Journal of Computational Physics 389 (2019) 22–47

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Kernel Flows: From learning kernels from data into the abyss

Houman Owhadi [a,*], Gene Ryan Yoo [b]

[a] California Institute of Technology, 1200 E California Blvd, MC 9-49, Pasadena, CA 91125, USA
[b] California Institute of Technology, 1200 E California Blvd, MC 253-47, Pasadena, CA 91125, USA

ARTICLE INFO

Article history:
Received 28 September 2018
Received in revised form 17 March 2019
Accepted 20 March 2019
Available online 28 March 2019

Keywords:
Learning Kernels
Kriging
Deep learning
Data driven dynamical system
Support vector machine
Reproducing Kernel Hilbert space

ABSTRACT

Learning can be seen as approximating an unknown function by interpolating the training data. Although Kriging offers a solution to this problem, it requires the prior specification of a kernel and it is not scalable to large datasets. We explore a numerical approximation approach to kernel selection/construction based on the simple premise that a kernel must be good if the number of interpolation points can be halved without significant loss in accuracy (measured using the intrinsic RKHS norm $\|\cdot\|$ associated with the kernel). We first test and motivate this idea on a simple problem of recovering the Green's function of an elliptic PDE (with inhomogeneous coefficients) from the sparse observation of one of its solutions. Next we consider the problem of learning non-parametric families of deep kernels of the form $K_1(F_n(x), F_n(x'))$ with $F_{n+1} = (I_d + \epsilon G_{n+1}) \circ F_n$ and $G_{n+1} \in \text{span}\{K_1(F_n(x_i), \cdot)\}$. With the proposed approach constructing the kernel becomes equivalent to integrating a stochastic data driven dynamical system, which allows for the training of very deep (bottomless) networks and the exploration of their properties. These networks learn by constructing flow maps in the kernel and input spaces via incremental data-dependent deformations/perturbations (appearing as the cooperative counterpart of adversarial examples) and, at profound depths, they (1) can achieve accurate classification from only one data point per class (2) appear to learn archetypes of each class (3) expand distances between points that are in different classes and contract distances between points in the same class. For kernels parameterized by the weights of Convolutional Neural Networks, minimizing approximation errors incurred by halving random subsets of interpolation points, appears to outperform training (the same CNN architecture) with relative entropy and dropout.

1. Introduction

Despite their popularity and impressive achievements [14] Artificial Neural Networks (ANNs) remain difficult to analyze. From a deep kernel learning perspective [35], the action of the last layer of an ANN can be seen as that of regressing the data with a kernel parameterized by the weights of all the previous layers. Therefore analysing the problem of performing a regression of the data with a kernel that is also learnt from the data could help understand ANNs and elaborate a rigorous theory for deep learning. Hierarchical Bayesian Inference [29] (placing a prior on a space of kernels and conditioning on the data) and Maximum Likelihood Estimation [34] (choosing the kernel which maximizes the probability of observing the data) are well known approaches for learning the kernel. In this paper we explore a numerical approximation approach (motivated

* Corresponding author.
E-mail addresses: owhadi@caltech.edu (H. Owhadi), gyoo@caltech.edu (G.R. Yoo).

https://doi.org/10.1016/j.jcp.2019.03.040
0021-9991/

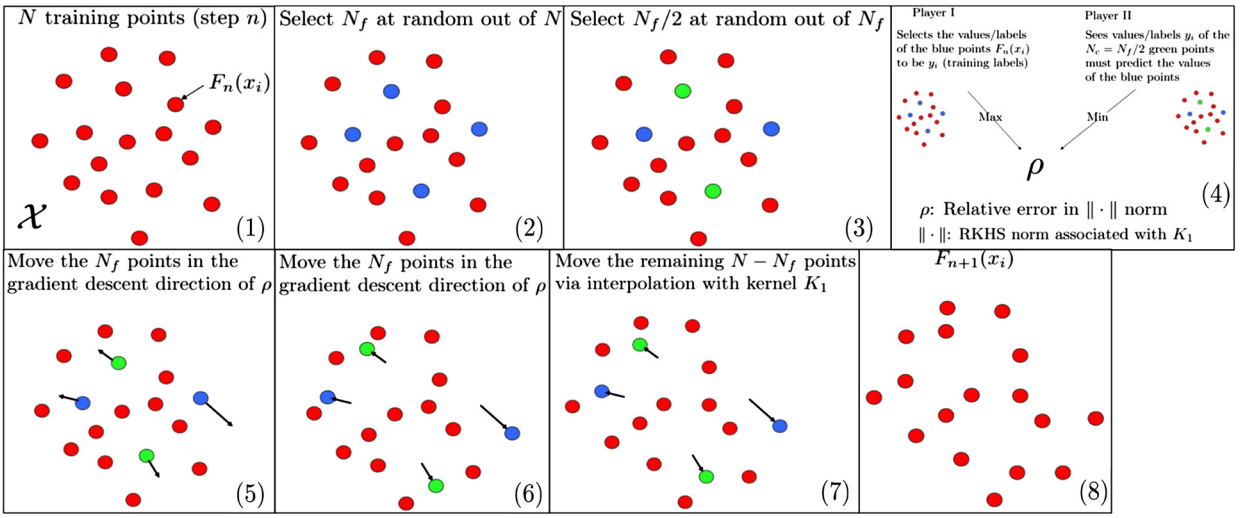

Fig. 1. The game theoretic interpretation of the step $n \to n+1$ of Kernel Flow. (1) Starting from F_n and the N data points $(F_n(x_i), y_i)$ (2) Select N_f indices out of N (3) Select $N_f/2$ indices out of N_f (4) Consider the zero sum adversarial game where Player I chooses the labels of the N_f points to be y_i and Player II sees half of them and tries to guess the other and let ρ be the loss of Player II in that game (using relative error in the RKHS norm associated with K_1) (5, 6) Move the N_f selected points $F_n(x_i)$ to decrease the loss of Player II (7) Move the remaining $N - N_f$ (and any other point x) points via interpolation with the kernel K_1 (this specifies F_{n+1}) (8) Repeat.

by interplays between Gaussian Process Regressions and Numerical Homogenization [21]) based on the simple premise that a kernel must be good if the number of points N used to perform the interpolation of data can be reduced to $N/2$ without significant loss in accuracy (measured using the intrinsic RKHS norm $\|\cdot\|$ associated with the kernel). Writing u and v for the interpolation of the data with N and $N/2$ points, the relative error $\rho = \frac{\|u-v\|^2}{\|u\|^2}$ induces a data dependent ordering on the space of kernels. The Fréchet derivative of ρ identifies the direction of the gradient descent and leads to a simple algorithm (Kernel Flow) for its minimization: (1) Select N_f ($\leq N$) points (at random, uniformly, without replacement) from the N training data points (2) Select $N_c = N_f/2$ points (at random, uniformly, without replacement) from the N_f points (3) Perturb the kernel in the gradient descent direction of ρ (computed from the current kernel and the N_f, N_c points) (4) Repeat.

To provide some context for this algorithm, we first summarize (in Section 2) interplays between Kriging, Gaussian Process Regression, Game Theory and Optimal Recovery. The identification (in Sec. 3) of ρ and its Fréchet derivative leads (in Sec. 4) to the proposed algorithm in a parametric setting.

In Sec. 5 we describe interplays between the proposed algorithm and Numerical Homogenization [21] by implementing and testing the parametric version of Kernel Flow for the (simple and amenable to analysis) problem of (1) recovering the unknown conductivity a of the PDE $-\mathrm{div}(a\nabla u) = f$ based on seeing $u \in \mathcal{H}_0^1((0,1))$ at a finite number N of points (2) approximating u between measurement points. For this problem, a, u and f are all unknown, we only know that $f \in L^2$ and we try to learn the Green's function of the PDE (seen as a kernel parameterized by the unknown conductivity a). Experiments suggest that, by minimizing ρ (parameterized by the conductivity), the algorithm can recover the conductivity and significantly improve the accuracy of the interpolation.

Next (in Sec. 6) we derive a non parametric version of the proposed algorithm that learns a kernel of the form

$$K_n(x,x') = K_1(F_n(x), F_n(x'))\,, \tag{1.1}$$

where K_1 is a standard kernel (e.g. Gaussian $K_1(x,x') = e^{-\gamma|x-x'|^2}$) and F_n maps the input space into itself, $n \to F_n$ is a discrete flow in the input space, and F_{n+1} is obtained from F_n by interrogating random subsets of the training data as described in Fig. 1 (which also summarizes the game theoretic interpretation of the proposed Algorithm, note that the game is incrementally rigged to minimize the loss of Player II).

The proposed algorithm (see Fig. 2 for a summary of its structure) can be reduced to an iteration of the form

$$K_n(x,x') = K_{n-1}(x + \epsilon G_n(x), x + \epsilon G_n(x')). \tag{1.2}$$

Writing x_i for the training points and $F_1(x) = x$ and $F_n(x) = (I_d + \epsilon G_n) \circ F_{n-1}(x)$, the network F_n (composed of n layers) is learnt from the data in a recursive manner (across layers) by (1) using $g_i^{(n)} := G_n \circ F_{n-1}(x_{s_{f,n}(i)})$ for a random subset $\{x_{s_{f,n}(i)} | 1 \leq i \leq N_f\}$ of the points $\{x_i | 1 \leq i \leq N\}$ as training parameters and interpolating G_n with the kernel K_1 in between the points $F_{n-1}(x_{s_{f,n}(i)})$ (2) selecting $g_i^{(n)}$ in the direction of the gradient descent of ρ at each step.

