1 The Mirror Descent Algorithm

Suppose we are in the same problem setting as described in Section 3.1. Given a strictly convex differentiable function, $\psi(\cdot) : \mathbb{R}^p \to \mathbb{R}$, which we call the *potential function*, and step size $\eta > 0$, the update rule of MD is defined as follows

$$\boxed{\nabla\psi(w_t) = \nabla\psi(w_{t-1}) - \eta\nabla L(w_{t-1}).} \tag{4.1}$$

$\nabla\psi(\cdot)$ defines a "mirrored" domain in which we perform updates. Because $\psi(\cdot)$ is strictly convex, we can compute a unique w_t at each iteration t:

$$w_t = \nabla\psi^{-1}\left(\nabla\psi(w_{t-1}) - \eta\nabla L(w_{t-1})\right).$$

Like with gradient descent, because the computation of $L(w)$ can be unwieldy and computationally expensive, we can introduce stochastic mirror descent (SMD). In particular, we write the update rule of mini-batch MD, which is analogous to the mini-batch update of GD given in (3.2).

$$\boxed{\nabla\psi(w_t) = \nabla\psi(w_{t-1}) - \frac{\eta}{|B|}\sum_{i\in B}\nabla L_i(w_{t-1}),} \tag{4.2}$$

where B is the set of data points in the chosen mini-batch. Note that because the loss is averaged across mini-batch B, the magnitude of the update at each step is batch size invariant.

4.2 Convergence Properties of MD

We outline the convergence properties of MD, as presented and proven in [5]. Suppose the per sample loss function is of the form $L_i(w) = \ell(y_i - f_i(w))$, where $\ell(\cdot)$ is convex and has global minimum at 0 (one example is the square loss). Given a highly overparameterized model $f(x,w)$, we define the set of interpolating solutions

$$\begin{aligned} \mathcal{W} &= \{w \in \mathbb{R}^p \mid f_i(w) = y_i, \quad i = 1, \dots, n\} && (4.3)\\ &= \{w \in \mathbb{R}^p \mid L_i(w) = 0, \quad i = 1, \dots n\}. && (4.4) \end{aligned}$$

We also define the interpolating solution $w^* \in \mathcal{W}$ that is closest to the initialization w_0 in Bregman divergence

$$w^* = \underset{w\in\mathcal{W}}{\arg\min}\ D_\psi(w, w_0). \tag{4.5}$$

Note that when w is initialized to $w_0 = \arg\min_w\ \psi(w)$, this reduces to

$$\boxed{w^* = \underset{w\in\mathcal{W}}{\arg\min}\ \psi(w).} \tag{4.6}$$

Note that (3.5) and (4.6) are equivalent.

It has been shown in [3] that for linear models, $f(x, w) = x^T w$ initialized at w_0, the iterates of MD with potential function $\psi(\cdot)$, converge to w^*, given that the learning rate η is sufficiently small. For nonlinear models, this still holds in the approximate sense, under certain conditions [5]. More formally, we first define $D_{L_i}(w, w') := L_i(w) - L_i(w') - \nabla L_i(w')^T(w - w')$, which is analogous to the Bregman divergence but defined for loss functions. However, note that $L_i(\cdot) = \ell(y_i - f_i(w))$ does not need to be convex due to the nonlinearity of $f(\cdot, \cdot)$. We also denote the Hessian of $f_i(\cdot)$ by H_{f_i}.

In determining the convergence properties of MD in the overparameterized regime, [5] proposes the following assumptions.

Assumption 4.2.1. Denote the initial point by w_0. There exists $w \in \mathcal{W}$ and a region $\mathcal{B} = \{w' \in \mathbb{R}^d \mid D_\psi(w, w') \leq \epsilon\}$ containing w_0, such that $D_{L_i}(w, w') \geq 0, i = 1 \ldots, n$, for all $w' \in \mathcal{B}$.

Assumption 4.2.2. Consider the region $\mathcal{B}$ in Assumption 4.2.1. $f_i(\cdot)$ have bounded gradient and Hessian on the convex hull of $\mathcal{B}$, i.e. $||\nabla f_i(w')|| \leq \gamma$, and $\alpha \leq \lambda_{\min}(H_{f_i}(w')) \leq \lambda_{\max}(H_{f_i}(w')) \leq \beta, i = 1, \ldots, n$, for all $w' \in \operatorname{conv} \mathcal{B}$.

To summarize in more plain language, Assumption 4.2.1 states that w_0 is close to the set of global minima $\mathcal{W}$, which is a reasonable assumption to make in the overparameterized setting [1]. Assumption 4.2.2 states that the first and second derivatives of the model are locally bounded.

Given that these assumptions hold, [5] proves the following theorem.

Theorem 4.2.3 (Azizan et al. [5]). *Consider the set of interpolating solutions $\mathcal{W} = \{w \in \mathbb{R}^d \mid f_i(w) = y_i, i = 1, \ldots, n\}$, the closest such solution $w^* = \arg\min_{w \in \mathcal{W}} D_\psi(w, w_0)$, and the MD iterates given in* (4.1) *initialized at w_0, where every data point is revisited after some steps. Under Assumptions 4.2.1 and 4.2.2, for sufficiently small step size, i.e., for any $\eta > 0$ for which $\psi(\cdot) - \eta L_i(\cdot)$ is strictly convex on $\mathcal{B}$ for all i, the following holds.*

1. *The iterates converge to $w_\infty \in \mathcal{W}$.*

2. *$D_\psi(w^*, w_\infty) = o(\epsilon)$.*

In other words, MD converges to the interpolating solution that is "close" to w^*, which is itself the global minimum (among potentially many minima) that minimizes the Bregman divergence to the initialization. When $w_0 = \arg\min_w \psi(w)$, this global minimum can be described as the "smallest" with respect to the geometry of the problem, as can be seen in the reduction in (4.6).

4.3 q-norm GD

One important family of mirror descent algorithms is mirror descent with a potential function of the form $\psi(\cdot) = \frac{1}{q}||\cdot||_q^q$, where $q > 1$, i.e., the potential function is chosen to be the ℓ_q-norm. Note that standard gradient descent (GD) is a special case of this class of MD where $\psi(\cdot) = \frac{1}{2}||\cdot||^2$, or the ℓ_2-norm. We will denote these MD algorithms *q-norm GD*, or q-GD, for short.

By substituting this potential function into (4.1), we arrive at the update rule for q-GD:

$$\begin{aligned} w_t[j] = \left||w_{t-1}[j]|^{q-1}\operatorname{sign}(w_{t-1}[j]) - \eta\nabla L(w_{t-1})[j]\right|^{\frac{1}{q-1}} \times \\ \operatorname{sign}\left(|w_{t-1}[j]|^{q-1}\operatorname{sign}(w_{t-1}[j]) - \eta\nabla L(w_{t-1})[j]\right), \end{aligned} \tag{4.7}$$

where $w_t[j]$ denotes the j-th element of w_t, the weight vector at step t. Similarly, $\nabla L(w_{t-1})[j]$ denotes the j-th element of the gradient vector at step t. A stochastic update rule for q-GD can also be derived by substituting $\psi(\cdot) = \frac{1}{q}||\cdot||_q^q$ into (4.1).

As highlighted in [5], this particular choice of potential for mirror descent is notable because it leads to an update rule that is *separable* in coordinates. In other words, only the j-th element of w and $\nabla L(w)$ will factor into the update for the j-th element of w. This means that q-GD can be implemented in a parallel and therefore efficient way, comparable to the computational overhead of GD.

Implementation

Listing 4.1 provides a sample implementation of q-GD in PyTorch as a proof-of-concept, demonstrating that q-GD is easily implemented. The `qnormSGD` class is an optimizer that

Listing 4.1: Sample PyTorch implementation of q-GD.

```python
import torch
from torch.optim import Optimizer

class qnormSGD(Optimizer):

    def __init__(self, params, lr=0.01, qnorm=2):
        if not 0.0 <= lr:
            raise ValueError(f'Invalid learning rate: {lr}')
        # q-norm must be strictly greater than 1
        if not 1.01 <= qnorm:
            raise ValueError(f'Invalid q-norm: {qnorm}')

        defaults = dict(lr=lr, qnorm=qnorm)
        super(qormSGD, self).__init__(params, defaults)

    def __setstate__(self, state):
        super(qnormSGD, self).__setstate__(state)

    def step(self, closure=None):
        loss = None
        if closure is not None:
            loss = closure()

        for group in self.param_groups:
            lr = group['lr']
            qnorm = group['qnorm']

            for p in group['params']:
                if p.grad is None:
                    continue
                x = p.data
                dx = p.grad.data

                # q-norm potential function
                update = torch.pow(torch.abs(x), qnorm-1) * \
                      torch.sign(x) - lr * dx
                p.data = torch.sign(update) * \
                      torch.pow(torch.abs(update), 1/(qnorm-1))

        return loss
```

can substitute for any existing PyTorch optimizer, and thus can be easily used with only very minor changes to existing code. This implementation was used in the remainder of the experiments in this thesis wherever q-GD was required (see Chapter 6).

Computational Overhead

To verify that we can train models with q-GD efficiently, we ran a simple benchmark of the runtime of q-GD against the standard PyTorch implementation of GD (`optim.SGD`). We train ResNet-18 [20] on the popular CIFAR-10 [25] dataset with a batch size of 128 and time how long it takes to train 10 and 20 epochs. These experiments were run on a single Nvidia V100 GPU. While we limit sources of nondeterminism[1] as much as possible to ensure the accuracy of these measurements, we note that these results may not be entirely precise due to the use of shared computing resources.

