Deep regularization and direct training of the inner layers of Neural Networks with Kernel Flows

Gene Ryan Yoo, Houman Owhadi

PII: S0167-2789(21)00110-X
DOI: https://doi.org/10.1016/j.physd.2021.132952
Reference: PHYSD 132952

To appear in: *Physica D*

Received date : 31 December 2020
Revised date : 17 March 2021
Accepted date : 13 April 2021

Deep regularization and direct training of the inner layers of Neural Networks with Kernel Flows

Gene Ryan Yoo[*] Houman Owhadi[†]

March 5, 2021

Abstract

We introduce a new regularization method for Artificial Neural Networks (ANNs) based on the Kernel Flow (KF) algorithm. The algorithm was introduced in [16] as a method for kernel selection in regression/kriging based on the minimization of the loss of accuracy incurred by halving the number of interpolation points in random batches of the dataset. Writing $f_\theta(x) = \big(f^{(n)}_{\theta_n} \circ f^{(n-1)}_{\theta_{n-1}} \circ \cdots \circ f^{(1)}_{\theta_1}\big)(x)$ for the functional representation of compositional structure of the ANN (where θ_i are the weights and biases of the layer i), the inner layers outputs $h^{(i)}(x) = \big(f^{(i)}_{\theta_i} \circ f^{(i-1)}_{\theta_{i-1}} \circ \cdots \circ f^{(1)}_{\theta_1}\big)(x)$ define a hierarchy of feature maps and a hierarchy of kernels $k^{(i)}(x,x') = \exp(-\gamma_i \|h^{(i)}(x) - h^{(i)}(x')\|_2^2)$. When combined with a batch of the dataset, these kernels produce KF losses $e_2^{(i)}$ (defined as the L^2 regression error incurred by using a random half of the batch to predict the other half) depending on the parameters of the inner layers $\theta_1, \ldots, \theta_i$ (and γ_i). The proposed method simply consists of aggregating (as a weighted sum) a subset of these KF losses with a classical output loss (e.g., cross-entropy). We test the proposed method on Convolutional Neural Networks (CNNs) and Wide Residual Networks (WRNs) without alteration of their structure nor their output classifier and report reduced test errors, decreased generalization gaps, and increased robustness to distribution shift without a significant increase in computational complexity relative to standard CNN and WRN training (with Drop Out and Batch Normalization). We suspect that these results might be explained by the fact that while conventional training only employs a linear functional (a generalized moment) of the empirical distribution defined by the dataset and can be prone to trapping in the Neural Tangent Kernel regime (under over-parameterizations), the proposed loss function (defined as a nonlinear functional of the empirical distribution) effectively trains the underlying kernel defined by the CNN beyond regressing the data with that kernel.

1 Introduction

Despite the industrial successes of Deep Learning, training ANNs is oftentimes compared to an art requiring careful tuning. Indeed, writing (X_i, Y_i) for the input/output

[*]Caltech, MC 253-37, Pasadena, CA 91125, USA, gyoo@caltech.edu

[†]Caltech, MC 9-94, Pasadena, CA 91125, USA, owhadi@caltech.edu

training data points, θ for the parameters of the ANN, $f_\theta(x)$ for its output, identifying θ by minimizing an empirical risk taking the form of an average error between $f(X_i)$ and Y_i is oftentimes not sufficient to obtain a desirable result. This paper proposes a principled regularization approach motivated by the observation that regressors obtained from ANNs are essential ridge regressors (kernel interpolants) with a kernel learned from data.

While current kernel perspectives for regularizing deep neural networks [6, 34, 31] are based on the principle of regularizing a fixed given kernel, our approach is based on learning a (new) kernel from data.

1.1 Connections with dynamical systems

[18] shows that the kernel learned by residual neural networks [10] (ResNet) (with L^2 regularization on weights and biases) is a warping kernel [24, 21, 25] of the form

$$K^v := K(\phi^v(x), \phi^v(x')), \tag{1.1}$$

where K is a base kernel and $\phi^v(x)$ is a warping of the space learned from data. More precisely, the regressor obtained from ResNet is of the form $f \circ \phi^v(x, 1)$ where f is the minimizer of[1]

$$\min_f \lambda \|f\|^2_{K^v} + \|f \circ \phi^v(X, 1) - Y\|^2_{\mathcal{Y}^N} \tag{1.2}$$

where $\|\cdot\|_{K^v}$ is the RKHS norm defined by the kernel (1.1), $\phi^v(x,t)$ is the flow map of v defined as the solution of

$$\begin{cases} \dot\phi(x,t) = v\big(\phi(x,t), t\big) & \text{for } (x,t) \in \mathcal{X} \times [0,1] \\ \phi(x,0) = x & \text{for } x \in \mathcal{X}\,. \end{cases} \tag{1.3}$$

and v is a minimizer of (writing $\|\cdot\|_\Gamma$ for the RKHS norm associated with a given kernel Γ) of

$$\min_{v,f} \frac{\nu}{2}\int_0^1 \|v(\cdot,t)\|^2_\Gamma\, dt + \lambda\|f\|^2_K + \|f \circ \phi^v(X,1) - Y\|^2_{\mathcal{Y}^N}\,. \tag{1.4}$$

Moreover [18] shows that a minimizer of (1.4) admits the representation

$$\dot\phi^v(x,t) = \Gamma(\phi^v(x,t), q)p\,, \tag{1.5}$$

where the position and momentum variables (q,p) are in $\mathcal{X}^N \times \mathcal{X}^N$ (writing $\mathcal{X}$ for the input space), started from $q(0) = X$, and following the dynamic defined by the Hamiltonian

$$\mathfrak{H}(q,p) = \frac{1}{2}p^T\Gamma(q,q)p\,, \tag{1.6}$$

i.e.,

$$\begin{cases} \dot q = \Gamma(q,q)p \\ \dot p = -\partial_q\big(\frac{1}{2}p^T\Gamma(q,q)p\big)\,, \end{cases} \text{with initial value } (q(0) = X, p(0))\,. \tag{1.7}$$

[1]Writing $\mathcal{Y}$ for the output space, N for the number of data points, $\|\cdot\|^N_{\mathcal{Y}}$ for the product norm on $\mathcal{Y}^N$, X and Y for the N-vectors whose entries are the input/output data points and $f \circ \phi^v(X,1)$ for the N-vector with entries $f \circ \phi^v(X_i, 1)$.

