AdaFlood: Adaptive Flood Regularization

Wonho Bae *whbae@cs.ubc.ca*
University of British Columbia

Yi Ren *renyi.joshua@gmail.com*
University of British Columbia

Mohamed Osama Ahmed *mohamed.o.ahmed@borealisai.com*
Borealis AI

Frederick Tung *frederick.tung@borealisai.com*
Borealis AI

Danica J. Sutherland *dsuth@cs.ubc.ca*
University of British Columbia & Amii

Gabriel L. Oliveira *gabriel.oliveira@borealisai.com*
Borealis AI

Reviewed on OpenReview: *https://openreview.net/forum?id=2s5YU6CSEz*

Abstract

Although neural networks are conventionally optimized towards zero training loss, it has been recently learned that targeting a non-zero training loss threshold, referred to as a flood level, often enables better test time generalization. Current approaches, however, apply the same constant flood level to all training samples, which inherently assumes all the samples have the same difficulty. We present AdaFlood, a novel flood regularization method that adapts the flood level of each training sample according to the difficulty of the sample. Intuitively, since training samples are not equal in difficulty, the target training loss should be conditioned on the instance. Experiments on datasets covering four diverse input modalities—text, images, asynchronous event sequences, and tabular—demonstrate the versatility of AdaFlood across data domains and noise levels.

1 Introduction

Preventing overfitting is an important problem of great practical interest in training deep neural networks, which often have the capacity to memorize entire training sets, even ones with incorrect labels (Neyshabur et al., 2015; Zhang et al., 2021). Common strategies to reduce overfitting and improve generalization performance include weight regularization (Krogh & Hertz, 1991; Tibshirani, 1996; Liu & Ye, 2010), dropout (Wager et al., 2013; Srivastava et al., 2014; Liang et al., 2021), label smoothing (Yuan et al., 2020), and data augmentation (Balestriero et al., 2022).

Although neural networks are conventionally optimized towards zero training loss, it has recently been shown that targeting a non-zero training loss threshold, referred to as a flood level, provides a surprisingly simple yet effective strategy to reduce overfitting (Ishida et al., 2020; Xie et al., 2022). The original Flood regularizer (Ishida et al., 2020) drives the *mean* training loss towards a constant, non-zero flood level, while the state-of-the-art iFlood regularizer (Xie et al., 2022) applies a constant, non-zero flood level to *each* training instance.

Training samples are, however, not uniformly difficult: some instances have more irreducible uncertainty than others (*i.e.* heteroskedastic noise), while some instances are simply easier to fit than others. It may not be beneficial to aggressively drive down the training loss for training samples that are outliers, noisy, or mislabeled. We explore this difference in the difficulty of training samples further in Section 3.1. To address this issue, we present Adaptive Flooding (AdaFlood), a novel flood regularizer that adapts the flood level of each training sample according to the difficulty of the sample (Section 3.2). We present theoretical support for AdaFlood in Section 3.4.

Like previous flood regularizers, AdaFlood is simple to implement and compatible with any optimizer. AdaFlood determines the appropriate flood level for each sample using an auxiliary network that is trained on a subset of the training dataset. Adaptive flood levels need to be computed for each instance only once, in a pre-processing step prior to training the main network. The results of this pre-processing step are not specific to the main network, and so can be shared across multiple hyper-parameter tuning runs. Furthermore, we propose a significantly more efficient way to train an auxiliary model based on fine-tuning, which saves substantially in memory and computation, especially for overparameterized neural networks (Sections 3.3 and 4.6).

Our experiments (Section 4) demonstrate that AdaFlood generally outperforms previous flood methods on a variety of tasks, including image and text classification, probability density estimation for asynchronous event sequences, and regression for tabular datasets. Models trained with AdaFlood are also more robust to noise (Section 4.3) and better-calibrated (Section 4.4) than those trained with other flood regularizers.

2 Related Work

Regularization techniques have been broadly explored in the machine learning community to improve the generalization ability of neural networks. Regularizers augment or modify the training objective and are typically compatible with different model architectures, base loss functions, and optimizers. They can be used to achieve diverse purposes including reducing overfitting (Hanson & Pratt, 1988; Ioffe & Szegedy, 2015; Krogh & Hertz, 1991; Liang et al., 2021; Lim et al., 2022; Srivastava et al., 2014; Szegedy et al., 2016; Verma et al., 2019; Yuan et al., 2020; Zhang et al., 2018), addressing data imbalance (Cao et al., 2019; Gong et al., 2022), and compressing models (Ding et al., 2019; Li et al., 2020; Zhuang et al., 2020).

AdaFlood is a regularization technique for reducing overfitting. Commonly adopted techniques for reducing overfitting include weight decay (Hanson & Pratt, 1988; Krogh & Hertz, 1991), dropout (Liang et al., 2021; Srivastava et al., 2014), batch normalization (Ioffe & Szegedy, 2015), label smoothing (Szegedy et al., 2016; Yuan et al., 2020), and data augmentation (Lim et al., 2022; Verma et al., 2019; Zhang et al., 2018). Inspired by work on double descent (Belkin et al., 2019; Nakkiran et al., 2021), Ishida et al. (2020); Xie et al. (2022) proposed Flood and iFlood, respectively, to prevent the training loss from reaching zero by maintaining a small constant value. In contrast to the original flood regularizer, which encourages the *overall* training loss towards a constant target, iFlood drives *each* training sample's loss towards some constant b.

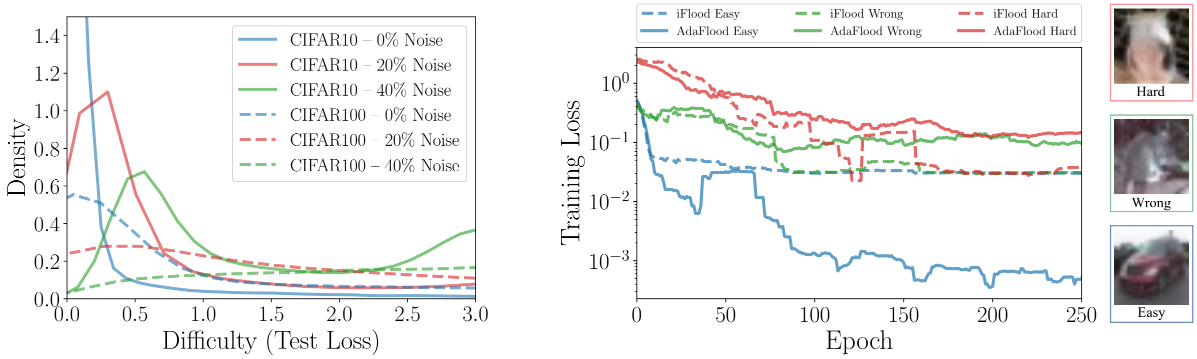
AdaFlood instead uses an auxiliary model trained on a heldout dataset to assign an adaptive flood level to each training sample. Using a heldout dataset to condition the training of the primary model is an effective strategy in machine learning, and is regularly seen in meta-learning (Bertinetto et al., 2019; Franceschi et al., 2018), batch or data selection (Fan et al., 2018; Mindermann et al., 2022), and neural architecture search (Liu et al., 2019; Wang et al., 2021), among other areas.

3 Adaptive Flooding

Adaptive Flooding (AdaFlood) is a general regularization method for training neural networks; it can accommodate any typical loss function and optimizer.

3.1 Problem Statement

Background Given a labeled training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathcal{X}$ are data samples and $y_i \in \mathcal{Y}$ are labels, we train a neural network $f : \mathcal{X} \to \widehat{\mathcal{Y}}$ by minimizing a training loss $\ell : \mathcal{Y} \times \widehat{\mathcal{Y}} \to \mathbb{R}$. In supervised learning we usually have $\ell \geq 0$, but in settings such as density estimation it may be negative.



(a) Dispersion of difficulty

(b) Training dynamics by difficulty

Figure 1: (a) Illustration of how difficulties of examples are dispersed with and without label noise (where the relevant portion of examples have their label switched to a random other label). (b) Comparison of training dynamics on some examples between iFlood and AdaFlood. The "Hard" example is labeled *horse*, but models usually predict *cow*; the "Wrong" example is incorrectly labeled in the dataset as *cat* (there is no *rat* class).

While conventional training procedures attempt to minimize the average training loss, this can lead to overfitting on training samples.

