Guided Learning of Nonconvex Models through Successive Functional Gradient Optimization

Rie Johnson [1] **Tong Zhang** [2]

Abstract

This paper presents a framework of successive functional gradient optimization for training nonconvex models such as neural networks, where training is driven by mirror descent in a function space. We provide a theoretical analysis and empirical study of the training method derived from this framework. It is shown that the method leads to better performance than that of standard training techniques.

1. Introduction

This paper presents a new framework to train nonconvex models such as neural networks. The goal is to learn a vector-valued function $f(\theta; x)$ that predicts an output y from input x, where θ is the model parameter. For example, for K-class classification where $y \in \{1, 2, \ldots, K\}$, $f(\theta; x)$ is K-dimensional, and it can be linked to conditional probabilities via the soft-max logistic function. Given a set of training data S, the standard method for solving this problem is to use stochastic gradient descent (SGD) for finding a parameter that minimizes on S a loss function $L(f(\theta; x), y)$ with a regularization term $R(\theta)$: $\min_\theta \left[\frac{1}{|S|} \sum_{(x,y)\in S} L(f(\theta; x), y) + R(\theta) \right]$.

In this paper, we consider a new framework that *guides training through successive functional gradient descent* so that training proceeds with alternating the following:

- Generate a guide function so that it is ahead (but not too far ahead) of the current model with respect to the minimization of the loss. This is done by functional gradient descent.
- 'Push' the model towards the guide function.

[1]RJ Research Consulting, Tarrytown, New York, USA [2]Hong Kong University of Science and Technology, Hong Kong. Correspondence to: Rie Johnson <riejohnson@gmail.com>, Tong Zhang <tongzhang@tongzhang-ml.org>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

Our original motivation was functional gradient learning of additive models in *gradient boosting* (Friedman, 2001). In our framework, essentially, training proceeds with repeating a local search, which limits the searched parameter space to the functional neighborhood of the current parameter at each iteration, instead of searching the entire space at once as the standard method does. This is analogous to ε-boosting where the use of a very small step-size (for successively expanding the ensemble of weak functions) is known to achieve better generalization (Friedman, 2001).

For measuring the distances between models, we use the Bregman divergence (see e.g., (Bubeck, 2015)) by applying it to the model output. Given a convex function h, the Bregman divergence D_h is defined by

$$D_h(u, v) = h(u) - h(v) - \nabla h(v)^\top (u - v). \quad (1)$$

This is the difference between $h(u)$ and the approximation of $h(u)$ based on the first-order Taylor expansion around v. This means that when $u - v$ is small,

$$D_h(u, v) \approx \frac{1}{2}(u - v)^\top (\mathrm{H}\,(h(v)))(u - v), \quad (2)$$

where $\mathrm{H}\,(h(v))$ denotes the Hessian matrix of h with respect to v. Therefore, use of the Bregman divergence has the beneficial effect of utilizing the second-order information.

We show that the parameter update rule of an induced method generalizes that of *distillation* (Hinton et al., 2014). That is, our framework subsumes iterative *self-distillation* as a special case.

Distillation was originally proposed to *transfer knowledge* from a high-performance but cumbersome model to a more manageable model. Various forms of self-distillation, which applies distillation to the models of the same architecture, has been empirically studied (Xu & Liu, 2019; Yang et al., 2019a; Lan et al., 2018; Furlanello et al., 2018; Anil et al., 2018; Zhang et al., 2018; Tarvainen & Valpola, 2017; Yim et al., 2017). One trend is to add to the original scheme, e.g., adding a term to the update rule, data distortion/division, more models for mutual learning, and so forth. However, we are not aware of any work on theoretical understanding such as a convergence analysis of the basic self-learning scheme.

Our theoretical analysis of the proposed framework provides a new functional gradient view of self-distillation, and we show a version of the generalized self-distillation procedure converges to a stationary point of a regularized loss function. Our empirical study shows that the iterative training of the derived method goes through a 'smooth path' in a restricted region with good generalization performance. This is in contrast to standard training, where the entire (and therefore much larger) parameter space is directly searched, and thus complexity may not be well controlled.

Notation $\nabla h(v)$ denotes the gradient of a scalar function h with respect to v. We omit the subscript of ∇ when the gradient is with respect to the first argument, e.g., we write $\nabla f(\theta; x)$ for $\nabla_\theta f(\theta; x)$. $\mathrm{H}\,(h(v))$ denotes the Hessian matrix of a scalar function h with respect to v. We use x and y for input data and output data, respectively. We use $<>$ to indicate the mean, e.g., $\langle F(x,y)\rangle_{(x,y)\in S} = \frac{1}{|S|}\sum_{(x,y)\in S} F(x,y)$. $L(u,y)$ is a loss function with y being the true output. We also let $L_y(u) = L(u,y)$ when convenient.

2. Guided Learning through Successive Functional Gradient Optimization

In this section, after presenting the framework in general terms, we develop concrete algorithms and analyze them.

2.1. Framework

We first describe the framework in general terms so that the models to be trained are not limited to parameterized ones. Let f be the model we are training. Starting from some initial f, training proceeds by repeating the following:

1. Generate a *guide function* f^* by applying functional gradient descent for reducing the loss to the current model f, so that f^* is an improvement over f in terms of loss but not too far from f.
2. Move the model f in the direction of the guide function f^* according to some distance measure.

We use the Bregman divergence D_h, defined in (1), for representing the distances between models.

Step 1: Guide going ahead We formulate Step 1 as

$$f^*(x,y) := \operatorname*{argmin}_q \left[D_h(q, f(x)) + \alpha \nabla L_y(f(x))^\top q \right], \quad (3)$$

where α is a meta-parameter. The second term pushes the guide function towards the direction of reducing loss, and the first term pulls back the guide function towards the current model f. Thus, f^* is ahead of f but not too far ahead. Note that we use the knowledge of the true output y here; therefore, f^* takes y as the second argument. The function value for each data point (x,y) can be found approximately by solving the optimization problem by SGD if there is no analytical solution. Also, this formulation is equivalent to finding f^* such that

$$\nabla h(f^*(x,y)) = \nabla h(f(x)) - \alpha \nabla L_y(f(x)). \quad (4)$$

This is mirror descent (see e.g., (Bubeck, 2015)) performed in a function space.

Due to the relation of the Bregman divergence to the Hessian matrix stated in (2), (3) implies that

$$f^*(x,y) \approx f(x) - \alpha (\mathrm{H}\,(h(f(x))))^{-1} \nabla L_y(f(x)). \quad (5)$$

Therefore, if we set $h(f) = L_y(f)$, (5) becomes

$$f^*(x,y) \approx f(x) - \alpha (\mathrm{H}\,(L_y(f(x))))^{-1} \nabla L_y(f(x)), \quad (6)$$

which is approximately a second-order functional gradient step (one step of the relaxed Newton method) with step-size α for minimizing the loss.

If we set $h(f) = \frac{1}{2}\|f\|^2$, then the optimization problem (3) has an analytical solution

$$f^*(x,y) = f(x) - \alpha \nabla L_y(f(x)),$$

which is a first-order functional gradient step with step-size α for minimizing the loss.

Taking m steps in Step 1 For further generality, let us also consider m steps of functional gradient descent by extending f^* in (3) to f^*_m recursively defined as follows.

$$f^*_0(x,y) := f(x)$$
$$f^*_{i+1}(x,y) := \arg\min_q \left[D_h(q, f^*_i(x)) + \alpha \nabla L_y(f^*_i(x))^\top q \right].$$

Then, in parallel to (5), we have

$$f^*_{i+1}(x,y) \approx f^*_i(x) - \alpha (\mathrm{H}\,(h(f^*_i(x))))^{-1} \nabla L_y(f^*_i(x)).$$

Step 2: Following the guide Using the Bregman divergence D_h, we formulate Step 2 above as an update of the model f to reduce

$$\left\langle D_h(f(x), f^*(x,y)) \right\rangle_{(x,y)\in S} + R(f) \quad (7)$$

so that the model f approaches the guide function f^* in terms of the Bregman divergence. $R(f)$ is a regularization term.

