Do We Need Zero Training Loss After Achieving Zero Training Error?

Takashi Ishida[1,2] Ikko Yamane[1] Tomoya Sakai[3]
Gang Niu[2] Masashi Sugiyama[2,1]

[1]The University of Tokyo [2]RIKEN [3]NEC Corporation

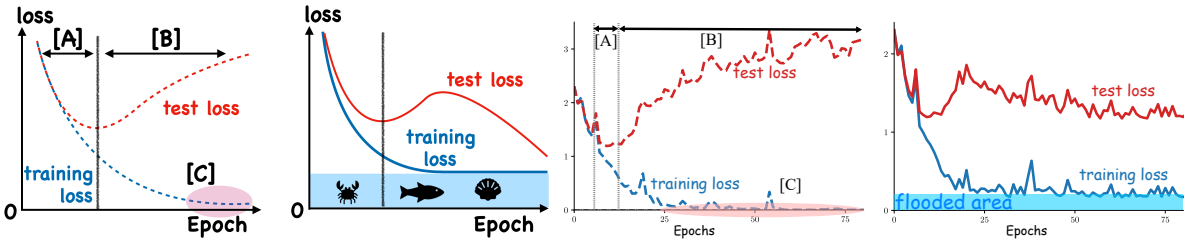
Abstract

Overparameterized deep networks have the capacity to memorize training data with zero training error. Even after memorization, the training loss continues to approach zero, making the model overconfident and the test performance degraded. Since existing regularizers do not directly aim to avoid zero training loss, they often fail to maintain a moderate level of training loss, ending up with a too small or too large loss. We propose a direct solution called *flooding* that intentionally prevents further reduction of the training loss when it reaches a reasonably small value, which we call the *flooding level*. Our approach makes the loss float around the flooding level by doing mini-batched gradient descent as usual but gradient ascent if the training loss is below the flooding level. This can be implemented with one line of code, and is compatible with any stochastic optimizer and other regularizers. With flooding, the model will continue to "random walk" with the same non-zero training loss, and we expect it to drift into an area with a flat loss landscape that leads to better generalization. We experimentally show that flooding improves performance and as a byproduct, induces a double descent curve of the test loss.

1 Introduction

"Overfitting" is one of the biggest interests and concerns in the machine learning community [Belkin et al., 2018, Caruana et al., 2000, Ng, 1997, Roelofs et al., 2019, Werpachowski et al., 2019]. One way of identifying if overfitting is happening or not, is to see whether the generalization gap, the test minus the training loss, is increasing or not [Goodfellow et al., 2016]. We can further decompose this situation of the generalization gap into two concepts: The first concept is the situation where both the training and test losses are decreasing, but the training loss is decreasing faster than the test loss ([A] in Fig. 1(a).) The next concept is the situation where the training loss is decreasing, but the test loss is increasing. This tends to occur after the first concept ([B] in Fig. 1(a)).

Within the concept [B], after learning for even more epochs, the training loss will continue to decrease and may become (near-)zero. This is shown as [C] in Fig. 1(a). If you continue training



(a) w/o Flooding **(b)** w/ Flooding **(c)** CIFAR-10 w/o Flooding **(d)** CIFAR-10 w/ Flooding

Figure 1: (a) shows 3 different concepts related to overfitting. [A] shows the generalization gap increases, while training & test losses decrease. [B] also shows the increasing gap, but the test loss starts to rise. [C] shows the training loss becoming (near-)zero. We avoid [C] by *flooding* the bottom area, visualized in (b), which forces the training loss to stay around a constant. This leads to a decreasing test loss once again. We confirm these claims in experiments with CIFAR-10 shown in (c)–(d).

even after the model has memorized [Arpit et al., 2017, Belkin et al., 2018, Zhang et al., 2017] the training data completely with zero error, the training loss can easily become (near-)zero especially with overparametrized models. Recent works on overparametrization and double descent curves [Belkin et al., 2019, Nakkiran et al., 2020] have shown that learning until zero training error is meaningful to achieve a lower generalization error. However, whether zero training *loss* is necessary after achieving zero training *error* remains an open issue.

In this paper, we propose a method to make the training loss *float* around a small constant value, in order to prevent the training loss from approaching zero. This is analogous to *flooding* the bottom area with water, and we refer to the constant value as the *flooding level*. Note that even if we add flooding, we can still memorize the training data. Our proposal only forces the training *loss* to become positive, which does not necessarily mean the training *error* will become positive, as long as the flooding level is not too large. The idea of flooding is shown in Fig. 1(b), and we show learning curves before and after flooding with benchmark experiments in Fig. 1(c) and Fig. 1(d).[1]

Algorithm and implementation Our algorithm of flooding is surprisingly simple. If the original learning objective is J, the proposed modified learning objective $\widetilde{J}$ with flooding is

$$\widetilde{J}(\boldsymbol{\theta}) = |J(\boldsymbol{\theta}) - b| + b, \tag{1}$$

where $b > 0$ is the flooding level specified by the user, and $\boldsymbol{\theta}$ is the model parameter.

The gradient of $\widetilde{J}$ w.r.t. $\boldsymbol{\theta}$ will point in the same direction as that of $J(\boldsymbol{\theta})$ when $J(\boldsymbol{\theta}) > b$ but in the opposite direction when $J(\boldsymbol{\theta}) < b$. This means that when the learning objective is above the flooding level, there is a "gravity" effect with gradient *descent*, but when the learning objective is below the flooding level, there is a "buoyancy" effect with gradient *ascent*. In practice, this will be performed with a mini-batch, and will be compatible with any stochastic optimizers. It can also be used along with other regularization methods.

[1]For the details of these experiments, see Appendix D.

During flooding, the training loss will repeat going below and above the flooding level. The model will continue to "random walk" with the same non-zero training loss, and we expect it to drift into an area with a flat loss landscape that leads to better generalization [Chaudhari et al., 2017, Keskar et al., 2017, Li et al., 2018]. [2]

Since it is a simple solution, this modification can be incorporated into existing machine learning code easily: Add one line of code for Eq. (1), after evaluating the original objective function $J(\boldsymbol{\theta})$. A minimal working example with a mini-batch in PyTorch [Paszke et al., 2019] is demonstrated below to show the additional one line of code:

```
outputs = model(inputs)
loss = criterion(outputs, labels)
flood = (loss-b).abs()+b  # This is it!
optimizer.zerograd()
flood.backward()
optimizer.step()
```

It may be hard to set the flooding level without expert knowledge on the domain or task. We can circumvent this situation easily, by treating the flooding level as a hyper-parameter. We may use a naive search, which exhaustively evaluates the accuracy for the predefined hyper-parameter candidates with a validation dataset. This procedure can be performed in parallel.

Previous regularization methods Many previous regularization methods also aim at avoiding *training too much* in various ways, e.g., restricting the parameter norm to become small by decaying the parameter weights [Hanson and Pratt, 1988], raising the difficulty of training by dropping activations of neural networks [Srivastava et al., 2014], smoothing the training labels [Szegedy et al., 2016], or simply stopping training at an earlier phase [Morgan and Bourlard, 1990]. These methods can be considered as indirect ways to control the training loss, by also introducing additional assumptions, e.g., the optimal model parameters are close to zero. Although making the regularization effect stronger would make it harder for the training loss to approach zero, it is still hard to maintain the right level of training loss till the end of training. In fact, for overparametrized deep networks, applying a small regularization parameter would not stop the training loss becoming (near-)zero, making it even harder to choose a hyper-parameter that corresponds to a specific level of loss.

Flooding, on the other hand, is a direct solution to the issue that the training loss becomes (near-)zero. Flooding intentionally prevents further reduction of the training loss when it reaches a reasonably small value, and the flooding level corresponds to the level of training loss that the user wants to keep.

Contributions Our proposed regularizer called *flooding* makes the training loss float around a small constant value, instead of making it head towards zero loss. Flooding is a regularizer that is domain-, task-, and model-independent. Theoretically, we find that the mean squared error can be reduced with flooding under certain conditions. Not only do we show test accuracy improving

[2] In Appendix F, we show that during this period of random walk, there is an increase in flatness of the loss function.

after flooding, we also observe that even after we avoid zero training loss, memorization with zero training error still takes place.

2 Backgrounds

In this section, we review regularization methods (summarized in Table 1), recent works on overparametrization and double descent curves, and the area of weakly supervised learning where similar techniques to flooding has been explored.

2.1 Regularization Methods

The name "regularization" dates back to at least Tikhonov regularization for the ill-posed linear least-squares problem [Tikhonov and Arsenin, 1977, Tikhonov, 1943]. One example is to modify $\boldsymbol{X}^\top \boldsymbol{X}$ (where $\boldsymbol{X}$ is the design matrix) to become "regular" by adding a term to the objective function. ℓ_2 regularization is a generalization of the above example and can be applied to non-linear models. These methods implicitly assume that the optimal model parameters are close to zero.