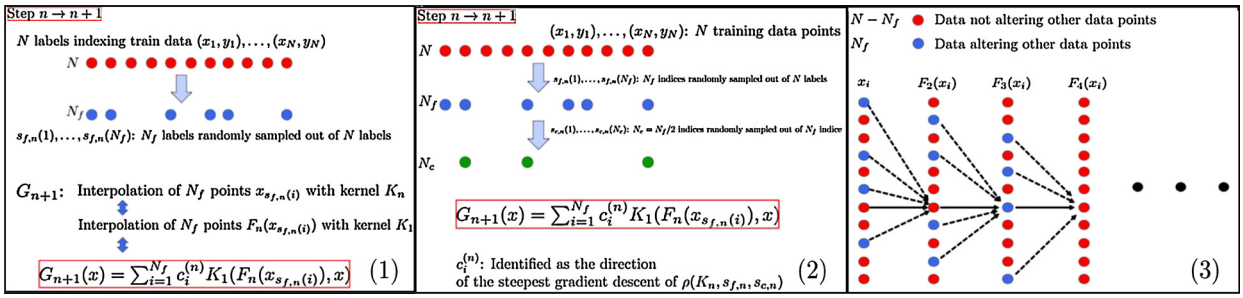

Fig. 2. The Kernel Flow Algorithm. (1) N_f indices $s_{f,n}(1), \ldots, s_{f,n}(N_f)$ are randomly sampled out of N and G_{n+1} belongs to the linear span of the $K(F(x_{s_{f,n}(i)}^{(n)}), x)$ (2) the coefficients in the representation of G_{n+1} are found as the direction of the gradient descent of ρ (3) The value of $F_{n+1}(x_i)$ is the sum of $F_n(x_i)$ a small perturbation depending on the joint values of the $F_n(x_{s_{f,n}(j)})$.

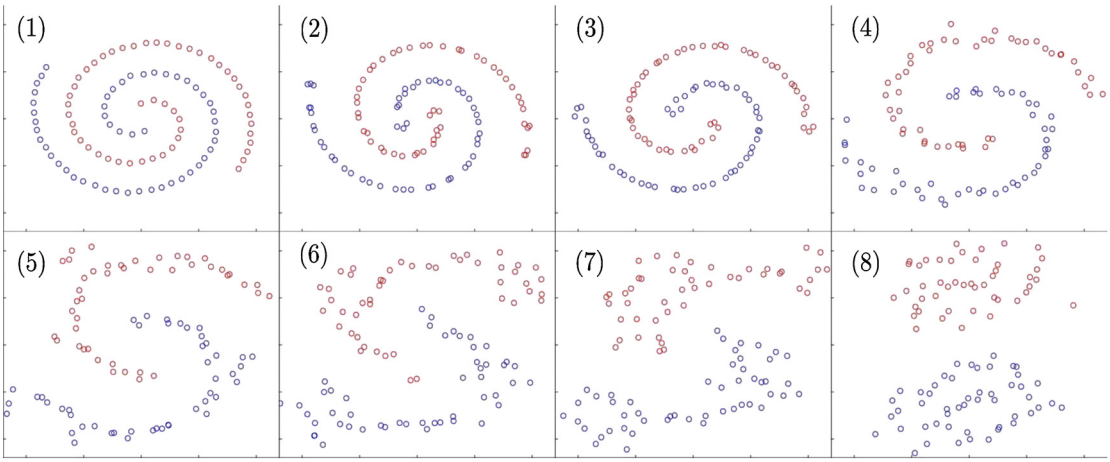

Fig. 3. Swiss Roll Cheesecake. $N = 100$. Red points have label -1 and blue points have label 1. $F_n(x_i)$ for 8 different values of n. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

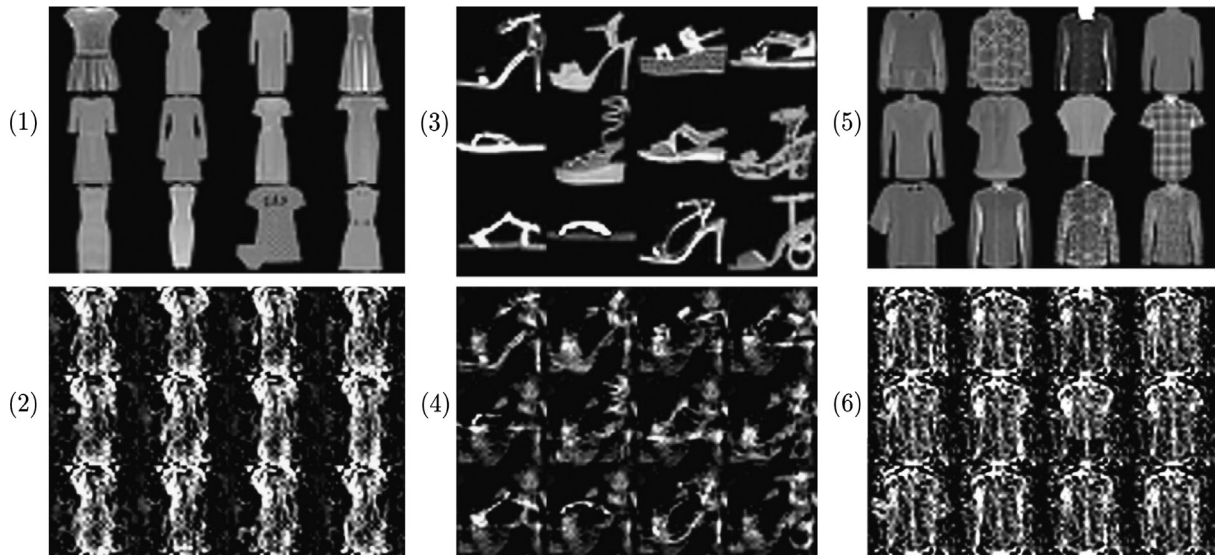

Fig. 4. Results for Fashion-MNIST. $N = 60000$, $N_f = 600$ and $N_c = 300$. (1, 3, 5) Training data x_i (class 3, 5 and 6) (2, 4, 6) $F_n(x_i)$ (class 3, 4 and 6) for $n = 50000$.

Writing (x_i, y_i) for the N training data points, G_{n+1} ends up being of the form

$$G_{n+1}(x) = \sum_{i=1}^{N_f} c_i^{(n)} K_1(F_n(x_{s_{f,n}(i)}), x) \tag{1.3}$$

where $s_{f,n}(1), \ldots, s_{f,n}(N_f)$ are N_f indices sampled at random (uniformly without replacement) from $\{1, \ldots, N\}$. Note that the kernel K_n produced at any step n, is a Deep Hierarchical Kernel (in the sense of [35,31]) satisfying the nesting equations

$$K_n(x, x') = K_{n-1}(x + \epsilon \sum_{i=1}^{N_f} c_i^{(n-1)} K_{n-1}(x_{s_{f,n-1}(i)}, x), x' + \epsilon \sum_{i=1}^{N_f} c_i^{(n-1)} K_{n-1}(x_{s_{f,n-1}(i)}, x')) \,. \tag{1.4}$$

Furthermore, the structure of the network defined by that kernel is randomized through the random selection of the N_f points (see Fig. 2.3). The coefficients $c_i^{(n)}$ in (1.3) are identified through one step of gradient descent of the relative error ρ (measured in the RKHS norm associated with K_1) of the approximation of the labels (the $y_{s_{f,n}(i)}$) of those N_f points upon seeing half of them. From the game theoretic perspective of Fig. 1, ρ is the loss of Player II (attempting the guess the unseen labels) and the points $F_{n+1}(x_{s_{f,n}(i)})$ are perturbations of the points $F_n(x_{s_{f,n}(i)})$ in a direction which seeks to minimize this loss. Note also that the proposed (Kernel Flow) algorithm produces a flow F_n (randomized through sampling of the training data) in the input space and a (stochastic) dynamical system $K_1(F_n(x), F_n(y))$ in the kernel space. Since learning becomes equivalent to integrating a dynamical system, it does not require back-propagation nor guessing the architecture of the network, which enables the construction of very deep networks and the exploration of their properties.

We implement this algorithm (and visualize its flow) for MNIST [37], Fashion-MNIST [36], the Swiss Roll Cheesecake. See Fig. 3 and Fig. 4 for illustrations of the flow $F_n(x)$ for the Swiss Roll Cheesecake and Fashion-MNIST. For these datasets we observe that (1) the flow F_n unrolls the Swiss Roll Cheesecake (2) the flow F_n expands distances between points that are in different classes and contracts distances between points in the same class (towards archetypes of each class) (3) at profound depths ($n = 12000$ layers for MNIST and $n = 50000$ layers for Fashion-MNIST) the resulting kernel K_n achieves a small average error (1.5% for MNIST and 10% for Fashion-MNIST) using only 10 points as interpolation points (i.e. one point for each class) (4) the incremental data-dependent perturbations ϵG_{n+1} seem to take advantage of a cooperative mechanism appearing as the counterpart of the one associated with adversarial examples [32,23].