Parameterization Although there can be many variations of this scheme, in this work, we parameterize the model f so that we can train neural networks. Thus, we replace $f(x)$ by $f(\theta; x)$ with parameter θ. This does not affect Step 1, and to reduce (7) in Step 2, we repeatedly update the model parameter θ by descending the stochastic gradient

$$\nabla_\theta \left[\left\langle D_h(f(\theta; x), f^*(x,y)) \right\rangle_{(x,y)\in B} + R(\theta) \right], \quad (8)$$

where B is a mini-batch sampled from a training set S.

2.2. Algorithms

Putting everything together, we obtain Algorithm 1, which performs mirror descent in a function space in Line 3. We call it (and its derivatives) a method of *GUided Learning through successive Functional gradient optimization (GULF)*. We now instantiate function h used by the Bregman divergence D_h to derive concrete algorithms. In general we allow h to vary for each data point. That is, it may depend on (x, y). Here we use two functions discussed above, which correspond to the first-order and the second-order methods, respectively; however, note that choice of h is not limited to these two.

Algorithm 1 GULF in the most general form. **Input:** θ_0, training set S. Meta-parameters: m, α, T. **Output:** θ_T.

1: $\theta \leftarrow \theta_0$
2: **for** $t = 0$ **to** $T - 1$ **do**
3: Define f_m^* by: $f_0^*(x, y) := f(\theta_t; x)$, $f_{i+1}^*(x, y) := \arg\min_q \left[D_h(q, f_i^*(x, y)) + \alpha \nabla L_y(f_i^*(x, y))^\top q\right]$
4: **repeat**
5: Sample a mini-batch B from S.
6: Update θ by descending the stochastic gradient $\nabla_\theta \left[\left\langle D_h(f(\theta; x), f_m^*(x, y))\right\rangle_{(x,y)\in B} + R(\theta)\right]$ for optimizing $Q_t(\theta) := \left\langle D_h(f(\theta; x), f_m^*(x, y))\right\rangle_{(x,y)\in S} + R(\theta)$.
7: **until** some criteria are met
8: $\theta_{t+1} \leftarrow \theta$
9: **end for**

GULF1 (1st-order, Algorithm 2) With $h(u) = \frac{1}{2}\|u\|^2$, we obtain Algorithm 2. Derivation is straightforward. This algorithm performs m steps of first-order functional gradient descent (Line 3) to push the guide function ahead of the current model and then let the model follow the guide by reducing the 2-norm between them.

Algorithm 2 GULF1 ($h(u) = \frac{1}{2}\|u\|^2$): **Input:** θ_0, training set S. Meta-parameters: m, α, T. **Output:** θ_T.

1: $\theta \leftarrow \theta_0$
2: **for** $t = 0$ **to** $T - 1$ **do**
3: Define f_m^* by: $f_0^*(x, y) = f(\theta_t; x)$, $f_{i+1}^*(x, y) = f_i^*(x, y) - \alpha \nabla L_y(f_i^*(x, y))$
4: **repeat**
5: Sample a mini-batch B from S.
6: Update θ by descending the stochastic gradient $\nabla_\theta \left[\left\langle \frac{1}{2}\|f(\theta; x) - f_m^*(x, y)\|^2\right\rangle_{(x,y)\in B} + R(\theta)\right]$
7: **until** some criteria are met
8: $\theta_{t+1} \leftarrow \theta$
9: **end for**

GULF2 (2nd order, Algorithm 3) We consider the case of $h(p) = L_y(p)$ (i.e., h returns loss given prediction p). (6) has shown that in this case Step 1 becomes approximately the second-order functional gradient descent. Also, with this choice of h, Algorithm 1 can be converted to a simpler form where we do not have to compute the values of the guide function f_m^* explicitly, and where we have one fewer meta-parameter. This simpler form is shown in Algorithm 3, which has the following relationship to Algorithm 1.

Algorithm 3 GULF2 ($h(p) = L_y(p)$): **Input:** θ_0, training set S. Meta-parameters: $\alpha \in (0, 1)$, T. **Output:** θ_T. Notation: $f_\theta = f(\theta; x)$ and $f_{\theta_t} = f(\theta_t; x)$.

$\theta \leftarrow \theta_0$
for $t = 0$ **to** $T - 1$ **do**
repeat
Sample a mini-batch B from S.
Update θ by descending the stochastic gradient $\nabla_\theta \left[\left\langle D_{L_y}(f_\theta, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top f_\theta\right\rangle_{(x,y)\in B} + R(\theta)\right]$
until some criteria are met
$\theta_{t+1} \leftarrow \theta$
end for

Proposition 2.1 *When $h(p) = L_y(p)$ that returns loss given prediction p, Algorithm 1 with $\alpha = \gamma$ is equivalent to Algorithm 3 with $\alpha = 1 - (1 - \gamma)^m$.*

The proofs are all provided in the supplementary material.

To simplify notation, let $f_\theta = f(\theta; x)$, which is the model that we are updating, and $f_{\theta_t} = f(\theta_t; x)$, which is a model that was frozen when time changed from $t - 1$ to t. In the stage associated with time t, Algorithm 3 minimizes

$$\left\langle D_{L_y}(f_\theta, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top f_\theta\right\rangle_{(x,y)\in S} + R(\theta) \quad (9)$$

approximately through mini-batch SGD. The second term $\alpha \nabla L_y(f_{\theta_t})^\top f_\theta$ pushes the model f_θ towards the direction of reducing loss, and the first term $D_{L_y}(f_\theta, f_{\theta_t})$ pulls it back towards the frozen model f_{θ_t}. With a certain family of loss functions, (9) can be further transformed as follows.

Proposition 2.2 *Let y be a vector representation such as a K-dim vector representing K classes. Assume that the gradient of the loss function can be expressed as*

$$\nabla L(f, y) = \nabla L_y(f) = p(f) - y \quad (10)$$

with $p(f)$ not depending on y. Let

$$J_t(\theta) = \left\langle D_{L_y}(f_\theta, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top f_\theta\right\rangle_{(x,y)\in S} \quad (11)$$

$$J'_t(\theta) = \left\langle (1 - \alpha) L(f_\theta, p(f_{\theta_t})) + \alpha L_y(f_\theta)\right\rangle_{(x,y)\in S} \quad (12)$$

Then we have

$$J_t(\theta) = J'_t(\theta) + c_t \ ,$$

where c_t is independent of θ. This implies that

$$\arg\min_\theta \left[J_t(\theta) + R(\theta)\right] = \arg\min_\theta \left[J'_t(\theta) + R(\theta)\right] .$$

Both the cross-entropy loss and squared loss satisfy (10). In particular, when $L_y(f)$ is the cross-entropy loss, $p(f)$ becomes the soft max function. In this case, (12) is the *distillation* formula with the frozen model f_{θ_t} playing the role of a cumbersome source model, and therefore, the parameter update rule of Algorithm 3 involving (11) becomes that of distillation. Thus, Algorithm 3 can be regarded as a generalization of self-distillation for arbitrary loss functions.

2.3. Convergence Analysis

Let us define α*-regularized loss*

$$\ell_\alpha(\theta) := \left\langle L\left(f(\theta; x), y\right)\right\rangle_{(x,y)\in S} + \frac{1}{\alpha}R(\theta). \qquad (13)$$

The following theorem shows that Algorithm 1 with step-size α always approximately reduces the α-regularized loss if α is appropriately set.

Theorem 2.1 *In the setting of Algorithm 1 with $m = 1$, assume that there exists $\beta > 0$ such that $D_h(f, f') \geq \beta D_{L_y}(f, f')$ for any f and f', and assume that $\alpha \in (0, \beta]$. Assume also that $Q_t(\theta)$ defined in Algorithm 1 is $1/\eta$ smooth in θ: $\|\nabla Q_t(\theta) - \nabla Q_t(\theta')\| \leq (1/\eta)\|\theta - \theta'\|$.*

Assume that θ_{t+1} is an improvement of θ_t with respect to minimizing Q_t so that $Q_t(\theta_{t+1}) \leq Q_t(\tilde{\theta})$, where

$$\tilde{\theta} = \theta_t - \eta\nabla Q_t(\theta_t). \qquad (14)$$

Then we have

$$\ell_\alpha(\theta_{t+1}) \leq \ell_\alpha(\theta_t) - \frac{\alpha\eta}{2}\|\nabla\ell_\alpha(\theta_t)\|^2.$$

For Algorithm 3, we have $h(\cdot) = L_y(\cdot)$ and thus $\beta = 1$, leading to $\alpha \in (0, 1]$. (14) is the parameter update step of Algorithm 1 except that the algorithm stochastically estimates the mean over S from a mini-batch B sampled from S. Therefore, the theorem indicates that each stage (corresponding to t) of the algorithm approximately reduces the α-regularized loss $\ell_\alpha(\theta)$. In other words, while the guide function changes from stage to stage, a quantity that does *not* depend on the guide function goes down throughout training, namely, the α-regularized loss ℓ_α.