1.2 MAP estimation vs cross-validation

While the variational formulation (1.4) leading to the identification of f and v is that of a MAP estimator associated with the composition of residual Gaussian processes defined Γ, and K [18], warping kernels of the form (1.5) were also recently introduced in [16] and trained using a variant of cross-validation (described in Sec. 2). Recall that while MAP and MLE are optimal when the model is well-specified, cross-validation is near-optimal and has some degree of robustness to misspecification [5, 7]. Our proposed method is based on Kernel Flows and can be interpreted as replacing MAP estimation (and the implicit simulation of a dynamical system akin to (1.7)) by cross-validation in the training of the inner layers of the ANN. To describe this method, we will first provide a reminder on Kernel Flows.

2 A reminder on Kernel Flows

Kernel Flows were introduced in [16] as a method for kernel selection/design in Kriging/Gaussian Process Regression (GPR) (see also [7] for a rigorous consistency analysis and [9] for an application to the learning of dynamical systems from data). As a reminder on KFs, consider the problem of approximating an unknown function $u^\dagger$ mapping $\mathcal{X}$ to $\mathbb{R}$ based on the input/output dataset $(x_i, y_i)_{1\leqslant i\leqslant N}$ ($u^\dagger(x_i) = y_i$). Any non-degenerate (positive-definite) kernel $K(x, x')$ can be used to approximate $u^\dagger$ with the interpolant

$$u(x) = K(x, X)K(X, X)^{-1}Y\,, \tag{2.1}$$

writing $Y := (y_1, \ldots, y_N)^T$, $X := (x_1, \ldots, x_N)$, $K(X, X)$ for the $N \times N$ Gram matrix $K(x_i, x_i)$ and $K(x, X)$ for the N dimensional vector with entries $K(x, x_i)$. The kernel selection problem concerns the identification of a good kernel for performing this interpolation. The KF approach to this problem is to simply use the loss of accuracy incurred by removing half of the dataset as a loss of kernel selection. The application of this process to minibatches results in a loss that is randomized by both (1) the selection of the minibatch (2) the half sub-sampling of the minibatch. An iterated steepest descent minimization of this loss then results in stochastic gradient descent algorithm (where the minibatch and its half-subset are re-sampled at each step). Given a family of kernels $K_\theta(x, x')$ parameterized by θ, the resulting algorithm can then be described as follows: (1) Select random subvectors X^b and Y^b of X and Y (through uniform sampling without replacement in the index set $\{1, \ldots, N\}$) (2) Select random subvectors X^c and Y^c of X^b and Y^b (by selecting, at random, uniformly and without replacement, half of the indices defining X^b) (3) Let $\rho(\theta, X^b, Y^b, X^c, Y^c)$ be the squared relative error (in the RKHS norm $\|\cdot\|_{K_\theta}$ defined by K_θ) between the interpolants u^b and u^c obtained from the two nested subsets of the dataset and the kernel K_θ, i.e.[2]

$$\rho(\theta, X^b, Y^b, X^c, Y^c) := 1 - \frac{Y^{c,T} K_\theta(X^c, X^c)^{-1} Y_c}{Y^{c,T} K_\theta(X^b, X^b)^{-1} Y^b}\,. \tag{2.2}$$

(4) evolve θ in the gradient descent direction of ρ, i.e. $\theta \leftarrow \theta - \delta \nabla_\theta \rho$ (5) repeat.

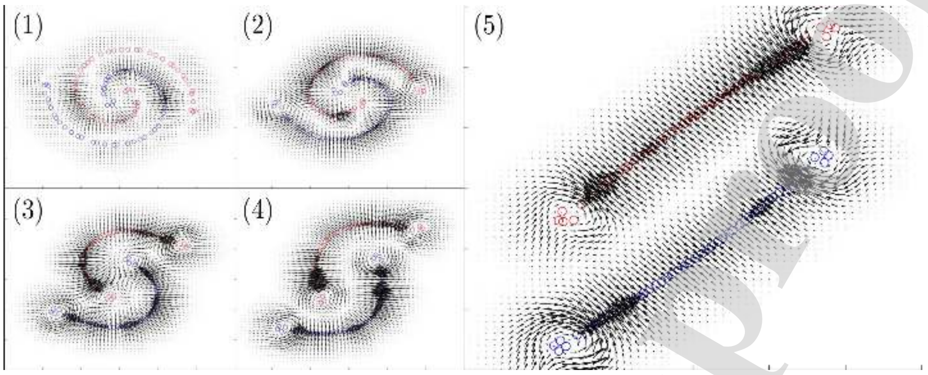

Figure 1: [16, Fig. 13]. Application of the original Kernel Flow approach to the selection of a kernel $K_F(x, x') = \exp(-\gamma \|F(x) - F(x')\|^2)$ parameterised by a deformation $F : \mathbb{R}^2 \to \mathbb{R}^2$ of the input space ($\mathcal{X} = \mathbb{R}^2$). The dataset is the swissroll cheesecake (red points have labels $+1$ and blue points have labels -1). Subfigures (1-5) show the deformed dataset $F_n(X)$ (mapped points $(F_n(x_i))_{1 \leqslant i \leqslant N}$ (dots)) for increasing values of n along with the gradient $-\nabla_F \rho$ averaged over 300 steps (deformation gradient $10(F_{n+300}(x) - F_n(x))/300$ (arrows)). Note that n large enough the two classes become linearly separable under the deformation learned by KF.