It is known that weight decay [Hanson and Pratt, 1988], dropout [Srivastava et al., 2014], and early stopping [Morgan and Bourlard, 1990] are equivalent to ℓ_2 regularization under certain conditions [Bishop, 1995, Goodfellow et al., 2016, Loshchilov and Hutter, 2019, Wager et al., 2013], implying that there is a similar assumption on the optimal model parameters. There are other penalties based on different assumptions, such as the ℓ_1 regularization [Tibshirani, 1996] based on the sparsity assumption that the optimal model has only a few non-zero parameters.

Modern machine learning tasks are applied to complex problems where the optimal model parameters are not necessarily close to zero or may not be sparse, and it would be ideal if we can properly add regularization effects to the optimization stage without such assumptions. Our proposed method does not have assumptions on the optimal model parameters and can be useful for more complex problems.

More recently, "regularization" has further evolved to a more general meaning, including various methods that alleviate overfitting, but do not necessarily have a step to regularize a singular matrix or add a regularization term to the objective function. For example, Goodfellow et al. [2016] defines regularization as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." In this paper, we adopt this broader meaning of "regularization."

Examples of the more general regularization category include mixup [Zhang et al., 2018] and data augmentation methods like cropping and flipping or adjusting brightness or sharpness [Shorten and Khoshgoftaar, 2019]. These methods have been adopted in many papers to obtain state-of-the-art performance [Berthelot et al., 2019, Verma et al., 2019] and are becoming essential regularization tools for developing new systems. However, these regularization methods have the drawback of being domain-specific: They are designed for the vision domain and require some efforts when applying to other domains [Guo et al., 2019, Thulasidasan et al., 2019]. Other regularizers such as label smoothing [Szegedy et al., 2016] is used for problems with class labels,

Table 1: Conceptual comparisons of various regularizers. "tr." stands for "training", "Indep." stands for "independent", ✓ stands for yes, and × stands for no.

Regularization and other methods	Target tr. loss	Domain indep.	Task indep.	Model indep.	Main assumption
ℓ_2 regularization [Tikhonov, 1943]	×	✓	✓	✓	Optimal model params are close to 0
Weight decay [Hanson and Pratt, 1988]	×	✓	✓	✓	Optimal model params are close to 0
Early stopping [Morgan and Bourlard, 1990]	×	✓	✓	✓	Overfitting occurs in later epochs
ℓ_1 regularization [Tibshirani, 1996]	×	✓	✓	✓	Optimal model has to be sparse
Dropout [Srivastava et al., 2014]	×	✓	✓	×	Weight scaling inference rule
Batch normalization [Ioffe and Szegedy, 2015]	×	✓	✓	×	Existence of internal covariate shift
Label smoothing [Szegedy et al., 2016]	×	✓	×	✓	True posterior is not a one-hot vector
Mixup [Zhang et al., 2018]	×	×	×	✓	Linear relationship between $\boldsymbol{x}$ and y
Image augment. [Shorten and Khoshgoftaar, 2019]	×	×	✓	✓	Input is invariant to the translations
Flooding (proposed method)	✓	✓	✓	✓	Learning until zero loss is harmful

and harder to use with regression or ranking, meaning they are task-specific. Batch normalization [Ioffe and Szegedy, 2015] and dropout [Srivastava et al., 2014] are designed for neural networks and are model-specific.

Although these regularization methods—both the *special* and *general* ones—already work well in practice and have become the de facto standard tools [Bishop, 2011, Goodfellow et al., 2016], we provide an alternative which is even more general in the sense that it is domain-, task-, and model-independent.

That being said, we want to emphasize that the most important difference between flooding and other regularization methods is whether it is possible to target a specific level of training loss other than zero. While flooding allows the user to choose the level of training loss directly, it is hard to achieve this with other regularizers.

2.2 Double Descent Curves with Overparametrization

Recently, there has been increasing attention on the phenomenon of "double descent," named by Belkin et al. [2019] to explain the two regimes of deep learning: The first one (underparametrized regime) occurs where the model complexity is small compared to the number of training samples, and the test error as a function of model complexity decreases with low model complexity but starts to increase after the model complexity is large enough. This follows the classical view of machine learning that excessive complexity leads to poor generalization. The second one (overparametrized regime) occurs when an even larger model complexity is considered. Then increasing the complexity only decreases test error, which leads to a double descent shape. The phase of decreasing test error often occurs after the training error becomes zero. This follows the modern view of machine learning that bigger models lead to better generalization.[3]

As far as we know, the discovery of double descent curves dates back to at least Krogh and

[3] https://www.eff.org/ai/metrics

Hertz [1992], where they theoretically showed the double descent phenomenon under a linear regression setup. Recent works [Belkin et al., 2019, Nakkiran et al., 2020] have shown empirically that a similar phenomenon can be observed with deep learning methods. Nakkiran et al. [2020] observed that the double descent curves can be shown not only as a function of model complexity, but also as a function of the epoch number.

We want to note a byproduct of our flooding method: We were able to produce the epoch-wise double descent curve for the test loss with about 100 epochs. Investigating the connection between our accelerated double descent curves and previous double descent curves [Belkin et al., 2019, Krogh and Hertz, 1992, Nakkiran et al., 2020] is out of the scope of this paper but is an important future direction.

2.3 Lower-Bounding the Empirical Risk

Lower-bounding the empirical risk has been used in the area of *weakly supervised learning*: There were a common phenomenon where the empirical risk goes below zero [Kiryo et al., 2017], when an equivalent form of the risk expressed with the given weak supervision was alternatively used [Cid-Sueiro et al., 2014, du Plessis et al., 2014, 2015, Natarajan et al., 2013, Patrini et al., 2017, van Rooyen and Williamson, 2018]. A gradient ascent technique was used to force the empirical risk to become non-negative in Kiryo et al. [2017]. This idea has been generalized and applied to other weakly supervised settings [Han et al., 2018, Ishida et al., 2019, Lu et al., 2020].

Although we also set a lower bound on the empirical risk, the motivation is different: First, while Kiryo et al. [2017] and others aim to fix the negative empirical risk to become lower bounded by zero, our empirical risk already has a lower bound of zero. Instead, we are aiming to sink the original empirical risk, by placing a *positive* lower bound. Second, the problem settings are different. Weakly supervised learning methods require certain loss corrections or sample corrections [Han et al., 2018] before the non-negative correction, but we work on the original empirical risk without any setting-specific modifications.

3 Flooding: How to Avoid Zero Training Loss

In this section, we propose our regularization method, *flooding*. Note that this section and the following sections only consider multi-class classification for simplicity.

3.1 Preliminaries

Consider input variable $\boldsymbol{x} \in \mathbb{R}^d$ and output variable $y \in \{1, \ldots, K\}$, where K is the number of classes. They follow an unknown joint probability distribution with density $p(\boldsymbol{x}, y)$. We denote the score function by $\boldsymbol{g} : \mathbb{R}^d \to \mathbb{R}^K$. For any test data point $\boldsymbol{x}_0$, our prediction of the output label will be given by $\widehat{y}_0 := \arg\max_{z \in \{1,\ldots,K\}} g_z(\boldsymbol{x}_0)$, where $g_z(\cdot)$ is the z-th element of $\boldsymbol{g}(\cdot)$, and in case of a tie, $\arg\max$ returns the largest argument. Let $\ell : \mathbb{R}^K \times \{1, 2, \ldots, K\} \to \mathbb{R}$ denote a loss

function. ℓ can be the *zero-one loss*,

$$\ell_{01}(\boldsymbol{v}, z') := \begin{cases} 0 & \text{if} \quad \arg\max_{z \in \{1,\dots,K\}} v_z = z', \\ 1 & \text{otherwise}, \end{cases} \tag{2}$$

where $\boldsymbol{v} := (v_1, \dots, v_K)^\top \in \mathbb{R}^K$, or a surrogate loss such as the softmax cross-entropy loss,

$$\ell_{\mathrm{CE}}(\boldsymbol{v}, z') := -\log \frac{\exp(v_{z'})}{\sum_{z \in \{1,\dots,K\}} \exp(v_z)}. \tag{3}$$

For a surrogate loss ℓ, we denote the *classification risk* by

$$R(\boldsymbol{g}) := \mathbb{E}_{p(\boldsymbol{x},y)}[\ell(\boldsymbol{g}(\boldsymbol{x}), y)] \tag{4}$$

where $\mathbb{E}_{p(\boldsymbol{x},y)}[\cdot]$ is the expectation over $(\boldsymbol{x}, y) \sim p(\boldsymbol{x}, y)$. We use $R_{01}(\boldsymbol{g})$ to denote Eq. (4) when $\ell = \ell_{01}$ and call it the *classification error*.

The goal of multi-class classification is to learn $\boldsymbol{g}$ that minimizes the classification error $R_{01}(\boldsymbol{g})$. In optimization, we consider the minimization of the risk with a almost surely differentiable surrogate loss $R(\boldsymbol{g})$ instead to make the problem more tractable. Furthermore, since $p(\boldsymbol{x}, y)$ is usually unknown and there is no way to exactly evaluate $R(\boldsymbol{g})$, we minimize its empirical version calculated from the training data instead:

$$\widehat{R}(\boldsymbol{g}) := \frac{1}{n} \sum_{i=1}^{n} \ell(\boldsymbol{g}(\boldsymbol{x}_i), y_i), \tag{5}$$

where $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ are i.i.d. sampled from $p(\boldsymbol{x}, y)$. We call $\widehat{R}$ the *empirical risk*.