Furthermore, we obtain from Theorem 2.1 that

$$\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla\ell_\alpha(\theta_t)\|^2 \leq \frac{2(\ell_\alpha(\theta_0) - \ell_\alpha(\theta_T))}{\alpha\eta T}.$$

Assuming $\ell_\alpha(\theta) \geq 0$, this implies that as T goes to infinity, the right-hand side goes to zero, and so Algorithm 3 converges with $\nabla\ell_\alpha(\theta_T) \to 0$. Therefore, when T is sufficiently large, θ_T finds a stationary point of ℓ_α.

The convergence result indicates that having a regularization term $R(\theta)$ in the algorithm effectively causes minimization of the α-regularized loss. However, our empirical results (shown later) indicate that GULF models are very different from standard models trained directly to minimize the α-regularized loss. For example, standard models trained with $\ell_{0.01}$ suffers from severe underfitting, but GULF model with α=0.01 produces high performance. This is because each step of guided learning tries to find a good solution which is near the previous solution (guidance). The complexity of each iterate is better controlled, and hence this approach leads to better generalization performance. We will come back to this point in the next section.

3. Empirical study

While the proposed framework is general, our empirical study places a major focus on GULF2 (Algorithm 3) with the cross-entropy loss, due to its connection to distillation (Proposition 2.2). In particular, we set up our implementation so that one instance of GULF2 coincides with self-distillation to provide empirical insight into it from a functional gradient viewpoint.

First, with the goal of understanding the empirical behavior of the algorithm, we examine obtained models in reference to our theoretical findings. We use relatively small neural networks for this purpose. Next, we study the case of larger networks with consideration of practicality.

3.1. Implementation

To implement the algorithms presented above, methods of parameter initialization and optimization need to be considered. To observe the basic behavior, our strategy in this work is to keep it as simple as possible.

Initial parameter θ_0 As the functions of interest are nonconvex, the outcome depends on the initial parameter θ_0. The most natural (and simplest) choice is random parameters. This option is called 'ini:random' below. We also considered two more options. One is to start from a *base model* obtained by regular training, called 'ini:base'. This option enables study of self-distillation. The other is to start from a shrunk version of the base model, and details of this option will be provided later.

Parameter update To update parameter θ by descending the stochastic gradient, standard techniques can be used such as momentum, Rmsprop (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2015), and so forth. As is the case for regular training, learning rate scheduling is beneficial. Among many possibilities, we chose to repeatedly use for each t, the same method that works well for regular training. For example, a standard method for CIFAR10 is to use momentum and decay the learning rate only a few times, and therefore we use this scheme for each stage on CIFAR10. That is, the learning rate is reset to the initial rate for each t; however

Algorithm 4 base-loop (simplified SGDR): **Input:** θ_0, training set S. Meta-parameter: T. **Output:** θ_T.

for $t = 0$ **to** $T-1$ **do**
  $\theta_{t+1} \leftarrow \arg\min_\theta \left[\langle L_y(f(\theta; x)) \rangle_{(x,y)\in S} + R(\theta) \right]$
    where θ is initialized by θ_t.
end for

	#class	train	dev.	test
CIFAR10	10	49000	1000	10000
CIFAR100	100	49000	1000	10000
SVHN	10	599388	5000	26032
ImageNet	1000	1271167	10000	50000

Table 1. Data. For each dataset, we randomly split the official training set into a training set and a development set to use the development set for meta-parameter tuning. For ImageNet, following custom, we used the official validation set as our 'test' set.

note that θ is not reset. Although this is perhaps not the best strategy in terms of computational cost, its advantage is that at the end of each stage, we obtain "clean" intermediate models with θ_t that were optimized for intermediate goals. (If instead, we used one decay schedule from the beginning to the end, the convergence theorem still holds, but θ_t would be noisy when the learning rate is still high.) This strategy enables to study how a model changes as the guide function gradually goes ahead, and also relates the method to self-distillation.

Since θ is not reset when the learning rate is reset, this schedule can be regarded as a simplified fixed-schedule version of *SGD with warm restarts (SGDR)* (Loshchilov & Hutter, 2017b). (SGDR instead does sophisticated scheduling with cosine-shape decay and variable epochs.) For comparison, we test the same schedule with the standard optimization objective ('base-loop'; Algorithm 4).

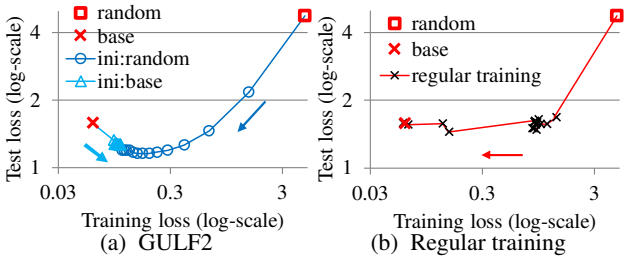
Enabling study of self-distillation We study classification tasks with the standard cross-entropy loss, which satisfies the condition of Proposition 2.2. Combined with the choice of learning rate scheduling above, GULF2 with the ini:base option (which initializes θ_0 with a trained model) essentially becomes self-distillation. Thus, one aspect of our experiments is to study self-distillation from the viewpoint of functional gradient learning.

3.2. Experimental setup

Table 1 summarizes the data we used. As for network architectures, we mainly used ResNet (He et al., 2016a;b) and wide ResNet (WRN) (Zagoruyko & Komodakis, 2016). Following the original work, the regularization term $R(\theta)$ was set to be $R(\theta) = \frac{\lambda}{2}\|\theta\|^2$ where λ is the weight decay. We fixed mini-batch size to 128 and used the same learning rate decay schedule for all but ImageNet. Due to the page limit, details are described in the supplementary material. However, note that the schedule we used for all but ImageNet is 3–4 times longer than those used in the original ResNet or WRN study for CIFAR datasets. This is because we used the "train longer" strategy (Loshchilov & Hutter, 2017a), and accordingly, the base model performance visibly improved from the original work. This, in fact, made it harder to obtain large performance gains over the base models (not only for GULF but also for all other tested methods) as the bar was set higher. We feel that this is more realistic testing than using the original shorter schedule.

We applied the standard mean/std normalization to images and used the standard image augmentation. In particular, for ImageNet, we used the same data augmentation scheme as used for training the pre-trained models provided as part of TorchVision, since we used these models as our base model.

The default value of α is 0.3.



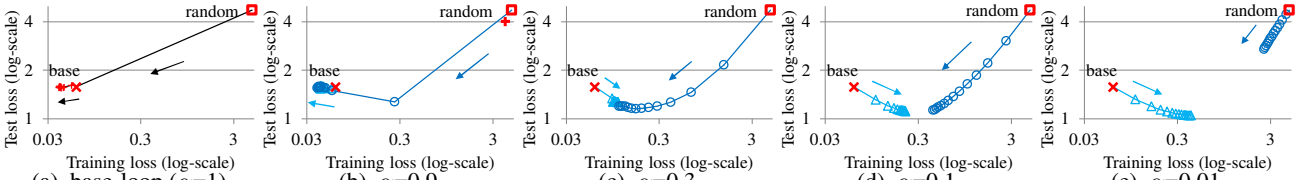
(a) GULF2 (b) Regular training

Figure 1. Test loss in relation to training loss. The arrows indicate the direction of time flow. CIFAR100. ResNet-28.

3.3. Smooth path

We start with examining training of a relatively small network ResNet-28 (0.4M parameters) on CIFAR100. In this setting, optimization is fast, and so a relatively large T (the number of stages) is feasible.