Furthermore, since N_f can be chosen independently from N, the proposed algorithms appears to provide a solution toward making Kriging/SVM scalable by producing a kernel capable of generalization from few samples.

Finally (in Sec. 10) we derive an ANN version of the algorithm by identifying the action of the last layer of the ANN as that of regressing the data with a kernel parameterized by the weights of all the previous layers (learnt by minimizing ρ or its analogous L^2 version). This algorithm is then tested for the MNIST and Fashion-MNIST data sets and shown to contract in-class distances and inter-class distances in a similar manner as above (thereby achieving accuracies comparable to the state of the art with a small number of interpolation points). For kernels parameterized by the weights of a given Convolutional Neural Network, minimizing ρ or its L^2 version, appears, for the MNIST and fashion MNIST data sets, to outperform training (the same CNN architecture) with relative entropy and dropout.

This paper is not aimed at identifying the state of the art algorithm in terms of accuracy nor complexity. It is simply motivated by an attempt to offer some insights (from a numerical approximation perspective) on mechanisms that may be at play in deep learning.

2. Learning as an interpolation problem

It is well understood [27] that "learning techniques are similar to fitting a multivariate function to a certain number of measurement data", e.g. solving the following problem.

Problem 1. Given input/output data $(x_1, y_1), \ldots, (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$ recover an unknown function $u^\dagger$ mapping $\mathcal{X}$ to $\mathcal{Y}$ such that

$$u^\dagger(x_i) = y_i \text{ for } i \in \{1, \ldots, N\} \,. \tag{2.1}$$

Optimal recovery In the setting of optimal recovery [16] the ill posed Problem 1 can be turned into a well posed one by restricting candidates for u to belong to a space of functions $\mathcal{B}$ endowed with a norm $\|\cdot\|$ and identifying the optimal recovery as the minimizer of the relative error

$$\min_v \max_u \frac{\|u - v\|^2}{\|u\|^2} \,, \tag{2.2}$$

where the max is taken over $u \in \mathcal{B}$ and the min is taken over candidates in $v \in \mathcal{B}$ such that $v(x_i) = u(x_i)$. Observe that $\mathcal{B}^*$, the dual space of $\mathcal{B}$, must contain delta Dirac functions

$$\phi_i(\cdot) := \delta(\cdot - x_i)\,, \tag{2.3}$$

for the validity of the constraints $u(x_i) = y_i$. Consider now the case where $\|\cdot\|$ is quadratic, i.e. such that

$$\|u\|^2 = [Q^{-1}u, u]\,, \tag{2.4}$$

where $[\phi, u]$ stands for the duality product between $\phi \in \mathcal{B}^*$ and $u \in \mathcal{B}$ and $Q : \mathcal{B}^* \to \mathcal{B}$ is a positive symmetric linear bijection (i.e. such that $[\phi, Q\phi] \geq 0$ and $[\phi, Q\varphi] = [\varphi, Q\phi]$ for $\varphi, \phi \in \mathcal{B}^*$). In that case the optimal solution of (2.2) has the explicit form (writing y_i for $u(x_i)$)

$$v^\dagger = \sum_{i,j=1}^{N} y_i A_{i,j} Q\phi_j\,, \tag{2.5}$$

where $A = \Theta^{-1}$ and Θ is the $N \times N$ Gram matrix with entries $\Theta_{i,j} = [\phi_i, Q\phi_j]$. Furthermore $v^\dagger$ can also be identified as the minimizer of

$$\begin{cases} \text{Minimize } \|\psi\| \\ \text{Subject to } \psi \in \mathcal{B} \text{ and } [\phi_i, \psi] = y_i, \quad i \in \{1, \ldots, N\}. \end{cases} \tag{2.6}$$

Kriging Defining K as the kernel

$$K(x, x') = [\delta(\cdot - x), Q\,\delta(\cdot - x')]\,, \tag{2.7}$$

$(\mathcal{B}, \|\cdot\|)$ can be seen as a Reproducing Kernel Hilbert Space endowed with the norm

$$\|u\|^2 = \sup_{\phi \in \mathcal{B}^*} \frac{(\int \phi(x)u(x)\,dx)^2}{\int \phi(x)K(x,y)\phi(y)\,dx\,dy}\,, \tag{2.8}$$

and (2.5) corresponds to the classical representer theorem

$$v^\dagger(\cdot) = y^T A K(x_\cdot, \cdot)\,, \tag{2.9}$$

using the vectorial notations $y^T A K(x_\cdot, \cdot) = \sum_{i,j=1}^{N} y_i A_{i,j} K(x_j, \cdot)$ with $A = \Theta^{-1}$ and $\Theta_{i,j} = K(x_i, x_j)$.

Gaussian process regression numerical approximation games Writing ξ for the centered Gaussian Process with covariance function K, (2.9) can also be recovered via Gaussian Process Regression as

$$v^\dagger(x) = \mathbb{E}\big[\xi(x) \mid \xi(x_i) = y_i\big]\,. \tag{2.10}$$

This link between Numerical Approximation and Gaussian Process Regression emerges naturally by viewing (2.2) as an adversarial zero sum game [21,18,20,28] between two players (I and II where I tries to maximize the relative error and II tries to minimize it after seeing the values of u at the points x_i) and observing that ξ and v are optimal mixed/randomized strategies for players I and II (forming a saddle point for the minimax lifted to measures over functions).

3. What is a good kernel?

Although the optimal recovery of $u^\dagger$ has a well established theory, it relies on the prior specification of a quadratic norm $\|\cdot\|$ or equivalently of a kernel K. In practical applications the performance of the interpolant (2.9) (e.g. when employed in a classification problem) is sensitive to the choice of K. How should K be selected to achieve generalization? Although ANNs [14] seem to address this question (by performing variants of the interpolation (2.9) with the last layer of the network using a kernel K parameterized by the weights of the previous layers and learnt by adjusting those weights) they remain difficult to analyze and the introduction of regularization steps (such as dropout or early stop) introduced to achieve generalization appear to be discovered through a laborious process of trial and error [38].

Is there a systematic way of identifying a good kernel? What is good kernel?

We will now explore these questions from the perspective of interplays between numerical approximation and inference [21] and the simple premise that a kernel must be good if the number of points N used to perform the interpolation of data can be reduced to $m = \text{round}(N/2)$ without significant loss in accuracy (measured using the intrinsic RKHS norm associated with the kernel).

To label the m sub-sampled (test) data points, let $s(1), \ldots, s(m)$ be a selection of m distinct elements of $\{1 \ldots, N\}$. Observe that $\{x_{s(1)}, \ldots, x_{s(m)}\}$ forms a strict subset of $\{x_1, \ldots, x_N\}$. Write v^s for the optimal recovery of $u^\dagger$ upon seeing its values at the points $x_{s(1)}, \ldots, x_{s(m)}$, and observe that $v^s(\cdot) = \sum_{i=1}^{m} \bar{y}_i \bar{A}_{i,j} K(x_{s(j)}, i)$ with $\bar{y}_i = y_{s(i)}$ and $\bar{A} = \bar{\Theta}^{-1}$ with $\bar{\Theta}_{i,j} = \Theta_{s(i),s(j)}$. Let π be the corresponding $m \times N$ sub-sampling matrix defined by $\pi_{i,j} = \delta_{s(i),j}$ and observe that

$$v^s = y^T \tilde{A} K(x_\cdot, \cdot) \tag{3.1}$$

with $\tilde{A} = \pi^T \bar{A} \pi$ and $\bar{A} = (\pi \Theta \pi^T)^{-1}$.

Proposition 3.1. *For $v^\dagger$ and v^s defined as in (2.9) and (3.1), we have*

$$\|v^\dagger - v^s\|^2 = y^T A y - y^T \tilde{A} y\,. \tag{3.2}$$

Proof. Proposition 3.1 is particular case of [21, Prop. 13.29]. The proof follows simply from $\|v^\dagger\|^2 = y^T A y$, $\|v^s\|^2 = \bar{y}^T \bar{A} \bar{y} = y^T \tilde{A} y$ and the orthogonal decomposition $\|v^\dagger\|^2 = \|v^s\|^2 + \|v^\dagger - v^s\|^2$ implied by the fact that v^s is the minimizer of $\|\psi\|^2$ subject to the constraints $[\phi_{s(i)}, \psi] = y_{s(i)}$ and that $v^\dagger$ satisfies those constraints. □

Let ρ be the ratio

$$\rho := \frac{\|v^\dagger - v^s\|^2}{\|v^\dagger\|^2}\,. \tag{3.3}$$

Note that a value of ρ close to zero indicates that v^s is a good approximation of $v^\dagger$ (and that most of the energy of $v^\dagger$ is contained in v^s) which is a desirable condition for the kernel K to achieve generalization. Furthermore, Proposition 3.1 implies that $\rho \in [0, 1]$ and

$$\rho = 1 - \frac{y^T \tilde{A} y}{y^T A y}\,. \tag{3.4}$$

Fixing y and π, ρ can be seen as a function of A which we will write $\rho(A)$. Since $A = \Theta^{-1}$, ρ can also be viewed as a function of Θ which, abusing notations, we will write $\rho(\Theta)$. Motivating by the application of ρ to the ordering of space of kernels (a small ρ being indicative of a good kernel) we will, in the following proposition, compute its Fréchet derivative with respect to small perturbations of A or of Θ.