We would like to clarify some of the undefined terms used in the title and the introduction. The "train/test loss" is the empirical risk with respect to the surrogate loss function ℓ over the training/test data, respectively. We refer to the "training/test error" as the empirical risk with respect to ℓ_{01} over the training/test data, respectively (which is equal to one minus accuracy) [Zhang, 2004].

Finally, we formally define the Bayes risk as

$$R^* := \inf_{\boldsymbol{h}} R(\boldsymbol{h}), \tag{6}$$

where the infimum is taken over all vector-valued functions $\boldsymbol{h} \colon \mathbb{R}^d \to \mathbb{R}^K$. The Bayes risk is often referred to as the *Bayes error* if the zero-one loss is used:

$$\inf_{\boldsymbol{h}} R_{01}(\boldsymbol{h}). \tag{7}$$

3.2 Algorithm

With flexible models, $\widehat{R}(\boldsymbol{g})$ w.r.t. a surrogate loss can easily become small if not zero, as we mentioned in Section 1; see [C] in Fig. 1(a). We propose a method that "floods the bottom area and sinks the original empirical risk" as in Fig. 1(b) so that the empirical risk cannot go below the flooding level. More technically, if we denote the flooding level as b, our proposed training objective with flooding is a simple fix:

Definition 1. *The* flooded empirical risk *is defined as*[4]

$$\widetilde{R}(\boldsymbol{g}) = |\widehat{R}(\boldsymbol{g}) - b| + b. \tag{8}$$

Note that when $b = 0$, then $\widetilde{R}(\boldsymbol{g}) = \widehat{R}(\boldsymbol{g})$. The gradient of $\widetilde{R}(\boldsymbol{g})$ w.r.t. model parameters will point to the same direction as that of $\widehat{R}(\boldsymbol{g})$ when $\widehat{R}(\boldsymbol{g}) > b$ but in the opposite direction when $\widehat{R}(\boldsymbol{g}) < b$. This means that when the learning objective is above the flooding level, we perform gradient *descent* as usual (gravity zone), but when the learning objective is below the flooding level, we perform gradient *ascent* instead (buoyancy zone).

The issue is that in general, we seldom know the optimal flooding level in advance. This issue can be mitigated by searching for the optimal flooding level b^* with a hyper-parameter optimization technique. In practice, we can search for the optimal flooding level by performing the exhaustive search in parallel.

3.3 Implementation

For large scale problems, we can employ mini-batched stochastic optimization for efficient computation. Suppose that we have M disjoint mini-batch splits. We denote the empirical risk (5) with respect to the m-th mini-batch by $\widehat{R}_m(\boldsymbol{g})$ for $m \in \{1, \ldots, M\}$. Then, our mini-batched optimization performs gradient descent updates in the direction of the gradient of $\widetilde{R}_m(\boldsymbol{g})$. By the convexity of the absolute value function and Jensen's inequality, we have

$$\widetilde{R}(\boldsymbol{g}) \leq \frac{1}{M} \sum_{m=1}^{M} \left(|\widehat{R}_{\mathrm{m}}(\boldsymbol{g}) - b| + b \right). \tag{9}$$

This indicates that mini-batched optimization will simply minimize an upper bound of the full-batch case with $\widetilde{R}(\boldsymbol{g})$.

3.4 Theoretical Analysis

In the following theorem, we will show that the mean squared error (MSE) of the proposed risk estimator with flooding is smaller than that of the original risk estimator without flooding.

Theorem 1. *Fix any measurable vector-valued function $\boldsymbol{g}$. If the flooding level b satisfies $\widehat{R}(\boldsymbol{g}) < b < R(\boldsymbol{g})$, we have*

$$\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) > \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})). \tag{10}$$

If $b \leq \widehat{R}(\boldsymbol{g})$, we have

$$\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) = \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})). \tag{11}$$

A proof is given in Appendix A. If we regard $\widehat{R}(\boldsymbol{g})$ as the training loss and $R(\boldsymbol{g})$ as the test loss, we would want b to be between those two for the MSE to improve.

[4]Strictly speaking, Eq. (1) is different from Eq. (8), since Eq. (1) can ignore constant terms of the original empirical risk. We will refer to Eq. (8) for the flooding operator for the rest of the paper.

4 Experiments

In this section, we show experimental results with synthetic and benchmark datasets. The implementation is based on PyTorch [Paszke et al., 2019] and demo code will be available[5]. Experiments were carried out with NVIDIA GeForce GTX 1080 Ti, NVIDIA Quadro RTX 5000 and Intel Xeon Gold 6142.

4.1 Synthetic Experiments

The aim of our synthetic experiments is to study the behavior of flooding with a controlled setup. We use three types of synthetic data described below.

Two Gaussians Data: We perform binary classification with two 10-dimensional Gaussian distributions with covariance matrix identity and means $\mu_{\mathrm{P}} = [0, 0, \dots, 0]^\top$ and $\mu_{\mathrm{N}} = [m, m, \dots, m]^\top$, where $m \in \{0.8, 1.0\}$. The Bayes risk for $m = 1.0$ and $m = 0.8$ are 0.14 and 0.24, respectively, where proofs are shown in Appendix B. The training, validation, and test sample sizes are 25, 10000, and 10000 per class respectively.

Sinusoid Data: The sinusoid data [Nakkiran et al., 2019] are generated as follows. We first draw input data points uniformly from the inside of a 2-dimensional ball of radius 1. Then we put class labels based on

$$y = \mathrm{sign}(\boldsymbol{x}^\top \boldsymbol{w} + \sin(\boldsymbol{x}^\top \boldsymbol{w}')),$$

where $\boldsymbol{w}$ and $\boldsymbol{w}'$ are any two 2-dimesional vectors such that $\boldsymbol{w} \perp \boldsymbol{w}'$. The training, validation, and test sample sizes are 100, 100, and 20000, respectively.

Spiral Data: The spiral data [Sugiyama, 2015] are two-dimensional synthetic data. Let $\theta_1^+ := 0, \theta_2^+, \dots, \theta_{n^+}^+ := 4\pi$ be equally spaced n^+ points in the interval $[0, 4\pi]$, and $\theta_1^- := 0, \theta_2^-, \dots, \theta_{n^-}^- := 4\pi$ be equally spaced n^- points in the interval $[0, 4\pi]$. Let positive and negative input data points be

$$\begin{aligned}
\boldsymbol{x}_{i^+}^+ &:= \theta_{i^+}^+ [\cos(\theta_{i^+}^+), \sin(\theta_{i^+}^+)]^\top + \tau \boldsymbol{\nu}_{i^+}^+, \\
\boldsymbol{x}_{i^-}^- &:= (\theta_{i^-}^- + \pi) [\cos(\theta_{i^-}^-), \sin(\theta_{i^-}^-)]^\top + \tau \boldsymbol{\nu}_{i^-}^-
\end{aligned}$$

for $i_+ = 1, \dots, n^+$ and $i_- = 1, \dots, n^-$ where τ controls the magnitude of the noise, $\boldsymbol{\nu}_i^+$ and $\boldsymbol{\nu}_i^-$ are i.i.d. distributed according to the two-dimensional standard normal distribution. Then, we make data for classification by $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n := \{(\boldsymbol{x}_{i^+}^+, +1)\}_{i^+=1}^{n^+} \cup \{(\boldsymbol{x}_{i^-}^-, -1)\}_{i^-=1}^{n^-}$, where $n := n^+ + n^-$. The training, validation, and test sample sizes are 100, 100, and 10000 per class respectively.

For Two Gaussians, we use a one-hidden-layer feedforward neural network with 500 units in the hidden layer with the ReLU activation function [Nair and Hinton, 2010]. We train the network for 1000 epochs with the logistic loss and vanilla gradient descent with learning rate of 0.05. The flooding level is chosen from $b \in \{0, 0.01, 0.02, \dots, 0.40\}$. For Sinusoid and Spiral, we use a four-hidden-layer feedforward neural network with 500 units in the hidden layer, with the

[5] https://github.com/takashiishida/flooding

Table 2: Experimental results for the synthetic data. Sub-table (A) shows the results without early stopping. Sub-table (B) shows the results with early stopping. The better method is shown in **bold** in each of sub-tables (A) and (B). "BR" stands for the Bayes risk. For Two Gaussians, the distance between the positive and negative distributions is larger for larger m. See the description in Section 4.1 for the details.