The original flood regularizer (Ishida et al., 2020) defines a global flood level for the average training loss, attempting to reduce the "incentive" to overfit. Denote the average training loss by $\mathcal{L}(f, \mathcal{B}) = \frac{1}{B}\sum_{i=1}^{B} \ell(y_i, f(x_i))$, where $f(x_i)$ denotes the model prediction and $\mathcal{B} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{B}$ is a mini-batch with size of B. Instead of minimizing $\mathcal{L}$, Flood (Ishida et al., 2020) regularizes the training by minimizing

$$\mathcal{L}_{\text{Flood}}(f, \mathcal{B}, b) = |\mathcal{L}(f, \mathcal{B}) - b| + b\,, \tag{1}$$

where the hyperparameter b is a fixed flood level. Individual Flood (iFlood) instead assigns a "local" flood level, trying to avoid instability observed with Flood (Xie et al., 2022):

$$\mathcal{L}_{\text{iFlood}}(f, \mathcal{B}, b) = \frac{1}{B}\sum_{i=1}^{B}\big(|\ell(y_i, f(\boldsymbol{x}_i)) - b| + b\big). \tag{2}$$

Motivation Training samples are, however, not uniformly difficult: some are inherently easier to fit than others. Figure 1a shows the dispersion of difficulty on CIFAR10 and 100 with various levels of added label noise, as measured by the heldout cross-entropy loss from cross-validated models. Although difficulties on CIFAR10 without noise are concentrated around difficulty ≤ 0.5, as the noise increases, they vastly spread out. CIFAR100 has a wide spread in difficulty, even without noise. A constant flood level as used in iFlood may be reasonable for un-noised CIFAR10, but it seems less appropriate for CIFAR100 or noisy-label cases.

Moreover, it may not be beneficial to aggressively drive the training loss for training samples that are outliers, noisy, or mislabeled. In Figure 1b, we show training dynamics on an *easy*, *wrong*, and a *hard* example from the training set of CIFAR10. With iFlood, each example's loss converges to the pre-determined flood level (0.03); with AdaFlood, the *easy* example converges towards zero loss, while the *wrong* and *hard* examples maintain higher loss.

3.2 Proposed Method: AdaFlood

Many advances in efficient neural network training and inference, such as batch or data selection (Coleman et al., 2020; Fan et al., 2018; Mindermann et al., 2022) and dynamic neural networks (Li et al., 2021; Verelst & Tuytelaars, 2020), stem from efforts to address the differences in per-sample difficulty. AdaFlood connects this observation to flooding. Intuitively, easy training samples (e.g. a correctly-labeled image of a *cat* in

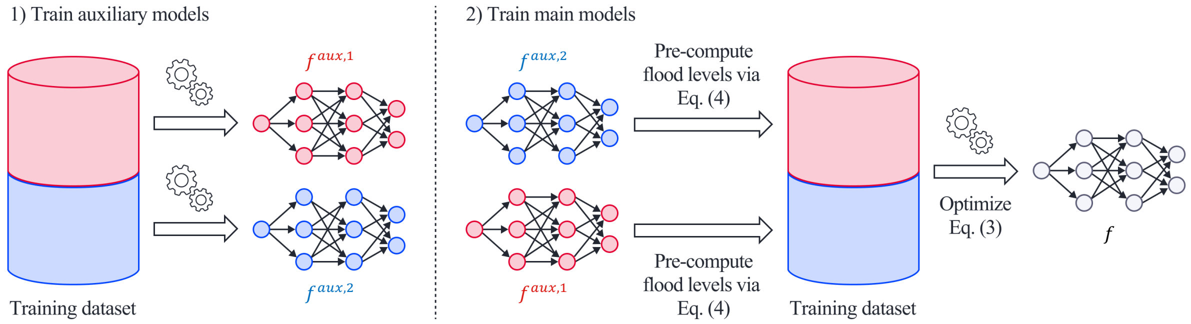


Figure 2: AdaFlood for settings where training data is limited and acquiring additional data is impractical. In the first stage, we partition the training set into two halves and train two auxiliary networks $f^{\text{aux},1}$ and $f^{\text{aux},2}$: one on each half. In the second stage, we use each auxiliary network to set the adaptive flood level of training samples from the half it has not seen, via equation 4. The main network f is then trained on the entire training set, minimizing the AdaFlood-regularized loss, equation 3. Note that the flood levels are fixed over the course of training f and need to be pre-computed once only.

a typical pose) can be driven more aggressively to zero training loss without overfitting the model, while doing so for noisy, outlier, or incorrectly-labeled training samples may cause overfitting. These types of data points behave differently during training (Ren et al., 2022), and so should probably not be treated the same. AdaFlood differentiates training samples by setting a sample-specific flood level $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^B$ in its objective:

$$\mathcal{L}_{\text{AdaFlood}}(f, \mathcal{B}, \boldsymbol{\theta}) = \frac{1}{B} \sum_{i=1}^{B} \left(|\ell(y_i, f(\boldsymbol{x}_i)) - \theta_i| + \theta_i \right). \tag{3}$$

Here the sample-specific parameters θ_i should be set according to the individual sample's difficulty. AdaFlood estimates this quantity according to

$$\theta_i = \ell(y_i, \phi_\gamma(f^{\text{aux},i}(\boldsymbol{x}_i), y_i)), \tag{4}$$

where $f^{\text{aux},i}$ is an auxiliary model trained with cross-validation such that $\boldsymbol{x}_i$ is in its heldout set, and $\phi_\gamma(\cdot)$ is a "correction" function explained in a moment. Figure 2 illustrates the training process using equation 3, Section 3.5 gives further motivation, and Section 3.4 provides further theoretical support.

The flood targets θ_i are fixed over the course of training the main network f, and can be pre-computed for each training sample prior to the first epoch of training f. We typically use five-fold cross-validation as a reasonable trade-off between computational expense and good-enough models to estimate θ_i, but see further discussion in Section 3.3. The cost of this pre-processing step can be further amortized over many training runs of the main network f since different variations and configurations of f can reuse the adaptive flood levels.

Correction function. Unfortunately, the predictions from auxiliary models are not always correct even when trained on most of the training set—if they were, our model would be perfect already. In particular, the adaptive flood levels θ_i can be arbitrarily large for any difficult examples where the auxiliary model is incorrect; this could lead to strange behavior when we encourage the primary model f to be very incorrect. We thus "correct" the predictions with the correction function ϕ_γ, which mixes between the dataset's label and the heldout model's signal.

For **regression tasks**, the predictions $f(\boldsymbol{x}_i) \in \mathbb{R}$ should simply be close to the labels $y_i \in \mathbb{R}$. The correction function linearly interpolates the predictions and labels as,

$$\phi_\gamma(f^{\text{aux}}(\boldsymbol{x}_i), y_i) = (1 - \gamma) f^{\text{aux}}(\boldsymbol{x}_i) + \gamma y_i. \tag{5}$$

Algorithm 1 Training of Auxiliary Network(s) and AdaFlood

```
Train a single auxiliary network $f^{\text{aux}}$ on the entire training set $\mathcal{D}$        ▷ Fine-tuning method only
for $\mathcal{D}^{\text{aux},i}$ in $\{\mathcal{D}^{\text{aux},i}\}_{i=1}^{n}$ do
    Train $f^{\text{aux},i}$, either from scratch or by fine-tuning $f^{\text{aux}}$, on $\mathcal{D} \setminus \mathcal{D}^{\text{aux},i}$
    Save the adaptive flood level $\theta_i$ for each $\boldsymbol{x}_i \in \mathcal{D}^{\text{aux},i}$ using $f^{\text{aux},i}$ on $\boldsymbol{x} \in \mathcal{D}^{\text{aux},i}$
end for
Train the main model $f$ using Equation (3) and adaptive flood levels $\theta$ computed above
```

Here $\gamma = 0$ fully trusts the auxiliary models (no "correction"), while $\gamma = 1$ disables flooding.

For K-way **classification tasks**, $f(\boldsymbol{x}_i) \in \mathbb{R}^K$ is a vector of output probabilities (following a softmax layer) and the label is $y_i \in [0,1]^K$, usually considered as a one-hot vector. Cross-entropy loss is then computed as: $\ell(y_i, f(\boldsymbol{x}_i)) = -\sum_{k=1}^{K} y_{i,k} \log f(\boldsymbol{x}_i)_k$. Similar to the regression tasks, we define the correction function $\phi_\gamma(f^{\text{aux}}(x_i), y_i)$ for classification tasks as a linear interpolation between the predictions and labels as:

$$\phi_\gamma(f^{\text{aux},i}(\boldsymbol{x}_i), y_i) = (1-\gamma) f^{\text{aux},i}(\boldsymbol{x}_i) + \gamma y_i \,. \tag{6}$$

Again, for $\gamma = 0$ there is no "correction," and for $\gamma = 1$ flooding is disabled, as $\theta_i = -\sum_{k=1}^{K} y_{i,k} \log y_{i,k}$ is the lower bound of the cross-entropy loss.

The hyperparameter $\gamma \in [0,1]$ is perhaps simpler to interpret and search for than directly identifying a flood level as in Flood or iFlood; in those cases, the level is unbounded (in $[0,\infty)$ for supervised tasks and all of $\mathbb{R}$ for density estimation) and the choice is quite sensitive to the particular task.

3.3 Efficiently Training Auxiliary Networks

Although the losses from auxiliary networks can often be good measures for the difficulties of samples, this is only true when the number of folds n is reasonably large; otherwise the training set of size about $\frac{n-1}{n}|\mathcal{D}|$ may be too much smaller than $\mathcal{D}$ for the model to have comparable performance. The computational cost scales roughly linearly with n, however, since we must train n auxiliary networks: if we do this in parallel it requires n times the computational resources, or if we do it sequentially it takes n times as long as training a single model.