Table 4.1: Comparison of the training time of q-GD, as implemented in Listing 4.1, and the standard PyTorch implementation of GD (`optim.SGD`). The mean $\pm$ std. dev. of the training time for 10 and 20 epochs across six trials is reported.

	Training Time (seconds)	
Algorithm	**10 epochs**	**20 epochs**
$q = 1.1$	145.8 ± 3.5	291.7 ± 7.3
$q = 2$	146.9 ± 3.5	293.8 ± 7.8
$q = 3$	147.5 ± 5.8	293.0 ± 9.5
$q = 10$	146.2 ± 4.6	292.1 ± 8.5
GD	131.1 ± 3.4	261.3 ± 6.8

From Table 4.1, it can be observed that q-GD trains at similar speeds regardless of the value of q, which is expected given that the update steps for each value of q require the same number of operations. We also expect that q-GD should have little computational overhead over standard GD; although q-GD requires more arithmetic operations, this should not significantly affect training times because the computational bottleneck in training is differentiation. However, Table 4.1 shows us that q-GD trains approximately 10% slower than PyTorch's GD implementation, even for $q = 2$. We believe that this discrepancy is likely due to detailed optimization of the PyTorch implementation, and that, with similar care given to optimization of p-GD, this gap can be closed.

[1]We follow the steps provided in `https://pytorch.org/docs/stable/notes/randomness.html` to limit nondeterminism.

Chapter 5

Proposed Method: Regularizer Mirror Descent (RMD)

In this section, we propose a novel method for training overparameterized models with explicit regularization. This method builds upon mirror descent, which is discussed in detail in Section 4, and is termed ***Regularizer Mirror Descent (RMD)***.

5.1 Derivation of RMD

When it is undesirable to reach zero training error, e.g., due to a high amount of noise in the data, one cannot rely solely on the implicit bias of the optimization algorithm to avoid overfitting. That is because these algorithms would still converge to a solution with zero training error and interpolate the noise as well. This suggests the use of explicit regularization methods, such as weight decay, as defined in (3.3). Unfortunately, as shown in Figure 1-1, standard explicit regularization methods like weight decay, which simply employs GD to solve (3.3), have very inconsistent convergence behavior. Motivated by this issue, we propose a new algorithm called Regularizer Mirror Descent (RMD), which, under appropriate conditions, provably regularizes the weights for any desired differentiable, strictly convex regularizer. Specifically, RMD converges to a weight vector close to the minimizer of (3.3). In the following, we describe the derivation of RMD.

We are interested in solving the explicitly regularized optimization problem (3.3). Let us define an auxiliary variable $z \in \mathbb{R}^n$ with elements $z[1], \ldots, z[n]$. The optimization problem (3.3) can be transformed into the following form

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{n} \sum_{i=1}^{n} \frac{z^2[i]}{2} + \lambda \psi(w) \\ \text{s.t.} \quad & z[i] = \sqrt{2L_i(w)}, \quad \forall\, i = 1, \ldots, n. \end{aligned}$$

This can be arranged to the equivalent problem

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{\lambda n} \sum_{i=1}^{n} \frac{z^2[i]}{2} + \psi(w) \\ \text{s.t.} \quad & z[i] = \sqrt{2L_i(w)}, \quad \forall\, i = 1, \ldots, n. \end{aligned} \tag{5.1}$$

The objective of this optimization problem is a strictly convex function

$$\hat{\psi}\,(w, z) = \psi(w) + \frac{1}{\lambda n} \cdot \frac{\|z\|^2}{2},$$

and there are n equality constraints. We can therefore think of the model that is solved in this problem as an "augmented" model of $p+n$ dimensions, i.e., with two sets of weights: w and z. To enforce the constraints $z[i] = \sqrt{2L_i(w)}$, we can define a "constraint-enforcing" loss $\hat{\ell}\left(z[i] - \sqrt{2L_i(w)}\right)$, where $\hat{\ell}(\cdot)$ is a differentiable convex function with a unique root at 0 (e.g., the square loss $\hat{\ell}(\cdot) = \frac{(\cdot)^2}{2}$). Thus, (5.1) can be rewritten as

$$\boxed{\begin{aligned} \min_{w,z} \quad & \hat{\psi}(w, z) \\ \text{s.t.} \quad & \hat{\ell}\left(z[i] - \sqrt{2L_i(w)}\right) = 0, \quad \forall\, i = 1, \ldots, n. \end{aligned}} \tag{5.2}$$

Notice that (5.2) matches the implicitly-regularized problem given in in (3.5), which is solved by MD under certain conditions as seen in Section 4.2. Therefore, we can solve (5.2) by minimizing $\hat{\ell}$ via mirror descent with the potential function $\hat{\psi}$. To do so, we need to follow the update rule given in (4.1) and compute the gradients of the potential $\hat{\psi}(\cdot,\cdot)$ as well as the loss $\hat{\ell}\left(z[i] - \sqrt{2L_i(w)}\right)$, with respect to w and z. We omit the details of this straightforward calculation and simply state the result, which we call the *Regularizer*

Algorithm 1 Regularizer Mirror Descent (RMD)

```
Require: $\lambda, \eta, w_0$
 1: Initialization: $w \leftarrow w_0$, $z \leftarrow 0$
 2: repeat
 3:     for $i \leftarrow 1$ to $n$ do
 4:         $c_i \leftarrow \eta \hat{\ell}' \left( z[i] - \sqrt{2L_i(w)} \right)$
 5:     end for
 6:     $w \leftarrow \nabla\psi^{-1} \left( \nabla\psi(w) + \frac{1}{n} \sum_{i=1}^{n} \frac{c_i}{\sqrt{2L_i(w)}} \nabla L_i(w) \right)$
 7:     for $i \leftarrow 1$ to $n$ do
 8:         $z[i] \leftarrow z[i] - \lambda \cdot c_i$
 9:     end for
10: until convergence
11: return $w$
```

Mirror Descent (RMD) algorithm.

At time t, the update rule of RMD can be written as follows:

$$\begin{aligned} \nabla\psi(w_t) &= \nabla\psi(w_{t-1}) + \frac{1}{n} \sum\nolimits_{i=1}^{n} \frac{c_{t,i}}{\sqrt{2L_i(w_{t-1})}} \nabla L_i(w_{t-1}), \\ z_t[i] &= z_{t-1}[i] - \lambda c_{t,i}, \quad \forall\, i = 1, \dots, n, \end{aligned} \tag{5.3}$$

where $c_{t,i} = \eta \cdot \hat{\ell}' \left(z_{t-1}[i] - \sqrt{2L_i(w_{t-1})} \right)$, $\hat{\ell}'(\cdot)$ is the derivative of the constraint-enforcing loss function, and the weights are initialized with $w_0 = \arg\min_w \psi(w)$ (which is the origin for all norms, for example), and $z_0 = 0$. Note that because of the strict convexity of the regularizer $\psi(\cdot)$, its gradient $\nabla\psi(\cdot)$ is an invertible function, and the above update rule is well-defined. Algorithm 1 summarizes the procedure. As will be shown in Chapter 7, under suitable conditions, RMD provably solves the optimization problem (3.3).

One can choose the constraint-enforcing loss as $\hat{\ell}(\cdot) = \frac{(\cdot)^2}{2}$, which implies $\hat{\ell}'(\cdot) = (\cdot)$, to simply obtain the same update rule as in (5.3) with $c_{t,i} = \eta(z_{t-1}[i] - \sqrt{2L_i(w_{t-1})})$.

5.1.1 Mini-Batch RMD

As mentioned previously, in practice, gradient updates are computed on mini-batches instead of the full dataset – we apply this principle to arrive at the mini-batch implementation of RMD, which is summarized in Algorithm 2. Like with the mini-batch version of MD described in Section 4.1, averaging the gradient over the mini-batch in the w update ensures that the magnitude of the update is normalized with respect to the batch size.

Algorithm 2 Mini-batch Regularizer Mirror Descent (RMD)

```
Require: $\lambda, \eta, w_0$
 1: Initialization: $w \leftarrow w_0$, $z \leftarrow 0$
 2: repeat
 3:     Take a mini batch $B$
 4:     for $i \in B$ do
 5:         $c_i \leftarrow \eta \cdot \hat{\ell}' \left( z[i] - \sqrt{2L_i(w)} \right)$
 6:     end for
 7:     $w \leftarrow \nabla\psi^{-1} \left( \nabla\psi(w) + \frac{1}{|B|} \sum_{i \in B} \frac{c_i}{\sqrt{2L_i(w)}} \nabla L_i(w) \right)$
 8:     for $i \in B$ do
 9:         $z[i] \leftarrow z[i] - \lambda \cdot c_i$
10:     end for
11: until convergence
12: return $w$
```

5.1.2 Reduction of RMD to MD for $\lambda \to 0$

For $\lambda \to 0$, RMD reduces to the standard MD, which jives with the fact that the optimization problem it solves, i.e., (3.3), for $\lambda \to 0$ reduces to the optimization problem that MD solves, i.e., (3.5).

Note that when $\lambda \to 0$ and $z_0 = 0$, the update rule for z_t in (5.3) vanishes, and we have $z_t = 0$ for all t. Therefore, the update becomes

$$\nabla\psi(w_t) = \nabla\psi(w_{t-1}) + \frac{\eta}{n} \sum_{i=1}^{n} \frac{\hat{\ell}' \left(-\sqrt{2L_i(w_{t-1})} \right)}{\sqrt{2L_i(w_{t-1})}} \nabla L_i(w_{t-1}).$$

For $\hat{\ell}(\cdot) = \frac{(\cdot)^2}{2}$, we have $\hat{\ell}'(\cdot) = (\cdot)$, and the update rule further reduces to

$$\nabla\psi(w_t) = \nabla\psi(w_{t-1}) - \frac{\eta}{n} \sum_{i=1}^{n} \nabla L_i(w_{t-1}),$$

which is precisely the update rule for MD. We also note that when the potential is $\psi(\cdot) =$

$\frac{1}{2}\|\cdot\|_2^2$, we further reduce the update rule to standard gradient descent.