Proposition 3.2. *Write $z := A^{-1}\tilde{A} y$ with $\tilde{A} := \pi^T(\pi A^{-1}\pi^T)^{-1}\pi$ defined as above.*[1] *It holds true that*

$$\rho(A + \epsilon S) = \rho(A) + \epsilon \frac{(1 - \rho(A)) y^T S y - z^T S z}{y^T A y} + \mathcal{O}(\epsilon^2)\,, \tag{3.5}$$

and, writing[2] $\hat{y} := \Theta^{-1} y$ *and* $\hat{z} := \pi^T(\pi \Theta \pi^T)^{-1}\pi y$,

$$\rho(\Theta + \epsilon T) = \rho(\Theta) - \epsilon \frac{(1 - \rho(\Theta)) \hat{y}^T T \hat{y} - \hat{z}^T T \hat{z}}{\hat{y}^T \Theta \hat{y}} + \mathcal{O}(\epsilon^2)\,. \tag{3.6}$$

Proof. Observe that

$$\rho(A + \epsilon S) = 1 - \frac{y^T \pi^T (\pi (A + \epsilon S)^{-1} \pi^T)^{-1} \pi y}{y^T (A + \epsilon S) y} \tag{3.7}$$

and recall the approximation

$$(A + \epsilon S)^{-1} = A^{-1} - \epsilon A^{-1} S A^{-1} + \mathcal{O}(\epsilon^2)\,. \tag{3.8}$$

(3.5) then follows from straightforward calculus. The proof of (3.6) is identical and can also be obtained from (3.5) and the first order approximation $(\Theta + \epsilon T)^{-1} = \Theta^{-1} - \epsilon \Theta^{-1} T \Theta^{-1} + \mathcal{O}(\epsilon^2)$. □

4. The algorithm with a parametric family of kernels

Let $\mathcal{W}$ be a finite dimensional linear space and let $K(x, x', W)$ be a family of kernels parameterized by $W \in \mathcal{W}$. Let $N_f \leq N$ and $N_c = \text{round}(N_f/2)$. Let $s_f(1), \ldots, s_f(N_f)$ be a selection of N_f distinct elements of $\{1, \ldots, N\}$. Let $s_c(1), \ldots, s_c(N_c)$ be a selection of N_c distinct elements of $\{1, \ldots, N_f\}$. Let π be the corresponding $N_c \times N_f$ sub-sampling matrix defined by $\pi_{i,j} = \delta_{s_c(i),j}$. Let $y_f \in \mathbb{R}^{N_f}$ and $y_c \in \mathbb{R}^{N_c}$ be the corresponding subvectors of y defined by $y_{f,i} = y_{s_f(i)}$ and $y_{c,i} = y_{f,s_c(i)}$.

Using the notations of sections 2 and 3 write $\Theta(W)$ for the $N_f \times N_f$ matrix with entries $\Theta_{i,j} = K(x_{s_f(i)}, x_{s_f(j)}, W)$ and let

$$\rho(W, s_f, s_c) := 1 - \frac{y_c^T (\pi \Theta \pi^T)^{-1} y_c}{y_f^T \Theta^{-1} y_f}\,. \tag{4.1}$$

The following corollary derived from Proposition 3.2 allows us to compute the gradient of ρ respect to W.

[1] The operator $P := \tilde{A} A^{-1}$ is a projection with $\text{Im}(P) = \text{Im}(\pi^T)$ and $\text{Ker}(P) = A\,\text{Ker}(\pi)$ and from the perspective of numerical homogenization $\tilde{A}$ can be interpreted as the homogenized version of A [21, Sec. 13.10.3], see [21, Chap. 13.10] for further geometric properties.

[2] Note that $\hat{z} = \Theta^{-1} z = \tilde{A} y$.

Corollary 4.1. *Write* $\Theta := \Theta(W)$, $\hat{y} := \Theta^{-1} y_f$ *and* $\hat{z} := \pi^T(\pi\Theta\pi^T)^{-1}\pi y_f$. *Write* W_i *for the entries of the vector* W. *It holds true that*

$$\partial_{W_i}\rho(W) = -\frac{(1-\rho(W))\hat{y}^T(\partial_{W_i}\Theta(W))\hat{y} - \hat{z}^T(\partial_{W_i}\Theta(W))\hat{z}}{y_f^T\Theta^{-1}y_f}. \tag{4.2}$$

Proof. Proposition 3.2 implies that

$$\rho(W+\epsilon W') = \rho(W) - \epsilon\frac{(1-\rho(W))\hat{y}^T T\hat{y} - \hat{z}^T T\hat{z}}{y_f^T\Theta^{-1}y_f} + \mathcal{O}(\epsilon^2), \tag{4.3}$$

with $T = (W')^T\nabla_W\Theta(W)$, which proves the result. □

The purpose of Algorithm 1 is to learn the parameters W (of the kernel K) from the data. The value of N_f (and hence N_c) corresponds to the size of a batch. The initialization of W in step 1 may be problem dependent or at random.

Algorithm 1 Learning W in the $K(\cdot,\cdot,W)$.

1: Initialize W
2: **repeat**
3: Select $s_f(1),\dots,s_f(N_f)$ out of $\{1,\dots,N\}$.
4: Select $s_c(1),\dots,s_c(N_c)$ out of $\{1,\dots,N_f\}$.
5: $W' = -\nabla_W\rho(W, s_f, s_c)$
6: $W = W + \epsilon W'$
7: **until** End criterion

5. A simple PDE model

To motivate, illustrate and study the proposed approach, it is useful to start with an application to the following simple PDE model amenable to detailed analysis [21]. Let u be the solution of

$$\begin{cases} -\operatorname{div}\big(a(x)\nabla u(x)\big) = f(x) & x\in\Omega;\\ u = 0 \quad \text{on} \quad \partial\Omega, \end{cases} \tag{5.1}$$

where $\Omega\subset\mathbb{R}^d$, is a regular subset and a is a uniformly elliptic symmetric matrix with entries in $L^\infty(\Omega)$. Write $\mathcal{L} := -\operatorname{div}(a\nabla\cdot)$ for the corresponding linear bijection from $\mathcal{H}_0^1(\Omega)$ to $\mathcal{H}^{-1}(\Omega)$.

In this proposed simple application we seek to recover the solution of (5.1) from the data $(x_i, y_i)_{1\le i\le N}$ and the information $u(x_i) = y_i$. If the conductivity a is known then [21,20] interpolating the data with the kernel (1) $\mathcal{L}^{-1}$ leads to a recovery that is minimax optimal in the (energy) norm $\|u\|^2 = \int_\Omega(\nabla u)a\nabla u$ ($d=1$ is required to ensure the continuity of the kernel). (2) $(\mathcal{L}^T\mathcal{L})^{-1}$ leads to a recovery that is minimax optimal in the norm $\|u\| = \|\operatorname{div}(a\nabla u)\|_{L^2(\Omega)}$ ($d\le 3$, the recovery is equivalent to interpolating with Rough Polyharmonic Splines [24]). (3) $(\mathcal{L}^T\Delta\mathcal{L})^{-1}$ leads to a recovery that is minimax optimal in the norm $\|u\| = \|\operatorname{div}(a\nabla u)\|_{\mathcal{H}_0^1(\Omega)}$ ($d\le 5$).

Which kernel should be used for the recovery of u when the conductivity a is unknown? Consider the case when $d=1$ and Ω is the interval $(0,1)$. For $b\in\mathcal{L}^\infty(\Omega)$ with $\operatorname{essinf}_\Omega(b) > 0$ write G_b for the Green's function of the operator $-\operatorname{div}(b\nabla\cdot)$ mapping $\mathcal{H}_0^1(\Omega)$ to $\mathcal{H}^{-1}(\Omega)$. Observe that the $\{G_b|b\}$ is a set of kernels parameterized b and any kernel in that set could be used to interpolate the data. Which one should we pick? The answer proposed in Sec. 3 and 4 is to use the ordering induced by ρ to select the kernel.