To alleviate the computational overhead for training auxiliary networks, we sometimes instead approximate the process by fine-tuning a single auxiliary network. More specifically, we first train a single base model f^{aux} on the entire training set $\mathcal{D}$. We then train each of the n auxiliary models by randomly re-initializing the last few layers, then re-training with the relevant fold held out. The overall process is illustrated in Algorithm 1 and $n = 2$ case is described in Figure 3.

Although this means that $\boldsymbol{x}_i$ does slightly influence the final prediction $f^{\text{aux},i}(\boldsymbol{x}_i)$ ("training on the test set"), it is worth remembering that we use θ_i only as a parameter in our model, not to evaluate its performance: $\boldsymbol{x}_i$ is in fact a training data point for the overall model f being trained. This procedure is justified by recent understanding in the field that in typical settings, a single data point only loosely influence the early layers of a network. In highly over-parameterized settings (the "kernel regime") where neural tangent kernel theory is a good approximation to the training of f^{aux} (Jacot et al., 2018), re-initializing the last layer would completely remove the effect of $\boldsymbol{x}_i$ on the model. Even in more realistic settings, although the mechanism is not yet fully understood, last layer re-training seems to do an excellent job at retaining "core" features and removing "spurious" ones that are more specific to individual data points (Kirichenko et al., 2023; LaBonte et al., 2023).

For smaller models with fewer than a million parameters, we use 2- or 5-fold cross-validation, since training multiple auxiliary models is not much of a computational burden. For larger models such as ResNet18, however, we use the fine-tuning method. This substantially reduces training time, since each fine-tuning gradient step is less expensive and the models converge much faster given strong features from lower levels than they do starting from scratch; Section 4.6 gives a comparison.

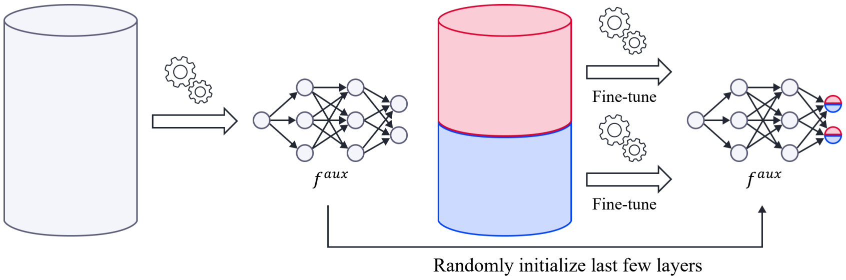


Figure 3: Efficient fine-tuning method for training a auxiliary network when held-out split is $n = 2$. First, a single model f^{aux} is trained on the entire training set $\mathcal{D}$. Then, the last few layers of each of the n auxiliary models are randomly re-initialized and re-trained with the relevant fold held out.

To validate the quality of the flood levels from the fine-tuned auxiliary network, we compare them to the flood levels from $n = 50$ auxiliary models using ResNet18 (He et al., 2016) on CIFAR10 (Krizhevsky et al., 2009); with $n = 50$, each model is being trained on 98% of the full dataset, and thus should be a good approximation to the best that this kind of method can achieve. The Spearman rank correlation between the flood levels θ_i from the fine-tuned method and the full cross-validation is 0.63, a healthy indication that this method provides substantial signal for the "correct" θ_i. Our experimental results also reinforce that this procedure chooses a reasonable set of parameters.

3.4 Theoretical Intuition

For a deeper understanding of AdaFlood's advantages, we now examine a somewhat stylized supervised learning setting: an overparameterized regime where the θ_i are nonetheless optimal.

Proposition 1. *Let $\mathcal{F}$ be a set of candidate models, and suppose there exists an optimal model $f_{opt} \in \arg\min_{f\in\mathcal{F}} \mathbb{E}_{\boldsymbol{x},y}\ell(y, f(\boldsymbol{x}))$, where ℓ is a nonnegative loss function. Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, let f_{emp} denote a minimizer of the empirical loss $\mathcal{L}(f, \mathcal{D}) = \frac{1}{N}\sum_{i=1}^N \ell(y_i, f(\boldsymbol{x}_i))$; suppose that, as in an overparameterized setting, $\mathcal{L}(f_{emp}, \mathcal{D}) = 0$. Also, let f_{ada} be a minimizer of the AdaFlood loss equation 3 using "perfect" flood levels $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^N$ where $\theta_i = \ell(y_i, f_{opt}(\boldsymbol{x}_i))$. Then we have that*

$$\mathcal{L}(f_{emp}, \mathcal{D}) = 0 \leq \mathcal{L}(f_{opt}, \mathcal{D}) = \mathcal{L}(f_{ada}, \mathcal{D}). \tag{7}$$

Furthermore, we have that

$$\mathcal{L}_{AdaFlood}(f_{emp}, \mathcal{D}, \boldsymbol{\theta}) = 2\mathcal{L}(f_{opt}, \mathcal{D}) \geq \mathcal{L}(f_{opt}, \mathcal{D}) = \mathcal{L}_{AdaFlood}(f_{opt}, \mathcal{D}, \boldsymbol{\theta}) = \mathcal{L}_{AdaFlood}(f_{ada}, \mathcal{D}, \boldsymbol{\theta}). \tag{8}$$

Proof. We know that $\mathcal{L}(f_{\text{opt}}, \mathcal{D})$ will be approximately the Bayes risk, the irreducible distributional error achieved by f_{opt}; this holds for instance by the law of large numbers, since f_{opt} is independent of the random sample $\mathcal{D}$. Thus, if the Bayes risk is nonzero and the θ_i are optimal, we can see that empirical risk minimization of overparametrized models will find f_{emp}, and disallow f_{opt}; minimizing $\mathcal{L}_{\text{AdaFlood}}$, on the other hand, will allow the solution f_{opt} and disallow the empirical risk minimizer f_{emp}.

With this choice of θ_i, we have that

$$\mathcal{L}_{\text{AdaFlood}}(f, \mathcal{D}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\Big(|\ell(y_i, f(\boldsymbol{x}_i)) - \ell(y_i, f_{\text{opt}}(\boldsymbol{x}_i))| + \ell(y_i, f_{\text{opt}}(\boldsymbol{x}_i))\Big).$$

Since $|\cdot|$ is nonnegative, we have $\mathcal{L}_{\text{AdaFlood}}(f, \mathcal{D}, \boldsymbol{\theta}) \geq \mathcal{L}(f_{\text{opt}}, \mathcal{D})$ for any f, and $\mathcal{L}_{\text{AdaFlood}}(f_{\text{opt}}, \mathcal{D}, \boldsymbol{\theta}) = \mathcal{L}(f_{\text{opt}}, \mathcal{D})$; this establishes that f_{opt} minimizes $\mathcal{L}_{\text{AdaFlood}}$, and that any minimizer f_{ada} must achieve $\ell(y_i, f_{\text{ada}}(\boldsymbol{x}_i)) = \theta_i$ for each i, so $\mathcal{L}(f_{\text{ada}}, \mathcal{D}) = \mathcal{L}(f_{\text{opt}}, \mathcal{D})$. Using that $\ell(y_i, f_{\text{emp}}(\boldsymbol{x}_i)) = 0$ for each i, as is necessary for $\ell \geq 0$ when $\mathcal{L}(f_{\text{emp}}, \mathcal{D}) = 0$, shows $\mathcal{L}_{\text{AdaFlood}}(f_{\text{emp}}, \mathcal{D}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^N 2\theta_i = 2\mathcal{L}(f_{\text{opt}}, \mathcal{D})$. □

In settings where θ_i is not perfect (and we would not expect the auxiliary models to obtain *perfect* estimates of the loss) the comparison will still approximately hold. If θ_i consistently overestimates the f_{opt} loss, f_{opt} will

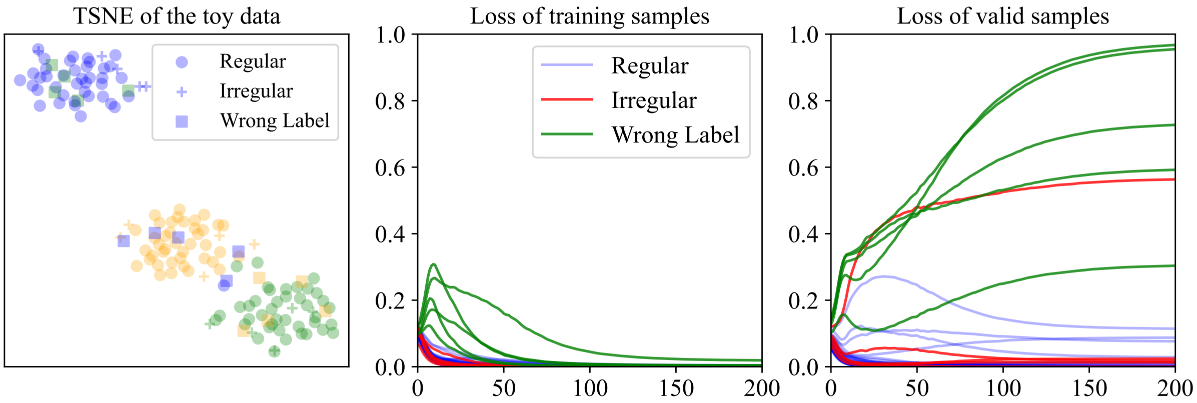


Figure 4: Left: the t-SNE (Van der Maaten & Hinton, 2008) of toy Gaussian example; middle: loss of different samples in the training set; right: loss of different samples in the validation set.