5.1.3 Special Case: q-norm Potential

An important special case of RMD is when the potential function $\psi(\cdot)$ is chosen to be the ℓ_q-norm, i.e., $\psi(w) = \frac{1}{q}\|w\|_q^q = \frac{1}{q}\sum_{k=1}^{p} |w[k]|^q$, for a real number $q > 1$. Let the current gradient for each data point i be denoted $g_i := \nabla L_i(w_{t-1})$. In this case, the update rule can be written as

$$w_t[k] = \left|\xi_{t,i}\right|^{\frac{1}{q-1}} \operatorname{sign}\left(\xi_{t,i}\right), \quad \forall\, k$$

$$z_t[i] = z_{t-1}[i] - \lambda c_{t,i}, \quad \forall\, i = 1, \ldots, n$$

for $\xi_{t,i} = |w_{t-1}[k]|^{q-1} \operatorname{sign}(w_{t-1}[k]) + \frac{1}{n}\sum_{i=1}^{n} \frac{c_{t,i}}{\sqrt{2L_i(w_{t-1})}} g_i[k]$, where $w_t[k]$ denotes the k-th element of w_t (the weight vector at time t) and $g_i[k]$ is the k-th element of the current gradient g_i. Note that for this choice of the potential function, the update rule is *coordinate-wise separable*, in the sense that the update for the k-th element of the weight vector requires only the k-th element of the weight and gradient vectors. This allows for efficient parallel implementation of the algorithm, which is crucial for large-scale tasks.

Even among the family of q-norm RMD algorithms, there can be a wide range of regularization effects for different values of q. Some important examples are as follows:

ℓ_1-norm regularization promotes sparsity in the weights. Sparsity is often desirable for reducing the storage and/or computational load, given the massive size of state-of-the-art DNNs. However, since the ℓ_1-norm is neither differentiable nor strictly convex, one may use $\psi(w) = \frac{1}{1+\epsilon}\|w\|_{1+\epsilon}^{1+\epsilon}$ for some small $\epsilon > 0$ [2] as a proxy.

ℓ_∞-norm regularization promotes a bounded and small range of weights. With this choice of potential, the weights tend to concentrate around a small interval. This is often desirable in various implementations of neural networks since it provides a small dynamic range for quantization of the weights, which reduces the production cost and computational complexity. However, since ℓ_∞ is, again, not differentiable, one can choose a large value for q and use $\psi(w) = \frac{1}{q}\|w\|_q^q$ to achieve the desirable regularization effect of ℓ_∞-norm ($q = 10$ is used in [5]).

ℓ_2-norm still promotes small weights, similar to ℓ_1-norm, but to a lesser extent. The update rule is

$$\begin{aligned} w_t[k] &= w_{t-1}[k] + \frac{1}{n}\sum_{i=1}^{n} \frac{c_{t,i}}{\sqrt{2L_i(w_{t-1})}} g_i[k], \quad \forall\, k \\ z_t[i] &= z_{t-1}[i] - \lambda c_{t,i}, \quad \forall\, i = 1 \dots n \end{aligned} \tag{5.4}$$

5.1.4 Special Case: Negative Entropy Potential

One can choose the potential function $\psi(\cdot)$ to be the negative entropy, i.e., $\psi(w) = \sum_{k=1}^{p} w[k]\log(w[k])$. For this particular choice, the associated Bregman divergence [11, 4] reduces to the Kullback–Leibler divergence. Let the current gradient for each data point i be denoted by $g_i := \nabla L_i(w_{t-1})$. The update rule would be

$$\begin{aligned} w_t[k] &= w_{t-1}[k] \exp\left(\frac{1}{n}\sum_{i=1}^{n} \frac{c_{t,i}}{\sqrt{2L_i(w_{t-1})}} g[k] \right), \quad \forall\, k \\ z_t[i] &= z_{t-1}[i] - \lambda c_{t,i}, \quad \forall\, i = 1, \dots, n \end{aligned}$$

This update rule requires the weights to be positive. Like in the special case of the q-norm potential, the update rule for the negative entropy potential is separable and therefore allows for efficient parallel implementation.

5.2 Implementation of q-norm RMD

We provide a proof-of-concept implementation of the q-norm RMD in PyTorch with $\hat{\ell}(\cdot) = \frac{(\cdot)^2}{2}$. An implementation of q-norm RMD is particularly useful, because of the special case where $q = 2$. In this case, RMD attempts to solve the same objective as weight decay, allowing us to perform a direct comparison between the two, which will be carried out in Chapter 6.

This implementation is not immediately apparent, as RMD requires computations on the per-sample empirical losses $L_i(\cdot)$ and gradients, as can be seen in Algorithm 1 and 2, whereas PyTorch typically takes a single gradient of the loss averaged or summed over the batch of data points in each update step. However, we note that the update rule for the weights w can be rewritten

$$\begin{aligned}\nabla\psi(w_t) &= \nabla\psi(w_{t-1}) + \frac{\eta}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w_{t-1}}\hat{\ell}\left(z_{t-1}[i] - \sqrt{2L_i(w_{t-1})}\right)\\ &= \nabla\psi(w_{t-1}) + \eta\frac{\partial}{\partial w_{t-1}}\left(\frac{1}{n}\sum_{i=1}^{n}\hat{\ell}\left(z_{t-1}[i] - \sqrt{2L_i(w_{t-1})}\right)\right)\end{aligned}$$

Thus, to implement RMD, we create a new loss function module which computes the constraint-enforcing loss $\hat{\ell}\left(z[i] - \sqrt{2L_i(w)}\right) = \frac{1}{2}\left(z[i] - \sqrt{2L_i(w)}\right)^2$, which is given in Listing 5.1. The parameters `sample_loss` and `num_samples` represent the per-sample empirical loss function $L_i(w) = \ell(y_i, f_i(w))$ (for example, cross entropy loss) and the total number of training samples, respectively.

Listing 5.1: Sample PyTorch implementation of the RMD loss function.

```python
import torch
import torch.nn as nn

class RMDLoss(nn.Module):
    def __init__(self, sample_loss, num_samples, reduction='mean'):
        super(RMD_Loss, self).__init__()
        self.sample_loss_func = sample_loss
        self.z = torch.zeros(num_samples)
        self.idx = 0
        self.reduction = reduction

    def set_z_values(self, z, idx):
        self.z = z
        self.idx = idx

    def forward(self, predictions, target):
        sample_loss = self.sample_loss_func(predictions, target)
        adj_loss = torch.sqrt(2 * sample_loss)
        sq_loss = 0.5 * torch.square(self.z[self.idx] - adj_loss)

        if self.reduction == 'mean':
            return torch.mean(sq_loss)
        elif self.reduction == 'sum':
            return torch.sum(sq_loss)
        else:
            return squared_loss
```

An example training script for one epoch is provided in Listing 5.2. In the case of q-norm RMD, the `optimizer` should be `qnormSGD`, as given in Listing 4.1. The constraint enforcing loss `rmd_criterion` can be implemented as the `RMDLoss` module provided in Listing 5.1, and `per_sample_criterion` is the corresponding per-sample empirical loss $\ell(\cdot,\cdot)$. The data loader `trainloader` should provide the indices of each sample in the batch, along with the data and labels. The auxiliary z variables are maintained separately from the base model as a tensor of length `num_samples`.

Thus, RMD can easily be substituted into a standard PyTorch training loop with only a few additional lines of code. This implementation was utilized in the experiments in Chapter 6, demonstrating that RMD is practically feasible.

Listing 5.2: Sample PyTorch implementation of training RMD for a single epoch.

```python
def train_one_epoch():
    for data, labels, indexes in trainloader:
        # Run the forward pass
        outputs = model(data)
        # per-sample loss in each batch
        sample_loss = per_sample_criterion(outputs, labels)

        rmd_criterion.set_z_values(z, idx)
        rmd_loss = rmd_criterion(outputs, labels)

        # Backprop
        optimizer.zero_grad()
        rmd_loss.backward()
        optimizer.step()

        # Compute c_i for batch
        c_batch = lr * (z[idx] - \
            torch.sqrt(2*sample_loss.clone().detach()))
        c_batch = c_batch.clone().detach()

        # Update z-auxiliary variables
        z[idx] = z[idx] - (lmbda * c_batch)
```

5.2.1 Computational Overhead

As in Section 4.3, we verify that models can be trained efficiently with RMD by running a simple benchmark comparing the runtime of RMD (with the ℓ_2-norm as the potential) against the standard PyTorch implementation of weight decay (`optim.SGD`). We train ResNet-18 [20] on CIFAR-10 [25] with 20% of the training labels corrupted and a batch size of 128 and time how long it takes to train 10 and 20 epochs. See Chapter 6 for details. These experiments were run on a single Nvidia A30 GPU. We similarly limit sources of nondeterminisim as in Section 4.3, though we again acknowledge that the results are likely inprecise due to the use of shared computing resources.

Table 5.1: Comparison of the training time of RMD, as implemented in Listing 5.1 amd 5.2, and the standard PyTorch implementation of weight decay (`optim.SGD`). The mean $\pm$ std. dev. of the training time for 10 and 20 epochs across five trials is reported.