		(A) Without Early Stopping			(B) With Early Stopping		
Data	Setting	Without Flooding	With Flooding	Chosen b	Without Flooding	With Flooding	Chosen b
Two Gaussians	m: 1.0, BR: 0.14	87.96%	**92.25%**	0.28	91.63%	**92.25%**	0.27
Two Gaussians	m: 0.8, BR: 0.24	82.00%	**87.31%**	0.33	86.57%	**87.29%**	0.35
Sinusoid	Label Noise: 0.01	93.84%	**94.46%**	0.01	**92.54%**	**92.54%**	0.00
Sinusoid	Label Noise: 0.05	91.12%	**95.44%**	0.10	93.26%	**94.60%**	0.01
Sinusoid	Label Noise: 0.10	86.57%	**96.02%**	0.17	**96.70%**	**96.70%**	0.00
Spiral	Label Noise: 0.01	**98.96%**	97.85%	0.01	98.60%	**98.88%**	0.01
Spiral	Label Noise: 0.05	93.87%	**96.24%**	0.04	**96.58%**	95.62%	0.14
Spiral	Label Noise: 0.10	89.70%	**92.96%**	0.16	89.70%	**92.96%**	0.16

ReLU activation function [Nair and Hinton, 2010], and batch normalization [Ioffe and Szegedy, 2015]. We train the network for 500 epochs with the logistic loss and Adam [Kingma and Ba, 2015] optimizer with 100 mini-batch size and learning rate of 0.001. The flooding level is chosen from $b \in \{0, 0.01, 0.02, \ldots, 0.20\}$. Note that training with $b = 0$ is identical to the baseline method without flooding. We report the test accuracy of the flooding level with the best validation accuracy. We first conduct experiments without early stopping, which means that the last epoch was chosen for all flooding levels.

Results The results are summarized in Table 2. It is worth noting that for Two Gaussians, the chosen flooding level b is larger for the smaller distance between the two distributions, which is when the classification task is harder and the Bayes risk becomes larger since the two distributions become less separated. We see similar tendencies for Sinusoid and Spiral data: a larger b was chosen for larger flipping probability for label noise, which is expected to increase the Bayes risk. This implies the positive correlation between the optimal flooding level and the Bayes risk, as is also partially suggested by Theorem 1. Another interesting observation is that the chosen b is close to but higher than the Bayes risk for Two Gaussians data. This may look inconsistent with Theorem 1. However, it makes sense to adopt larger b with stronger regularization effect that allows some bias as a trade-off for reducing the variance of the risk estimator. In fact, Theorem 1 does not deny the possibility that some $b \geq R(\boldsymbol{g})$ achieves even better estimation.

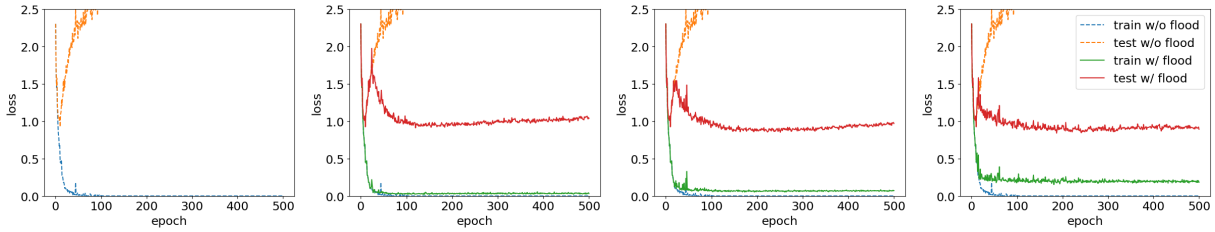
From (A) in Table 2, we can see that the method with flooding often improves test accuracy over the baseline method without flooding. As we mentioned in the introduction, it ca be harmful to keep training a model until the end without flooding. However, with flooding, the model at the final epoch has good prediction performance according to the results, which implies that flooding

Table 3: Results with benchmark datasets. We report classification accuracy for all combinations of weight decay (✓ and ×), early stopping (✓ and ×) and flooding (✓ and ×). The second column shows the training/validation split used for the experiment. W stands for weight decay, E stands for early stopping, and F stands for flooding. "—" means that flooding level of zero was optimal. "N/A" means that we skipped the experiments because zero weight decay was optimal in the case without flooding. The best and equivalent are shown in **bold** by comparing "with flooding" and "without flooding" for two columns with the same setting for W and E, e.g., the first and fifth columns out of the 8 columns. The best performing combination is highlighted.

Dataset	tr/va split	W:	×	×	✓	✓	×	×	✓	✓
		E:	×	✓	×	✓	×	✓	×	✓
		F:	×	×	×	×	✓	✓	✓	✓
MNIST	0.8		98.32%	98.30%	**98.51%**	98.42%	**98.46%**	**98.53%**	98.50%	**98.48%**
	0.4		97.71%	97.70%	97.82%	**97.91%**	**97.74%**	**97.85%**	—	97.83%
Fashion-MNIST	0.8		89.34%	89.36%	N/A	N/A	—	—	N/A	N/A
	0.4		88.48%	88.63%	88.60%	88.62%	—	—	—	—
Kuzushiji-MNIST	0.8		91.63%	91.62%	91.63%	91.71%	**92.40%**	**92.12%**	**92.11%**	**91.97%**
	0.4		89.18%	89.18%	89.58%	89.73%	**90.41%**	**90.15%**	**89.71%**	**89.88%**
CIFAR-10	0.8		**73.59%**	73.36%	73.65%	73.57%	73.06%	**73.44%**	—	**74.41%**
	0.4		66.39%	66.63%	69.31%	69.28%	**67.20%**	**67.58%**	—	—
CIFAR-100	0.8		42.16%	42.33%	42.67%	42.45%	**42.50%**	**42.36%**	—	—
	0.4		34.27%	34.34%	37.97%	38.82%	**34.99%**	**35.14%**	—	—
SVHN	0.8		92.38%	92.41%	93.20%	92.99%	**92.78%**	**92.79%**	—	**93.42%**
	0.4		90.32%	90.35%	90.43%	90.49%	**90.57%**	**90.61%**	**91.16%**	**91.21%**

helps the late-stage training improve test accuracy.

We also conducted experiments with early stopping, meaning that we chose the model that recorded the best validation accuracy during training. The results are reported in sub-table (B) of Table 2. Compared with sub-table (A), we see that early stopping improves the baseline method without flooding well in many cases. This indicates that training longer without flooding was harmful in our experiments. On the other hand, the accuracy for flooding combined with early stopping is often close to that with early stopping, meaning that training until the end with flooding tends to be already as good as doing so with early stopping. The table shows that flooding often improves or retains the test accuracy of the baseline method without flooding even after deploying early stopping. Flooding does not hurt performance but can be beneficial for methods used with early stopping.



(a) CIFAR-10 (0.00) **(b)** CIFAR-10 (0.03) **(c)** CIFAR-10 (0.07) **(d)** CIFAR-10 (0.20)

Figure 2: Learning curves of training and test loss for training/validation proportion of 0.8. (a) shows the learning curves without flooding. (b), (c), and (d) show the learning curves with different flooding levels.

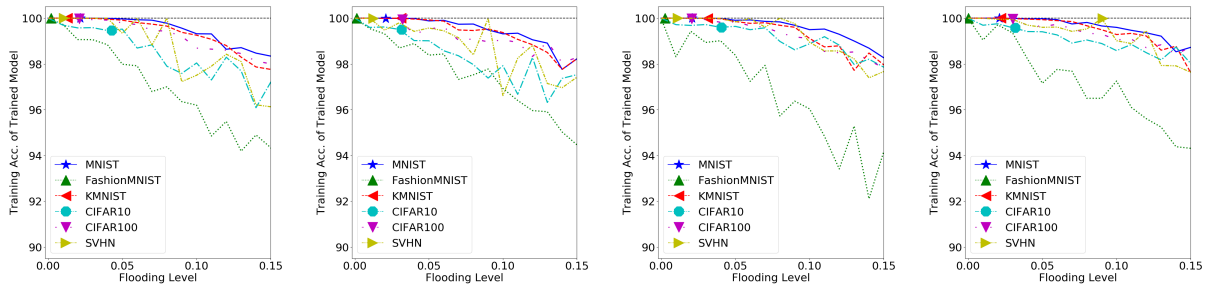
4.2 Benchmark Experiments

We next perform experiments with benchmark datasets. Not only do we compare with the baseline without flooding, we also compare or combine with other general regularization methods, which are early stopping and weight decay.

Settings We use the following six benchmark datasets: MNIST, Fashion-MNIST, Kuzushiji-MNIST, CIFAR-10, CIFAR-100, and SVHN. The details of the benchmark datasets can be found in Appendix C.1. We split the original training dataset into training and validation data with different proportions: 0.8 or 0.4 (meaning 80% or 40% was used for training and the rest was used for validation, respectively). We perform the exhaustive hyper-parameter search for the flooding level with candidates from $\{0.00, 0.01, \ldots, 0.20\}$. The number of epochs is 500. Stochastic gradient descent [Robbins and Monro, 1951] is used with learning rate of 0.1 and momentum of 0.9. For MNIST, Fashion-MNIST, and Kuzushiji-MNIST, we use a one-hidden-layer feedforward neural network with 500 units and ReLU activation function [Nair and Hinton, 2010]. For CIFAR-10, CIFAR-100, and SVHN, we used ResNet-18 [He et al., 2016]. We do not use any data augmentation or manual learning rate decay. We deployed early stopping in the same way as in Section 4.1.