still be preferred to f_{emp}: for instance, if $\theta_i = 2\ell(y_i, f_{\text{opt}}(\boldsymbol{x}_i))$, then $\mathcal{L}_{\text{AdaFlood}}(f_{\text{emp}}, \mathcal{D}, \boldsymbol{\theta}) = 4\mathcal{L}(f_{\text{opt}}, \mathcal{D}) \geq 3\mathcal{L}(f_{\text{opt}}, \mathcal{D}) = \mathcal{L}_{\text{AdaFlood}}(f_{\text{opt}}, \mathcal{D}, \boldsymbol{\theta})$. On the other hand, if $\theta_i = \frac{1}{2}\ell(y_i, f_{\text{opt}}(\boldsymbol{x}_i))$—a not-unreasonable situation when using a correction function—then $\mathcal{L}_{\text{AdaFlood}}(f_{\text{emp}}, \mathcal{D}, \boldsymbol{\theta}) = \mathcal{L}(f_{\text{opt}}, \mathcal{D}) = \mathcal{L}_{\text{AdaFlood}}(f_{\text{opt}}, \mathcal{D}, \boldsymbol{\theta})$. When θ_i is random, the situation is more complex, but we can expect that noisy θ_i which somewhat overestimate the loss of f_{opt} will still prefer f_{opt} to f_{emp}.

3.5 Discussion: Why We Calculate θ Using Held-out Data

In Section 3.2, we estimate θ_i for each training sample using the output of an auxiliary network $f^{\text{aux}}(x_i)$ that is trained on a held-out dataset. In fact, this adaptive flood level θ_i can be considered as the sample difficulty when training the main network. Hence, it is reasonable to consider existing difficulty measurements based on learning dynamics, like C-score (Jiang et al., 2021) or forgetting score (Maini et al., 2022). However, we find these methods are not robust when wrong labels exist in the training data, because the network will learn to remember the wrong label of $\boldsymbol{x}_i$, and hence provide a low θ_i for the wrong sample, which is harmful to our method. That is why we propose to split the whole training set into n parts and train $f^{\text{aux}}(\boldsymbol{x}_i)$ for n times (each with different $n-1$ parts).

Dataset and implementation To verify this, we conduct experiments on a toy Gaussian dataset, as illustrated in the first panel in Figure 4. Assume we have N samples, each sample in 2-tuple (x, y). To draw a sample, we first select the label $y = k$ following a uniform distribution over all K classes. After that, we sample the input signal $x \mid (y = k) \sim \mathcal{N}(\mu_k, \sigma^2 I)$, where σ is the noise level for all the samples. μ_k is the mean vector for all the samples in class k. Each μ_k is a 10-dim vector, in which each dimension is randomly selected from $\{-\delta_\mu, 0, \delta_\mu\}$. Such a process is similar to selecting 10 different features for each class. We consider 3 types of samples for each class: regular samples, the typical or easy samples in our training set, have a small σ; irregular samples have a larger σ; mislabeled samples have a small σ, but with a flipped label. We generate two datasets following this same procedure (call them datasets A and B). Then, we randomly initialize a 2-layer MLP with ReLU layers and train it on dataset A. At the end of every epoch, we record the loss of each sample in dataset A.

Result The learning paths are illustrated in the second panel in Figure 4. The model is clearly able remember all the wrong labels, as all the curves converge to a small value. If we calculate θ_i in this way, all θ_i would have similar values. However, if we instead train the model using dataset B, which comes from the same distribution but is different from dataset A, the learning curves of samples in dataset A will behave like the last panel in Figure 4. The mislabeled and some irregular samples can be clearly identified from the figure. Calculating θ_i in this way gives different samples more distinct flood values, which makes our method more robust to sample noise, as our experiments on various scenarios show.

NTPP	Method	Uber		Reddit			Stack Overflow		
		RMSE	NLL	RMSE	NLL	ACC	RMSE	NLL	ACC
Intensity-free	Unreg.	75.83	3.86	0.25	1.28	55.26	6.69	3.66	45.52
		(6.12)	(0.05)	(0.01)	(0.07)	(0.57)	(0.98)	(0.12)	(0.07)
	Flood	64.34	4.01	0.25	1.17	57.46	4.12	3.46	**45.76**
		(3.85)	(0.02)	(0.01)	(0.06)	(0.84)	(0.23)	(0.03)	**(0.03)**
	iFlood	67.07	3.97	**0.23**	1.11	56.59	4.12	3.46	**45.76**
		(3.12)	(0.06)	**(0.01)**	(0.12)	(0.92)	(0.23)	(0.03)	**(0.03)**
	AdaFlood	**59.69**	**3.75**	0.26	**1.09**	**59.02**	**3.26**	**3.45**	45.67
		(1.49)	**(0.01)**	(0.02)	**(0.13)**	**(0.91)**	**(0.25)**	**(0.04)**	(0.03)
THP$^+$	Unreg.	71.01	3.73	0.28	0.82	58.63	1.46	2.82	46.24
		(6.12)	(0.05)	(0.01)	(0.07)	(0.57)	(0.98)	(0.12)	(0.07)
	Flood	68.61	3.70	0.26	1.02	58.05	1.39	2.79	46.31
		(3.85)	(0.02)	(0.01)	(0.06)	(0.84)	(0.23)	(0.03)	(0.03)
	iFlood	68.61	3.70	**0.25**	0.92	58.93	1.46	2.82	46.24
		(4.76)	(0.17)	(0.01)	(0.23)	(1.26)	(0.06)	(0.04)	(0.08)
	AdaFlood	**54.85**	**3.55**	**0.25**	**0.80**	**61.34**	**1.38**	**2.77**	**46.41**
		(1.49)	**(0.01)**	**(0.02)**	**(0.13)**	**(0.91)**	**(0.25)**	**(0.04)**	**(0.03)**

Table 1: Comparison of flooding methods on asynchronous event sequence datasets. The numbers are the means and standard errors (in parentheses) over three runs.

4 Experiments

We demonstrate the effectiveness of AdaFlood on three tasks (probability density estimation, classification and regression) in four domains (asynchronous event sequences, image, text and tabular). We compare flooding methods on asynchronous event time in Section 4.1 and image classification tasks in Section 4.2. We also demonstrate that AdaFlood is more robust to various noisy settings in Section 4.3, and that it yields better-calibrated models in Section 4.4. Some ablation studies are provided in Sections 4.5 and 4.6.

4.1 Results on Asynchronous Event Sequences

We compare flooding methods on asynchronous event sequence datasets of which goal is to estimate the probability distribution of the next event time given the previous event times. Each event may have a class label. Asynchronous event sequences are often modeled as temporal point processes (TPPs).

Datasets We use two popular benchmark datasets, Stack Overflow (predicting the times at which users receive badges) and Reddit (predicting posting times). Following Bae et al. (2023), we also benchmark our method on a dataset with stronger periodic patterns: Uber (predicting pick-up times). We split each training dataset into train (80%) and validation (20%) sets. Details are provided in Appendix A.

Following the literature in TPPs, we use two metrics to evaluate models: *root mean squared error* (RMSE) and *negative log-likelihood* (NLL). As NLL can be misleadingly low if the probability density is mostly focused on the correct event time, RMSE is also considered a complementary metric. However, RMSE has its own limitation: if a baseline is directly trained on the ground truth event times as point estimation, the stochastic components of TPPs are ignored. Therefore, we train our TPP models on NLL and use RMSE at test time to ensure that we do not rely too heavily on RMSE scores and account for the stochastic nature of TPPs. When class labels for events are available, we also report the accuracy of class predictions.

Implementation For TPP models to predict the asynchronous event times, we employ Intensity-free models (Shchur et al., 2020) based on GRU (Chung et al., 2014), and Transformer Hawkes Processes (THP) (Zuo et al., 2020) based on Transformer (Vaswani et al., 2017). THP predicts intensities to compute log-likelihood and expected event times, but this approach can be computationally expensive due to the need to compute integrals, particularly double integrals to calculate the expected event times. To overcome this challenge

Method	SVHN		CIFAR10		CIFAR100	
	w/o L_2 reg.	w/ L_2 reg.	w/o L_2 reg.	w/ L_2 reg.	w/o L_2 reg.	w/ L_2 reg.
Unreg.	95.65 ± 0.05	96.07 ± 0.01	87.80 ± 0.31	90.35 ± 0.21	56.59 ± 0.32	61.49 ± 0.16
Flood	95.63 ± 0.02	96.13 ± 0.02	87.57 ± 0.16	90.09 ± 0.20	55.88 ± 0.18	60.96 ± 0.03
iFlood	95.63 ± 0.08	96.05 ± 0.02	87.96 ± 0.07	90.57 ± 0.12	56.32 ± 0.05	61.63 ± 0.12
KD	95.69 ± 0.02	96.08 ± 0.10	88.06 ± 0.23	90.65 ± 0.03	56.67 ± 0.15	61.29 ± 0.03
AdaFlood	**95.72 ± 0.01**	**96.16 ± 0.02**	**88.38 ± 0.18**	**90.82 ± 0.08**	**57.25 ± 0.14**	**62.31 ± 0.14**

Table 2: Comparison of flooding methods on image classification datasets with and without L_2 regularization. The numbers are the means and standard errors over three runs.

while maintaining performance, we follow Bae et al. (2023) in using a mixture of log-normal distributions, proposed in Shchur et al. (2020), for the decoder; we call this THP$^+$.