	Training Time (seconds)	
Algorithm	**10 epochs**	**20 epochs**
RMD	122.0 ± 0.4	244.1 ± 1.2
Weight Decay	101.6 ± 0.1	203.6 ± 0.2

From Table 5.1, it can be seen that RMD trains about 20% slower than the PyTorch implementation of weight decay. We believe that this is in part because RMD inherently requires more operations per update, but also due to the lack of optimization of our implementation.

5.2.2 Notes on Implementation

One small but important note about this implementation of RMD is that it is prone to numerical issues when the regularization parameter λ is small. This is because the per sample losses are able to reach near zero when training, resulting in undefined gradients. While this did not pose an issue for our experiments, since we wanted to observe the effects of explicit regularization and thus used larger values of λ, this problem remains a kink in the implementation of RMD that requires more attention in the future.

Chapter 6

RMD: Experimental Results

In this chapter, we demonstrate the power of RMD with experiments in a practical scenario where the training labels are corrupted. As mentioned in the introduction, there are many ways to regularize learned models and improve their generalization performance, including methods that perform data augmentation, a change to the network architecture, early stopping, etc. Since this paper is concerned with the effect of learning algorithms for explicit regularization, we first present a detailed comparison of RMD, with the ℓ_2-norm as the regularizer, with standard weight decay (which also attempts to explicitly regularize the ℓ_2-norm of the weights) and GD (which induces implicit regularization) as a baseline. To further demonstrate that explicit regularization is a good strategy for learning with corrupted labels, we also compare RMD against several state-of-the-art methods designed for this setting.

The code used for the experimental portion of this thesis can be found at `https://github.com/tiff-toff/RegularizerMirrorDescent`.

6.1 Experimental Setup

6.1.1 General Setup

Dataset. To test the ability of different regularization methods in avoiding overfitting, we need a training set that does not consist entirely of clean data. Thus, we took the popular CIFAR-10 dataset [25], which has 10 classes and $n = 50,000$ training data points, and created 2 new datasets by corrupting 20% and 40% of the data points via assigning them random labels. Note that for each of those images, there is a $9/10$ chance of being assigned a wrong label. Therefore, on average, there are about 18% and 36% incorrect labels in the aforementioned datasets, respectively. To have a standard baseline, we also run our experiments on the standard CIFAR-10 dataset (i.e., 0% corruption). We apply random crops and horizontal flips for data augmentation. No corruption is applied on the test data, i.e., the standard test set of CIFAR-10 is used for evaluating the test accuracies.

Network Architecture. We train a standard ResNet-18 [20] deep neural network, as implemented in `https://github.com/kuangliu/pytorch-cifar`, which is commonly used for the CIFAR-10 dataset. The network has 18 layers, and around 11 million parameters. Thus, it qualifies as a highly overparameterized model. We do not make any changes to the network, and we use the same structure in every experiment.

Initialization. The parameters w and z are initialized randomly around zero. For each trial, ResNet-18 is initialized with different weights, but the same set of initializations is used across algorithms.

6.1.2 Explicit Regularization Algorithms

We first consider two *explicit regularization* strategies, weight decay and RMD, and the standard SGD. We ran each of the two explicit regularization algorithms on a wide range of regularization parameter λ.

1. **SGD:** We use the standard stochastic gradient descent as a baseline. While there is no explicit regularization, this is still known to induce an implicit regularization, as discussed in Section 3.3.

2. **SGD + Weight decay:** We train the network with an ℓ_2-norm regularization, through weight decay.

3. **RMD:** We train the network with RMD, which provably regularizes with an ℓ_2-norm. We utilize the mini-batch implementation of RMD as summarized in Algorithm 2 and use the square loss to enforce the constraints, i.e., $\hat{\ell}(\cdot) = \frac{1}{2}(\cdot)^2$.

Mini-Batch. For all three algorithms, we train in mini-batches with a batch size of 128, which is a common choice for CIFAR-10.

Learning Rate. We used three different fixed learning rates for each of the algorithms: 0.001, 0.01 and 0.1. Among all, 0.1 provided the best convergence behavior for the SGD and RMD, whereas 0.01 worked the best for the explicit regularization via weight decay. The reported results are given for the best-performing learning rate choices.

Stopping Criterion. In order to determine the stopping point for each of the algorithms, we use the following stopping criteria.

1. For SGD, we train until the training data is interpolated, i.e., 100% training accuracy, in a similar matter as [5].

2. Note that it is not feasible to determine the stopping criterion based on the training accuracies for the explicit regularization via weight decay or RMD. Therefore, for the explicit regularization via weight decay, we consider the change in the total loss over the training set. We stop the training if the change in the total loss is less than 0.01% over 100 consecutive epochs.

3. For RMD, we know that the algorithm eventually interpolates the new manifold $\hat{\mathcal{W}}$, i.e., fits the constraints in (5.1). Thus, we can use the total change in the constraints, i.e.,
$$\sum_{i=1}^{n} \left| z[i] - \sqrt{2L_i(w)} \right| .$$
and we stop the training if this summation improves less than 0.01% over 100 consecutive epochs.

We should emphasize that, given our choices for the setup, *the only difference between the experiments on the same corruption level dataset is the optimization algorithm.*

6.1.3 Comparison Methods

To demonstrate the practical utility of RMD, we compare its generalization performance against several other state-of-the-art methods for learning with noisy labels. Each of the comparison methods is trained on the same datasets, network architecture, and weight initialization as described in Section 6.1.1. To best reproduce the performance of these results, we reuse the choices of hyperparameters as is given in the original papers.

1. **CDR:** We train the network with CDR [49] by adapting the code used in [49], which is available at `https://github.com/xiaoboxia/CDR`. Note that the experiments in [49] use ResNet-50 [20], whereas we train with ResNet-18. The remaining experimental setup is unchanged from [49].

2. **PHuber-CE:** We adapt code provided at `https://github.com/dmizr/phuber` [36], which is a re-implementation of the experiments run in [34]. We train the network with the partially Huberised cross entropy loss with hyperparameter τ set to 2 [34]. The remaining hyperparameters for training ResNet-18 are provided in the codebase: 200 epochs, batch size of 128, training with the SGD optimizer with Nesterov momentum of 0.9 and weight decay of 5×10^{-4}. The initial learning rate is 0.1 and decays by 0.2 at epochs 60, 120, and 160.

3. **T-revision:** We adapt code used in [50] at `https://github.com/xiaoboxia/T-Revision`, which employs the importance reweighting method [30] and modifies the learned transition matrix during training [50]. We modify the number of epochs used for estimating the transition matrix from 20 to 40. The remaining experimental setup is unchanged from [50].

4. **Label Smoothing:** We train with label smoothing directly via Pytorch's `nn.CrossEntropyLoss` [46]. [31] found that the performance of label smoothing improves as the hyperparameter ϵ increases above the corruption rate, so we train with ϵ set to 0.8. We follow the direct training method in [48] for remaining experimental details: we train for 200 epochs with batch size 128 and used the SGD optimizer with Nesterov momentum of 0.9 and weight decay 10^{-4}. The initial

learning rate is set to 0.1, and it decays by 0.1 at the 100-th and 150-th epochs. Note that the experiments in [48] employ ResNet-34 [20], whereas we used ResNet-18.

5. **Mixup:** We train with Mixup [54], adapting the the original code used in [54], which is provided at `https://github.com/facebookresearch/mixup-cifar10`. We set the hyperparameter α to 8 and 32 for 20% and 40% corruption levels, respectively. Note that the original experiments used the PreActivation ResNet-18 network [19], whereas we used vanilla ResNet-18. The remaining experimental setup is unchanged from [54].

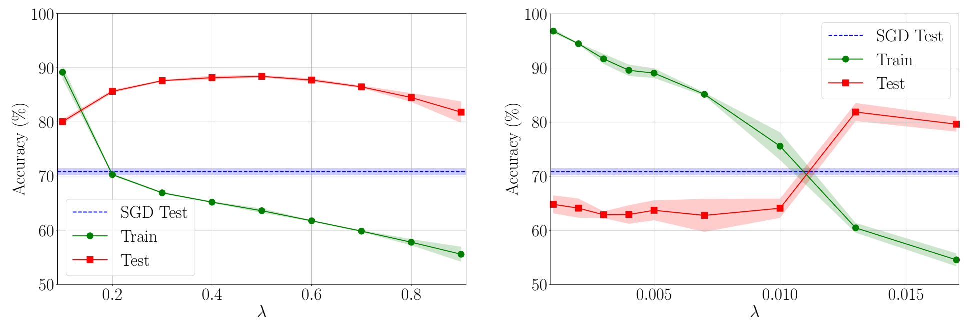
For these experiments, all trials trained with RMD, CDR, or PHuber-CE were performed on a single Nvidia V100 GPU. All trials trained with SGD, weight decay, T-revision, label smoothing, or Mixup were performed on a single Nvidia A30 GPU.

6.2 Explicit Regularization Results

The results for SGD, weight decay, and RMD are given in Fig. 6-1 and 6-2. Fig. 6-1a shows the results when 40% of the training data is corrupted, and Fig. 6-1b when the corrupted level is 20%. As expected, because the network is highly overparameterized, in all cases, SGD interpolates the training data and achieves 100% training accuracy.