Example. Fig. 1 shows an application of the proposed approach to the selection of a kernel $K_F(x, x') = \exp(-\gamma \|F(x) - F(x')\|^2)$ parameterised by a deformation $F : \mathbb{R}^2 \to \mathbb{R}^2$ of the input space ($\mathcal{X} = \mathbb{R}^2$).

The l_2-norm variant. In this paper we will consider[3] the l_2-norm variant of the KF algorithm (introduced in [16, Sec. 10]) in which the instantaneous loss ρ in (2.2) is replaced by the error (let $\|\cdot\|_2$ be the Euclidean l_2 norm) $e_2 := \|Y^b - u^c(X^b)\|_2^2$ of u^c in predicting the labels Y^b, i.e.

$$e_2(\theta, X^b, Y^b, X^c, Y^c) := \|Y^b - K_\theta(X^b, X^c) K_\theta(X^c, X^c)^{-1} Y^c\|_2^2\,. \tag{2.3}$$

[2] $\rho := \|u^b - u^c\|_{K_\theta}^2 / \|u^b\|_{K_\theta}^2$, with $u^b(x) = K_\theta(x, X^b) K_\theta(X^b, X^b)^{-1} Y^b$ and $u^c(x) = K_\theta(x, X^c) K_\theta(X^c, X^c)^{-1} Y^c$, and ρ admits [15, Prop. 13.29] the representation (2.2) enabling its computation. This representation (2.2) and its optimality in the setting of numerical homogenization/aproaximation [15] and some of the motivations for its utilization in the original method [16].

[3] Using (2.2) or (2.3) both showed improvements as regularizers, however (2.3) was slightly better.

3 Kernel Flow regularization of Neural Networks

Write

$$f_\theta(x) = \big(f^{(n)}_{\theta_n} \circ f^{(n-1)}_{\theta_{n-1}} \circ \cdots \circ f^{(1)}_{\theta_1}\big)(x) \tag{3.1}$$

for the compositional structure of an artificial neural network (ANN) with input x and n layers $f^{(i)}_{\theta_i}(z) = \phi(W_i z + b_i)$ parameterized by the weights and biases $\theta_i := (W_i, b_i)$, $\theta := \{\theta_1, \ldots, \theta_n\}$. We will use ReLU for the non-linearity ϕ in our experiments. For $i \in \{1, \ldots, n-1\}$ let $h^{(i)}(x)$ be the output of the i-th (inner) layer, i.e.

$$h^{(i)}_\theta(x) := \big(f^{(i)}_{\theta_i} \circ f^{(i-1)}_{\theta_{i-1}} \circ \cdots \circ f^{(1)}_{\theta_1}\big)(x)\,, \tag{3.2}$$

and let $h_\theta(x) := (h^{(1)}_\theta(x), \ldots, h^{(n-1)}_\theta(x))$ be the $(n-1)$-ordered tuple representing all inner layer outputs. Let $k_\gamma(\cdot,\cdot)$ be a family of kernels parameterized by γ and let $K_{\gamma,\theta}$ be the family of kernels parameterized by γ and θ defined by

$$K_{\gamma,\theta}(x, x') = k_\gamma(h_\theta(x), h_\theta(x'))\,. \tag{3.3}$$

Given the random mini-batch (X^b, Y^b) let $\mathcal{L}_{\text{c-e}}(f_\theta(X^b), Y^b) := \sum_i \mathcal{L}_{\text{c-e}}(f_\theta(X^b_i), Y^b_i)$ be the cross-entropy loss associated with that mini-batch. Given the (randomly sub-sampled) half sub-batch (X^c, Y^c), let $\mathcal{L}_{\text{KF}}(\gamma, \theta, X^b, Y^b, X^c, Y^c)$ be the loss function (with hyper-parameter $\lambda \geqslant 0$) defined by

$$\mathcal{L}_{\text{KF}} := \lambda \|Y^b - K_{\gamma,\theta}(X^b, X^c) K_{\gamma,\theta}(X^c, X^c)^{-1} Y^c\|_2^2 + \mathcal{L}_{\text{c-e}}(f_\theta(X^b), Y^b)\,. \tag{3.4}$$

Our proposed KF-regularization approach is then to train the parameters θ of the network f_θ via the steepest descent $(\gamma, \theta) \leftarrow (\gamma, \theta) - \delta \nabla_{\gamma,\theta} \mathcal{L}_{\text{KF}}$. Note that this algorithm (1) is randomized through both the sampling of the minibatch and its subsampling (2) adapts both θ and γ (since the KF loss term depends on both θ and γ) (3) simultaneously trains the accuracy of the output via the cross-entropy term[4] and the generalization properties of the feature maps defined by the inner layers via the KF loss term. Furthermore while the cross-entropy term is a linear functional of the empirical distribution $\frac{1}{N_b}\sum_i \delta_{(X^b_i, Y^b_i)}$ defined by the mini-batch (writing N_b for the number of indices contained in the mini-batch), the KF loss term is non-linear. While $K_{\gamma,\theta}$ may depend on the output of all the inner layers, for the sake of simplicity in our numerical experiments, we have restricted its dependence to the output of only one inner layer or used a weighted sum of such terms.