We performed GULF2 training with T=25 starting from random parameters (ini:random) as well as starting from a base model obtained by regular training (ini:base). Figure 1a plots test loss of these two runs in relation to training loss. Each point represents a model $f(\theta_t; x)$ at time $t = 1, 3, 5, \cdots, 25$, and the arrows indicate the direction of time flow. We observe that training proceeds on a *smooth path*. ini:random(○), which starts from random parameters (□), reduces both training loss and test loss. ini:base(△) starts from the base model (×) and increases training loss, but reduces test loss. ini:random and ini:base meet and complete one *smooth path* from a random state (□) to the base model (×). ini:random goes forward on this path while ini:base goes backward, and importantly, the path goes through the region where test loss is lower than that of the base model. The test error plotted against training loss also forms a U-shape path. Similar U-shape curves were observed across datasets and network architectures. The supplementary material shows a test error curve and a few more examples of test loss curves including a case of



(a) base-loop (α=1) (b) α=0.9 (c) α=0.3 (d) α=0.1 (e) α=0.01

Figure 2. Test loss of ini:base('△') and ini:random('○'). with five values of α (becoming smaller from left to right), in relation to training loss. GULF2. T=25. CIFAR100. ResNet-28. As α becomes smaller, the (potential) meeting point shifts further away from the base model. The left-most figure shows base-loop, which is equivalent to α=1.

DenseNet (Huang et al., 2017).

In the middle of this path, a number of models with good generalization performance lie. One might wonder if regular training also forms such a path. Figure 1b shows that this is not the case. This figure plots the loss of intermediate models in the course of regular training so that the i-th point represents a model after 20K$\times i$ steps of mini-batch SGD with the learning rate being reduced twice. The path of regular training from random initialization (□) to the final model (×) is rather bumpy and the test loss generally stays as high as the final outcome. The bumpiness is due to the fact that the learning rate is relatively high at the beginning of training. Comparing Figures 1a and 1b, GULF training clearly takes a very different path from regular training.

3.4. In relation to the theory

Going forward, going backward It might look puzzling why ini:base goes *backward* in the direction of *increasing* the training loss. Theorem 2.1 suggests that this is the effect of the regularization term $R(\theta)$, in this case $R(\theta) = \frac{\lambda}{2}\|\theta\|^2$ with weight decay λ. The theory indicates that for $\alpha \in (0, 1]$, the α-regularized loss

$$\ell_\alpha(\theta) = \left\langle L_y(f(\theta; x)) \right\rangle_{(x,y)\in S} + R(\theta)/\alpha$$

goes down and eventually converges as GULF2 proceeds. By contrast, The base model is a result of minimizing

$$\left\langle L_y(f(\theta; x)) \right\rangle_{(x,y)\in S} + R(\theta).$$

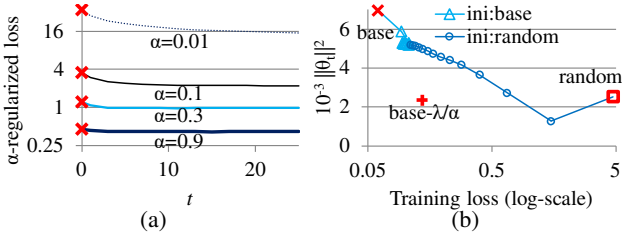
As we always set $\alpha < 1$ (0.3 in this case), i.e., $1/\alpha > 1$, GULF2 prefers smaller parameters than the base model does. Consequently, when GULF2 (with small α) starts from the base model (which has low training loss and high $R(\theta)$), GULF2 is likely to reduce $R(\theta)/\alpha$ at the expense of increasing loss (going backward). When GULF2 starts from random parameters, whose training loss is high, GULF2 is likely to reduce loss (going forward) at the expense of increasing $R(\theta)/\alpha$.

Effects of changing α With GULF2, the guide function f^* satisfies

$$f^* \approx f_{\theta_t} - \alpha(\mathrm{H}\left(L_y(f_{\theta_t})\right))^{-1}\nabla L_y(f_{\theta_t}),$$

thus, α serves as a step-size of functional gradient descent for *reducing loss*. The effects of changing α are shown in Figure 2 with T fixed to 25. The left-most graph is base-loop, which is equivalent to GULF2 with α=1 in this implementation. There are three things to note. First, with a very small step-size α=0.01 (the right most), ini:random cannot reach far from the random state for T=25. This is a straightforward effect of a small step size. Second, as step-size α becomes smaller (from left to right), the (potential) meeting/convergence point shifts further away from the base model; the convergence point of $\|\theta_t\|^2$ also shifts away from the base model and decreases (supplementary material). This is the effect of larger $R(\theta)/\alpha$ for smaller α. Finally, with a large step-size (0.9 and 1), the curve flattens and it no longer goes through the high-performance regions *slowly* or *smoothly*, and the benefit diminishes/vanishes.

α-regularized loss $\ell_\alpha(\theta)$ Figure 3a confirms that, as suggested by the theory, $\ell_\alpha(\theta)$ goes down and almost converges as training proceeds. This fact motivates examining standard models trained with this $\ell_\alpha(\theta)$ objective, which we call base-λ/α models. We found that base-λ/α models do not perform as well as GULF2 at all. In particular, with a very small α=0.01, which 100 times tightens regularization, test error of base-λ/α drastically degrades due to underfitting; in contrast, ini:base with α=0.01 performs well. Moreover, base-λ/α models are very different from GULF2 models with corresponding α even with a moderate α. For example, Figure 3b plots the parameter size $\|\theta_t\|^2$ in relation to training loss for α=0.3. base-λ/α is clearly far away from where ini:base and ini:random converge to.



(a) (b)

Figure 3. (a) α-regularized loss $\ell_\alpha(\theta)$ in relation to time t. GULF2 ini:base. (b) $\|\theta_t\|^2$ and training loss of base-λ/α in comparison with GULF2. α=0.3. CIFAR100. ResNet-28.

Benefit of guiding This fact illustrates the merit of guided learning (including self-distillation). GULF (indirectly and locally) minimizes the α-regularized loss $\ell_\alpha(\theta)$, but it does

this against the restraining force of *pulling the model back* to the current model. This serves as a form of regularization. Without such a force, training for, say, $\ell_{0.01}$ would make a big jump to rapidly reduce the parameter size and end up with a radical solution that suffers severely from underfitting. This is what happens with base-λ/α. By contrast, guided learning finds a more moderate solution with good generalization performance, and this is the benefit of extra regularization (in the form of pulling back) provided through the guide function. The regularization effect of distillation has been mentioned (Hinton et al., 2014), and our framework formalizes the notion through the functional gradient learning viewpoint.

3.5. With smaller networks

Now we review the test error results of using relatively small networks in Table 2. T for GULF2 and base-loop was fixed to 25 on CIFAR10/100 and 15 on SVHN. Step-size α was fixed to 0.3 for ini:random and chosen from { 0.01,0.03 } for ini:base. GULF2 is consistently better than the base model (Row 1) and generally better than the three baseline methods (Row 2–4). The base-λ/α results (Row 2) were obtained by α=0.3, and they are generally not much different from the base model. base-loop (Row 3) generally makes small improvement over the base model, but it generally falls short of GULF2. A common technique, label smoothing (Row 4) (Szegedy et al., 2016), 'softens' labels by taking a small amount of probability from the correct class and distributing it equally to the incorrect classes. It generally worked well, but the improvements were small. That is, the three baseline methods produced performance gains to some extent, but their gains are relatively small, and they are not as consistent as GULF2 across datasets.

ini:random In these experiments, ini:random performed as well as ini:base. This fact cannot be explained from the knowledge-transfer viewpoint of distillation, but it can be explained from our functional gradient learning viewpoint, as in the previous section.

3.6. With larger networks

The neural networks and the size of images (32×32) used above are relatively small. We now consider computationally more expensive cases.

			C10	C100	SVHN	
1	baselines	base model	6.42	30.90	1.86	1.64
2		base-λ/α	6.60	30.24	1.78	1.67
3		base-loop	6.20	30.09	1.93	**1.53**
4		label smooth	6.66	30.52	1.71	1.60
5	GULF2	ini:random	5.91	**28.83**	1.71	**1.53**
6		ini:base	**5.75**	29.12	**1.65**	1.56

Table 2. Test error (%). Median of 3 runs. Resnet-28 (0.4M parameters) for CIFAR10/100, and WRN-16-4 (2.7M parameters) for SVHN. Two numbers for SVHN are without and with dropout. base-λ/α: weight decay λ/α. base-loop: Algorithm 4.