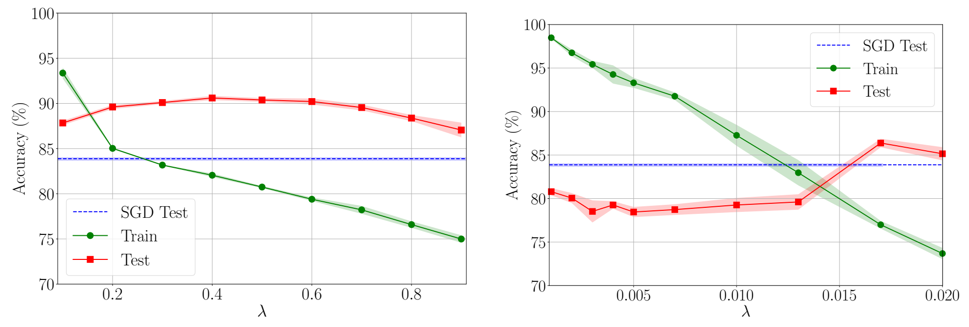
As seen in Fig. 6-1, RMD significantly outperforms both SGD and weight decay on corrupted datasets. In particular, despite attempting to minimize the same cost function, RMD and weight decay converge to very different solutions. Further, models trained via RMD achieve higher test accuracy than those trained with weight decay, even when the solutions reach similar training accuracy. At its peak, RMD surpasses weight decay's peak test accuracy by more than 4% and 6.5%, for when 20% and 40% of the labels are corrupted, respectively.

Additionally, RMD's behavior is significantly more consistent. Both training and test accuracies differ very little between trials. And the test accuracy changes gracefully as we adjust the regularization strength λ, where it follows a bell-shaped curve. Weight decay,



RMD Weight decay

(a) CIFAR-10 with 40% data points corrupted.



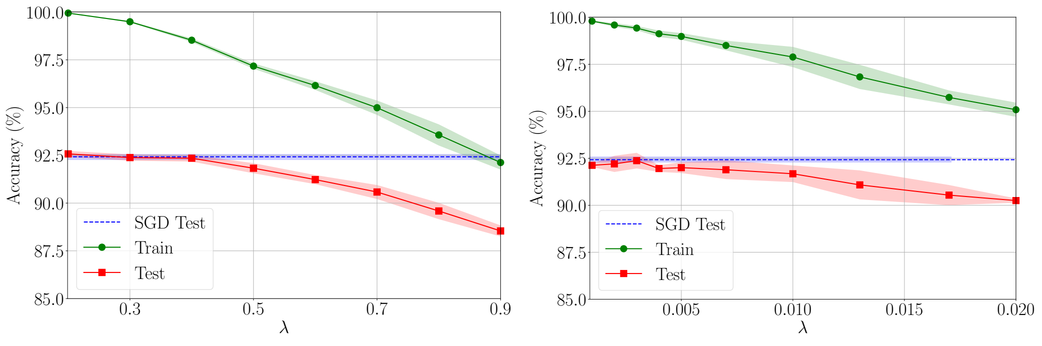
RMD Weight decay

(b) CIFAR-10 with 20% data points corrupted.

Figure 6-1: The accuracy (%) of ResNet-18 on CIFAR-10 with 40% (6-1a) and 20% (6-1b) of the training set corrupted when trained with RMD and weight decay over a range of regularization strengths λ. For each corruption level, the mean and standard deviation of the training (denoted in green) and test (denoted in red) accuracies over 5 trials are reported and compared against the test accuracy achieved by SGD when it fully interpolates the training data (denoted by the dotted blue line). The standard deviation is denoted by the shaded region. RMD achieves very small standard deviations, thus these regions are not entirely visible in the RMD plots. RMD is both more robust and more consistent than weight decay, outperforming weight decay with less variation between trials.

on the other hand, is highly sensitive to network initialization and regularization strength. Therefore, compared to weight decay, RMD produces more reliable results and requires less effort on hyperparameter tuning.

Finally, for the sake of completion, we show the results when training with SGD, weight decay, and RMD for the uncorrupted (clean) CIFAR-10 dataset in Fig. 6-2. As expected, since the data is uncorrupted, interpolating the data makes sense, and RMD and weight decay both demonstrate comparable performance to SGD when λ is small. Note that this empirical observation again highlights the fact that as $\lambda \to 0$, RMD reduces to mirror descent as shown in Section 5.1.2. However, RMD again demonstrates more consistent performance and slightly better generalization than weight decay.



(a) RMD (b) Weight Decay

Figure 6-2: The accuracy (%) of ResNet-18 on uncorrupted CIFAR-10 when trained with RMD and weight decay over a range of regularization strengths λ. The mean and standard deviation (indicated by the shaded region) of the training (denoted in green) and test (denoted in red) accuracies over 5 trials are reported and compared against the test accuracy achieved by SGD when it fully interpolates the training data (denoted by the dotted blue line). RMD achieves higher peak test accuracy compared to SGD and weight decay, and performs more consistently from trial to trial.

We acknowledge that there is a discrepancy in the scales of λ used to achieve similar training accuracy between weight decay and RMD in these experiments. We believe that this discrepancy occurs because of the distinct convergence behaviors of the two algorithms and due to the fact that weight decay most likely fails to find a minima in the training loss.

For a more complete set of results, see Appendix A.

6.2.1 Comparison with Other Methods

Table 6.1 compares RMD against state-of-the-art methods for learning problems with corrupted labels. As can be seen, for 20% corruption, Mixup achieves the best test accuracy, while RMD closely follows in second place. Further, for the larger 40% corruption, RMD outperforms all other methods, including Mixup. Additionally, as noted in Section 6.2, RMD is remarkably consistent and achieves small standard deviations in the test accuracy across 5 trials, particularly at the 40% corruption level, where the accuracy achieved by other methods varies more from trial to trial.

We note that some of the above methods are orthogonal to RMD, and can thus likely

Table 6.1: Comparison of the test accuracy (%) of RMD against methods for learning with corrupted labels. The mean $\pm$ std. dev. across 5 trials is reported. For weight decay and RMD, we report results for the best-performing value of λ. The method with the best mean test accuracy for each corruption level is highlighted in **boldface**.

Method / Corruption level	**20%**	**40%**
CDR [49]	$89.37 \pm .16$	$86.14 \pm .50$
PHuber-CE [34]	$90.26 \pm .27$	$86.70 \pm .23$
T-revision [50]	$89.56 \pm .30$	$86.79 \pm .52$
Label Smoothing [46]	$89.74 \pm .21$	$83.75 \pm .36$
Mixup [54]	$\mathbf{91.71 \pm .22}$	$87.70 \pm .38$
SGD	$83.88 \pm .18$	$70.83 \pm .63$
Weight Decay	$86.38 \pm .49$	81.86 ± 1.67
RMD **(ours)**	$90.60 \pm .24$	$\mathbf{88.42 \pm .22}$

be used in combination with RMD to achieve higher performance (e.g., training with RMD and Mixup together). However, because this paper focuses on the effects of the learning algorithm and considers the problem of learning with corrupted labels only as an example, we leave this study to future work.

It is particularly important to note that RMD is able to match or exceed the performance of these methods when training on corrupted labels simply by solving the regularized objective in a principled manner. This is in stark contrast with the methods we compare against, which introduce additional hyperparameters other than the regularization strength λ (Mixup, label smoothing, PHuber-CE) or rely on ad-hoc heuristics and stopping conditions tailored to the problem in order to achieve high performance (CDR, PHuber-CE, T-revision). Table 6.2 demonstrates the effects the hyperparameters and heuristics used for each method have on the test accuracy when trained on CIFAR-10 with 40% of the training labels corrupted.

CDR relies heavily on early stopping, as determined by when the validation accuracy is highest among the 100 training epochs. When we look at the final test accuracy achieved vs. the test accuracy given by early stopping, we see that there is a severe drop of nearly 20%. Similarly, for label smoothing, the test accuracy achieved by its last epoch is significantly worse than its best, indicating that these techniques still overfit to the corrupted labels.

Table 6.2: Demonstration of the effects of hyperparameter tuning and ad-hoc heuristics for state-of-the-art methods for 40% corruption. The mean $\pm$ std. dev. of the test accuracy (%) across 5 trials is reported for each version of each method. The best performing version of each method is highlighted in $\mathbf{boldface}$; this is the value reported in Table 6.1. For each method, the specific hyperparameter setting or heuristics used for stopping conditions has a strong effect on the resulting test accuracy. RMD achieves higher test accuracy with smaller standard deviation compared to these methods, without introducing ad-hoc heuristics or new hyperparameters.

Algorithm	**Test Accuracy**
CDR (early stopping)	$\mathbf{86.14 \pm .50}$
CDR (last)	66.31 ± 1.14
PHuber-CE	$\mathbf{86.70 \pm .23}$
PHuber-CE (extended)	$84.32 \pm .61$
T-revision (20 epoch est.)	65.37 ± 5.28
T-revision (40 epoch est.)	$\mathbf{86.79 \pm .52}$
Label Smoothing ($\epsilon = 0.2$, best)	$82.37 \pm .35$
Label Smoothing ($\epsilon = 0.2$, last)	$72.36 \pm .42$
Label Smoothing ($\epsilon = 0.8$, best)	$\mathbf{83.75 \pm .36}$
Label Smoothing ($\epsilon = 0.8$, last)	$68.97 \pm .21$
Mixup (best)	$\mathbf{87.70 \pm .38}$
Mixup (last)	$86.90 \pm .30$
RMD (best; **ours**)	$\mathbf{88.42 \pm .22}$

T-revision is multistage algorithm – its first stage is to estimate a transition matrix. If we train for 20 epochs in this first stage to obtain our estimate, the resulting test accuracy is more than 20% lower than if we used 40 epochs to obtain a more accurate initial estimate. We also note that both T-revision and CDR require estimates of the noise rate or corruption level (T-revision estimates the transition matrix, CDR explicitly requires the noise rate in its update rule), and thus have limited practical utility even for the problem of learning corrupted labels.