We first ran experiments with the following candidates for the weight decay rate: $\{1\times10^{-5}, 1\times10^{-4}, 4\times10^{-4}, 7\times10^{-4}, 1\times10^{-3}, 4\times10^{-3}, 7\times10^{-3}\}$. We choose the weight decay rate with the best validation accuracy, for each dataset and each training/validation proportion. Then, fixing the weight decay to the chosen one, we ran experiments with flooding level candidates from $\{0, 0.01, \ldots, 0.20\}$, to investigate whether weight decay and flooding have complementary effects, or if adding weight decay will diminish the accuracy gain of flooding.

Results We show the results in Table 3 and the chosen flooding levels in Table 4 in Appendix C.2. We can observe that flooding gives better accuracy for most cases. We can also see that combining flooding with early stopping or with both early stopping and weight decay may lead to even better accuracy in some cases.



(a) w/o early stop (0.8) **(b)** w/o early stop (0.4) **(c)** w/ early stop (0.8) **(d)** w/ early stop (0.4)

Figure 3: We show the optimal flooding level maintains memorization. The vertical axis shows the training accuracy. The horizontal axis shows the flooding level. We show results with and without early stopping, and for different training/validation splits with proportion of 0.8 or 0.4. The marks ($\star$, $\triangle$, $\triangleleft$, $\circ$, $\triangledown$, $\triangleright$) are placed on the flooding level that was chosen based on validation accuracy.

4.3 Memorization

Can we maintain memorization even after adding flooding? We investigate if the trained model has zero training error (100% accuracy) for the flooding level that was chosen with validation data. We show the results for all benchmark datasets and all training/validation splits with proportions 0.8 and 0.4. We also show the case without early stopping (choosing the last epoch) and with early stopping (choosing the epoch with the highest validation accuracy). The results are shown in Fig. 3.

All figures show downward curves, implying that the model will give up eventually on memorizing all training data as the flooding level becomes higher. A more interesting and important observation is the position of the optimal flooding level (the one chosen by validation accuracy which is marked with $\star$, $\triangle$, $\triangleleft$, $\circ$, $\triangledown$ or $\triangleright$). We can observe that the marks are often plotted at zero error, and in some cases there is a mark on the highest flooding level that maintains zero error. These results are consistent with recent empirical works that imply zero training error leads to lower generalization error [Belkin et al., 2019, Nakkiran et al., 2020], but we further demonstrate that zero training loss may be harmful under zero training error.

5 Conclusion

We proposed a novel regularization method called *flooding* that keeps the training loss to stay around a small constant value, to avoid zero training loss. In our experiments, the optimal flooding level often maintained memorization of training data, with zero error. With flooding, we showed that the test accuracy will improve for various benchmark datasets, and theoretically showed that the mean squared error will be reduced under certain conditions.

As a byproduct, we were able to produce a double descent curve for the test loss with a relatively few number of epochs, e.g., in around 100 epochs, shown in Fig. 2 and Fig. 4 in Appendix D. An important future direction is to study the relationship between this and the double descent curves

from previous works [Belkin et al., 2019, Krogh and Hertz, 1992, Nakkiran et al., 2020].

It would also be interesting to see if Bayesian optimization [Shahriari et al., 2016, Snoek et al., 2012] methods can be utilized to search for the optimal flooding level efficiently. We will investigate this direction in the future.

Acknowledgements

We thank Chang Xu, Genki Yamamoto, Kento Nozawa, Nontawat Charoenphakdee, Voot Tangkaratt, and Yoshihiro Nagano for the helpful discussions. TI was supported by the Google PhD Fellowship Program. MS and IY were supported by JST CREST Grant Number JPMJCR18A2 including AIP challenge program, Japan.

References

Devansh Arpit, Stanisffaw Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *ICML*, 2017.

Mikhail Belkin, Daniel Hsu, and Partha P. Mitra. Overfitting or perfect fitting? Risk bounds for classification and regression rules that interpolate. In *NeurIPS*, 2018.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical biasfi?!variance trade-off. *PNAS*, 116:15850–15854, 2019.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. MixMatch: A holistic approach to semi-supervised learning. In *NeurIPS*, 2019.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2011.

Christopher Michael Bishop. Regularization and complexity control in feed-forward networks. In *ICANN*, 1995.

Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *NeurIPS*, 2000.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. In *ICLR*, 2017.

Jesús Cid-Sueiro, Darío García-García, and Raúl Santos-Rodríguez. Consistency of losses for learning from weak labels. In *ECML-PKDD*, 2014.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical Japanese literature. In *NeurIPS Workshop on Machine Learning for Creativity and Design*, 2018.

Marthinus Christoffel du Plessis, Gang Niu, and Masashi Sugiyama. Analysis of learning from positive and unlabeled data. In *NeurIPS*, 2014.

Marthinus Christoffel du Plessis, Gang Niu, and Masashi Sugiyama. Convex formulation for learning from positive and unlabeled data. In *ICML*, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Hongyu Guo, Yongyi Mao, and Richong Zhang. Augmenting data with mixup for sentence classification: An empirical study. In *arXiv:1905.08941*, 2019.

Bo Han, Gang Niu, Jiangchao Yao, Xingrui Yu, Miao Xu, Ivor Tsang, and Masashi Sugiyama. Pumpout: A meta approach to robust deep learning with noisy labels. In *arXiv:1809.11008*, 2018.

Stephen Jose Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *NeurIPS*, 1988.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Takashi Ishida, Gang Niu, Aditya Krishna Menon, and Masashi Sugiyama. Complementary-label learning for arbitrary losses and models. In *ICML*, 2019.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017.

Diederik P. Kingma and Jimmy L. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Ryuichi Kiryo, Gang Niu, Marthinus Christoffel du Plessis, and Masashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator. In *NeurIPS*, 2017.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *NeurIPS*, 1992.

Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Nan Lu, Tianyi Zhang, Gang Niu, and Masashi Sugiyama. Mitigating overfitting in supervised classification from two unlabeled datasets: A consistent risk correction approach. In *AISTATS*, 2020.

N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *NeurIPS*, 1990.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Preetum Nakkiran, Gal Kaplun, Dimitris Kalimeris, Tristan Yang, Benjamin L. Edelman, Fred Zhang, and Boaz Barak. SGD on neural networks learns functions of increasing complexity. In *NeurIPS*, 2019.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *ICLR*, 2020.

Nagarajan Natarajan, Inderjit S. Dhillon, Pradeep K. Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *NeurIPS*, 2013.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Andrew Y. Ng. Preventing "overfitting" of cross-validation data. In *ICML*, 1997.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Giorgio Patrini, Alessandro Rozza, Aditya K. Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *CVPR*, 2017.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Rebecca Roelofs, Vaishaal Shankar, Benjamin Recht, Sara Fridovich-Keil, Moritz Hardt, John Miller, and Ludwig Schmidt. A meta-analysis of overfitting in machine learning. In *NeurIPS*, 2019.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Masashi Sugiyama. *Introduction to statistical machine learning*. Morgan Kaufmann, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jon Shlens. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Sunil Thulasidasan, Gopinath Chennupati, Jeff Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. In *NeurIPS*, 2019.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58:267–288, 1996.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill Posed Problems*. Winston, 1977.

Andrey Nikolayevich Tikhonov. On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39:195–198, 1943.

Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. In *IEEE Trans. PAMI*, 2008.

Brendan van Rooyen and Robert C. Williamson. A theory of learning with corrupted labels. *Journal of Machine Learning Research*, 18:1–50, 2018.

Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. Interpolation consistency training for semi-supervised learning. In *IJCAI*, 2019.

Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *NeurIPS*, 2013.

Roman Werpachowski, Andrs Gyrgy, and Csaba Szepesvri. Detecting overfitting via adversarial examples. In *NeurIPS*, 2019.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

Tong Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32:56–85, 2004.