4 Numerical experiments

We will now use the proposed KF-regularization method to train a simple Convolutional Neural Network (CNN) on MNIST and Wide Residual Networks (WRN) [35] on CIFAR-10 and CIFAR-100. Our goal is to test the proposed approach and compare its performance with popular ones (Batch Normalization and Drop Out).

[4]The training of which is relatively cheap when using typical batch sizes of around 100.

Layer Type	Number of filters	Filter size	Padding	Output shape
Input layer				$28 \times 28 \times 1$
Convolutional layer 1, ReLU	150	3×3	Valid	$26 \times 26 \times 150$
Convolutional layer 2, ReLU	150	3×3	Valid	$24 \times 24 \times 150$
Convolutional layer 3, ReLU Max Pool	150	5×5 2×2	Same	$24 \times 24 \times 150$ $12 \times 12 \times 150$
Convolutional layer 4, ReLU	300	3×3	Valid	$10 \times 10 \times 300$
Convolutional layer 5, ReLU	300	3×3	Valid	$8 \times 8 \times 300$
Convolutional layer 6, ReLU Max Pool Average Pool	300	5×5 2×2 4×4	Same	$8 \times 8 \times 300$ $4 \times 4 \times 300$ 300
Dense layer 1, ReLU				1200
Dense layer 2, ReLU				300
Dense layer 3				10
Softmax Output layer				10

Table 1: The architecture of the CNN used in KF-regularization experiments is charted. Convolutional layers are divided by horizontal lines. The middle block shows layer specifics, and the shapes of the outputs of each layer is on the right.

4.1 Kernel Flow regularization on MNIST

We consider a Convolutional Neural Network (CNN) with six convolutional layers and three fully connected layers, as charted in Table 1 (this CNN is a variant of a CNN presented in [3] with code used from [2]). Convolutional layers all have stride one in this network with the number of convolutional channels and the convolutional kernel size in the second and third columns from the left. "Valid" padding implies no 0-padding at the boundaries of the image while "same" 0-pads images to obtain convolutional outputs with the same sizes as the inputs. The "Max Pool" layers downsample their inputs by reducing each 2×2 square to their maximum values. The "Average Pool" layer in the final convolutional layer takes a simple mean over each channel. The final three layers are fully connected, each with outputs listed on the right column. All convolutional and dense layers include trainable biases. Using notations from the previous section, the outputs of the convolutional layers, which include ReLU and pooling, are $h^{(1)}(x)$ to $h^{(6)}(x)$ with output shapes described in the left column. The dense layers outputs are $h^{(7)}(x)$ to $h^{(9)}(x)$. We do not pre-process the data and, when employed, the data augmentation step, in this context, passes the original MNIST image to the network with probability $\frac{1}{3}$, applies an elastic deformation [26] with probability $\frac{1}{3}$, and a small random translation, rotation, and shear with probability $\frac{1}{3}$. The learning rate, as selected by validation, begins at 10^{-3} and smoothly exponentially decreases to 10^{-7} while training over 20 epochs.

4.1.1 Comparisons of dropout and KF-regularization

The first experiment presents results obtained by training the CNN with the architecture given in Table 1 and (1) Batch Normalization (BN) [11] (2) BN, and KF-regularization

(3) BN and dropout (DO) [27] (4) BN, KF-regularization, and DO. We use the same dropout structure as in [3], and use a rate of 0.3, as selected with validation.

Training Method	**Original MNIST**	**Data augmented**	**QMNIST**
BN only	$0.395 \pm 0.030\%$	$0.302 \pm 0.026\%$	$0.389 \pm 0.014\%$
BN+KF	$0.300 \pm 0.024\%$	$0.281 \pm 0.033\%$	$0.341 \pm 0.013\%$
BN+DO	$0.363 \pm 0.028\%$	$0.314 \pm 0.024\%$	$0.400 \pm 0.015\%$
BN+KF+DO	$0.296 \pm 0.023\%$	$0.287 \pm 0.022\%$	$0.344 \pm 0.015\%$

Table 2: A comparison of the average and standard deviation of testing errors each over 20 runs for networks. The first data column on the left shows networks trained and tested on original MNIST data. The middle is trained using data augmentation and uses original MNIST testing data. The right column shows the same data augmented trained network, but uses QMNIST testing data [33].

We present a KF-regularization experiment using the following Gaussian kernel on the final convolutional layer $h^{(6)}(x) \in \mathbb{R}^{300}$:

$$K^{(6)}_{\gamma_6,\theta}(x,x') = k^{(6)}_{\gamma_6}(h^{(6)}(x), h^{(6)}(x')) = e^{-\gamma_6 \|h^{(6)}(x)-h^{(6)}(x')\|^2} . \tag{4.1}$$

We optimize the loss function in (3.4) with kernel $K^{(6)}_{\gamma_6}$ over the parameters θ and γ_6. Specifically, given the random mini-batch (X^b, Y^b) and the (randomly sub-sampled) half sub-batch (X^c, Y^c), we evolve θ and γ_6 in the steepest descent direction of the loss

$$\mathcal{L}_{\text{KF}} = \lambda^{(6)} \|Y^b - K^{(6)}_{\gamma_6,\theta}(X^b, X^c) K^{(6)}_{\gamma_6,\theta}(X^c, X^c)^{-1} Y^c\|_2^2 + \mathcal{L}_{\text{c-e}}(f_\theta(X^b), Y^b) . \tag{4.2}$$

The second experiment is a slight variant where we use both $K^{(6)}$ and

$$K^{(3)}_{\gamma_3,\theta}(x,x') = k^{(3)}_{\gamma_3}(h^{(3)}(x), h^{(3)}(x')) = e^{-\gamma_3 \|a(h^{(3)}(x))-a(h^{(3)}(x'))\|^2} , \tag{4.3}$$

where a is a 12×12 average pooling reducing each channel to a single point.