For Mixup and PHuber-CE, the heuristic we examine (reporting best vs. last test accuracy for Mixup, and extending the number of epochs from 200 to 300 for PHuber), had much more mild effects on the test performance. Similarly, the ϵ hyperparameter of label smoothing has noticeable but relatively small effects on the performance of the method.

Similar analysis for the 20% corruption level and the uncorrupted dataset can be found in Appendix A

These methods all employ weight decay and also introduce new hyperparameters or ad-hoc heuristics like early stopping. Our experiments show that RMD removes the need for these new hyperparameters and heuristics and provides well-formulated guarantees on its behavior. Therefore, RMD is a strong candidate for training overparameterized deep networks with explicit regularization, including in other applications beyond learning with corrupted labels where regularization is also desirable.

Chapter 7

Convergence Properties of RMD

In this section, we provide convergence guarantees for RMD using the implicit regularization properties of mirror descent algorithms [3, 5] described in Chapter 4. In the following, in parallel with the construction in Section 4.2 for MD, we extend and formalize the convergence guarantee of RMD.

7.1 RMD Approximately Converges to the Optimal Explicitly Regularized Solution

As mentioned in Chapter 5, the optimization problem solved by RMD can be defined over the augmented parameters $\bar{w} := \begin{bmatrix} w \\ z \end{bmatrix} \in \mathbb{R}^{p+n}$. Following the definitions above, we define the learning problem over $\bar{w}$ with $\hat{f}_i(\bar{w}) = \sqrt{2L_i(w)} - z[i]$, $\hat{y}_i = 0$, and $\hat{L}_i(\bar{w}) = \hat{\ell}(\hat{y}_i - \hat{f}_i(\bar{w})) = \hat{\ell}(z[i] - \sqrt{2L_i(w)})$ for $i = 1, \ldots, n$. Note that in this new problem, we now have $p+n$ parameters and n constraints/data points, and since $p \gg n$, we have $p+n \gg n$, and we are still in the highly overparameterized regime (even more so). Thus, we can also define the set of interpolating solutions for the new problem as

$$\hat{\mathcal{W}} = \left\{ \bar{w} \in \mathbb{R}^{p+n} \mid \hat{f}_i(\bar{w}) = \hat{y}_i, \quad i = 1, \ldots, n \right\}. \tag{7.1}$$

As discussed in Chapter 5, define the new potential function $\hat{\psi}(\bar{w}) = \psi(w) + \frac{1}{2\lambda n}\|z\|^2$

and a corresponding MD update algorithm

$$\nabla\hat{\psi}\left(\bar{w}_t\right)=\nabla\hat{\psi}\left(\bar{w}_{t-1}\right)-\frac{\eta}{n}\sum\nolimits_{i=1}^{n}\nabla\hat{L}_i\left(\bar{w}_{t-1}\right),$$

initialized at $\bar{w}_0 := \begin{bmatrix} w_0 \\ 0 \end{bmatrix}$. It is straightforward to verify that this update rule is equivalent to that of RMD, i.e., (5.3). Moreover, following (4.5), we have

$$\begin{aligned} \hat{w}^* = \underset{w,z}{\arg\min} \quad & D_{\hat{\psi}}\left(\bar{w}, \bar{w}_0\right) \\ \text{s.t.} \qquad & \hat{f}_i\left(\bar{w}\right) = \hat{y}_i, \quad \forall\, i = 1, \dots, n. \end{aligned} \tag{7.2}$$

Plugging $D_{\hat{\psi}}\left(\bar{w}, \bar{w}_0\right) = D_{\psi}(w, w_0) + \frac{1}{2\lambda n}\|z\|^2$ and $\hat{f}_i\left(\bar{w}\right) = \sqrt{2L_i(w)} - z[i]$ into (7.2), we can show that (7.2) is equivalent to (5.1) for $w_0 = \arg\min_w \psi(w)$.

Before giving the formal analysis on the convergence guarantee, we have the following regularity assumptions on the underlying loss landscape and the learning model, which are standard in the non-convex optimization literature [29, 1, 13].

Assumption 7.1.1. Denote the initial point by $\begin{bmatrix} w_0 \\ 0 \end{bmatrix}$. There exists $\bar{w} \in \hat{\mathcal{W}}$ and a region $\hat{\mathcal{B}} = \left\{ \bar{w}' = \begin{bmatrix} w' \\ z' \end{bmatrix} \in \mathbb{R}^{p+n} \mid D_{\hat{\psi}}\left(\bar{w}, \bar{w}'\right) \le \epsilon \right\}$ containing $\begin{bmatrix} w_0 \\ 0 \end{bmatrix}$, such that $D_{\hat{L}_i}(\bar{w}, \bar{w}') \ge 0, i = 1, \dots, n$, for all $\bar{w}' \in \hat{\mathcal{B}}$.

Assumption 7.1.2. Consider the region $\hat{\mathcal{B}}$ in Assumption 7.1.1. $\hat{f}_i(\cdot)$ have bounded gradient and Hessian on the convex hull of $\hat{\mathcal{B}}$, i.e., $\|\nabla\hat{f}_i\left(\bar{w}'\right)\| \le \gamma$, and $\alpha \le \lambda_{\min}\left(H_{\hat{f}_i}\left(\bar{w}'\right)\right) \le \lambda_{\max}\left(H_{\hat{f}_i}\left(\bar{w}'\right)\right) \le \beta, i = 1, \dots, n$, for all $\bar{w}' \in \operatorname{conv}\hat{\mathcal{B}}$.

In a nutshell, Assumption 7.1.1 states that the initial point $\begin{bmatrix} w_0 \\ 0 \end{bmatrix}$ is close to the (new) $(p+n)$-dimensional manifold $\hat{\mathcal{W}}$ of global minima. This assumption arguably comes for free in highly overparameterized settings [1]. Assumption 7.1.2 states that the first and second derivatives of the augmented learning model are *locally* bounded. This requirement is again standard in non-convex optimization literature to have some regularity in the learning dynamics. With this assumption in place, we state the convergence guarantees of RMD.

Theorem 7.1.3. *Consider the set of interpolating solutions* $\hat{\mathcal{W}}$ *defined in* (7.1)*, the closest such solution* $\hat{w}^*$ *defined in* (7.2)*, and the RMD iterates given in* (5.3) *initialized at* $\begin{bmatrix} w_0 \\ 0 \end{bmatrix}$*. Under Assumption 7.1.1, for sufficiently small step size, i.e., for any* $\eta > 0$ *for which* $\hat{\psi}(\cdot) - \eta \hat{L}_i(\cdot)$ *is strictly convex on* $\hat{\mathcal{B}}$ *for all* i*, the following statements hold:*

1. *The iterates converge to* $\begin{bmatrix} w_\infty \\ z_\infty \end{bmatrix} \in \hat{\mathcal{W}}$*.*
2. $D_{\hat{\psi}}\left(\hat{w}^*, \begin{bmatrix} w_\infty \\ z_\infty \end{bmatrix}\right) = o(\epsilon)$*.*

The proof of this result follows from adapting the MD convergence guarantee to the augmented model of RMD. This result shows that, if RMD starts with an initialization that is $O(\epsilon)$ away from $\hat{\mathcal{W}}$ in terms of Bregman divergence $D_{\hat{\psi}}$, it converges to a point $\begin{bmatrix} w_\infty \\ z_\infty \end{bmatrix} \in \hat{\mathcal{W}}$ that is $o(\epsilon)$ away from $\hat{w}^*$, again in terms of Bregman divergence. In other words, the Bregman divergence of this point is $o(\epsilon)$ from the minimum value it can take.

It is important to note that Theorem 7.1.3 requires the step size to be just small enough to locally guarantee the strict convexity of $\hat{\psi}(\cdot) - \eta \hat{L}_i(\cdot)$ inside $\hat{\mathcal{B}}$, and not globally. Moreover, we should emphasize that while Theorem 7.1.3 states that RMD converges to the manifold $\hat{\mathcal{W}}$, it does *not* mean that it is fitting the training data points or achieving zero training error. That is because $\hat{\mathcal{W}} \in \mathbb{R}^{p+n}$ is a *different* (much higher-dimensional) manifold than $\mathcal{W} \in \mathbb{R}^p$ in (4.3), and interpolating the new problem would result in fitting the equality constraints defined by the explicitly regularized problem, i.e., $\sqrt{2L_i(w)} = z[i]$ for all i, which in general may happen when the empirical losses L_i are nonzero.

7.2 Experimental Validation

Using the results from Chapter 6, we are able to evaluate the theoretical claims of Section 7.1. As we have run extensive experiments training with RMD where $\psi(\cdot) = \frac{1}{2}\|\cdot\|^2$

Table 7.1: As in Chapter 6, we have used RMD with the ℓ_2-norm as regularizer $\psi(\cdot)$ to train ResNet-18 from 5 different initializations on CIFAR-10 with 40% of the labels corrupted for $\lambda = 0.5$. This gives us 5 different solutions in the manifold $\hat{\mathcal{W}}$ of global minima. The rows correspond to different initial points, and the column corresponds to the final solutions obtained when trained from each initialization. Each entry is the distance between the two, measured in the Bregman divergence relative to the modified potential $\hat{\psi}(\cdot)$. As can be seen, the smallest entry in each row is the one where the initial point and final point match, i.e., RMD has converged to the point in $\hat{\mathcal{W}}$ closest to the initialization it began at.