For each dataset, we conduct hyper-parameter tuning for learning rate and the weight for L_2 regularization with the unregularized baseline (we still apply early stopping and L_2 regularization by default). Once learning rate and weight decay parameters are fixed, we search for the optimal flood levels. The optimal flood levels are selected via a grid search on $\{-50, -45, -40 \ldots, 0, 5\} \cup \{-4, -3 \ldots, 3, 4\}$ for Flood and iFlood, and optimal γ on $\{0.0, 0.1 \ldots, 0.9\}$ for AdaFlood using the validation set. We use five auxiliary models.

Results In order to evaluate the effectiveness of various regularization methods, we present the results of our experiments in Table 1 (showing means and standard errors from three runs). This is the first time we know of where flooding methods have been applied in this domain; we see that all flooding methods improve the generalization performance here, sometimes substantially. Furthermore, AdaFlood often outperforms other flooding methods on various datasets, suggesting that instance-wise flooding level adaptation using auxiliary models can effectively enhance the generalization capabilities of TPP models. However, there are instances where AdaFlood's performance is comparable to or slightly worse than other methods, indicating that its effectiveness may vary depending on the specific context. Despite this variability, AdaFlood generally appears to be the best choice for training TPP models.

4.2 Results on Image Classification

Datasets We use SVHN (Netzer et al., 2011), CIFAR-10, and 100 (Krizhevsky et al., 2009) for image classification with random crop and horizontal flip as augmentation. Unlike Xie et al. (2022), we split each training dataset into train (80%) and validation (20%) sets for hyperparameter search; thus our numbers are generally somewhat worse than what they reported, as we do not directly tune on the test set.

Implementation Following Ishida et al. (2020) and similar to Xie et al. (2022), we consider training ResNet18 (He et al., 2016) with and without L_2 regularization (with a weight of 10^{-4}). All methods are trained with SGD for 300 epochs, with early stopping. We use a multi-step learning rate scheduler with an initial learning rate of 0.1 and decay coefficient of 0.2, applied at every 60 epochs. The optimal flood levels are selected based on validation performance with a grid search on $\{0.01, 0.02 \ldots, 0.1, 0.15, 0.2 \ldots, 1.0\}$ for Flood and iFlood, and $\{0.05, 0.1 \ldots, 0.95\}$ for AdaFlood. We use a single ResNet18 auxiliary network where its layer 3 and 4 are randomly initialized and fine-tuned on held-out sets with $n = 10$ splits.

Furthermore, we compare with knowledge distillation (KD) baselines following Hinton et al. (2014) for the implementation of its loss. For a mini-batch $\mathcal{B}$, the KD loss is defined as:

$$\mathcal{L}_{\mathrm{KD}}(f_{\mathrm{s}}, f_{\mathrm{t}}, \mathcal{B}, \tau, \alpha) = \alpha \mathcal{L}_{\mathrm{CE}}(f_{\mathrm{s}}, \mathcal{B}) + (1-\alpha)\mathcal{L}_{\mathrm{Distll}}(f_{\mathrm{s}}, f_{\mathrm{t}}, \mathcal{B}, \tau), \tag{9}$$

where τ denotes a temperature scale, which is an additional input to a student model f_{s} and teacher model f_{t}. Also, $\mathcal{L}_{\mathrm{CE}}$ and $\mathcal{L}_{\mathrm{Distill}}$ are defined as:

$$\mathcal{L}_{\mathrm{CE}}(f, \mathcal{B}) = \frac{1}{B}\sum_{i=1}^{B} \ell(y_i, f(x_i)),\ \mathcal{L}_{\mathrm{Distll}}(f_{\mathrm{s}}, f_{\mathrm{t}}, \mathcal{B}, \tau) = \frac{1}{\tau^2 B}\sum_{i=1}^{B} \ell(f_{\mathrm{t}}(x_i, \tau), f_{\mathrm{s}}(x_i, \tau)). \tag{10}$$

We set $\alpha = 0.5$ following one of the experiments in Hinton et al. (2014) so that all the methods have only one hyperparameter to tune. We tune the temperature scale τ with a grid search on $\{1, 2, 3, \cdots, 9, 10\}$.

Results The results are presented in Table 2. We report the means and standard errors of accuracies over three runs. We can observe that KD and flooding methods, including AdaFlood, are not significantly better than the unregularized baseline on SVHN. However, AdaFlood noticeably improves the performance over the other methods on harder datasets like CIFAR10 and CIFAR100, whereas iFlood is not obviously better than the baseline and Flood is worse than the baseline on CIFAR100. The gap between AdaFlood and KD is more noticeable on CIFAR100, particularly with L_2 regularization.

Discussion While iFlood is closely related to label smoothing, AdaFlood shares similarities with KD as both utilize auxiliary networks. However, a motivation behind two algorithms are fundamentally different. KD relies on predictions made on already-seen training examples, whereas AdaFlood leverages predictions on intentionally forgotten (or unseen) examples. Since the predictions of teacher networks in KD are based on already-seen examples, they do not serve as meaningful measures of uncertainty. In contrast, the predictions from an auxiliary network in AdaFlood can effectively measure uncertainty, and flood levels computed from these predictions can function as uncertainty regularizations. A disadvantage of AdaFlood, however, is the additional fine-tuning step required to forget already-seen examples, which is not necessary in KD.

4.3 Noisy Labels

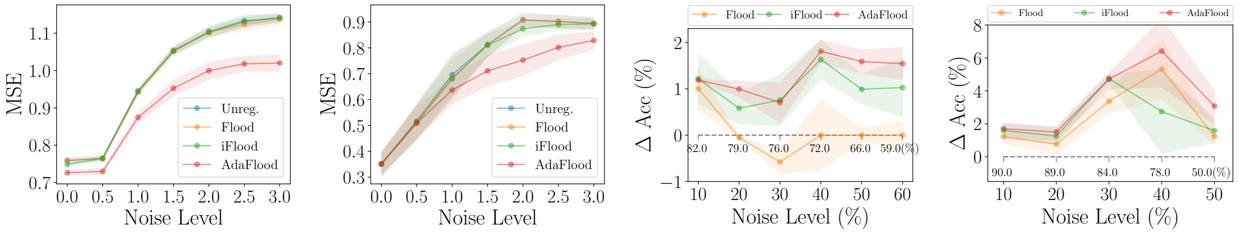
Datasets In addition to CIFAR10 for image classification, we also use the tabular datasets Brazilian Houses and Wine Quality from OpenML (Vanschoren et al., 2013), following Grinsztajn et al. (2022), for regression tasks. We further employ Stanford Sentiment Treebank (SST-2) for the text classification task, following Xie et al. (2022). Details of datasets are provided in Appendix A.

We report accuracy for classification tasks. For regression tasks, we report *mean squared error* (MSE) in the main body, as well as *mean absolute error* (MAE) and R^2 score in Figure 8 (Appendix C).

Implementation We inject noise for both image and text classification by changing the label to a uniformly randomly selected wrong class, following Xie et al. (2022). More specifically, for α% of the training data, we change the label to a uniformly random class other than the original label. For the regression tasks, we add errors sampled from a skewed normal distribution, with skewness parameter ranging from 0.0 to 3.0. Similar to the previous experiments, we tune learning rate and the weight for L_2 regularization with the unregularized baseline (with early stopping and L_2 regularization by default except for Figure 5c). Then, we tune the flood levels with the fixed learning rate and L_2 regularization.

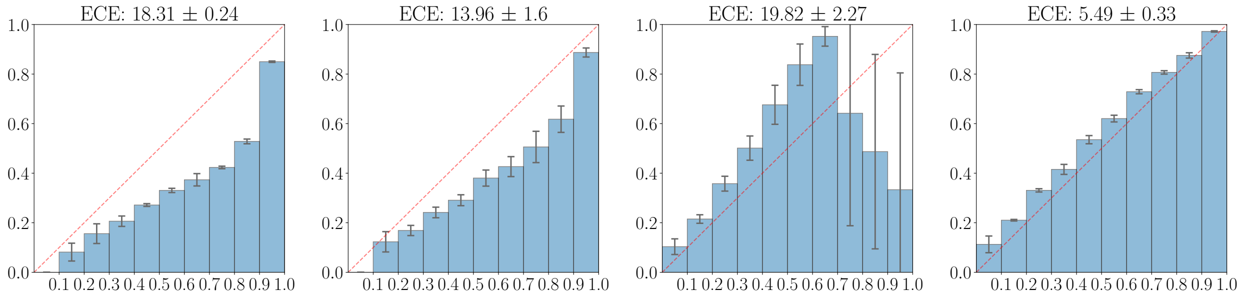
Results Figure 5 compares the flooding methods for noisy settings. We report the mean and standard error over three runs for CIFAR10, and five and seven runs for tabular datasets and SST-2, respectively. We provide ΔAcc (%) for CIFAR10 and SST-2 compared to the unregularized model: that is, we plot the accuracy of each method minus the accuracy of the unregularized method, to display the gaps between methods more clearly. The mean accuracies of the unregularized method are displayed below the zero line.