The comparison between the dropout and KF-regularization training methods, as well as their combination, is made in Table 2. KF-regularization and the network architecture were inspired by the work in [16, Sec. 10] (the GPR estimator on the final convolutional output space is here replaced by a fully connected network to minimize computational complexity). On a 12GB NVIDIA GeForce GTX TITAN X graphics card, training one network with BN+DO (20 epochs) takes 1605s to run, compared with 1629s for BN+KF+DO. Furthermore, this KF-regularization framework has another advantage of being flexible, both allowing the control of generalization properties of multiple layers of the network simultaneously and being able to be used concurrently with dropout.

For each of the training methods, we experiment with using original MNIST training and testing data, augmenting the MNIST training set and testing on the original data, and finally training on the augmented set, but testing on QMNIST, which is re-sampled MNIST test data [33]. These three regimes are presented in the data columns of Table 2 from left to right. The difference between the original data augmented and QMNIST testing errors quantifies the effect of the distributional shift of the testing data [23]. This effect is observed to be reduced when using KF-regularized trained networks, which suggests some degree of robustness to distributional shift.

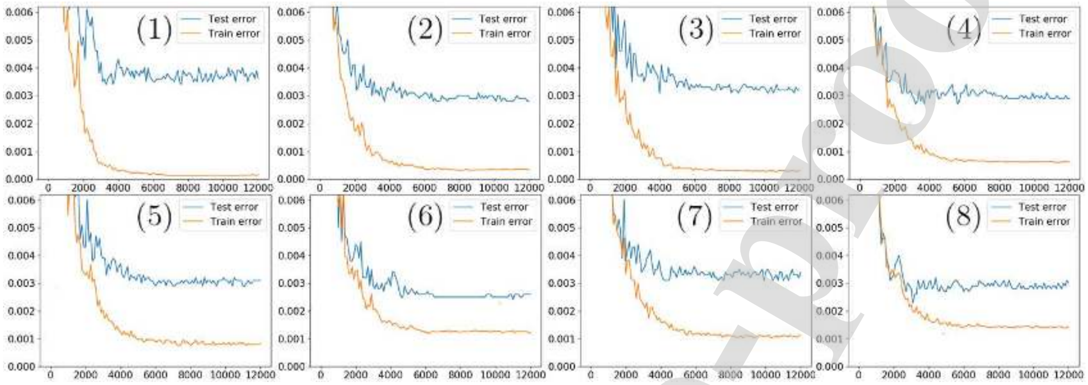

Figure 2: Training and testing errors are plotted over single runs trained with original data using (1) BN only (2) BN+KF (3) BN+DO (4) BN+KF+DO. Data augmented trained network errors are shown using (5) BN only (6) BN+KF (7) BN+DO (8) BN+KF+DO.

The training and testing errors of single runs of networks trained with BN only, BN+DO, BN+KF, and BN+KF+DO are plotted in Fig. 2. Observe that the generalization gap (the gap between the training and testing errors) decreases with the use of dropout, and that decrease is even more pronounced in the experiments with KF-regularization. We observe similar findings on networks trained using data augmentation, albeit less pronounced. We finally examine the KF-regularization component of the loss function as in equation (4.2). This KF loss, $\|Y^b - K^{(6)}_{\gamma_6,\theta}(X^b, X^c)K^{(6)}_{\gamma_6,\theta}(X^c, X^c)^{-1}Y^c\|_2^2$, is computed for batch normalization, dropout, and KF-regularized training in Fig. 3. Furthermore, we compare the average pairwise distance $\|h^{(6)}(x) - h^{(6)}(x')\|$ for x and x' when both in the same class and when they are in different classes. We observe in the figure that both BN+KF and BN+KF+DO reduce the KF loss and increase the ratio of inter-class and in-class pairwise distances within each batch. The class-dependent clustering within hidden layer outputs highlights the difference between traditional training techniques and KF-regularization.

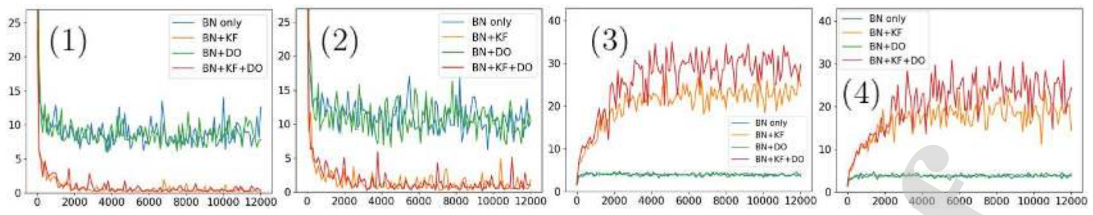

Figure 3: Single run with each of BN only, BN+KF, BN+DO, and BN+KF+DO training methods plotting (1) 6th layer KF-loss using the original MNIST training set (2) 6th layer KF-loss using an augmented training set (3) ratio of mean inter-class and in-class distances of 6th layer outputs using the original training set (4) ratio of mean inter-class and in-class distances of 6th layer outputs using an augmented set.