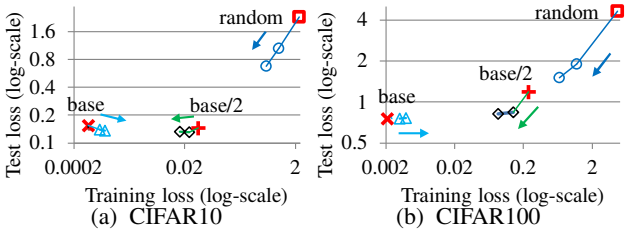


Figure 4. Test loss in relation to training loss. WRN-28-10 on CIFAR10 and CIFAR100. GULF2. ini:base/2 ('⋄') fills the gap between ini:random ('○') and ini:base ('△').

		CIFAR10	CIFAR100
1	base model	3.82	18.55
2	base-λ/α	3.70	27.89
3	base-loop	3.70	18.91
4	lab smooth	4.13	19.44
5	GULF1	**3.46**	18.14
6	GULF2	3.63	**17.95**

Table 3. Test error (%) results on CIFAR10 and CIFAR100. WRN-28-10 (36.5M parameters) without dropout. Median of 3 runs.

Parameter shrinking ini:random, the most natural option from the functional gradient learning viewpoint, unfortunately, turned out to be too costly in this large-network situation. Moreover, in this setting, it is useful to have an option of starting somewhere between the two end points ('random' and 'base') since that is where good models tend to lie according to our study with small networks. Therefore, we experimented with 'rewinding' a base model by shrinking its weights and bias of the last fully-connected linear layer by dividing them with $V > 1$ (a meta-parameter). We use these partially-shrunk parameters as the initial parameter θ_0 for GULF. Since doing so shrinks the model output $f(\theta_0; x)$ by the factor of V, this is closely related to *temperature scaling*, for *distillation* (Hinton et al., 2014) and post-training calibration (Guo et al., 2017). Parameter shrinking is, however, simpler than temperature scaling of distillation, which scales logits of both models, and fits well in our framework.

Figure 4 shows training loss (the x-axis) and test loss (the y-axis) obtained when parameter shrinking is applied to WRN-28-10 on CIFAR10 and CIFAR100. By shrinking with V=2, the loss values of the base model change from 'base' ($\times$) to 'base/2' ($+$). The location of base/2 is roughly the midpoint of two end points 'base' and 'random'. ini:base/2 ($\diamond$), which starts from the shrunk model, explores the space neither ini:random nor ini:base can reach in a few stages.

Larger ResNets on CIFAR10 and CIFAR100 Table 3 shows test error of ini:base/2 using WRN-28-10 on CIFAR10/100. T was fixed to 1. Compared with the base model, both GULF1 and 2 consistently improved performance, while the baseline methods mostly failed to make

improvements. GULF1 and 2 produced similar performances. This is the best WRN non-ensemble results on CIFAR10/100 among the self-distillation related studies that we are aware of.

ImageNet To test further scale-up, we experimented with ResNet-50 (25.6M parameters) and WRN50-2 (68.9M parameters) on the ILSVRC-2012 ImageNet dataset. As ImageNet training is resource-consuming, we only tested selected configurations, which are GULF2 with ini:base and ini:base/2 options. In these experiments, α was set to 0.5, but partial results suggested that 0.3 works well too. We used models pre-trained on ImageNet provided as part of TorchVision[1] as the base models. Table 4 shows that GULF2 consistently improves error rates over the base model. The best-performing ini:base/2 achieved lower error rates than a twice deeper counterpart of each network, ResNet-101 for ResNet-50 and WRN-101-2 for WRN-50-2 trained in a standard way (Rows 11–12). Thus, we confirmed that GULF2 scales up and brings performance gains on ImageNet. To our knowledge, this is one of the largest-scale ImageNet experiments among the self-distillation related studies.

	methods		Resnet-50		WRN50-2	
1	base model		23.87	7.14	21.53	5.91
2	base-loop	t=1	23.73	6.95	21.99	6.11
3		t=2	23.50	6.93	–	
4		t=3	23.36	6.78	–	
5	ini:base	t=1	22.79	6.43	21.17	5.65
6		t=2	22.49	6.27	–	
7		t=3	22.31	6.28		
8	ini:base/2	t=1	22.50	6.25	**20.69**	**5.35**
9		t=2	22.31	6.18	–	
10		t=3	**22.08**	**6.10**	–	
11	Resnet-101†		22.63	6.44	–	
12	WRN-101-2†		–		21.16	5.72

Table 4. ImageNet 224×224 single-crop results on the validation set. GULF2. top-1 and top-5 errors (%).
† The ResNet-101 and WRN-101-2 performances are from the description of the pre-trained torchvision models.

Additional experiments on text Finally, the experiments in this section used image data. Additional experiments using text data are presented in the supplementary material.

4. Discussion

Guided exploration of landscape GULF is an informed/guided exploration of the loss landscape, where the guidance is successively given as interim goals set in the neighborhood of the model at the time, and such guidance is provided by gradient descent in a function space. Another view of this process is an accumulation of successive greedy optimization. Instead of searching the entire space for the ultimate goal of loss minimization at once, guided learning proceeds with repeating a local search, which limits the space to be searched and leads to better generalization. Its benefit is analogous to that of ε-boosting.

GULF1 GULF2 uses the second-order information of loss in the functional gradient step for generating the guide function, and GULF1 does not. GULF2's update rule is equivalent to that of distillation, and GULF1's is not. GULF1 also differs from the logit least square fitting version of distillation. In our experiments (though limited due to our focus on self-distillation study), GULF1 performed as well as GULF2. If this is a general trend, this indicates that inclusion of the second-order information is not particularly helpful. If so, this could be because the second-order information is useful for accelerating optimization, but we would like to proceed slowly to obtain better generalization performance. This motivates further investigation of GULF1 as well as other instantiations of the framework.

Computational cost From a practical viewpoint, a shortcoming of the particular setup tested here (but not the general framework of GULF) is computational cost. Since we used the same learning rate scheduling as regular training in each stage, GULF training with T stages took more than T times longer than regular training. It is conceivable that training in each stage can be shortened without hurting performance since optimization should be easier as a results of aiming at a nearby goal. Schemes that decay the learning rate throughout the training without restarts or hybrid approaches might also be beneficial for reducing computation. Note that Theorem 2.1 does not require each stage to be performed to the optimum. On the other hand, testing (i.e., making predictions) of the models trained with GULF only requires the same cost as regular models. As shown in the ImageNet experiments, a model trained with GULF could perform better than a much larger (and so slow-to-predict) model; in that case, GULF can save the overall computational cost since the cost for making predictions can be significant for practical purposes.

Relation to other methods The proposed method seeks to improve generalization performances in a principled way that limits the searched parameter space. The relation to existing methods for similar purposes is at least two-fold. First, we view that this work gives theoretical insight into related methods such as self-distillation and label smoothing, which we hope can be used to improve them. Second, methods derived from this framework can be used *with* existing techniques that are based on different principles (e.g., weight decay and dropout) for *further* improvements.

Distillation Due to the connection discussed above, our theoretical and empirical analyses of GULF2 provide a new functional gradient view of distillation. Here we discuss a few self-distillation studies from this new viewpoint. (Furlanello et al., 2018) showed that iterative self-distillation

[1] https://pytorch.org/docs/stable/torchvision/models.html

improves performance over the base model. They set α to 0 (in our terminology) and reported that there were no performance gains on CIFAR10. According to our theory, when α goes to 0, the quantity reduced throughout the process is not the α-regularized loss but merely $R(\theta)$. Such an extreme setting might be risky. In *deep mutual learning* (Zhang et al., 2018), multiple models are simultaneously trained by reducing loss and aligning each other's model output. They were surprised by the fact that 'no prior powerful teacher' was necessary. This fact can be explained by our functional gradient view by relating their approach to our ini:random. Finally, the regularization effect of distillation has been noticed (Hinton et al., 2014). Our framework formalized the notion through the functional gradient learning viewpoint.

5. Conclusion

This paper introduces a new framework for guided learning of nonconvex models through successive functional gradient optimization. A convergence analysis is established for the proposed approach, and it is shown that our framework generalizes the popular self-distillation method. Since the guided learning approach learns nonconvex models in restricted search spaces, we obtain better generalization performance than standard training techniques.