A Proof of Theorem

Proof. If the flooding level is b, then the proposed flooding estimator is

$$\widetilde{R}(\boldsymbol{g}) = |\widehat{R}(\boldsymbol{g}) - b| + b. \tag{12}$$

Since the absolute operator can be expressed with a max operator with $\max(a, b) = \frac{a+b+|a-b|}{2}$, the proposed estimator can be re-expressed as,

$$\widetilde{R}(\boldsymbol{g}) = 2\max(\widehat{R}(\boldsymbol{g}), b) - \widehat{R}(\boldsymbol{g}) = A - \widehat{R}(\boldsymbol{g}). \tag{13}$$

For convenience, we used $A = 2\max(\widehat{R}(\boldsymbol{g}), b)$. From the definition of MSE,

$$\begin{aligned}
\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) &= \mathbb{E}[(\widehat{R}(\boldsymbol{g}) - R(\boldsymbol{g}))^2] && (14)\\
\mathrm{MSE}(\widetilde{R}(\boldsymbol{g})) &= \mathbb{E}[(\widetilde{R}(\boldsymbol{g}) - R(\boldsymbol{g}))^2] && (15)\\
&= \mathbb{E}[(A - \widehat{R}(\boldsymbol{g}) - R(\boldsymbol{g}))^2)] && (16)\\
&= \mathbb{E}[A^2] - 2\mathbb{E}[A(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g}))] + \mathbb{E}[(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g}))^2]. && (17)
\end{aligned}$$

We are interested in the sign of

$$\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) - \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})) = \mathbb{E}[-4\widehat{R}(\boldsymbol{g})R(\boldsymbol{g}) - A^2 + 2A(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g}))]. \tag{18}$$

Define the inside of the expectation as $B = -4\widehat{R}(\boldsymbol{g})R(\boldsymbol{g}) - A^2 + 2A(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g}))$. B can be divided into two cases, depending on the outcome of the max operator:

$$\begin{aligned}
B &= \begin{cases} -4\widehat{R}(\boldsymbol{g})R(\boldsymbol{g}) - 4\widehat{R}(\boldsymbol{g})^2 + 4\widehat{R}(\boldsymbol{g})(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g})) & \text{if} \quad \widehat{R}(\boldsymbol{g}) \geq b \\ -4\widehat{R}(\boldsymbol{g})R(\boldsymbol{g}) - 4b^2 + 4b(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g})) & \text{if} \quad \widehat{R}(\boldsymbol{g}) < b \end{cases} && (19)\\
&= \begin{cases} 0 & \text{if} \quad \widehat{R}(\boldsymbol{g}) \geq b \\ -4\widehat{R}(\boldsymbol{g})R(\boldsymbol{g}) - 4b^2 + 4b(\widehat{R}(\boldsymbol{g}) + R(\boldsymbol{g})) & \text{if} \quad \widehat{R}(\boldsymbol{g}) < b \end{cases} && (20)\\
&= \begin{cases} 0 & \text{if} \quad \widehat{R}(\boldsymbol{g}) \geq b \\ -4(b - \widehat{R}(\boldsymbol{g}))(b - R(\boldsymbol{g})) & \text{if} \quad \widehat{R}(\boldsymbol{g}) < b \end{cases}. && (21)
\end{aligned}$$

The latter case becomes positive when $\widehat{R}(\boldsymbol{g}) < b < R(\boldsymbol{g})$. Therefore, when $\widehat{R}(\boldsymbol{g}) < b < R(\boldsymbol{g})$,

$$\begin{aligned}
\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) - \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})) > 0 && (22)\\
\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) > \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})). && (23)
\end{aligned}$$

When $b \leq \widehat{R}(\boldsymbol{g})$,

$$\begin{aligned}
\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) - \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})) > 0 && (24)\\
\mathrm{MSE}(\widehat{R}(\boldsymbol{g})) = \mathrm{MSE}(\widetilde{R}(\boldsymbol{g})). && (25)
\end{aligned}$$

□

B Bayes Risk for Gaussian Distributions

In this section, we explain in detail how we derived the Bayes risk with respect to the surrogate loss in the experiments with Gaussian data in Section 4.1. Since we are using the logistic loss in the synthetic experiments, the loss of the margin is

$$\ell(yg(\boldsymbol{x})) = \log(1 + \exp(-yg(\boldsymbol{x}))), \tag{26}$$

where $g(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ is a scalar instead of the vector definition that was used previously, because the synthetic experiments only consider binary classification. Take the derivative to derive,

$$\frac{\partial \mathbb{E}[\ell(yg(\boldsymbol{x}))]}{\partial g(\boldsymbol{x})} = \frac{\mathbb{E}[\log(1 + \exp(-yg(\boldsymbol{x})))]}{\partial g(\boldsymbol{x})} = \mathbb{E}\left[\frac{-y\exp(-yg(\boldsymbol{x}))}{1 + \exp(-yg(\boldsymbol{x}))}\middle|\boldsymbol{x}\right] p(\boldsymbol{x}) \tag{27}$$

$$= \mathbb{E}\left[\frac{-y}{1 + \exp(yg(\boldsymbol{x}))}\middle|\boldsymbol{x}\right] p(\boldsymbol{x}) \tag{28}$$

$$= \mathbb{E}\left[\frac{y+1}{2}\frac{1}{\exp(-g(\boldsymbol{x})) + 1} + \frac{y-1}{2}\frac{-1}{\exp(g(\boldsymbol{x})) + 1}\middle|\boldsymbol{x}\right] p(\boldsymbol{x}) \tag{29}$$

$$= \mathbb{E}\left[\frac{y+1}{2}\middle|\boldsymbol{x}\right] \frac{1}{\exp(-g(\boldsymbol{x})) + 1} p(\boldsymbol{x}) + \mathbb{E}\left[\frac{y-1}{2}\middle|\boldsymbol{x}\right] \frac{-1}{\exp(g(\boldsymbol{x})) + 1} p(\boldsymbol{x}) \tag{30}$$

$$= p(y = +1|\boldsymbol{x})\frac{1}{\exp(-g(\boldsymbol{x})) + 1} p(\boldsymbol{x}) + p(y = -1|\boldsymbol{x})\frac{-1}{\exp(g(\boldsymbol{x})) + 1} p(\boldsymbol{x}) \tag{31}$$

Set this to zero, divide by $p(\boldsymbol{x}) > 0$ to obtain,

$$p(y = -1|\boldsymbol{x})\frac{1}{\exp(-g(\boldsymbol{x})) + 1} = p(y = +1|\boldsymbol{x})\frac{1}{\exp(g(\boldsymbol{x})) + 1} \tag{32}$$

$$\exp(g(\boldsymbol{x})) = \frac{p(y = +1|\boldsymbol{x})}{p(y = -1|\boldsymbol{x})} \tag{33}$$

$$g(\boldsymbol{x}) = \log \frac{p(y = +1|\boldsymbol{x})}{p(y = -1|\boldsymbol{x})} \tag{34}$$

Since we are interested in the surrogate loss under this classifier, we plug this into the logistic loss, to obtain the Bayes risk,

$$\mathbb{E}[\ell(yg(\boldsymbol{x}))] = \mathbb{E}\left[\log(1 + \frac{p(-y|\boldsymbol{x})}{p(y|\boldsymbol{x})})\right] = \mathbb{E}\left[\log(\frac{1}{p(y|\boldsymbol{x})})\right] = \mathbb{E}[-\log p(y|\boldsymbol{x})]. \tag{35}$$

In the experiments in Section 4.1, we report the empirical version of this with the test dataset as the Bayes risk.

C Details of Experiments

C.1 Benchmark Datasets

In the experiments in Section 4.2, we use six benchmark datasets explained below.

- MNIST[6] [Lecun et al., 1998] is a 10 class dataset of handwritten digits: $1, 2 \ldots, 9$ and 0. Each sample is a 28×28 grayscale image. The number of training and test samples are 60,000 and 10,000, respectively.
- Fashion-MNIST[7] [Xiao et al., 2017] is a 10 class dataset of fashion items: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. Each sample is a 28×28 grayscale image. The number of training and test samples are 60,000 and 10,000, respectively.
- Kuzushiji-MNIST[8] [Clanuwat et al., 2018] is a 10 class dataset of cursive Japanese ("Kuzushiji") characters. Each sample is a 28×28 grayscale image. The number of training and test samples are 60,000 and 10,000, respectively.
- CIFAR-10[9] is a 10 class dataset of various objects: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each sample is a colored image in $32 \times 32 \times 3$ RGB format. It is a subset of the 80 million tiny images dataset [Torralba et al., 2008]. There are 6,000 images per class, where 5,000 are for training and 1,000 are for test.
- CIFAR-100[10] is a 100 class dataset of various objects. Each class has 600 samples, where 500 samples are for training and 100 samples are for test. This is also a subset of the 80 million tiny images dataset [Torralba et al., 2008].
- SVHN[11] [Netzer et al., 2011] is a 10 class dataset of house numbers from Google Street View images, in $32 \times 32 \times 3$ RGB format. 73257 digits are for training and 26032 digits are for testing.