Fig. 5 provides a numerical illustration of that ordering. In that example Ω is discretized over 2^8 equally spaced interior points (and piecewise linear tent finite elements) and Fig. 5.1-3 shows a, f and u. For $k\in\{1,\dots,8\}$ and $i\in\mathcal{I}^{(k)} := \{1,\dots,2^k-1\}$ let $x_i^{(k)} = i/2^k$ and write $v_b^{(k)}$ for the interpolation of the data $(x_i^{(k)}, u(x_i^{(k)}))_{i\in\mathcal{I}^{(k)}}$ using the kernel G_b (note that $v_b^{(8)} = u$). Let $\|v\|_b$ be the energy norm $\|v\|_b^2 = \int_\Omega(\nabla v)^T b\nabla v$. Take $b\equiv 1$. Fig. 5.4 shows (in semilog scale) the values of $\rho(a) = \frac{\|v_a^{(k)} - v_a^{(8)}\|_a^2}{\|v_a^{(8)}\|_a^2}$ and $\rho(b) = \frac{\|v_b^{(k)} - v_b^{(8)}\|_b^2}{\|v_b^{(8)}\|_b^2}$ vs k. Note that the value of ratio ρ is much smaller when the kernel G_a is used for the interpolation of the data. The geometric decay $\rho(a)\le C2^{-2k}\frac{\|f\|_{L^2(\Omega)}}{\|u\|_a^2}$ is well known and has been extensively studied in Numerical Homogenization [21].

Fig. 5.5 shows (in semilog scale) the values of the average prediction errors $e(a)$ and $e(b)$ (vs k) defined (after normalization) to be proportional to $\|v_a^{(k)}(x) - u(x)\|_{L^2(\Omega)}$ and $\|v_b^{(k)}(x) - u(x)\|_{L^2(\Omega)}$. Note again that the prediction error is much smaller when the kernel G_a is used for the interpolation.

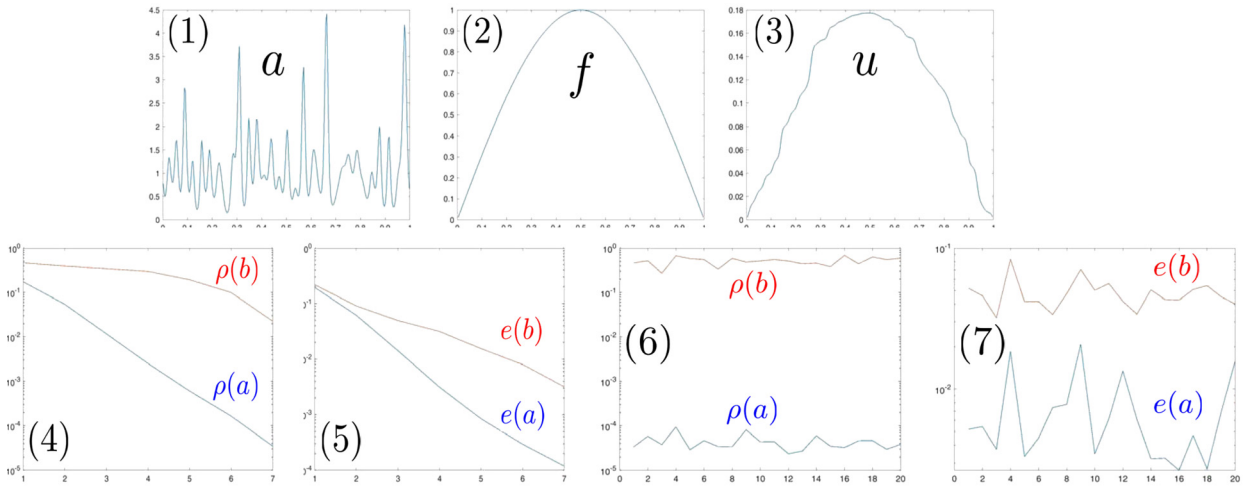

Fig. 5. (1) a (2) f (3) u (4) $\rho(a)$ and $\rho(b)$ vs k, geometric (5) $e(a)$ and $e(b)$ vs k, geometric (6) $\rho(a)$ and $\rho(b)$ vs k, random (5) $e(a)$ and $e(b)$ vs random realization.

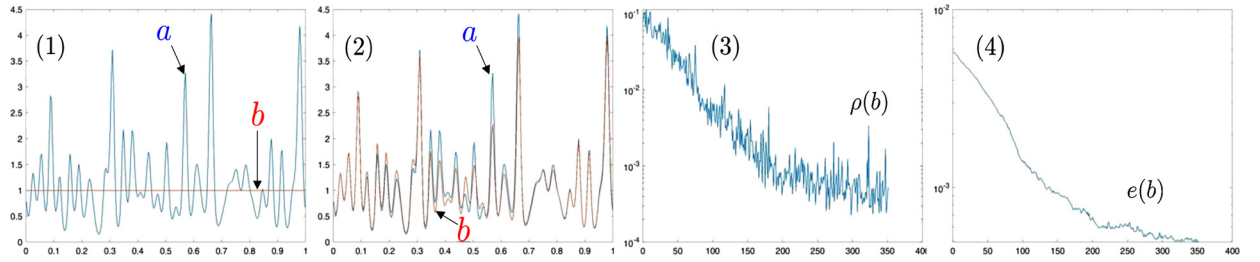

Fig. 6. (1) a and b for $n=1$ (2) a and b for $n=350$ (2) $\rho(b)$ vs n (4) $e(b)$ vs n.

Now let us consider the case where the interpolation points form a random subset of the discretization points. Take $N_f = 2^7$ and $N_c = 2^6$. Let $X = \{x_1, \ldots, x_{N_f}\}$ be a subset N_f distinct points of (the discretization points) $\{i/2^8 | i \in \mathcal{I}^{(8)}\}$ sampled with uniform distribution. Let $Z = \{z_1, \ldots, z_{N_c}\}$ be a subset of N_c distinct points of X sampled with uniform distribution. Write v_b^f for the interpolation of the data $(x_i, u(x_i))$ using the kernel G_b and write v_b^c for the interpolation of the data $(z_i, u(z_i))$ using the kernel G_b. Fig. 5.6 shows in (semilog scale) 20 independent random realizations of the values of $\rho(a) = \|v_a^f - v_a^c\|_a^2 / \|v_a^f\|_a^2$ and $\rho(b) = \|v_b^f - v_b^c\|_b^2 / \|v_b^f\|_b^2$. Fig. 5.7 shows in (semilog scale) 20 independent random realizations of the values of the prediction errors $e(a) \propto \|u - v_a^c\|_{L^2(\Omega)}$ and $e(b) \propto \|u - v_b^c\|_{L^2(\Omega)}$. Note again that the values of $\rho(a), e(a)$ are consistently and significantly lower than those of $\rho(b), e(b)$.

Fig. 6 provides a numerical illustration of an implementation of Algorithm 1 with $N_f = N = 2^7$, $N_c = 2^6$ and $n_c = 1$. In this implementation a, f and u are as in Fig. 5.1-3. The training data corresponds to N_f points $X = \{x_1, \ldots, x_{N_f}\}$ uniformly sampled (without replacement) from $\{i/2^8 | i \in \mathcal{I}^{(8)}\}$ (Since $N = N_f$ these points remain fixed during the execution the of the algorithm). n. The purpose of the algorithm is to learn the kernel G_a in the set of kernels $\{G_{b(W)} | W\}$ parameterized by the vector W via

$$\log b(W) = \sum_{i=1}^{2^6} (W_i^c \cos(2\pi i x) + W_i^s \sin(2\pi i x)) . \tag{5.2}$$

Using n to label its progression, Algorithm 1 is initialized at $n=1$ with the guess $b \equiv 1$ (i.e. $W \equiv 0$) (Fig. 6.1). At each step $(n \to n+1)$ the algorithm performs the following operations:

1. Select N_c points $Z = \{z_1, \ldots, z_{N_c}\}$ uniformly sampled (without replacement) from X.
2. Write v_b^f and v_f^c for the interpolation of the data $(x_i, u(x_i))$ and $(z_i, u(z_i))$ using the kernel G_b, and $\rho(W) = \|v_b^f - v_b^c\|_b^2 / \|v_b^f\|_b^2$. Compute the gradient $\nabla_W \rho(W)$ using Corollary 4.1 (and the identity $\partial_{W_i} \Theta(W) = -\pi_0 (A_0(W))^{-1} \times \partial_{W_i} A_0(W) (A_0(W))^{-1} \pi_0^T$ for $\Theta(W) = \pi_0 (A_0(W))^{-1} \pi_0^T$).
3. Update $W \to W - \lambda \nabla_W \rho(W)$ (with $\lambda \propto 0.01 / \|\nabla_W \rho(W)\|_{L^2}$).