Acknowledgements

We thank Professor Cun-Hui Zhang for his support of this research.

References

Anil, R., Pereyra, G., Passos, A., Ormandi, R., Dahl, G. E., and Hinton, G. E. Large scale distributed neural network training through online distillation. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Bubeck, S. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8: 231–358, 2015.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 2001. ISSN 0090-5364.

Furlanello, T., Lipton, Z. C., Tsehannen, M., Itti, L., and Anandkumar, A. Born-again neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016b.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *Proceedings of Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

Johnson, R. and Zhang, T. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

Lan, X., Zhu, X., and Gong, S. Knowledge distillation by on-the-fly native ensemble. In *Adances in Neural Information Processing Systems 31 (NeurIPS 2018)*, 2018.

Loshchilov, I. and Hutter, F. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 1731–1741, 2017a.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017b.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., , and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 2818–2826, 2016.

Tarvainen, A. and Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Adances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

Xie, Q., Dai, Z., Hovy, E., Luong, M.-T., and Le, Q. V. Unsupervised data augmentation for consistency training. *arXiv:1904.12848*, 2019.

Xu, T.-B. and Liu, C.-L. Data-distortion guided self-distillation for deep neural networks. In *Proceedings of The 33rd AAAI Conference on Artificial Intelligence)*, 2019.

Yang, C., Xie, L., Qiao, S., and Yuille, A. Training deep neural networks in generations: A more tolerant teacher educates better students. In *Proceedings of AAAI 2019*, 2019a.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. XLNet: Generalized autoregressive pretraining for language understanding. In *Adances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019b.

Yim, J., Joo, D., Bae, J., and Kim, J. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015.

Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. Deep mutual learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2018.

A. Proofs

In the proofs, we use abbreviated notation by dropping x and y and making θ a subscript, e.g., we write f_θ for $f(\theta; x)$.

A.1. Proof of Proposition 2.1

Proposition 2.1 *When $h(p) = L_y(p)$ that returns loss given prediction p, Algorithm 1 with $\alpha = \gamma$ is equivalent to Algorithm 3 with $\alpha = 1 - (1-\gamma)^m$.*

Proof From Algorithm 1 with $\alpha = \gamma$, we have

$$f_i^* = \arg\min_q \left[D_h(q, f_{i-1}^*) + \gamma \nabla L_y(f_{i-1}^*)^\top q \right]. \tag{15}$$

From $h(\cdot) = L_y(\cdot)$ and (15), we obtain

$$\nabla L_y(f_i^*) = \nabla L_y(f_{i-1}^*) - \gamma \nabla L_y(f_{i-1}^*) = (1-\gamma)\nabla L_y(f_{i-1}^*) \quad \text{for } i = 1, \cdots, m.$$

Since $f_0^* = f_{\theta_t}$, we have

$$\nabla L_y(f_m^*) = (1-\gamma)^m \nabla L_y(f_0^*) = (1-\gamma)^m \nabla L_y(f_{\theta_t}),$$

which implies

$$\begin{aligned} \nabla_{f_\theta} \left[D_h(f_\theta, f_m^*) \right] &= \nabla L_y(f_\theta) - \nabla L_y(f_m^*) = \nabla L_y(f_\theta) - (1-\gamma)^m \nabla L_y(f_{\theta_t}) \\ &= \nabla_{f_\theta} \left[D_{L_y}(f_\theta, f_{\theta_t}) + (1 - (1-\gamma)^m) \nabla L_y(f_{\theta_t})^\top f_\theta \right] \end{aligned}$$

and therefore, $\nabla_\theta \left[D_h(f_\theta, f_m^*) \right] = \nabla_\theta \left[D_{L_y}(f_\theta, f_{\theta_t}) + (1 - (1-\gamma)^m) \nabla L_y(f_{\theta_t})^\top f_\theta \right]$. The rest is trivial. ■

A.2. Proof of Proposition 2.2

Proposition 2.2 *Let y be a vector representation such as a K-dim vector representing K classes. Assume that the gradient of the loss function can be expressed as*

$$\nabla L(f, y) = \nabla L_y(f) = p(f) - y$$

with $p(f)$ not depending on y. Let

$$\begin{aligned} J_t(\theta) &= \left\langle D_{L_y}(f_\theta, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top f_\theta \right\rangle_{(x,y)\in S} \\ J_t'(\theta) &= \left\langle (1-\alpha) L(f_\theta, p(f_{\theta_t})) + \alpha L_y(f_\theta) \right\rangle_{(x,y)\in S} \end{aligned}$$

Then we have

$$J_t(\theta) = J_t'(\theta) + c_t,$$

where c_t is independent of θ. This implies that

$$\arg\min_\theta \left[J_t(\theta) + R(\theta) \right] = \arg\min_\theta \left[J_t'(\theta) + R(\theta) \right].$$

Proof

$$\begin{aligned} \nabla_{f_\theta} \left[D_{L_y}(f_\theta, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top f_\theta \right] &= \nabla L_y(f_\theta) - (1-\alpha) \nabla L_y(f_{\theta_t}) \\ &= (p(f_\theta) - y) - (1-\alpha)(p(f_{\theta_t}) - y) \\ &= (1-\alpha)(p(f_\theta) - p(f_{\theta_t})) + \alpha(p(f_\theta) - y) \\ &= \nabla_{f_\theta} \left[(1-\alpha) L(f_\theta, p(f_{\theta_t})) + \alpha L_y(f_\theta) \right]. \end{aligned}$$

This implies that $\nabla J_t(\theta) = \nabla J_t'(\theta)$. Therefore $J_t(\theta) - J_t'(\theta)$ is independent of θ. ■

A.3. Proof of Theorem 2.1

Theorem 2.1 *In the setting of Algorithm 1 with $m = 1$, assume that there exists $\beta > 0$ such that $D_h(f, f') \geq \beta D_{L_y}(f, f')$ for any f and f', and assume that $\alpha \in (0, \beta]$. Assume also that $Q_t(\theta)$ defined in Algorithm 1 is $1/\eta$ smooth in θ:*

$$\|\nabla Q_t(\theta) - \nabla Q_t(\theta')\| \leq (1/\eta)\|\theta - \theta'\|.$$

Assume that θ_{t+1} is an improvement of θ_t with respect to minimizing Q_t so that

$$Q_t(\theta_{t+1}) \leq Q_t(\tilde{\theta}),$$

where

$$\tilde{\theta} = \theta_t - \eta \nabla Q_t(\theta_t).$$

Then we have

$$\ell_\alpha(\theta_{t+1}) \leq \ell_\alpha(\theta_t) - \frac{\alpha\eta}{2}\|\nabla \ell_\alpha(\theta_t)\|^2.$$

Proof

We first define $\tilde{Q}_t(\theta)$ as follows:

$$\tilde{Q}_t(\theta) := \left\langle D_h(f_\theta, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top f_\theta \right\rangle_{(x,y)\in S} + R(\theta).$$

We can check that $Q_t(\theta) - \tilde{Q}_t(\theta)$ is independent of θ. Therefore optimizing θ with respect to $Q_t(\theta)$ is the same as optimizing θ with respect to $\tilde{Q}_t(\theta)$, and $\nabla Q_t(\theta) = \nabla \tilde{Q}_t(\theta)$.