[6] http://yann.lecun.com/exdb/mnist/
[7] https://github.com/zalandoresearch/fashion-mnist
[8] https://github.com/rois-codh/kmnist
[9] https://www.cs.toronto.edu/~kriz/cifar.html
[10] https://www.cs.toronto.edu/~kriz/cifar.html
[11] http://ufldl.stanford.edu/housenumbers/

C.2 Chosen Flooding Levels

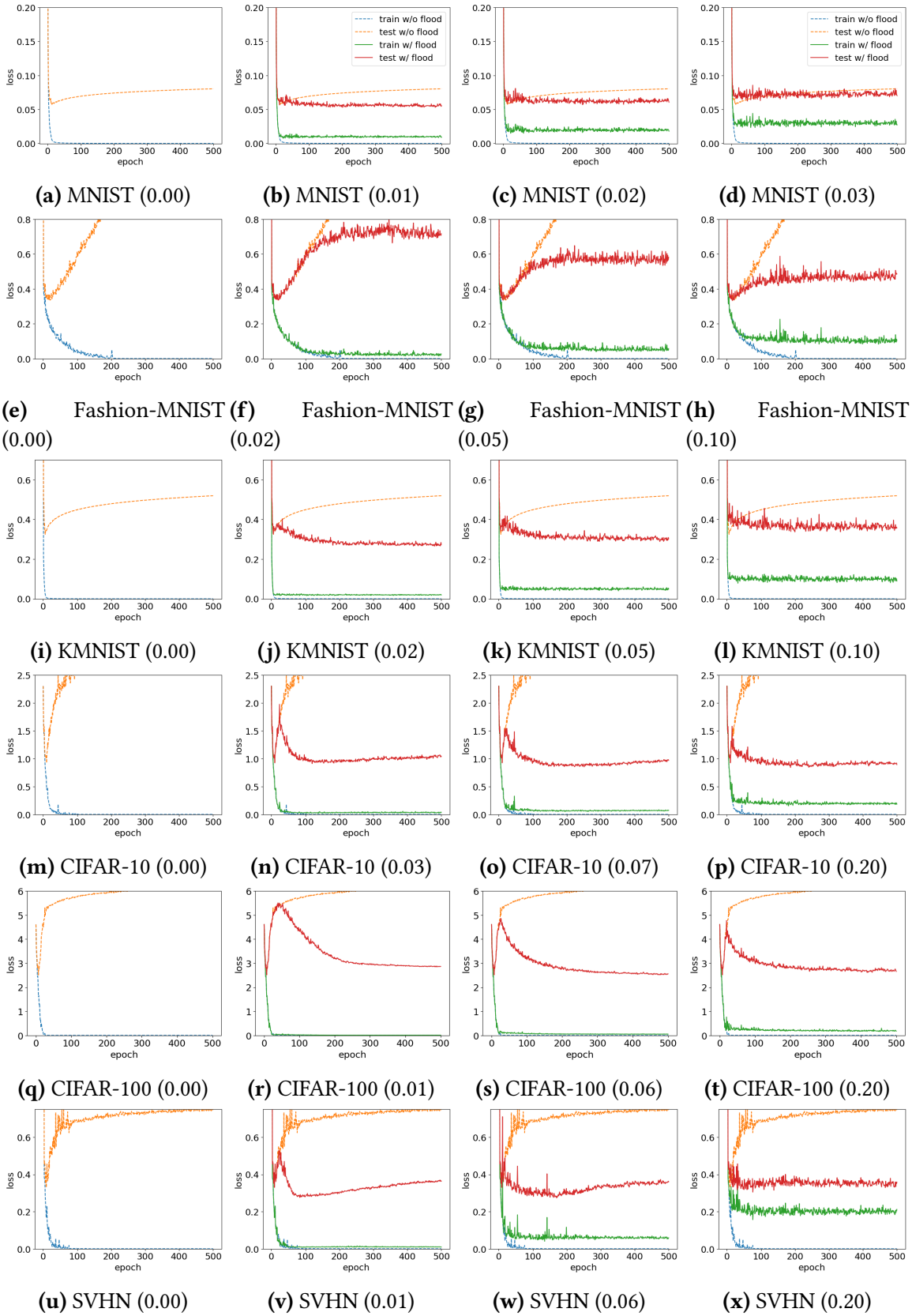
In Table 4, we report the chosen flooding levels for our experiments with benchmark datasets.

Table 4: The chosen flooding levels for benchmark experiments.

Early stopping	×	✓	×	✓
Weight decay	×	×	✓	✓
Flooding	✓	✓	✓	✓
MNIST (0.8)	0.02	0.02	0.03	0.02
MNIST (0.4)	0.02	0.03	0.00	0.02
Fashion-MNIST (0.8)	0.00	0.00	—	—
Fashion-MNIST (0.4)	0.00	0.00	—	—
Kuzushiji-MNIST (0.8)	0.01	0.03	0.03	0.03
Kuzushiji-MNIST (0.4)	0.03	0.02	0.04	0.03
CIFAR-10 (0.8)	0.04	0.04	0.00	0.01
CIFAR-10 (0.4)	0.03	0.03	0.00	0.00
CIFAR-100 (0.8)	0.02	0.02	0.00	0.00
CIFAR-100 (0.4)	0.03	0.03	0.00	0.00
SVHN (0.8)	0.01	0.01	0.00	0.02
SVHN (0.4)	0.01	0.09	0.03	0.03

D Learning Curves

In Figure 4, we visualize learning curves for all datasets (including CIFAR-10 which we already visualized in Figure 2). We only show the learning curves for training/validation proportion of 0.8, since the results for 0.4 were similar with 0.8. Note that Figure 1(c) shows the learning curves for the first 80 epochs for CIFAR-10 without flooding. Figure 1(d) shows the learning curves with flooding, when the flooding level is 0.18.



(a) MNIST (0.00) **(b)** MNIST (0.01) **(c)** MNIST (0.02) **(d)** MNIST (0.03)

(e) Fashion-MNIST (0.00) **(f)** Fashion-MNIST (0.02) **(g)** Fashion-MNIST (0.05) **(h)** Fashion-MNIST (0.10)

(i) KMNIST (0.00) **(j)** KMNIST (0.02) **(k)** KMNIST (0.05) **(l)** KMNIST (0.10)

(m) CIFAR-10 (0.00) **(n)** CIFAR-10 (0.03) **(o)** CIFAR-10 (0.07) **(p)** CIFAR-10 (0.20)

(q) CIFAR-100 (0.00) **(r)** CIFAR-100 (0.01) **(s)** CIFAR-100 (0.06) **(t)** CIFAR-100 (0.20)

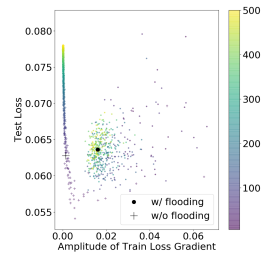
(u) SVHN (0.00) **(v)** SVHN (0.01) **(w)** SVHN (0.06) **(x)** SVHN (0.20)

Figure 4: Learning curves of training and test loss. The first figure in each row is the learning curves without flooding. The 2nd, 3rd, and 4th columns show the results with different flooding levels. The flooding level increases towards the right-hand side.

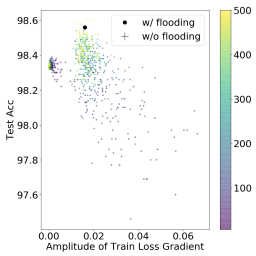
E Relationship between Performance and Gradients

Settings We visualize the relationship between test performance (loss or accuracy) and gradient amplitude of the training/test loss in Figure 5, where the gradient amplitude is the ℓ_2 norm of the *filter-nomalized gradient* of the loss. The filter-normalized gradient is the gradient appropriately scaled depending on the magnitude of the corresponding convolutional filter, similarly to Li et al. [2018]. More specicically, for each filter of every comvolutional layer, we multiply the corresponding elements of the gradient by the norm of the filter. Note that a fully connected layer is a special case of convolutional layer and subject to this scaling. We exclude Fashion-MNIST because the optimal flooding level was zero. We used the results with training/validation split ratio of 0.8.

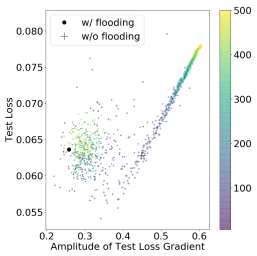
Results For the figures with gradient amplitude of training loss on the horizontal axis, "$\circ$" marks (w/ flooding) are often plotted on the right of "$+$" marks (w/o flooding), which implies that flooding prevents the model from staying a local minimum. For the figures with gradient amplitude of test loss on the horizontal axis, we can observe the method with flooding ("$\circ$") improves performance while the gradient amplitude becomes smaller. On the other hand, the performance with the method without flooding ("$+$") degenerates while the gradient amplitude of test loss keeps increasing.