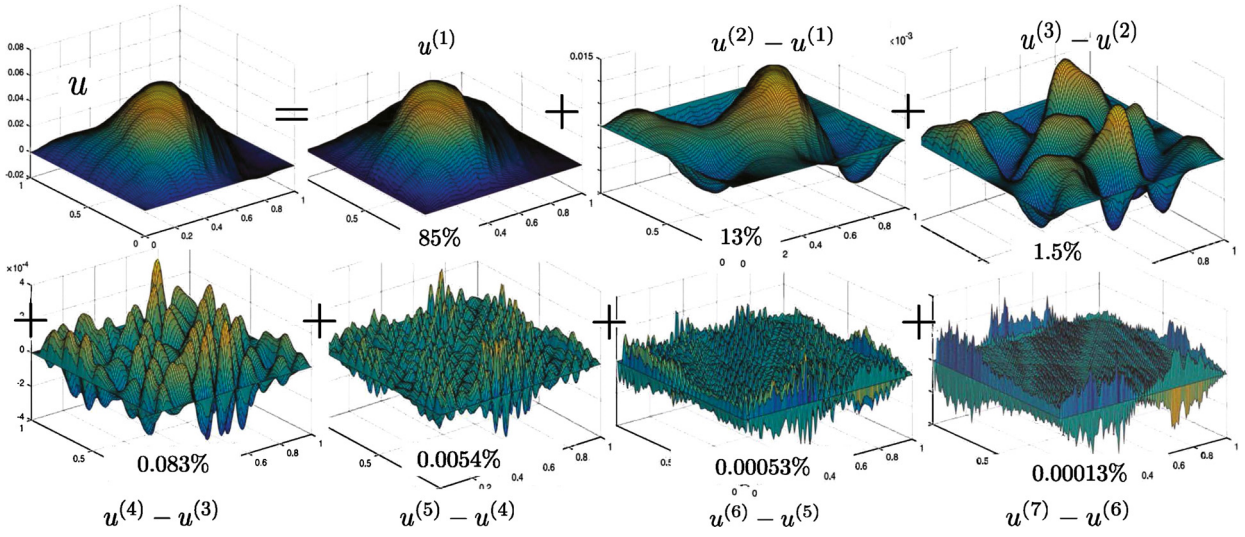

Fig. 7. u and $u^{(k)} - u^{(k-1)}$. Number below sub-figures show relative energy content $\|u^{(k)} - u^{(k-1)}\|^2/\|u\|^2$. Used from forthcoming book [21] with permission from Cambridge University Press.

Fig. 6.2 shows the value of b for $n = 350$. Fig. 6.3 shows the value of $\rho(b)$ vs n. Fig. 6.4 shows the value of the prediction error $e(b) \propto \|u - v_b^c\|_{L^2(\Omega)}$ vs n. The lack of smoothness of the plots of $\rho(b), e(b)$ vs n originate from the re-sampling of the set Z at each step n.

Remark 5.1. For $d \geq 1$, let $\Omega = (0,1)^d$ and for $k \geq 1$, let $\tau_i^{(k)}$ be a nested (in k) hierarchy of sub-cubes of $(0,1)^d$ with locations indexed by i. For $\mathcal{L} = -\operatorname{div}(a\nabla)$ let $\xi \sim \mathcal{N}(0, \mathcal{L}^{-1})$ and let $u^{(k)} := \mathbb{E}[\xi \mid \int_{\tau_i^{(k)}} \xi = \int_{\tau_i^{(k)}} \xi u \text{ for all } i]$. Fig. 7 shows (for $d = 2$) the corresponding increments $u^{(k)} - u^{(k-1)}$ and the relative energy content $\|u^{(k)} - u^{(k-1)}\|^2/\|u\|^2$ of each increment for a solution of (5.1) with $f \in L^2(\Omega)$. The quick decay of $\|u^{(k)} - u^{(k-1)}\|^2/\|u\|^2$ with respect to k illustrates the accuracy of the Green's function of (5.1) used as a kernel for interpolating partial linear measurements made on solutions of (5.1). This numerical homogenization phenomenon [21,18] is one motivation for minimizing ρ in the kernel identification problem described above ($\|u^{(k)} - u^{(k-1)}\|^2/\|u\|^2 \approx \rho_k$ with $\rho_k := \|u^{(k)} - u^{(k-1)}\|^2/\|u^{(k)}\|^2$).

6. Kernel Flows (KF)

6.1. Non parametric family of kernels and bottomless networks without guesswork

Composing a symmetric positive kernel with a function produces a symmetric positive kernel [4]. We will now use this property to learn a kernel from the data within a non-parametric family of kernels constructed by composing layers of functions. For $n \geq 2$ let $G_n : \mathcal{X} \to \mathcal{X}$ ($\mathcal{X}$ is the input space mentioned Pb. 1) be a sequence of functions determining the layers of this network. Let $\epsilon > 0$ be a small parameter, let $F_1 := I_d$ be the identity function and for $n \geq 2$, let F_n be the sequence of functions inductively defined by

$$F_{n+1} = (I_d + \epsilon G_{n+1}) \circ F_n \,. \tag{6.1}$$

Let K_n be the sequence of symmetric positive kernels obtained by composing a kernel K_1 with this sequence of functions, i.e. $K_n(x, x') = K_1(F_n(x), F_n(x'))$ and

$$K_{n+1}(x, x') = K_n\big(x + \epsilon G_{n+1}(x), x' + \epsilon G_{n+1}(x')\big) \,. \tag{6.2}$$

Our purpose is to use the training data $(x_1, y_1), \ldots, (x_N, y_N) \in \mathcal{X} \times \mathcal{Y}$ to learn the functions $G_1, \ldots, G_{n^*}$ and then approximate $u^\dagger$ with u_{n^*} obtained by interpolating a subset of the training data with K_{n^*}.

When applied to a classification problem with $n = 1$ the proposed algorithm is a support-vector network (in the sense of [3]) with kernel K_1. As n progresses the algorithm incrementally modifies the kernel via small perturbations of the identity operator (ϵG_{n+1} is reminiscent of the residual term of deep residual networks [8]). Since the training does not require any back propagation, achieving profound depths (with 10000 layers or more) is not difficult (since training is akin to simulating a stochastic dynamical system the network is essentially bottomless) and one purpose of this section is to explore properties of such bottomless networks (see [9] for a review of the motivations/challenges associated with the exploration of very deep networks).

6.2. The algorithm

We will adapt Algorithm 1 to learn the functions $G_1, \dots, G_n, \dots$ by induction over n. As in Sec. 4 let $N_f \leq N$ and $N_c = \text{round}(N_f/2)$.

For $n = 1$ let $x_i^{(n)} := x_i$ for $i \in \{1, \dots, N\}$.

Let $n \geq 1$ Assume $x_1^{(n)}, \dots, x_N^{(n)}$ to be known. Let $s_{f,n}(1), \dots, s_{f,n}(N_f)$ be N_f distinct elements of $\{1, \dots, N\}$ obtained through random sampling (with uniform distribution) without replacement. Let $s_{c,n}(1), \dots, s_{c,n}(N_c)$ be N_c distinct elements of $\{1, \dots, N_f\}$ also obtained through random sampling (with uniform distribution) without replacement. Let π be the corresponding $N_c \times N_f$ sub-sampling matrix defined by $\pi_{i,j}^{(n)} = \delta_{s_{c,n}(i),j}$. Let $y_f^{(n)} \in \mathbb{R}^{N_f}$ and $y_c^{(n)} \in \mathbb{R}^{N_c}$ be the corresponding subvectors of y defined by $y_{f,i}^{(n)} = y_{s_{f,n}(i)}$ and $y_{c,i}^{(n)} = y_{f,s_{c,n}(i)}$. For $i \in \{1, \dots, N_f\}$ let $x_{f,i}^{(n)} := x_{s_{f,n}(i)}^{(n)}$ and write $\Theta^{(n)}$ for the $N_f \times N_f$ matrix with entries

$$\Theta_{i,j}^{(n)} = K_1(x_{f,i}^{(n)}, x_{f,j}^{(n)}), \tag{6.3}$$

and let

$$\rho(n) := 1 - \frac{(y_c^{(n)})^T (\pi^{(n)} \Theta^{(n)} (\pi^{(n)})^T)^{-1} y_c^{(n)}}{(y_f^{(n)})^T (\Theta^{(n)})^{-1} y_f^{(n)}}. \tag{6.4}$$

Let $\hat{y}_f^{(n)} := (\Theta^{(n)})^{-1} y_f^{(n)}$, $\hat{z}_f^{(n)} := (\pi^{(n)})^T (\pi^{(n)} \Theta^{(n)} (\pi^{(n)})^T)^{-1} \pi^{(n)} y_f^{(n)}$ and for $i \in \{1, \dots, N_f\}$ let

$$\hat{g}_{f,i}^{(n)} := 2 \frac{(1 - \rho(n)) \hat{y}_{f,i}^{(n)} (\nabla_x K_1)(x_{f,i}^{(n)}, x_{f,\cdot}^{(n)}) \hat{y}_f^{(n)} - \hat{z}_{f,i}^{(n)} (\nabla_x K_1)(x_{f,i}^{(n)}, x_{f,\cdot}^{(n)}) \hat{z}_f^{(n)}}{y_f^T (\Theta^{(n)})^{-1} y_f} \tag{6.5}$$

Let G_{n+1} be the function obtained by interpolating the data $(x_{f,i}^{(n)}, \hat{g}_{f,i}^{(n)})$ with the kernel K_1, i.e.

$$G_{n+1}(x) = (\hat{g}_{f,\cdot}^{(n)})^T \left(K_1(x_{f,\cdot}^{(n)}, x_{f,\cdot}^{(n)})\right)^{-1} K_1(x_{f,\cdot}^{(n)}, x). \tag{6.6}$$