	Final 1	Final 2	Final 3	Final 4	Final 5
Initial 1	108.3	1226.1	1222.1	1222.2	1222.3
Initial 2	1216.9	122.3	1222.6	1223.9	1222.7
Initial 3	1217.2	1227.1	115.8	1223.6	1223.3
Initial 4	1216.0	1226.8	1222.5	120.3	1222.4
Initial 5	12173	1227.0	1222.8	1223.4	116.6

for different initializations, we can verify the convergence property presented in Theorem 7.1.3. For a specific corruption level and value of λ, we compute the distance between each of the five initializations and each model trained by RMD from each initialization, or more specifically the Bregman divergence $D_{\hat{\psi}}(\bar{w}, \bar{w}_0) = D_\psi(w, w_0) + \frac{1}{2\lambda n}||z||^2 = \frac{1}{2}||w - w_0||^2 + \frac{1}{2\lambda n}||z||^2$.

Table 7.1 shows the value of $D_{\hat{\psi}}(\bar{w}, \bar{w}_0)$ when computed on the solutions RMD obtained when training ResNet-18 from 5 different initializations on CIFAR-10 with a 40% noise rate in the labels and $\lambda = 0.5$. It can be seen that the smallest entries of each row lie along the diagonal, indicating that the solution in $\hat{\mathcal{W}}$ reached by RMD is indeed the "closest" to the initial point that RMD started at. Thus, the findings in Section 7.1 appear to hold true empirically.

Chapter 8

Conclusion

We presented Regularizer Mirror Descent (RMD), a novel and efficient algorithm for training DNNs with any desired strictly convex regularizer. The starting point for RMD is a standard cost which is the sum of the training loss and a differentiable, strictly convex regularizer of the network weights. For highly overparameterized models, RMD provably converges to a point "close" to the minimizer of this cost. The algorithm can be readily applied to any DNN and enjoys the same parallelization properties as SGD. To illustrate the utility of RMD, we consider an application in the problem of learning with corrupted labels. After applying RMD to this problem of learning with corrupted labels, we demonstrate that RMD not only significantly outperforms and is more consistent than weight decay, but also surpasses state-of-the-art methods for this problem setting. So we conclude that RMD is a viable and versatile approach to solving explicit regularization in practical settings.

8.1 Future Work

Given that RMD enables training any network efficiently with a desired regularizer, it opens up several new avenues for future research. In particular, an extensive experimental study of the effect of different regularizers on different datasets and different architectures would be instrumental to uncovering the role of regularization in modern learning problems.

There are additionally a couple open concerns left to address. While we have a hypothesis for the discrepancy in the ranges of λ used in Chapter 6, a more thorough investigation would be worthwhile to properly understand the cause. Additionally, as mentioned in Section 5.2, the current implementation is likely to result in numerical issues for small values of λ. Thus, fine tuning of the implementation of RMD is needed.

Bibliography

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019.

[2] Navid Azizan and Babak Hassibi. A characterization of stochastic mirror descent algorithms and their convergence properties. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[3] Navid Azizan and Babak Hassibi. Stochastic gradient/mirror descent: Minimax optimality and implicit regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[4] Navid Azizan and Babak Hassibi. A stochastic interpretation of stochastic mirror descent: Risk-sensitive optimality. In *2019 58th IEEE Conference on Decision and Control (CDC)*, pages 3960–3965, 2019.

[5] Navid Azizan, Sahin Lale, and Babak Hassibi. Stochastic mirror descent on overparameterized nonlinear models. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):7717–7727, 2022.

[6] Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

[7] Peter L Bartlett, Andrea Montanari, and Alexander Rakhlin. Deep learning: a statistical viewpoint. *arXiv preprint arXiv:2103.09177*, 2021.

[8] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[9] Mikhail Belkin, Daniel J Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[10] Nicholas M Boffi and Jean-Jacques E Slotine. Implicit regularization and momentum algorithms in nonlinearly parameterized adaptive control and prediction. *Neural Computation*, 33(3):590–673, 2021.

[11] Lev M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

[12] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

[13] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pages 1675–1685. PMLR, 2019.

[14] Heinz Werner Engl, Martin Hanke, and Andreas Neubauer. *Regularization of inverse problems*, volume 375. Springer Science & Business Media, 1996.

[15] Ian Goodfellow, Y Bengio, and A Courville. Regularization for deep learning. *Deep learning*, pages 216–261, 2016.

[16] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In *International Conference on Machine Learning*, pages 1827–1836, 2018.

[17] Suriya Gunasekar, Jason D Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[18] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6152–6160, 2017.

[19] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016.

[20] 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.

[21] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[22] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[23] Wei Hu, Zhiyuan Li, and Dingli Yu. Simple and effective regularization methods for training on noisily labeled data with generalization guarantee. In *International Conference on Learning Representations*, 2019.

[24] Ziwei Ji, Miroslav Dudík, Robert E Schapire, and Matus Telgarsky. Gradient descent follows the regularization path for general losses. In *Conference on Learning Theory*, pages 2109–2136. PMLR, 2020.

[25] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[26] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[27] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International conference on artificial intelligence and statistics*, pages 4313–4324. PMLR, 2020.

[28] Shiyu Liang. *The role of explicit regularization in overparameterized neural networks*. PhD thesis, 2021.

[29] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 59:85–116, 2022.

[30] Tongliang Liu and Dacheng Tao. Classification with noisy labels by importance reweighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(3):447–461, mar 2016.

[31] Michal Lukasik, Srinadh Bhojanapalli, Aditya Menon, and Sanjiv Kumar. Does label smoothing mitigate label noise? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6448–6458. PMLR, 13–18 Jul 2020.

[32] Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. In *International Conference on Learning Representations*, 2019.

[33] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3325–3334. PMLR, 2018.

[34] Aditya Krishna Menon, Ankit Singh Rawat, Sanjiv Kumar, and Sashank Reddi. Can gradient clipping mitigate label noise? In *International Conference on Learning Representations (ICLR)*, 2020.

[35] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[36] David Mizrahi, Oğuz Kaan Yüksel, and Aiday Marlen Kyzy. [re] can gradient clipping mitigate label noise? In *ML Reproducibility Challenge 2020*, 2021.

[37] Cesare Molinari, Mathurin Massias, Lorenzo Rosasco, and Silvia Villa. Iterative regularization for convex regularizers. In *International Conference on Artificial Intelligence and Statistics*, pages 1684–1692. PMLR, 2021.

[38] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[39] A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. 1983.

[40] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR (Workshop)*, 2015.

[41] Tomaso Poggio, Andrzej Banburski, and Qianli Liao. Theoretical issues in deep networks. *Proceedings of the National Academy of Sciences*, 117(48):30039–30045, 2020.

[42] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[43] Siddhartha Satpathi, Harsh Gupta, Shiyu Liang, and R Srikant. The role of regularization in overparameterized neural networks. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 4683–4688. IEEE, 2020.

[44] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

[45] Haoyuan Sun, Kwangjun Ahn, Christos Thrampoulidis, and Navid Azizan. Mirror descent maximizes generalized margin and can be implemented efficiently. In *Advances in Neural Information Processing Systems*, volume 35, pages 31089–31101, 2022.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[47] Matus Telgarsky. Margins, shrinkage, and boosting. In *International Conference on Machine Learning*, pages 307–315. PMLR, 2013.

[48] Jiaheng Wei, Hangyu Liu, Tongliang Liu, Gang Niu, Masashi Sugiyama, and Yang Liu. To smooth or not? When label smoothing meets noisy labels. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23589–23614. PMLR, 17–23 Jul 2022.

[49] 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.

[50] Xiaobo Xia, Tongliang Liu, Nannan Wang, Bo Han, Chen Gong, Gang Niu, and Masashi Sugiyama. Are anchor points really indispensable in label-noise learning? 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.

[51] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[52] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[53] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2018.

[54] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

Appendix A

Full Experimental Results

A.1 Results on the Training Set with 40% of Labels Corrupted

The training and test accuracies for the two explicitly regularized methods (RMD and weight decay) with various values of λ are reported in Table A.1. They are also compared against the baseline accuracies achieved by SGD.

SGD interpolates the training data due to overparameterization and achieves 100% training accuracy, but this results in overfitting. Hence, the test accuracy is only around 70%. Weight decay generally does not outperform the interpolating solution found by standard SGD until λ is relatively large, i.e., there is a sufficient amount of regularization.

RMD significantly outperforms weight decay for nearly all runs of either method and is not as sensitive to the value of λ in the ranges considered. Notably, for the 40% corruption level, RMD achieves peak test accuracy when the training accuracy is approximately equal to the fraction of uncorrupted points in the training set, i.e., 64%.

Table A.1: Comparison of the two explicitly regularized methods, RMD and weight decay, against the baseline of implicit regularization induced by SGD, for 40% corruption on the labels. The mean $\pm$ std. dev. of the training and test accuracies (%) across 5 trials are reported for each value of λ. The best performing λ is for RMD and weight decay highlighted in **boldface**; this is the value reported in Table 6.1. Note for $\lambda = 0.02$, weight decay failed to converge in 3 out of 5 trials. Thus only the mean $\pm$ std. dev. achieved by the 2 successful trials is reported in gray and is denoted by *.