- Wine Quality, Figure 5a: AdaFlood slightly outperforms the other methods at first, but the gap significantly increases as the noise level increases.
- Brazilian Houses, Figure 5b: There is no significant difference between the methods for small noise level, *e.g.* noise ≤ 1.5, but MSE for AdaFlood becomes significantly lower as the noise level increases.
- CIFAR10, Figure 5c: iFlood and AdaFlood significantly outperform Flood and unregularized. AdaFlood also outperforms iFlood when the noise level is high (*e.g.* $\geq 50\%$).
- SST-2, Figure 5d: Flooding methods outperform the unregularized method. AdaFlood is comparable to iFlood up to the noise level of 30%, but noticeably outperforms it as the noise level increases.



(a) Wine Quality (b) Brazilian Houses (c) CIFAR10 (d) SST-2

Figure 5: Comparison of flooding methods on tabular and image datasets with noise and bias.



(a) Unregularized (b) Flood (c) iFlood (d) AdaFlood

Figure 6: Calibration results of flooding methods with 10 bins on CIFAR100. The bars and errors are the means and standard errors over three runs, respectively.

Overall, AdaFlood is more robust to noise than other flooding methods, since the model pays less attention to samples with high losses.

4.4 Calibration

Datasets and implementation Miscalibration—neural networks being over or under-confident—has been a well-known issue in deep learning. We thus evaluate the quality of calibration with different flooding methods on CIFAR100, as measured by the Expected Calibration Error (ECE) metric. (Figure 9 does the same for CIFAR10, but since model predictions are usually quite confident, this becomes difficult to measure.) We use a ResNet18 with L_2 regularization with the optimal hyperparameters for the baseline and flooding methods. The optimal hyperparameter varies by seed for each run.

Result Figure 6 provides the calibration quality in ECE metric as well as a visualization over three runs, compared to perfect calibration (dotted red lines). We can observe that AdaFlood significantly improves the calibration, both in ECE and visually. Note that iFlood significantly miscalibrates at the bins corresponding to high probability *e.g.* bin ≥ 0.7, compared to the other methods, and also has high standard errors. This behavior is expected, since iFlood encourages the model not to predict higher than a probability of $\exp(-b)$, where b denotes the flood level used in iFlood.

4.5 Ablation study: Relationship with Other Regularization

In this ablation study, we design an experiment that shows how different regularization methods interact with flooding methods. We conduct the experiment on CIFAR100 with ResNet18, gradually adding regularization methods in the order of early stopping, L_2 regularization, dropout and CutMix (Yun et al., 2019), a popular data augmentation method, as shown in Table 3. Please note that the second row with early stopping and the third row with early stopping + L_2 regularization are the same as what we report in Table 2.

Similar to the results in Table 2, Flood is comparable to or slightly worse than the unregularized baseline for the case with dropout and with both dropout and CutMix. Although iFlood is generally better than the

Regularization				Flooding			
Early Stopping	L_2	Dropout	CutMix	Unreg.	Flood	iFlood	AdaFlood
				56.39 ± 0.25	56.07 ± 0.19	56.07 ± 0.06	**56.89 ± 0.19**
✓				56.59 ± 0.32	55.88 ± 0.18	56.32 ± 0.05	**57.25 ± 0.14**
✓	✓			61.49 ± 0.16	60.96 ± 0.03	61.63 ± 0.12	**62.31 ± 0.14**
✓	✓	✓		62.04 ± 0.17	61.73 ± 0.13	62.19 ± 0.27	**63.15 ± 0.10**
✓	✓	✓	✓	67.08 ± 0.28	67.11 ± 0.26	67.09 ± 0.08	**67.50 ± 0.16**

Table 3: Comparison between flooding methods with and without various regularization methods on CIFAR100. We report the means and standard errors of three runs.

unregularized baseline as shown in the third to fifth rows, the gap is limited. Compared to Flood or iFlood, AdaFlood shows more consistently larger improvement.

4.6 Ablation study: Fine-tuning vs. Multiple Auxiliaries

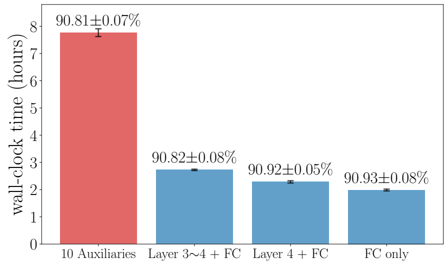


Figure 7: Comparison of aux. training

Figure 7 compares training of ten ResNet18 auxiliary networks (original proposal) to the single fine-tuned auxiliary network (efficient variant) in terms of wall-clock time for training the auxiliary network(s), and performance of the corresponding main model, on the test set of CIFAR10. For the efficient variant, we fine-tune different layers to show insensitivity to the choice of layers: $Layer3, 4 + FC$, $Layer3 + FC$, and FC, where $Layer3$ and 4 are the 3rd, 4th layers in ResNet18 and FC denotes the last fully connected layer. For example, $Layer4 + FC$ means we only fine-tune $Layer4$ and FC layers, freezing all the previous layers. Results show that training multiple auxiliary networks yields the same-quality model as fine-tuning, though training time is 3 to 4 times longer. There is also little difference in performance between different fine-tuning methods: it seems that fine-tuning only the FC layer is sufficient to forget the samples, with early-stopping regularizing well enough for similar generalization ability. We also compare AdaFlood with various architectures for auxiliary networks in Appendix F.

5 Conclusion

In this paper, we introduced the Adaptive Flooding (AdaFlood) regularizer, a novel reguralization technique that adaptively regularizes a loss for each sample based on the difficulty of the sample. Each flood level is computed only once through an auxiliary training procedure with held-out splitting, which we can make more efficient by fine-tuning the last few layers on held-out sets. Experimental results on various domains and tasks: density estimation for asynchronous event sequences, image and text classification tasks as well as regression tasks on tabular datasets, with and without noise, demonstrated that our approach is more robustly applicable to a varied range of tasks including calibration.

Limitation Although AdaFlood is a robust and effective regularizer on many different tasks, particularly in high-noise settings, an open question that we leave for future work is how to best apply AdaFlood in long-tailed learning. For long-tailed data, it is expected that samples from the rare classes will tend to have higher losses. During the training of the main model, AdaFlood will direct the model to keep the higher losses for rare classes and lower losses for common classes, which may not be desirable. One potential solution could be to adaptively adjust γ for different classes. Alternatively, imbalanced learning techniques such as resampling, reweighting, or two-stage training could be adopted.

Reproducibility For each experiment, we listed implementation details such as model, regularization, and search space for hyperparameters. We also specified datasets we used for each experiment, and how they were split and augmented, along with the description of metrics. The code is released with the final version.

References

Wonho Bae, Mohamed Osama Ahmed, Frederick Tung, and Gabriel L Oliveira. Meta temporal point processes. In *ICLR*, 2023.

Randall Balestriero, Leon Bottou, and Yann LeCun. The effects of regularization and data augmentation are class dependent. In *NeurIPS*, 2022.

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.

Luca Bertinetto, João Henriques, Philip H. S. Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *ICLR*, 2019.

Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss. In *NeurIPS*, 2019.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS*, 2014.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. In *ICLR*, 2020.

Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *CVPR*, 2019.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *KDD*, 2016.

Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. In *ICLR*, 2018.

Luca Franceschi, Paolo Frasconi, Saverio Salvo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 2018.

Yu Gong, Greg Mori, and Frederick Tung. RankSim: Ranking similarity regularization for deep imbalanced regression. In *ICML*, 2022.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS*, 2022.

Stephen Hanson and Lorien 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.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPSW*, 2014.

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

Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Do we need zero training loss after achieving zero training error? In *ICML*, 2020.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018.

Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C Mozer. Characterizing structural regularities of labeled data in overparameterized models. In *ICML*, 2021.

Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Last layer re-training is sufficient for robustness to spurious correlations. In *ICLR*, 2023.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *NeurIPS*, 1991.

Tyler LaBonte, Vidya Muthukumar, and Abhishek Kumar. Towards last-layer retraining for group robustness with fewer annotations. In *NeurIPS*, 2023.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *NeurIPS*, 2019.

Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *CVPR*, 2021.

Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *CVPR*, 2020.

Xiaobo Liang, Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. R-Drop: Regularized dropout for neural networks. In *NeurIPS*, 2021.

Soon Hoe Lim, N. Benjamin Erichson, Francisco Utrera, Winnie Xu, and Michael W. Mahoney. Noisy feature mixup. In *ICLR*, 2022.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

Jun Liu and Jieping Ye. Efficient l1/lq norm regularization. *arXiv preprint arXiv:1009.4766*, 2010.