Note that $G^{(n+1)}(x_{f,i}^{(n)}) = \hat{g}_{f,i}^{(n)}$. For $i \in \{1, \dots, N\}$, let

$$x_i^{(n+1)} = x_i^{(n)} + \epsilon G_{n+1}(x_i^{(n)}). \tag{6.7}$$

Remark 6.1. In the description above the input space $\mathcal{X}$ (of the function u to be interpolated) is assumed to a finite-dimensional vector space and the output space $\mathcal{Y}$ (of the function u to be interpolated) is assumed to be contained in the real line $\mathbb{R}$. If the output space $\mathcal{Y}$ is a finite-dimensional vector space (e.g. $\mathbb{R}^{d_{\mathcal{Y}}}$) then $y_f^{(n)}$ and $y_c^{(n)}$ are $N_f \times d_{\mathcal{Y}}$ and $N_c \times d_{\mathcal{Y}}$ matrices and (6.4) and (6.5) must be replaced by

$$\rho(n) := 1 - \frac{\text{Tr}\left[(y_c^{(n)})^T (\pi^{(n)} \Theta^{(n)} (\pi^{(n)})^T)^{-1} y_c^{(n)}\right]}{\text{Tr}\left[(y_f^{(n)})^T (\Theta^{(n)})^{-1} y_f^{(n)}\right]}. \tag{6.8}$$

and

$$\hat{g}_{f,i}^{(n)} := 2 \frac{\text{Tr}\left[(1 - \rho(n)) \hat{y}_{f,i}^{(n)} (\nabla_x K_1)(x_{f,i}^{(n)}, x_{f,\cdot}^{(n)}) \hat{y}_f^{(n)} - \hat{z}_{f,i}^{(n)} (\nabla_x K_1)(x_{f,i}^{(n)}, x_{f,\cdot}^{(n)}) \hat{z}_f^{(n)}\right]}{\text{Tr}\left[y_f^T (\Theta^{(n)})^{-1} y_f\right]}, \tag{6.9}$$

where, in (6.9), $\hat{y}_{f,i}^{(n)} \in \mathbb{R}^{d_{\mathcal{Y}}}$, $\hat{y}_f^{(n)} \in \mathbb{R}^{N_f \times d_{\mathcal{Y}}}$, $(\nabla_x K_1)(x_{f,i}^{(n)}, x_{f,\cdot}^{(n)}) \in \mathbb{R}^{d_{\mathcal{X}} \times N_f}$ (writing $d_{\mathcal{X}}$ for the dimension of the input space, $\nabla_x K_1$ refers to the gradient over the first component of K_1). The product results in a $d_{\mathcal{Y}} \times d_{\mathcal{X}} \times d_{\mathcal{Y}}$ tensor and the trace (taken with respect to the d_Y dimensions) results in a vector in $\mathbb{R}^{d_{\mathcal{X}}}$, i.e. writing x_s for the sth entry of x,

$$(\hat{g}_{f,i}^{(n)})_s = 2 \frac{\sum_{l=1}^{d_{\mathcal{Y}}} \sum_{t=1}^{N_f} (1 - \rho(n)) (\hat{y}_f^{(n)})_{i,l} \partial_{x_s} K_1(x_{f,i}^{(n)}, x_{f,t}^{(n)}) (\hat{y}_f^{(n)})_{t,l} - (\hat{z}_f^{(n)})_{i,l} \partial_{x_s} K_1(x_{f,i}^{(n)}, x_{f,t}^{(n)}) (\hat{z}_f^{(n)})_{t,l}}{\text{Tr}\left[y_f^T (\Theta^{(n)})^{-1} y_f\right]}.$$

This simple modification (via the trace operator) is equivalent to endowing the space of functions $v = (v_1, \dots, v_{d_{\mathcal{Y}}})$ mapping $\mathcal{X}$ to $\mathcal{Y}$ with the RKHS norm $\|v\|^2 = \sum_{i=1}^{d_{\mathcal{Y}}} \|v_i\|^2$ with $\|v_i\|^2 = \sup_\phi \frac{(\int_{\mathcal{X}} \phi(x) v_i(x)\, dx)^2}{\int_{\mathcal{X}^2} \phi(x) K_n(x,x') \phi(x')\, dx\, dx'}$ and using that norm to compute ρ and its gradient.

6.3. Rationale of the algorithm

Observe that the algorithm is randomized through the random samples $s_{f,n}(i)$ and $s_{c,n}(i)$ (taking values in the training data). Observe also that, given those random samples, the functions G_n, F_n and kernels K_n are entirely determined by the values of the learning parameters $\hat{g}_{f,i}^{(n)} = G^{(n+1)}(x_{f,i}^{(n)})$.

As in Algorithm 1, the $\hat{g}_{f,i}^{(n)}$ are selected in (6.5) in the direction of the gradient descent of the ratio ρ: we apply Proposition 3.2 to compute the Fréchet derivative of $\rho(n)$ (6.4) with respect to small perturbations $x_{f,i}^{(n)} + \epsilon g_{f,i}^{(n)}$ to the $x_{f,i}^{(n)}$ and select the $g_{f,i}^{(n)}$ in the direction of the gradient descent.

More precisely let $g_{f,i}^{(n)}$ be candidates for the values of $G^{(n+1)}(x_{f,i}^{(n)})$ and write $\tilde{x}_{f,i}^{(n+1)} = x_{f,i}^{(n)} + \epsilon g_{f,i}^{(n)}$. Write $\tilde{\Theta}^{(n+1)}$ for the $N_f \times N_f$ matrix with entries

$$\tilde{\Theta}_{i,j}^{(n+1)} = K_1(\tilde{x}_{f,i}^{(n+1)}, \tilde{x}_{f,j}^{(n+1)}), \tag{6.10}$$

and let

$$\tilde{\rho}(n+1) := 1 - \frac{(y_c^{(n)})^T (\pi^{(n)} \tilde{\Theta}^{(n+1)} (\pi^{(n)})^T)^{-1} y_c^{(n)}}{(y_f^{(n)})^T (\tilde{\Theta}^{(n+1)})^{-1} y_f^{(n)}}. \tag{6.11}$$

Then the following proposition identifies $\hat{g}_{f,i}^{(n)}$ as the direction of the gradient descent of $\tilde{\rho}(n+1)$ with respect to the parameters $g_{f,i}^{(n)}$.

Proposition 6.2. *It holds true that*

$$\tilde{\rho}(n+1) = \rho(n) - \epsilon \sum_{i=1}^{N_f} (g_{f,i}^{(n)})^T \hat{g}_{f,i}^{(n)} + \mathcal{O}(\epsilon^2), \tag{6.12}$$

where the $\hat{g}_{f,i}^{(n)}$ are as in (6.5).

Proof. We deduce from (6.2) that, to the first order in ϵ,

$$\begin{aligned} K_{n+1}(x, x') \approx & K_n\big(x, x'\big) + \epsilon \big((G_{n+1} \circ F_n(x))^T (\nabla_x K_1)\big(F_n(x), F_n(x')\big) \\ & + (G_{n+1} \circ F_n(x'))^T (\nabla_{x'} K_1)\big(F_n(x), F_n(x')\big). \end{aligned} \tag{6.13}$$

Therefore, using the symmetry of K_1, we have

$$\tilde{\Theta}^{(n+1)} = \Theta^{(n)} + \epsilon T^{(n)} + \mathcal{O}(\epsilon^2) \tag{6.14}$$

with

$$T_{i,j}^{(n)} = (g_{f,i}^{(n)})^T (\nabla_x K_1)(x_{f,i}^{(n)}, x_{f,j}^{(n)}) + (g_{f,j}^{(n)})^T (\nabla_x K_1)\big(x_{f,j}^{(n)}, x_{f,i}^{(n)}\big). \tag{6.15}$$

We deduce from Proposition 3.2 that

$$\rho(\tilde{\Theta}^{(n+1)}) = \rho(\Theta^{(n)}) - \epsilon \frac{(1 - \rho(\Theta^{(n)}))(\hat{y}_f^{(n)})^T T^{(n)} \hat{y}_f^{(n)} - (\hat{z}_f^{(n)})^T T^{(n)} \hat{z}_f^{(n)}}{(\hat{y}_f^{(n)})^T \Theta^{(n)} \hat{y}_f^{(n)}} + \mathcal{O}(\epsilon^2), \tag{6.16}$$

which implies the result after simplification. □

The following corollary is a direct application of Proposition 6.2.