The smoothness assumption implies that

$$\tilde{Q}_t(\theta - \Delta\theta) \leq \tilde{Q}_t(\theta) - \nabla Q_t(\theta)^\top \Delta\theta + \frac{1}{2\eta}\|\Delta\theta\|^2.$$

Therefore

$$\begin{aligned}
\tilde{Q}_t(\theta_{t+1}) \leq & \tilde{Q}_t(\tilde{\theta}) = \tilde{Q}_t(\theta_t - \eta \nabla Q_t(\theta_t)) \\
\leq & \tilde{Q}_t(\theta_t) - \eta\|\nabla Q_t(\theta_t)\|^2 + \frac{1}{2\eta}\|\eta \nabla Q_t(\theta_t)\|^2 \\
= & \tilde{Q}_t(\theta_t) - \frac{\eta}{2}\|\nabla Q_t(\theta_t)\|^2.
\end{aligned}$$

Note also that

$$\begin{aligned}
\tilde{Q}_t(\theta_{t+1}) - \tilde{Q}_t(\theta_t) \geq & \left\langle \beta D_{L_y}(f_{\theta_{t+1}}, f_{\theta_t}) + \alpha \nabla L_y(f_{\theta_t})^\top (f_{\theta_{t+1}} - f_{\theta_t}) \right\rangle_{(x,y)\in S} + [R(\theta_{t+1}) - R(\theta_t)] \\
= & \left\langle (\beta - \alpha) D_{L_y}(f_{\theta_{t+1}}, f_{\theta_t}) + \alpha L_y(f_{\theta_{t+1}}) - \alpha L_y(f_{\theta_t}) \right\rangle_{(x,y)\in S} + [R(\theta_{t+1}) - R(\theta_t)] \\
\geq & \left\langle \alpha L_y(f_{\theta_{t+1}}) - \alpha L_y(f_{\theta_t}) \right\rangle_{(x,y)\in S} + [R(\theta_{t+1}) - R(\theta_t)] \\
= & \alpha \ell_\alpha(\theta_{t+1}) - \alpha \ell_\alpha(\theta_t).
\end{aligned}$$

The second inequality is due to the non-negativity of the Bregman divergence.

By combining the two inequalities, we obtain

$$\alpha \ell_\alpha(\theta_{t+1}) \leq \alpha \ell_\alpha(\theta_t) - \frac{\eta}{2}\|\nabla Q_t(\theta_t)\|^2.$$

Now, observe that $\nabla Q_t(\theta_t) = \nabla \tilde{Q}_t(\theta_t) = \alpha \nabla \ell_\alpha(\theta_t)$, and we obtain the desired bound. ■

B. On the Empirical Study

In this section, we first provide experimental details and additional figures regarding the experiments reported in the main paper, and then we report additional experiments using text data. Our code is provided at a repository under `github.com/riejohnson`.

B.1. Details of the experiments in the main paper

B.1.1. CIFAR10, CIFAR100, AND SVHN

This section describes the experimental details of all but the ImageNet experiments.

The mini-batch size was set to 128. We used momentum 0.9. The following learning rate scheduling was used: 200K steps with η, 40K steps with 0.1η, and 40K steps with 0.01η. The initial learning rate η was set to 0.1 on CIFAR10/100 and 0.01 on SVHN, following (Zagoruyko & Komodakis, 2016). The weight decay λ was 0.0001 except that it was 0.0005 for (CIFAR100, WRN-28-10) and SVHN.

We used the standard mean/std normalization on all and the standard shift and horizontal flip image augmentation on CIFAR10/100.

We report the median of three runs with three random seeds. The meta-parameters were chosen based on the performance on the development set. All the results were obtained by using only the 'train' portion (shown in Table 1 of the main paper) of the official training set as training data.

For label smoothing, the amount of probability taken away from the true class was chosen from $\{0.1, 0.2, 0.3, 0.4\}$.

To obtain the results reported in Table 2 (with smaller networks), T was fixed to 25 for CIFAR10/100, and 15 for SVHN. α for ini:random was fixed to 0.3. For ini:base, we chose α from $\{0.3, 0.01\}$. We excluded $\alpha = 0.01$ for ini:random, as it takes too long. When dropout was applied in the SVHN experiments, the dropout rate was set to 0.4, following (Zagoruyko & Komodakis, 2016). To obtain the results reported in Table 3 (with larger networks), T was fixed to 1. For GULF2, α was chosen from $\{0.3, 0.01\}$. For GULF1, α was fixed to 0.3, and m (the number of functional gradient steps) was chosen from $\{1, 2, 5\}$. On CIFAR datasets, the choice of α or m did not make much difference, and the chosen values tended to vary among the random seeds. On SVHN, α=0.01 tended to be better when no dropout was used, and 0.3 was better when dropout was used.

To perform random initialization of the parameter for ini:random and the baseline methods, we used Kaiming normal initialization (He et al., 2015), following the previous work.

B.1.2. IMAGENET

Each stage of the training for ImageNet followed the code used for training the pre-trained models provided as part of TorchVision: `https://github.com/pytorch/examples/blob/master/imagenet/main.py`. That is, for both ResNet-50 and WRN-50-2, the learning rate was set to η, 0.1η, and 0.01η for 30 epochs each, i.e., 90 epochs in total, and the initial rate η was set to 0.1. The mini-batch size was set to 256, and the weight decay was set to 0.0001. The momentum was 0.9. α was fixed to 0.5. We used two GPUs for ResNet-50 and four GPUs for WRN-50-2.

We used the standard mean/std normalization and the standard image augmentation for ImageNet – random resizing, cropping and horizontal flip, which is the same data augmentation scheme as used for training the pre-trained models provided as part of TorchVision.

B.2. Additional figures

Figure 5 shows test error (%) in relation to training loss with a small ResNet on CIFAR100. Additional examples of test-loss curves are shown in Figure 6. Figure 7 shows the parameter size $\|\theta_t\|^2$ in relation to training loss, in the settings of Figure 2 in the main paper.

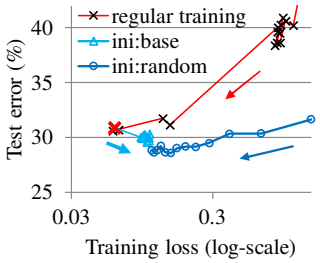
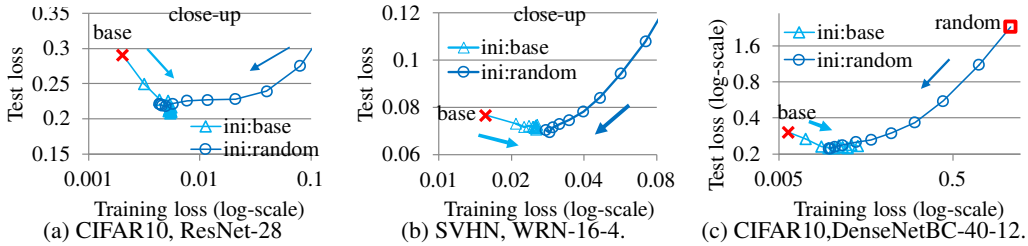
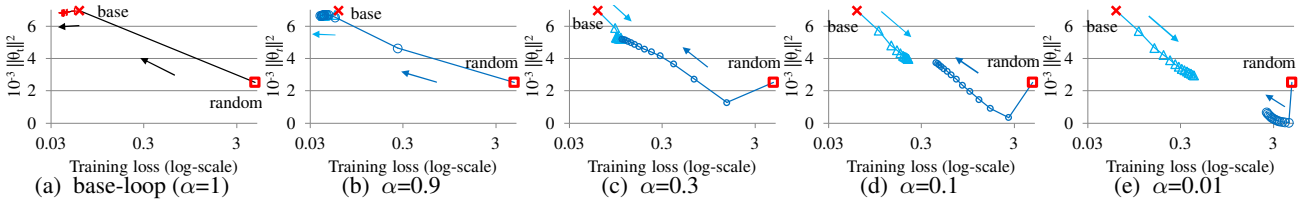


Figure 5. Test error (%) in relation to training loss. The arrows indicate the direction of time flow. GULF2. CIFAR100. ResNet-28.



(a) CIFAR10, ResNet-28 (b) SVHN, WRN-16-4. (c) CIFAR10,DenseNetBC-40-12.

Figure 6. Additional examples of test loss curves of GULF2. The arrows indicate the direction of time flow.



(a) base-loop (α=1) (b) α=0.9 (c) α=0.3 (d) α=0.1 (e) α=0.01

Figure 7. Parameter size $\|\theta_t\|^2$ of ini:base('△') and ini:random('○'). with five values of α (becoming smaller from left to right), in relation to training loss. GULF2. T=25. CIFAR100. ResNet-28. Matching figures with Figure 2. As α becomes smaller, the (potential) meeting point shifts further away from the base model. The left-most figure is base-loop, which is equivalent to α=1. The arrows indicate the direction of time flow.