Layer/Block name	**Number of filters**	**Filter size**	**Number of residual layers**	**Output shape**
Input layer				$32 \times 32 \times 3$
Convolutional block 1	16	3×3	1	$32 \times 32 \times 16$
Convolutional block 2	$16k$	3×3	N	$32 \times 32 \times 16k$
Convolutional block 3 Max Pool	$32k$	3×3 2×2	N	 $16 \times 16 \times 32k$
Convolutional block 4 Max Pool Average Pool	$64k$	3×3 2×2 8×8	N	 $8 \times 8 \times 64k$ $64k$
Dense layer				$64k$
Softmax Output layer				10

Table 3: The architecture of the WRN used in KF-regularization experiments with CIFAR input images. Convolutional blocks are divided with horizontal lines. The middle portion shows block specifics such as filter width and depth in each block and the shapes of the outputs of each layer is on the right. Note that max pooling occurs within the last residual layer of each block.

Neural collapse. The neural collapse recently featured in [20] is closely related to the inner layers class clustering presented in Fig. 3 (and also observed in [16]).

4.2 Kernel Flow regularization on CIFAR

We now consider the Wide Residual Network (WRN) structure described in [35, Table 1] with the addition of a dense layer. For convenience, we show this architecture in Table 3. Note that there are four convolutional blocks, each with a certain number of residual layers, which are as described in [35, Fig. 1c,d] for BN and BN+DO training, respectively. Each layer consists of two convolutional blocks, with dropout applied between the blocks in dropout training, added to an identity mapping from the input of the layer. In our dropout experiments, we drop each neuron in the network with probability

0.3, as selected with cross-validation in [35]. Note that k and N are hyper-parameters of the WRN architecture governing width and depth, respectively, and a network with such k, N is written WRN-k-N. In these presented WRN experiments, we use data augmentation where training images are randomly translated and horizontally flipped. In our implementations, we have modified the code from [1] (which uses TensorFlow). Batches consisting of 100 images are used in these experiments. In CIFAR-10, each half batch contains 5 random images from each of the 10 classes. Meanwhile, in CIFAR-100, we require each class represented in the testing sub-batch to also be represented in the training sub-batch.

We write the outputs of each of the four convolutional blocks as $h^{(1)}(x), \ldots, h^{(4)}(x)$. Again defining a as the average pooling operator, we have $a(h^{(1)}(x)) \in \mathbb{R}^{16}$, $a(h^{(2)}(x)) \in \mathbb{R}^{16k}$, $a(h^{(3)}(x)) \in \mathbb{R}^{32k}$, and $a(h^{(4)}(x)) = h^{(4)}(x) \in \mathbb{R}^{64k}$. We define corresponding RBF kernels

$$K^{(l)}_{\gamma_l}(x, x') = k^{(l)}_{\gamma_l}(h^{(l)}(x), h^{(l)}(x')) = e^{-\gamma_l \|a(h^{(l)}(x)) - a(h^{(l)}(x'))\|^2} . \tag{4.4}$$

Given the random mini-batch (X^b, Y^b) and the (randomly sub-sampled) half sub-batch (X^c, Y^c), we evolve θ (and γ) in the steepest descent direction of the loss

$$\mathcal{L}_{\text{KF}} = \sum_{l=1}^{4} \lambda^{(l)} \|Y^b - K^{(l)}_{\gamma_l,\theta}(X^b, X^c) K^{(l)}_{\gamma_l,\theta}(X^c, X^c)^{-1} Y^c\|_2^2 + \mathcal{L}_{\text{c-e}}(f_\theta(X^b), Y^b) . \tag{4.5}$$

4.2.1 Comparison to Dropout

Training Method	**CIFAR-10**	**CIFAR-10.1**	**CIFAR-100**
BN	$4.72 \pm 0.17\%$	$11.07 \pm 0.55\%$	$20.42 \pm 0.25\%$
BN+KF	$4.43 \pm 0.12\%$	$10.38 \pm 0.40\%$	$20.37 \pm 0.27\%$
BN+DO	$4.39 \pm 0.08\%$	$10.50 \pm 0.39\%$	$19.58 \pm 0.41\%$
BN+KF+DO	$4.05 \pm 0.11\%$	$10.20 \pm 0.32\%$	$19.38 \pm 0.18\%$

Table 4: A comparison of the average and standard deviation of test errors over 5 runs for networks trained on augmented data on CIFAR-10, CIFAR-10.1, and CIFAR-100. The second column to the right trains on augmented CIFAR-10 data but tests on CIFAR-10.1 data [22, 30].

Table 4 compares the test errors obtained after training with only batch normalization (BN) with the incorporation of dropout (DO), KF-regularization, as well as a combination of all three. The network architecture WRN-16-8 is used, and the testing error statistics over five runs are listed. We train with step exponentially decreasing learning rates over 200 epochs with identical hyperparameters as [35]. We observe that KF-regularization improves testing error rates against training with BN and BN+DO.

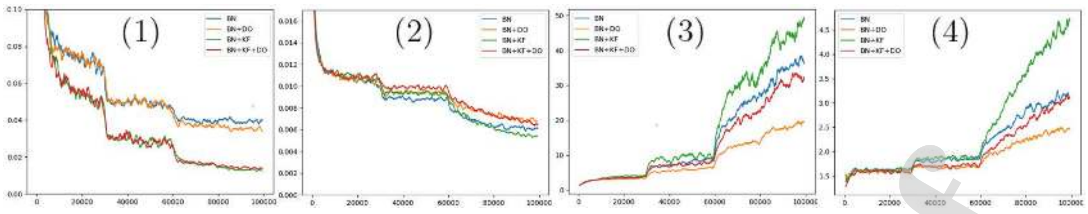

Figure 4: Single run using WRN-16-8 with each of BN only, BN+KF, BN+DO, and BN+KF+DO plotting (1) CIFAR-10 KF loss (2) CIFAR-100 KF loss (3) CIFAR-10 ratio of mean inter-class and in-class distances $h^{(4)}$ (4) CIFAR-100 ratio of mean inter-class and in-class distances $h^{(4)}$.