Pratyush Maini, Saurabh Garg, Zachary Lipton, and J Zico Kolter. Characterizing datapoints via second-split forgetting. In *NeurIPS*, volume 35, pp. 30044–30057, 2022.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *ICML*, 2022.

Mohamad Amin Mohamadi, Wonho Bae, and Danica J Sutherland. A fast, well-founded approximation to the empirical neural tangent kernel. In *ICML*, 2023.

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.

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

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.

Yi Ren, Shangmin Guo, and Danica J. Sutherland. Better supervisory signals by observing learning paths. In *ICLR*, 2022.

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *ICLR*, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, pp. 1929–1958, 2014.

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

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *ECCV*, 2020.

Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, pp. 49–60, 2013.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *CVPR*, 2020.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, 2019.

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

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable NAS. In *ICLR*, 2021.

Yuexiang Xie, WANG Zhen, Yaliang Li, Ce Zhang, Jingren Zhou, and Bolin Ding. iFlood: A stable and effective regularizer. In *ICLR*, 2022.

Li Yuan, Francis E. H. Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *CVPR*, 2020.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

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

Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. In *NeurIPS*, 2020.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer Hawkes process. In *ICML*, 2020.

A Details about Datasets

Stack Overflow It contains 6,633 sequences with 480,414 events where an event is the acquisition of badges received by users. The maximum number of sequence length is 736 and the number of marks is 22. The dataset is provided by Du et al. (2016); we use the first folder, following Shchur et al. (2020) and Bae et al. (2023).

Reddit It contains 10,000 sequences with 532,026 events where an event is posting in Reddit. The maximum number of sequence length is 736 and the number of marks is 22. Marks represent sub-reddit categories.

Uber It contains 791 sequences with 701,579 events where an event is pick-up of customers. The maximum number of sequence length is 2,977 and there is no marks. It is processed and provided by Bae et al. (2023).

Brazilian Houses It contains information of 10,962 houses to rent in Brazil in 2020 with 13 features. The target is the rent price for each house in Brazilian Real. According to OpenML (Vanschoren et al., 2013) where we obtained this dataset, since the data is web-scrapped, there are some values in the dataset that can be considered outliers.

Wine Quality It contains 6,497 samples with 11 features and the quality of wine is numerically labeled as targets. This dataset is also obtained from OpenML (Vanschoren et al., 2013).

SST-2 The Stanford Sentiment Treebank (SST-2) is a dataset containing fully annotated parse trees, enabling a comprehensive exploration of how sentiment influences language composition. Comprising 11,855 individual sentences extracted from film reviews, this dataset underwent parsing using the Stanford parser, resulting in a collection of 215,154 distinct phrases.

B Additional Results on Image Classification

Datasets We use ImageNet100 (Tian et al., 2020) for image classification with random crop, horizontal flip, and color jitter as augmentation. We also add 30% of label noise as done in Section 4.3.

Implementation We train ResNet34 (He et al., 2016) on the dataset with L_2 regularization (with a weight of 0.0001). All methods are trained for 200 epochs with early stopping using SGD. We use a multi-step learning rate scheduler with an initial learning rate of 0.1 and decay coefficient of 0.5, applied at every 25 epochs. The optimal flood levels are selected based on validation performance with a grid search on $\{0.01, 0.02..., 0.1, 0.15, 0.2..., 0.3\}$ for Flood and iFlood, and $\{0.05, 0.1..., 0.95\}$ for AdaFlood. We use a single ResNet34 auxiliary network where its last FC layer is randomly initialized and fine-tuned on held-out sets with $n = 10$ splits.

Results Table 4 (Left) compares flooding methods on ImageNet100 dataset with and without 30% of label noise. We report test accuracies along with expected calibration error (ECE) on the right. Although Flood and iFlood do not improve the performance over the unregularized model, AdaFlood improves the performance by about 0.80% over the unregularized baseline. Given the size of the dataset, the gap is not marginal. This gap is even larger than that we observed in SVHN and CIFAR datasets Table 2. We conjecture it is because ImageNet contains more noisy samples. It is well-known that there are many ImageNet images containing multiple objects although the label says there is only one object (Beyer et al., 2020).

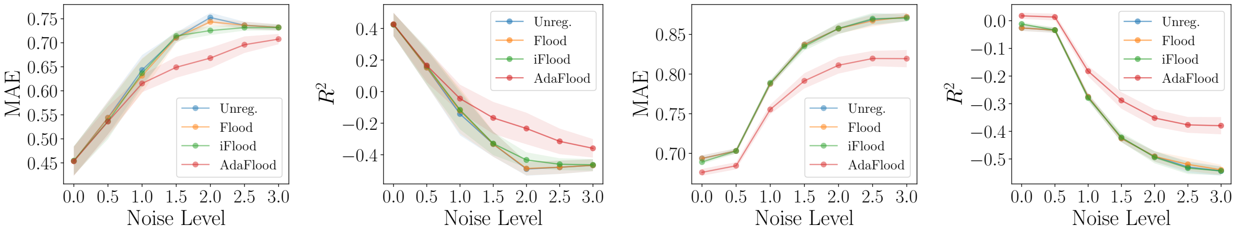
C Additional Results on Tabular Regression

Datasets We use NYC Taxi Tip dataset from OpenML (Vanschoren et al., 2013), one of the largest tabular dataset used in Grinsztajn et al. (2022), for regression tasks. NYC Taxi Tip dataset contains $581,835$ rows and 9 features. As the name of the dataset implies the target variable is “tip amount". To increase the importance of other features, the creator of the dataset deliberately ignores “fare amount" or “trip distance".

Method	Mislabled Sample Rate	
	0%	30%
Unreg.	81.00 / 6.64	68.12 (17.23)
Flood	81.18 (6.44)	68.19 (17.71)
iFlood	81.04 (6.86)	68.24 (20.67)
AdaFlood	**81.79**(**4.81**)	**69.22**(**17.45**)

Method	Noise Level		
	0.0	1.5	3.0
Unreg.	0.2373(0.3335)	0.3707 (-0.0409)	0.3910 (-0.0978)
Flood	0.2373(0.3335)	0.3707 (-0.0409)	0.3904 (-0.0980)
iFlood	0.2370(0.3374)	0.3652 (-0.0255)	0.3902 (-0.0986)
AdaFlood	**0.2369**(**0.3348**)	**0.3520**(**0.0250**)	**0.3465**(**0.0119**)

Table 4: Comparison of flooding methods on ImageNet100 (Left) and NYC Taxi Tip (Right) datasets with and without label noise. The numbers on the right on ImageNet100 represents ECE metric whereas on NYC Taxi Tip, they are R^2 scores.



(a) Brazilian House (MAE) (b) Brazilian House (R^2) (c) Wine Quality (MAE) (d) Wine Quality (R^2)

Figure 8: Additional results in various metrics on tabular datasets with noise and bias

Implementation As with Section 4.3, we use a model tailored for tabular dataset proposed by (Grinsztajn et al., 2022) and add errors sampled from a skewed normal distribution, with skewness parameter ranging from 0.0 to 3.0.

Results Table 4 (Right) compares flooding methods on NYC Taxi Tip dataset (Grinsztajn et al., 2022) with and without noises. We report mean square error (MSE) and R2 score on the right. Note that R2 score is usually in between 0 and 1 but when predictions are bad, it can go below 0.

From the table, we can observe that all flooding methods perform similar to the unregularized baseline when there is no noise. Although it continues for Flood and iFlood even under noisy settings, AdaFlood significantly outperforms (lower MSE and higher R^2 scores) the other methods when noise level is 1.5 and 3.0. In particular, while R2 scores of other methods go below 0, it does not happen with AdaFlood, which demonstrates the robustness of AdaFlood even for the large-scale dataset like NYC Taxi Tip. It is consistent with the results we provided in Section 4.3.

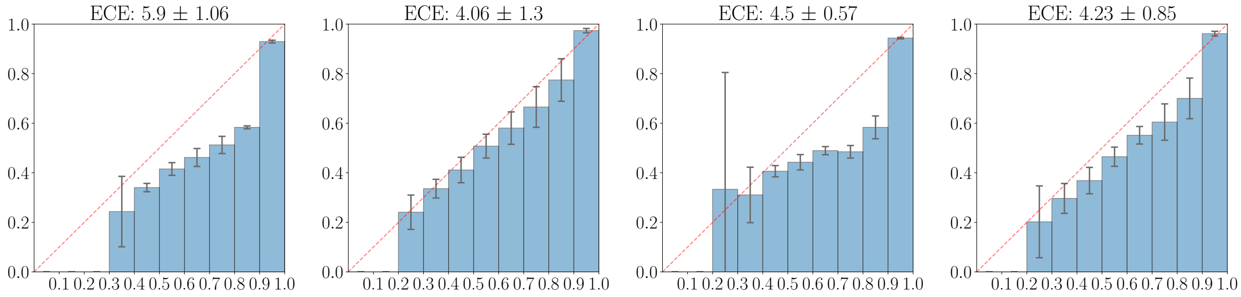
For additional information, in Figure 8, we also provide experiment results in MAE and R^2 on Brazilian House and Wine Quality datasets of which results in MSE is provided in Figures 5a and 5b.