Algorithm	**Lambda**	**Training Accuracy**	**Test Accuracy**
SGD	N/A	$100.00 \pm .00$	$70.83 \pm .63$
RMD **(ours)**	0.1	89.18 ± 1.23	$80.06 \pm .45$
	0.2	$70.27 \pm .22$	$85.65 \pm .29$
	0.3	$66.91 \pm .16$	$87.63 \pm .15$
	0.4	$65.18 \pm .15$	$88.18 \pm .36$
	0.5	$\mathbf{63.60 \pm .40}$	$\mathbf{88.42 \pm .22}$
	0.6	$61.74 \pm .13$	$87.74 \pm .32$
	0.7	$59.83 \pm .16$	$86.48 \pm .24$
	0.8	$57.77 \pm .57$	$84.52 \pm .81$
	0.9	55.55 ± 1.39	81.81 ± 1.98
Weight Decay	0.001	$96.83 \pm .39$	64.80 ± 1.66
	0.002	$94.49 \pm .13$	64.09 ± 1.77
	0.003	$91.70 \pm .93$	$62.87 \pm .62$
	0.004	89.58 ± 1.06	62.92 ± 1.72
	0.005	$89.05 \pm .90$	63.69 ± 1.85
	0.007	$85.14 \pm .32$	62.75 ± 3.05
	0.01	75.55 ± 2.57	64.076 ± 1.78
	0.013	$\mathbf{60.44 \pm .90}$	$\mathbf{81.86 \pm 1.67}$
	0.017	54.52 ± 1.20	79.62 ± 1.38
	0.02	$^*50.80 \pm .12$	$^*75.00 \pm .41$

A.2 Results on the Training Set with 20% of Labels Corrupted

The training and test accuracies for SGD, weight decay, and RMD with various values of λ for the 20% corruption level are summarized in Table A.2. As mentioned before, because the network is highly overparameterized, SGD expectedly interpolates the training data and achieves almost 100% training accuracy. The test accuracy for SGD is around 84%. RMD again outperforms weight decay at this corruption level and achieves peak test accuracy when the training accuracy is around 82%, i.e., the fraction of uncorrupted samples in the training set.

Table A.3 presents results from a similar set of experiments as in Table 6.2, but for the 20% corruption level. The findings are consistent with those of Table 6.2 but at a much smaller scale. The choice of heuristic has much less noticeable effects given the lower corruption rate. Thus, we again see that RMD can be trained to comparable or superior test accuracies when the training set is corrupted, without introducing new hyperparameters and ad-hoc heuristics.

Table A.2: Comparison of the two explicitly regularized methods, RMD and weight decay, against the baseline of implicit regularization induced by SGD, for 20% corruption on the labels. The mean $\pm$ std. dev. of the training and test accuracies (%) across 5 trials are reported for each value of λ. The best performing λ for RMD and weight decay is highlighted in **boldface**; this is the value reported in Table 6.1.

Algorithm	Lambda	Training Accuracy	Test Accuracy
SGD	N/A	$100.00 \pm .00$	$83.88 \pm .18$
RMD	0.1	$93.36 \pm .61$	$87.84 \pm .24$
	0.2	$85.03 \pm .08$	89.61 ± 0.22
	0.3	$83.18 \pm .06$	$90.10 \pm .17$
	$\mathbf{0.4}$	$\mathbf{82.05 \pm .20}$	$\mathbf{90.60 \pm .24}$
	0.5	$80.74 \pm .12$	$90.38 \pm .21$
	0.6	$79.39 \pm .20$	$90.21 \pm .34$
	0.7	$78.22 \pm .43$	$89.55 \pm .32$
	0.8	$76.59 \pm .36$	$88.38 \pm .34$
	0.9	$74.99 \pm .42$	$87.06 \pm .82$
Weight Decay	0.001	$98.48 \pm .12$	$80.79 \pm .41$
	0.002	$96.75 \pm .39$	$80.05 \pm .58$
	0.003	$95.42 \pm .41$	78.52 ± 1.27
	0.004	94.26 ± 1.05	$79.26 \pm .44$
	0.005	$93.30 \pm .58$	$78.45 \pm .58$
	0.007	$91.76 \pm .43$	$78.73 \pm .61$
	0.01	87.27 ± 1.20	$79.26 \pm .81$
	0.013	82.97 ± 1.46	$79.61 \pm .89$
	$\mathbf{0.017}$	$\mathbf{76.99 \pm .44}$	$\mathbf{86.38 \pm .49}$
	0.02	$73.68 \pm .65$	$85.15 \pm .76$

Table A.3: Demonstration of the effects of hyperparameter tuning and ad-hoc heuristics for state-of-the-art methods for 20% corruption. The mean $\pm$ std. dev. of the test accuracy (%) across 5 trials is reported for each version of each method. The best performing version of each method is highlighted in **boldface**; this is the value reported in Table 6.1. For each method, the specific hyperparameter setting or heuristics used for stopping conditions has a strong effect on the resulting test accuracy. RMD matches or achieves higher test accuracy compared to these methods, without introducing ad-hoc heuristics or new hyperparameters.

Algorithm	Test Accuracy
CDR (early stopping)	$\mathbf{89.37 \pm .16}$
CDR (last)	$81.12 \pm .60$
PHuber-CE	$\mathbf{90.26 \pm .27}$
PHuber-CE (extended)	$89.95 \pm .23$
T-revision (20 epoch est.)	85.24 ± 3.73
T-revision (40 epoch est.)	$\mathbf{89.56 \pm .30}$
Label smoothing ($\epsilon = 0.2$, best)	$86.35 \pm .28$
Label smoothing ($\epsilon = 0.2$, last)	$85.72 \pm .28$
Label smoothing ($\epsilon = 0.8$, best)	$\mathbf{89.74 \pm .21}$
Label smoothing ($\epsilon = 0.8$, last)	$85.37 \pm .50$
Mixup (best)	$\mathbf{91.71 \pm .22}$
Mixup (last)	$91.44 \pm .23$
RMD (best; **ours**)	$\mathbf{90.60 \pm .24}$

A.3 Results on the Uncorrupted (Clean) Training Set

The training and test accuracies for SGD, weight decay, and RMD with various values of λ for the uncorrupted dataset are presented in Table A.4.

Table A.5 presents results from a similar set of experiments as in Table 6.2, but when training with uncorrupted CIFAR-10. From the results in Table A.5, we can see that, when the dataset is uncorrupted, RMD performs similarly to PHuber, but is outperformed by most other methods in our comparison. However, this is expected given that the other methods build upon standard SGD to improve performance. We also note that because we are in the overparameterized regime and because our data is uncorrupted, it is desirable to train to zero training error. Thus, there is little discrepancy across choices of heuristics for each of these methods, and RMD does not necessarily achieve superior performance.

Table A.4: Comparison of the two explicitly regularized methods, RMD and weight decay, against the baseline of implicit regularization induced by SGD, on the uncorrupted dataset. The mean $\pm$ std. dev. of the training and test accuracies (%) across 5 trials are reported for each value of λ. The best performing λ for RMD and weight decay is highlighted in **boldface**.

Algorithm	Lambda	Training Accuracy	Test Accuracy
SGD	N/A	$100 \pm .00$	$92.42 \pm .15$
RMD	**0.2**	$\mathbf{99.95 \pm .01}$	$\mathbf{92.58 \pm .15}$
	0.3	$99.50 \pm .04$	$92.38 \pm .16$
	0.4	$98.53 \pm .10$	$92.35 \pm .18$
	0.5	$97.17 \pm .14$	$91.83 \pm .27$
	0.6	$96.15 \pm .23$	$91.23 \pm .24$
	0.7	$94.99 \pm .38$	$90.58 \pm .37$
	0.8	$93.57 \pm .55$	$89.59 \pm .43$
	0.9	$92.12 \pm .37$	$88.54 \pm .29$
Weight Decay	0.001	$99.80 \pm .03$	$92.13 \pm .09$
	0.002	$99.59 \pm .10$	$92.20 \pm .43$
	0.003	$\mathbf{99.43 \pm .11}$	$\mathbf{92.37 \pm .42}$
	0.004	$99.12 \pm .17$	$91.95 \pm .18$
	0.005	$98.98 \pm .18$	$92.00 \pm .29$
	0.007	$98.50 \pm .26$	$91.89 \pm .50$
	0.01	$97.89 \pm .54$	$91.68 \pm .44$
	0.013	$96.83 \pm .65$	$91.08 \pm .77$
	0.017	$95.74 \pm .37$	$90.54 \pm .54$
	0.02	$95.08 \pm .38$	$90.25 \pm .11$

Table A.5: Demonstration of the effects of hyperparameter tuning and ad-hoc heuristics for state-of-the-art methods for the uncorrupted dataset. The mean $\pm$ std. dev. of the test accuracy (%) across 5 trials is reported for version of each method. The best performing version of each method is highlighted in $\mathbf{boldface}$.

Algorithm	**Test Accuracy**
CDR (early stopping)	$93.10 \pm .20$
CDR (last)	$\mathbf{93.11 \pm .17}$
PHuber-CE	$92.63 \pm .19$
PHuber-CE (extended)	$\mathbf{92.66 \pm .25}$
T-revision (20 epoch est.)	$92.17 \pm .14$
T-revision (40 epoch est.)	$\mathbf{92.90 \pm .18}$
Label smoothing ($\epsilon = 0.2$, best)	$93.82 \pm .12$
Label smoothing ($\epsilon = 0.2$, last)	$93.67 \pm .10$
Label smoothing ($\epsilon = 0.8$, best)	$\mathbf{94.02 \pm .24}$
Label smoothing ($\epsilon = 0.8$, last)	$93.95 \pm .26$
Mixup (best)	$\mathbf{94.90 \pm .21}$
Mixup (last)	$94.63 \pm .25$
RMD (best; **ours**)	$\mathbf{92.58 \pm .15}$