We also run a distributional shift experiment for CIFAR-10 using the data set CIFAR-10.1, [22] which is sampled from [30]. As with the QMNIST experiment, we also observe improvements with the addition of KF-regularization

We finally compare the KF loss, $\mathcal{L}_{\text{KF}}$, and ratios of inter-class and in-class Euclidean distances on the output of the final convolutional layers within each batch in Figure 4. These statistics are plotted over runs of WRN trained with CIFAR-10 and CIFAR-100. We again observe reduced KF losses and increased ratios of mean inter-class and in-class distances on the final convolutional layer output $h^{(4)}$ when comparing between BN and BN+KF as well as BN+DO and BN+KF+DO. That is, KF-regularization reduces the distance (defined on the outputs of the inner layers) between images in the same class and increases the distance between images in distinct classes (thereby enhancing the separation). The opposite effect is observed with the addition of dropout in training, suggesting they improve testing errors through distinct mechanisms.

5 Concluding remarks

5.1 On our contributions and the novelty of the method

While deep kernels [32], and network regularization techniques are not new; the proposed framework for simultaneously training inner and output layers inspired by kernel flows is new. Furthermore, the proposed Kernel Flow (KF) regularization is not the same as the one presented in [16]. The proposed KF regularization approach uses a kernel to learn a network classifier, while [16] uses a network to learn a kernel classifier. Furthermore, our method is not the same as and has a different purpose than the early exit BranchyNet approach [29]. The early exit BranchyNet approach leverages the observation that "features learned at an early layer of a network may often be sufficient for the classification of many data points." Our method establishes a framework that utilizes the early (and other) layers of a network to train a classifier on the output of the network. The kernel interpretation is only used to construct a total loss that incorporates the status of the network's inner layers. A Kernel Flows loss (which is a variant of cross-validation) is then used to quantify the quality of the feature maps learned by those inner layers (this loss has a theoretical justification in numerical approximation [7]).

5.2 Why the proposed Kernel flow regularizer can improve performance?

To answer this question, observe that ANNs can be interpreted as ridge regression methods [18] performed with a feature map learned from data and parameterized by the inner layers for the network [18]. While minimizing the training loss lets the ANN fit the training data, it does not guarantee the learning a good kernel. In particular, in the overparameterized regime, training Neural Networks (or models) $f(x, \theta)$ with gradient descent and cross-entropy or mean squared loss is essentially equivalent to interpolating the data with the Neural Tangent Kernel $K(x, x') = \nabla_\theta f(x, \theta_0) \cdot \nabla_\theta f(x', \theta_0)$ [12, 14], i.e., when combined with gradient descent, losses defined as linear functionals (generalized moments) of the empirical distribution simply interpolate the data with a kernel fixed at initialization (θ_0). Kernel Flows, on the other hand, use nonlinear functionals of the empirical distribution designed to actually train the underlying kernel defined by the architecture of the Neural Network. Furthermore, while dropout prevents overfitting (see [18] for a rigorous interpretation and alternative in terms of adding nuggets (multiple of the identity matrix) to the kernel representation of the ANN), it also does not guarantee the learning of a good kernel.

The importance of learning adapted kernels is well understood in numerical approximation [17, 19, 15] where it can be shown that interpolants/regressors obtained from non-adapted kernels can be arbitrarily slow to converge as the number of data points goes to infinity [4]. Beyond using the structure of the underlying problem [17], there are essentially two main approaches to learning adapted kernels: The hierarchical Bayes approach (which includes MLE and MAP estimation) and the cross-validation approach. Adding l_2-regularization on weights and biases is equivalent to using Bayesian MAP estimation for learning the kernel [18]. Although the Bayesian approach (e.g., MLE or MAP) is optimal (in MSE), when the model is well-specified, it is not robust to model misspecification, whereas cross-validation is nearly optimal and has some degree of robustness [28, 13, 5, 7]. As a variant of the cross-validation, Kernel Flow [16] inherits these properties, and it has been rigorously shown to achieve an optimal rate of convergence in the approximation of target functions sampled from a GP with a Matérn covariance function with unknown parameters. Its application to regression problems and to learning dynamical systems has already been shown to not only be effective on synthetic problems [9], it has also been shown to outperform (in terms of both accuracy and complexity) both PDE and ANN-based methods for the extrapolation of geophysical time-series [8].

Acknowledgments The authors gratefully acknowledge support by the Air Force Office of Scientific Research under award number FA9550-18-1-0271 (Games for Computation and Learning), Beyond Limits (Learning Optimal Models), and NASA/JPL (Earth 2050). We also thank an anonymous referee for comments and suggestions.

References

[1] *wrn-tensorflow*, 2018 (accessed January, 2020). https://github.com/dalgu90/wrn-tensorflow.

[2] *Tensorflow and deep learning without a PhD*, 2019 (accessed December, 2019). https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd.

[3] *How to choose CNN Architecture MNIST*, 2019 (accessed January, 2020). https://www.kaggle.com/cdeotte/how-to-choose-cnn-architecture-mnist.

[4] Ivo Babuška and John Osborn. Can a finite element method perform arbitrarily badly? *Mathematics of computation*, 69(230):443–462, 2000.