D Additional Results on Calibration

We provide calibration results on CIFAR10 in Figure 9. As briefly mentioned in Section 4.4, since model predictions are generally accurate and quite confident, the gap between different methods are not as significant as on CIFAR100 shown in Figure 6.

E Initialization of the Main Model using an Auxiliary Network

Even though we efficiently fine-tune an auxiliary network to compute flood levels, it may be still too expensive to train both the auxiliary network f^{aux} and main model f. To reduce computation further, we may utilize a pre-trained auxiliary network when we use the same architecture for the auxiliary and main model. Instead of randomly initialize the main model, we can initialize the parameters of the main model using the parameters from the pre-trained auxiliary network (before fine-tuning).



(a) Unregularized (b) Flood (c) iFlood (d) AdaFlood

Figure 9: Calibration results of flooding methods with 10 bins on CIFAR10.

Random Init	Freezing first k layers			
	$k = 0$	$k = 1$	$k = 2$	$k = 3$
90.83 / 3.52	91.30 / 2.50	90.90 / 3.44	90.68 / 3.96	89.99 / 4.69

Table 5: Comparison between random initialization and initialization with the pre-trained auxiliary network. ECE metrics are reported on the right.

We conduct an experiment to validate if it actually saves computation without hurting performance on CIFAR10 with ResNet18. We first trained an auxiliary network on the whole training set. The pre-trained auxiliary network is then used for both computing θ through the fine-tuning step described in Section 3.3 and initializing the main model. When we "fine-tune" the main model, we freeze the first k layers (among four layers of ResNet18) to save computation, and randomly initialize the last fully connected layer to have the model forget some information.

Table 5 shows that if we initialize the parameters of the main model with those of the pre-trained auxiliary model without freezing any layers, it performs better than random initialization. As we freeze more layers to save computation, the performance gradually goes down compared to that of random initialization but it is still comparable up to $k = 2$ case.

F Ablation study: Various Architectures for Auxiliary Networks

	VGG11	VGG19	ResNet18 Small	ResNet18
# of params	9.8 M	21.6 M	2.8 M	11.2 M
AdaFlood	90.74 ± 0.13	91.09 ± 0.06	90.73 ± 0.12	90.82 ± 0.08

Table 6: Comparison of various architectures for auxiliary networks in AdaFlood.

To investigate the robustness of AdaFlood in terms of the choice of architectures for auxiliary networks, we conduct an ablation study on AdaFlood with various architectures for auxiliary networks: VGG11, VGG19, ResNet18 small and ResNet18. Here, we use a ResNet18 for the main model, and utilize the efficient fine-tuning method to train auxiliary networks. We report the mean and standard error of three runs in Table 6.

With VGG11 where its number of parameters is slightly less than ResNet18, the mean test accuracy of the main model is lower but the gap is marginal. With VGG19 which is larger than ResNet18, the performance slightly improves. We also try with a smaller variant of ResNet18 for the auxiliary network where its number of parameters is a quarter of the original ResNet18. Even with this significantly smaller architecture, the mean test accuracy of the main model is only slightly worse than much larger models e.g. VGG11 and ResNet18, which implies that what is important from the auxiliary network is the relative magnitude of losses (or flood levels) not the absolute values of losses.

NTPP	Method	Uber	Reddit	Stack Overflow
Intensity-free	Flood	{4.0, 4.0, 3.0}	{-1.0, -5.0, -3.0}	{0.0, 3.0, 1.0}
	iFlood	{4.0, 0.0, 1.0}	{-2.0, -3.0, -4.0}	{2.0, 2.0, 3.0}
	AdaFlood	{0.2, 0.0, 0.3}	{0.0, 0.1, 0.1}	{0.1, 0.4, 0.2}
THP$^+$	Flood	{0.0, -5.0, 0.0}	{-1.0, -2.0, -15.0}	{0.0, 2.0, 1.0}
	iFlood	{-1.0, 4.0, 3.0}	{-4.0, -4.0, -5.0}	{0.0, 0.0, -1.0}
	AdaFlood	{0.0, 0.1, 0.3}	{0.0, 0.0, 0.1}	{0.3, 0.3, 0.3}

Table 7: Choice of flood levels for (i)Flood, and γ for AdaFlood on TPP datasets with three seeds.

Method	SVHN		CIFAR10		CIFAR100	
	w/o L_2 reg.	w/ L_2 reg.	w/o L_2 reg.	w/ L_2 reg.	w/o L_2 reg.	w/ L_2 reg.
Flood	{0.04, 0.02, 0.03}	{0.02, 0.03, 0.04}	{0.02, 0.04, 0.05}	{0.07, 0.02, 0.01}	{0.01, 0.01, 0.01}	{0.01, 0.01, 0.05}
iFlood	{0.05, 0.09, 0.02}	{0.05, 0.06, 0.06}	{0.07, 0.04, 0.07}	{0.07, 0.03, 0.01}	{0.45, 0.35, 0.30}	{0.30, 0.40, 0.65}
AdaFlood	{0.75, 0.85, 0.70}	{0.35, 0.50, 0.65}	{0.95, 0.70, 0.65}	{0.75, 0.95, 0.65}	{0.40, 0.50, 0.45}	{0.50, 0.50, 0.55}

Table 8: Choice of flood levels for (i)Flood, and γ for AdaFlood on image classification with three seeds.

G Theoretic Intuition for Efficient Training of Auxiliary Networks

In this section, we provide theoretic intuition for efficient training of auxiliary networks. Following (Lee et al., 2019), we approximate the predictions of a neural network f on a test sample x_i

$$f(x_i) \approx \sum_{j=1}^{n} \alpha_j \mathrm{eNTK}(x_i, x_j) \tag{11}$$

where eNTK stands for empirical neural tangent kernel (NTK) following (Mohamadi et al., 2023) and $\{x_j\}_{j=1}^{n}$ denotes data from a training set. Equation (11) says we can approximate a prediction on x_i as an interpolation of $\mathrm{eNTK}(x_i, \cdot)$ with some weights α.

Suppose $f(x) = V\phi(x)$ where $\phi(x) \in \mathbb{R}^h$ denotes a feature from the penultimate layer and $V \in \mathbb{R}^{k \times h}$ denotes the weights of the last fully connected layer (k being the number of classes), consisting of $v_j \in \mathbb{R}^h$ for j-th row. If $v_{j,i}$, i-th entry of v_j, is from $N(0, \sigma^2)$, then Mohamadi et al. (2023) haven shown that,

$$\mathrm{eNTK}_w(x_1, x_2)_{jj'} = v_j^T \mathrm{eNTK}^{\phi}_{w \setminus V}(x_1, x_2) v_{j'} + \mathbb{1}(j = j')\phi(x_1)^T \phi(x_2) \tag{12}$$

where w denotes a set of all the model parameters and $w \setminus V$ means a set of all the parameters except the last fully connected layer.

With this frame, we can approximate the predictions from the efficiently trained auxiliary network denoted as f_{tune}, as follows,

$$f_{\text{tune}}(x_i) \approx \sum_{j \neq i} \alpha_j^{\text{tune}} \left(v^T \mathrm{eNTK}^{\phi,\text{trained}}_{w \setminus V}(x_j, x_i) v + \phi^{\text{trained}}(x_j)^T \phi^{\text{trained}}(x_i) \right). \tag{13}$$

Here, superscript "trained" means the model parameters are pre-trained on the whole training set.

On the other hand, the original "direct" training algorithm of auxiliary networks trains an auxiliary network from scratch, on the training set excluding the samples that we measure the difficulty on. Similarly, the predictions of a single auxiliary network denoted as f_{direct} is approximated as,

$$f_{\text{direct}}(x_i) \approx \sum_{j \neq i} \alpha_j^{\text{direct}} \left(v^T \mathrm{eNTK}^{\phi,\text{untrained}}_{w \setminus V}(x_j, x_i) v + \phi^{\text{untrained}}(x_j)^T \phi^{\text{untrained}}(x_i) \right) \tag{14}$$

where superscript "untrained" means the model parameters are randomly initialized.

In **NTK regime** where a neural network has infinite-width, the terms in the parentheses (the term except α's) are the same for Equation (13) and Equation (14). Therefore, the difficulty measures from the efficient training of a single auxiliary network and direct training of multiple auxiliary networks are equivalent in highly-overparameterized regime.

H Choice of Flood Levels

In Table 7 and Table 8, we report the choice of flood levels for Flood and iFlood, and γ for AdaFlood (recall that γ is the hyperparameter for correct function adjusting interpolation level) on TPP and image classification tasks with three different random seeds.

One interesting observation is that flood levels for iFlood on CIFAR100 is significantly larger than on the other datasets. It is because CIFAR100 is particularly harder than the other two datasets. However, even though a flood level is sometimes high *e.g.* 0.65, it is still a reasonable choice because 0.65 flood level implies that the model's highest predicted probabilities do not deviate much from $0.52 \sim \exp(-0.65)$. With this high regularization, the model is not too overconfident for its predictions.