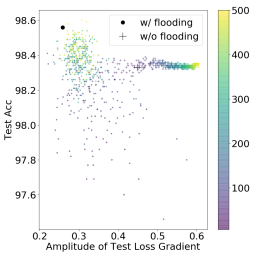
(a) MNIST, x:train, y:loss



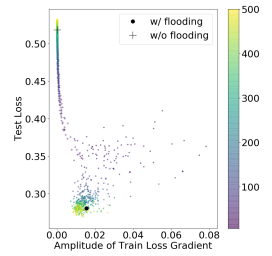
(b) MNIST, x:train, y:acc



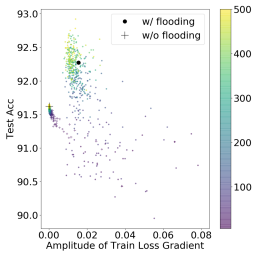
(c) MNIST, x:test, y:loss



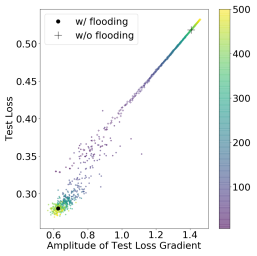
(d) MNIST, x:test, y:acc



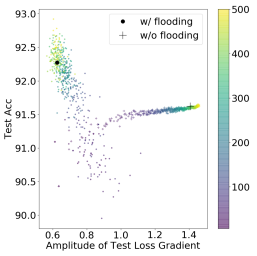
(e) K-M, x:train, y:loss



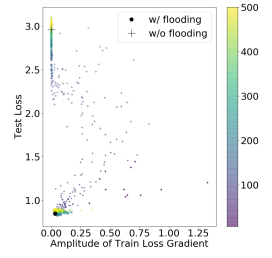
(f) K-M, x:train, y:acc



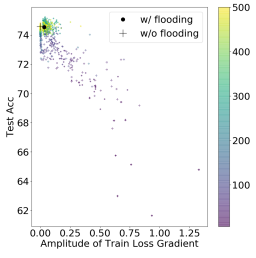
(g) K-M, x:test, y:loss



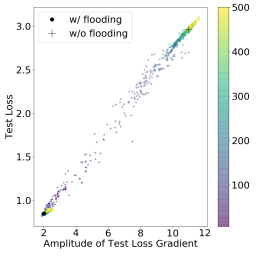
(h) K-M, x:test, y:acc



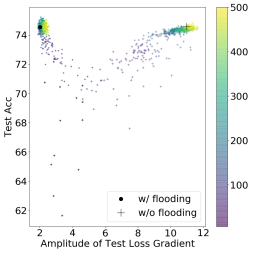
(i) C10, x:train, y:loss



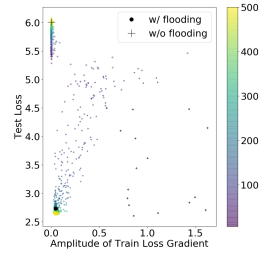
(j) C10, x:train, y:acc



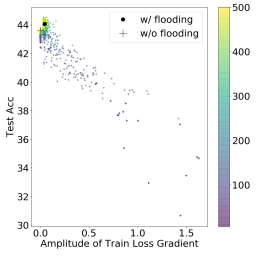
(k) C10, x:test, y:loss



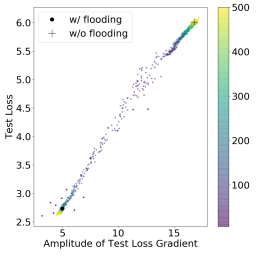
(l) C10, x:test, y:acc



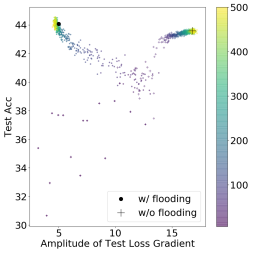
(m) C100, x:train, y:loss



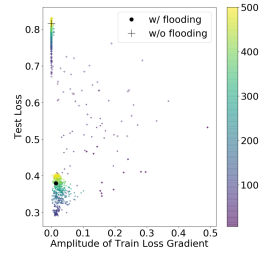
(n) C100, x:train, y:acc



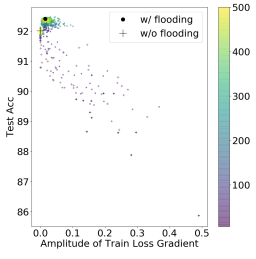
(o) C100, x:test, y:loss



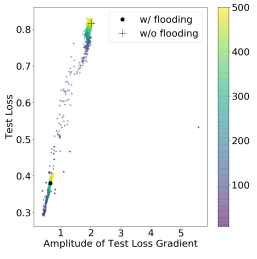
(p) C100, x:test, y:acc



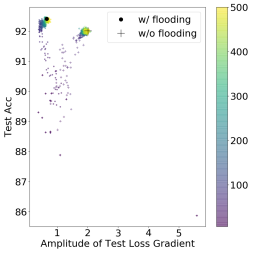
(q) SVHN, x:train, y:loss



(r) SVHN, x:train, y:acc



(s) SVHN, x:test, y:loss



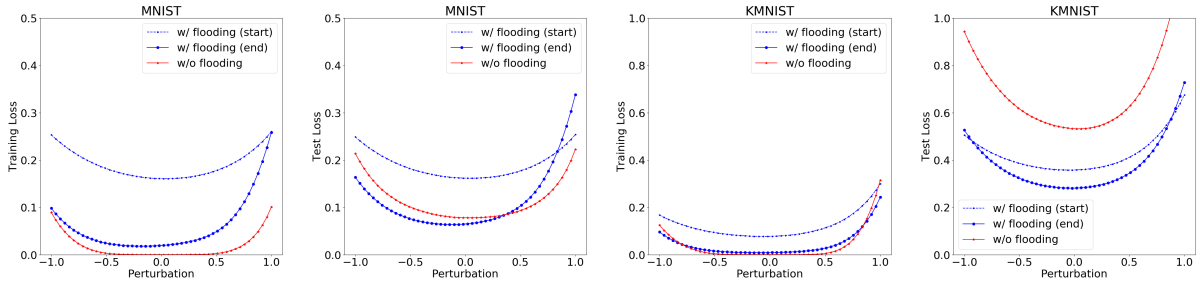
(t) SVHN, x:test, y:acc

Figure 5: Relationship between test performance (loss or accuracy) and amplitude of gradient (with training or test loss). Each point ("○" or "+") in the figures corresponds to a single model at a certain epoch. We remove the first 5 epochs and plot the rest. "○" is used for the method with flooding and "+" is used for the method without flooding. The large black "○" and "+" show the epochs with early stopping. The color becomes lighter (purple → yellow) as the training proceeds. K-M, C10, and C100 stand for Kuzushiji-MNIST, CIFAR-10, and CIFAR-100.

F Flatness

Settings We follow Li et al. [2018] and give a one-dimensional visualization of flatness for each dataset in Figure 6. We exclude Fashion-MNIST because the optimal flooding level was zero. We used the results with training/validation split ratio of 0.8. We compare the flatness of the model right after the empirical risk with respect to a mini-batch becomes smaller than the flooding level, $\widehat{R}_m(\boldsymbol{g}) < b$, for the first time (dotted blue line) and the model after training (solid blue line). We also compare them with the model trained by the baseline method without flooding, and training is finished (solid red line).

Results According to Figure 6, the test loss becomes lower and more flat during the training with flooding. Note that the training loss, on the other hand, continues to float around the flooding level until the end of training after it enters the flooding zone. We expect that the model makes a random walk and escapes regions with sharp loss landscapes during the period. This may be a possible reason for better generalization results with our proposed method.

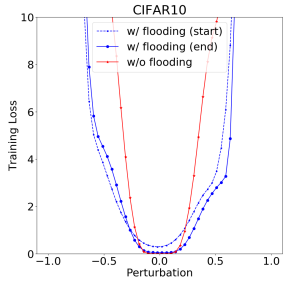


(a) MNIST (train)

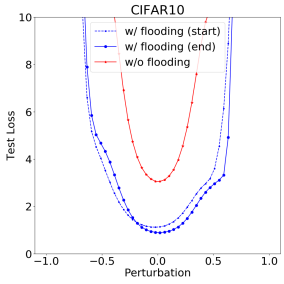
(b) MNIST (test)

(c) Kuzushiji-MNIST (train)

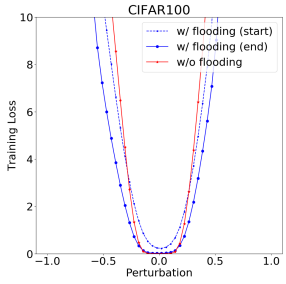
(d) Kuzushiji-MNIST (test)



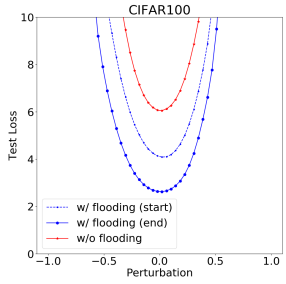
(e) CIFAR-10 (train)



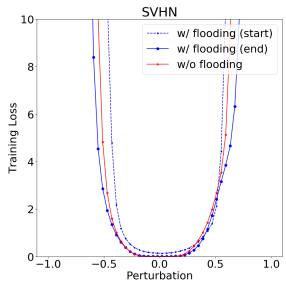
(f) CIFAR-10 (test)



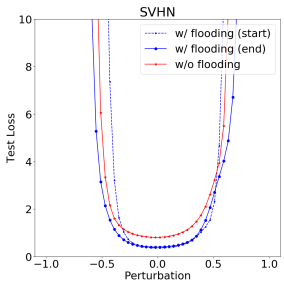
(g) CIFAR-100 (train)



(h) CIFAR-100 (test)



(i) SVHN (train)



(j) SVHN (test)

Figure 6: One-dimensional visualization of flatness. We visualize the training/test loss with respect to perturbation. We depict the results for 3 models: the model when the empirical risk with respect to training data is below the flooding level for the first time during training (dotted blue), the model at the end of training with flooding (solid blue), and the model at the end of training without flooding (solid red).