[5] François Bachoc. Cross validation and maximum likelihood estimations of hyper-parameters of gaussian processes with model misspecification. *Computational Statistics & Data Analysis*, 66:55–69, 2013.

[6] Alberto Bietti, Grégoire Mialon, Dexiong Chen, and Julien Mairal. A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning*, pages 664–674. PMLR, 2019.

[7] Yifan Chen, Houman Owhadi, and Andrew M Stuart. Consistency of empirical Bayes and kernel flow for hierarchical parameter estimation. *arXiv preprint arXiv:2005.11375*, 2020.

[8] Boumediene Hamzi, Romit Maulik, and Houman Owhadi. Data-driven geophysical forecasting: Simple, low-cost, and accurate baselines with kernel methods.

[9] Boumediene Hamzi and Houman Owhadi. Learning dynamical systems from data: a simple cross-validation perspective. *arXiv preprint arXiv:2007.05074*, 2020.

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

[11] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[13] Robert Kohn, Craig F Ansley, and David Tharm. The performance of cross-validation and maximum likelihood estimators of spline smoothing parameters. *Journal of the american statistical association*, 86(416):1042–1050, 1991.

[14] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, pages 8570–8581, 2019.

[15] H. Owhadi and C. Scovel. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design*, volume 35. Cambridge University Press, 2019.

[16] H. Owhadi and G.R. Yoo. Kernel Flows: From learning kernels from data into the abyss. *Journal of Computational Physics*, 389:22–47, 2019. https://doi.org/10.1016/j.jcp.2019.03.040.

[17] Houman Owhadi. Bayesian numerical homogenization. *Multiscale Modeling & Simulation*, 13(3):812–828, 2015.

[18] Houman Owhadi. Do ideas have shape? plato's theory of forms as the continuous limit of artificial neural networks. *arXiv preprint arXiv:2008.03920*, 2020.

[19] Houman Owhadi, Clint Scovel, and Florian Schäfer. Statistical numerical approximation. *Notices of the AMS*, 2019.

[20] Vardan Papyan, XY Han, and David L Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

[21] O Perrin and P Monestiez. Modelling of non-stationary spatial structure using parametric radial basis deformations. In *GeoENV II—Geostatistics for Environmental Applications*, pages 175–186. Springer, 1999.

[22] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do CIFAR-10 classifiers generalize to CIFAR-10? *arXiv preprint arXiv:1806.00451*, 2018.

[23] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? *arXiv preprint arXiv:1902.10811*, 2019.

[24] Paul D Sampson and Peter Guttorp. Nonparametric estimation of nonstationary spatial covariance structure. *Journal of the American Statistical Association*, 87(417):108–119, 1992.

[25] Alexandra M Schmidt and Anthony O'Hagan. Bayesian inference for non-stationary spatial covariance structure via spatial deformations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(3):743–758, 2003.

[26] P.Y. Simard, D. Steinkraus, and J. Platt. Best practices for Convolutional Neural Networks applied to visual document analysis. Institute of Electrical and Electronics Engineers, Inc., August 2003.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, Jun 2014. https://dl.acm.org/doi/abs/10.5555/2627435.2670313.

[28] Michael L Stein. A comparison of generalized cross validation and modified maximum likelihood for estimating the parameters of a stochastic process. *The Annals of Statistics*, pages 1139–1157, 1990.

[29] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[30] A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008. https://doi.org/10.1109/TPAMI.2008.128.

[31] Colin Wei, Jason D Lee, Qiang Liu, and Tengyu Ma. Regularization matters: Generalization and optimization of neural nets vs their induced kernel. In *Advances in Neural Information Processing Systems*, pages 9712–9724, 2019.

[32] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378, 2016.

[33] Yadav and L. C., Bottou. Cold case: The lost MNIST digits. In *Advances in Neural Information Processing Systems*, pages 13443–13452, 2019. http://papers.nips.cc/paper/9500-cold-case-the-lost-mnist-digits.pdf.

[34] Kai Yu, Wei Xu, and Yihong Gong. Deep learning with kernel regularization for visual recognition. *Advances in Neural Information Processing Systems*, 21:1889–1896, 2008.

[35] S. Zagoruyko and N. Komodakis. Wide Residual Networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016. https://dx.doi.org/10.5244/C.30.87.

- Neural networks are trained with kernels parameterized by inner layer outputs.
- Kernel Flows (cross-validation variant) enables the direct training of inner layers.
- KF regularization improves generalization error, robustness to distributional shift.
- The underlying inner layer kernels are learned from data.

- GRY: Conceptualization, methodology, numerical experiments, writing-original draft preparation.
- HO: Conceptualization, writing, reviewing, and editing.

CALTECH **Gene Ryan Yoo**

Gene Ryan Yoo
Graduate student
Department of Mathematics
MC 253-37, California Institute of Technology
Pasadena, CA 91125
Email: ryanyoo2008@gmail.com
Phone: (310) 283-8163

January 1, 2021

Dear editors,

I am submitting the manuscript "*Deep regularization and direct training of the inner layers of Neural Networks with Kernel Flows*" for publication in the Physica D: Nonlinear Phenomena. This work has no significant overlap with previous publications. This investigation develops the training of Artificial Neural Networks with Kernel Flows as introduced in [1]. The author has no conflicts of interest nor recent collaborators to report.

Sincerely,

Gene Ryan Yoo

[1] Houman Owhadi and Gene Ryan Yoo. Kernel flows: From learning kernels from data into the abyss. *Journal of Computational Physics*, 389:22–47, 2019.