B.3. Additional experiments on text data

We tested GULF on sentiment classification to predict whether reviews are positive or negative, using the polarized Yelp dataset (#train: 560K, #test: 38K) (Zhang et al., 2015). The best-performing models on this task are transformers pre-trained with language modeling on large and general text data such as Bert (Devlin et al., 2019) and XLnet (Yang et al., 2019b). However, these models are generally large and time-consuming to train using a GPU (i.e., without TPUs used in the original work). Therefore, instead, we used the deep pyramid convolutional neural network (DPCNN) (Johnson & Zhang, 2017) as our base model. In these experiments, we used GULF2.

Table 5 shows the test error results in five settings. The last three use relatively small training sets of 45K data points and validation sets of 5K data points, randomly chosen from the original training set, while the first two use the entire training set (560K data points) except for 5K data points held out for validation (meta-parameter tuning). DPCNNs optionally take additional features produced by embeddings of text regions that are trained with unlabeled data, similar to language modeling. Cases 1 and 3 exploited this option, training embeddings using the entire training set as unlabeled data; B.3.1 below provides the details. As in the image experiments, we used the cross entropy loss with softmax except for Case 5, where the quadratic hinge loss $L_y(f) = \max(0, 1 - yf)^2$ for $y \in \{-1, 1\}$ was used. This serves as an example of extending

Case#		1	2	3	4	5
Data		large-Yelp		small-Yelp		
Embedding learning?		Yes	No	Yes	No	
Loss function		cross-entropy				†
baselines	base model	2.81	2.98	3.80	5.43	5.32
	base-loop	2.63	2.88	3.90	5.43	5.34
	w/ dropout	2.70	2.95	3.90	5.34	5.35
GULF	ini:random	**2.34**	2.72	**3.70**	5.06	5.00
	ini:base	2.38	**2.70**	3.77	5.15	4.98
	ini:base/2	2.43	2.74	3.73	**4.99**	**4.96**

Table 5. Test error (%) on sentiment classification. Median of 3 runs. 7-block 250-dim DPCNN (10M parameters). † Squared hinge loss.

self-distillation (formulated specifically with the cross-entropy loss) to general loss functions.

In all the five settings, GULF achieves better test errors than the baseline methods, which shows the effectiveness of our approach in these settings. On this task, dropout turned out to be not very effective, which is, however, a reminder that the effectiveness of regularization methods can be data-dependent in general.

Case#		1	2		
		LM-like prep?		Runtime (sec/K)	Text for prep (GB)
		Yes	No		
(J & Z, 2017)	DPCNN	*2.64*	*3.30*	0.1	0.4
This work	Table 5 best	2.34	2.70	0.1	0.4
	Ensemble	**2.18**	**2.46**	0.9	0.4
(Devlin et al., 2019)	Bert base	2.25	6.19	5.7	13
	Bert large	*1.89*	–	17.9	13
(Yang et al., 2019b)	XLnet base	1.92	4.51	17.2	13
	XLnet large	*1.55*	–	40.5	126

Table 6. GULF ensemble results on Yelp in comparison with previous models. Test error (%) with or without embedding learning (DPCNN) or language modeling-based pre-training (Bert and XLnet), respectively, corresponding to Cases 1 & 2 of Table 5. Runtime: real time in seconds for labeling 1K instances using a single GPU with 11GB device memory, measured in the setting of Case 1; the average of 3 runs. The last column shows amounts of text data in giga bytes used for pre-training or embedding learning in Case 1. The test errors in *italics* were copied from the respective publications except that the Bert-large test error is from (Xie et al., 2019); other test errors and runtime were obtained by our experiments. Our ensemble test error results are in bold.

It is known that performance can be improved by making an ensemble of models from different stages of self-distillation, e.g., (Furlanello et al., 2018). In Table 6, we report ensemble performances of DPCNNs trained with GULF, in comparison with the previous best models. Test errors with and without embedding learning (or language modeling-based pre-training for Bert and XLnet) are shown, corresponding to Cases 1 and 2 in Table 5. The ensemble results were obtained by adding after applying softmax the output values of 20 DPCNNs (or 10 in Case 2) of last 5 stages of GULF training with different training options; details are provided in B.3.1.

With embedding learning, the ensemble of DPCNNs trained with GULF achieved test error 2.18%, which slightly beats 2.25% of pre-trained Bert-base, while testing (i.e., making predictions) of this ensemble is more than 6 times faster than Bert-base, as shown in the 'Runtime' column. (Note, however, that runtime depends on implementation and hardware/software configurations.) That is, using GULF, we were able to obtain a classifier that is as accurate as and much faster than a pre-trained transformer.

(Yang et al., 2019b) and (Xie et al., 2019) report 1.55% and 1.89% using a pre-trained large transformer, XLnet-large and Bert-large, respectively. We observe that the runtime and the amounts of text used for pre-training (the last two columns) indicate that their high accuracies come with steep cost at every step: pre-training, fine-tuning, and testing. Compared with them, an ensemble of GULF-trained DPCNNs is a much lighter-weight solution with an appreciable accuracy. Also, our ensemble without embedding learning outperforms Bert-base and XLnet-base without pre-training, with relatively large differences (Case 2). A few attempts of training Bert-large and XLnet-large from scratch also resulted in underperforming DPCNNs, but we omit the results as we found it infeasible to complete meta-parameter tuning in reasonable time.

On the other hand, it is plausible that the accuracy of the high-performance pre-trained transformers can be further improved by applying GULF to their fine-tuning, which would further push the state of the art. Though currently precluded by our computational constraints, this may be worth investigating in the future.

B.3.1. DETAILS OF THE TEXT EXPERIMENTS

Embedding learning It was shown in (Johnson & Zhang, 2017) that classification accuracy can be improved by training an embedding of small text regions (e.g., 3 consecutive words) for predicting neighboring text regions ('target regions') on unlabeled data (similar to language modeling) and then using the learned embedding function to produce additional features for the classifier. In this work, we trained the following two types of models with respect to use of embedding learning.

- Type-0 did not use any additional features from embedding learning.
- Type-1 used additional features from the following two types of embedding simultaneously:
  - the embedding of 3-word regions as a function of a bag of words to a 250-dim vector, and
  - the embedding of 5-word regions as a function of a bag of word {1,2,3}-grams to a 250-dim vector.

Embedding training was done using the entire training set (560K reviews, 391MB) as unlabeled data disregarding the labels.

It is worth mentioning that our implementation of embedding learning differs from the original DPCNN work (Johnson & Zhang, 2017), as a result of pursuing an efficient implementation in pyTorch (the original implementation was in C++). The original work used the bag-of-word representation for target regions (to be predicted) and minimized squared error with negative sampling. In this work we minimized the log loss without sampling where the target probability was set by equally distributing the probability mass among the words in the target regions.

Table 5 Optimization was done by SGD. The learning rate scheduling of the base model and each stage of base-loop and GULF was fixed to 9 epochs with the initial learning rate η followed by 1 epoch with 0.1η. The mini-batch size was 32 for small training data and 128 for large training data. We chose the weight decay parameter from {1e-4, 2e-4, 5e-4, 1e-3} and the initial learning rate from {0.25, 0.1, 0.05}, using the validation data, except that for GULF on the large training data, we simply used the values chosen for the base model, which were weight decay 1e-4 and learning rate 0.1 (with embedding learning) and 0.25 (without embedding learning).

For GULF, we chose the number of stages T from {1,2,...,25} and α from {0.3, 0.5}, using the validation data. $\alpha = 0.5$ was chosen in most cases.

Table 6 The ensemble performances were obtained by combining

- 20 DPCNNs ($T \in \{21, 22, \ldots, 25\} \times$ {ini:random, ini:base} $\times$ {Type-0, Type-1}) in Case 1, and
- 10 DPCNNs ($T \in \{21, 22, \ldots, 25\} \times$ {ini:random, ini:base} $\times$ {Type-0}) in Case 2.

To make an ensemble, the model output values were added after softmax.

Transformers The Bert and XLnet experiments were done using HuggingFace's Transformers[2] in pyTorch. Following the original work, optimization was done by Adam with linear decay of learning rate. For enabling and speeding up training using a GPU, we combined the techniques of gradient accumulation and variable-sized mini-batches (for improving parallelization) so that weights were updated after obtaining the gradients from approximately 128 data points. 128 was chosen, following the original work. To measure runtime of transformer testing, we used variable-sized mini-batches for speed-up by improving the parallelism on a GPU.

[2] https://huggingface.co/transformers/