Corollary 6.3. *Let $K_1(x, x') = e^{-\gamma |x - x'|^2}$. It holds true that*

$$\rho(\tilde{\Theta}^{(n+1)}) = \rho(\Theta^{(n)}) - \epsilon \sum_{i=1}^{N_f} (g_i^{(n)})^T \hat{g}_i^{(n)} + \mathcal{O}(\epsilon^2), \tag{6.17}$$

with $\hat{g}_i^{(n)} := \frac{4\gamma}{(y_f^{(n)})^T (\Theta^{(n)})^{-1} y_f^{(n)}} \sum_{j=1}^{N_f} \Gamma_{i,j}^{(n)} x_{f,j}^{(n)}$,

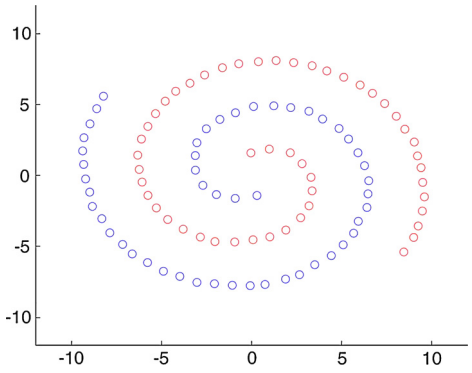

Fig. 8. Swiss Roll Cheesecake. $N = 100$. Red points have label -1 and blue points have label 1.

$$\begin{aligned}\Gamma^{(n)}_{i,j} =& \delta_{i,j}\hat{z}^{(n)}_{f,i}(\Theta^{(n)}\hat{z}^{(n)})_{f,i} - \hat{z}^{(n)}_{f,i}\Theta^{(n)}_{i,j}\hat{z}^{(n)}_{f,j} \\ & - (1-\rho(\Theta^{(n)}))\delta_{i,j}\hat{y}^{(n)}_{f,i}y^{(n)}_{f,i} + (1-\rho(\Theta^{(n)}))\hat{y}^{(n)}_{f,i}\Theta^{(n)}_{i,j}\hat{y}^{(n)}_{f,j}.\end{aligned} \tag{6.18}$$

and $\hat{z}^{(n)}_f = (\pi^{(n)})^T(\pi^{(n)}\Theta^{(n)}(\pi^{(n)})^T)^{-1}\pi^{(n)}y_f$, $\hat{y}^{(n)}_f = (\Theta^{(n)})^{-1}y^{(n)}_f$.

7. The flow of the KF algorithm on the Swiss Roll Cheesecake

From a numerical analysis perspective, the flow $F_n(x)$ associated with the Sec. 6 KF algorithm approximates, in the sense of Smoothed Particle Hydrodynamics [7], a flow $F(t,x)$ mapping $\mathbb{R}\times\mathcal{X}$ into $\mathcal{X}$. Writing x_i for the N training data points in $\mathcal{X}$, as $\epsilon \downarrow 0$, $F_n(x_i)$ approximates $X_i(t) := F(t,x_i)$ which (after averaging the effect of the randomized batches) can be identified as the solution of a system of ODEs of the form

$$\frac{dX}{dt} = \mathcal{G}(X, N, N_f, K_1)\,. \tag{7.1}$$

7.1. Implementation of the KF algorithm

We will now explore a few properties of this approximation by implementing the KF algorithm for the Swiss Roll Cheesecake illustrated in Fig. 8. The dataset is composed of $N = 100$ points x_i in $\mathbb{R}^2$ in the shape of two concentric spirals. Red points have label -1 and blue points have label 1. Since our purpose is limited to illustrating properties of the discrete flow $F_n(x)$ associated with the Kernel Flow algorithm we will consider all those points as training points (will not introduce a test dataset) and visualise the trajectories $n \to F_n(x_i)$ of the data points x_i.

The KF algorithm is implemented with the Gaussian kernel of Corollary 6.3 with $\gamma^{-1} = 4$. Training is done in random batches of size N_f and we use $N_c = N_f/2$ to compute the ratio ρ and learn the parameters of the network. We start with $N_f = N$ and as training progresses points of the same color start merging. Therefore to avoid near singular matrices caused by batches with points of the same color sharing nearly identical coordinates, once the distance between two points of the same color is smaller than 10^{-4} (in Fig. 3, 9 and 10) we drop one of them out of the set of possible candidates for the batch and decrease N_f by 1 (the point left out remains advected by ϵG_{n+1} but is simply no longer available as an interpolation point defining G_{n+1}).

Fig. 3 shows $F_n(x_i)$ vs n for $1 \le n \le 7000$. In Fig. 3 the value of ϵ is chosen at each step n so that the perturbation of each data point x_i of the batch is no greater than p% with $p = 10$ for $1 \le n \le 1000$ and $p = 10/\sqrt{n/1000}$ for $n \ge 1000$. The two spirals quickly unroll into two linearly separable clusters.

Fig. 9 shows $F_n(x_i)$ vs n for $1 \le n \le 500000$. The value of ϵ is chosen at each step n so that the perturbation of each data point x_i of the batch is no greater than 0.5%. The two intertwined spirals unroll into straight (vibrating) segments. The final unrolled configuration appears to be unstable (some red points are ejected out of the unrolled red segment towards the end of the simulation) and although this instability seems to be alleviated by adjusting ϵ to a smaller value at the end of the simulation it seems to also be caused by a combination of (1) the stiffness of the flow being simulated (2) the process of permanently removing points from the pool of possible candidates for the batch. Although these points remain advected by the flow and are initially at distance less than 10^{-4} of a point of the same color, the stiffness of the flow can quickly increase this distance and separate the point from its group if ϵ is not small enough.

7.2. Addition of nuggets

The instabilities observed in Fig. 9 seem to vanish (even with significantly larger values of ϵ) after the addition of a nugget (white noise kernel accounting for measurement noise in kriging) to the kernel K_1. In Fig. 10 we consider a longer

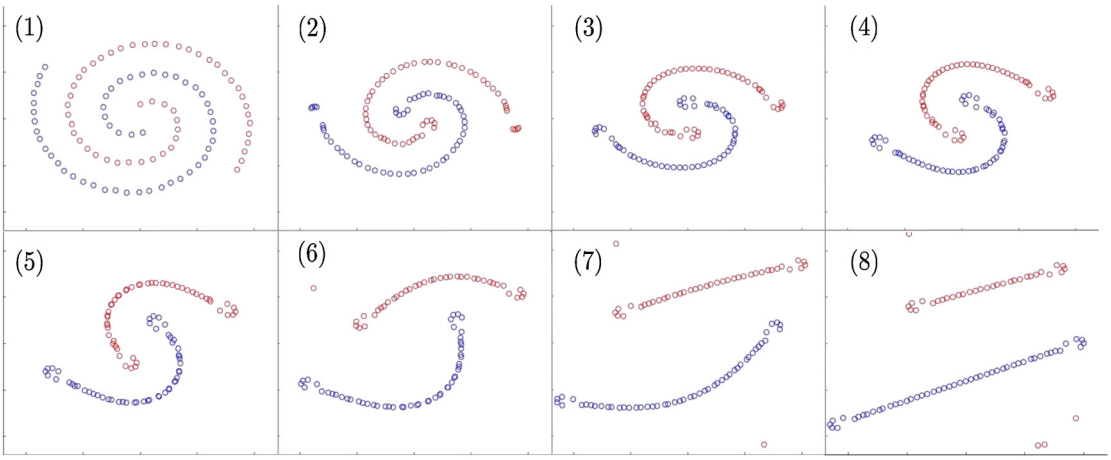

Fig. 9. $F_n(x_i)$ for 8 different values of n. ϵ is chosen at each step n so that the perturbation of each data point x_i of the batch is no greater than 0.5%.

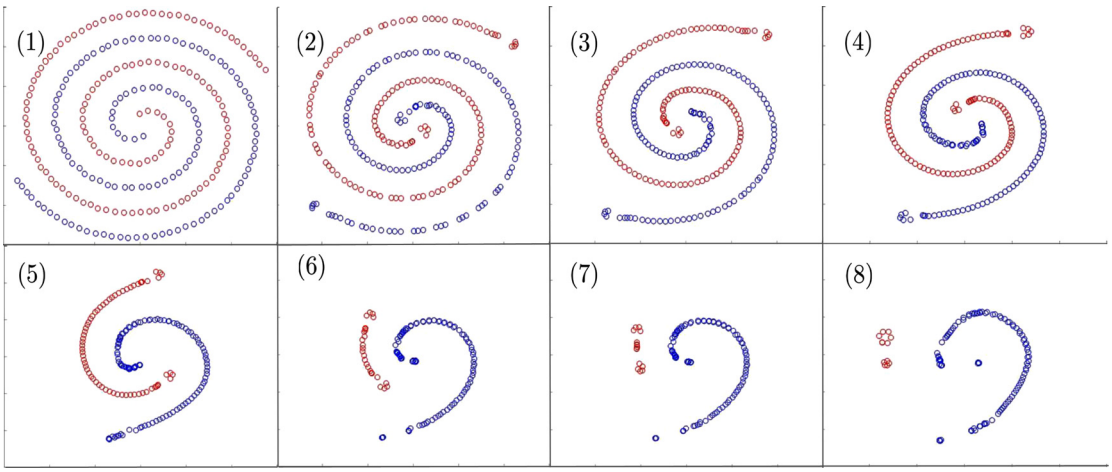

Fig. 10. $N = 250$. $F_n(x_i)$ for 8 different values of n. ϵ is chosen at each step n so that the absolute perturbation of each data point x_i of the batch is no greater than 0.05. $K_1(x,x') = e^{-\gamma|x-x'|^2} + \delta(x-x')e^{-\gamma 6^2}$.

version of the Swiss Roll Cheesecake ($N = 250$). The kernel is $K_1(x,x') = e^{-\gamma|x-x'|^2} + \delta(x-x')e^{-\gamma 6^2}$ ($\gamma^{-1} = 4$). ϵ is chosen at each step n so that the absolute perturbation (maximum translation) of each data point x_i of the batch is no greater than 0.05. Fig. 10 shows $F_n(x_i)$ vs n for $1 \leq n \leq 500000$. The two intertwined spirals unroll into stable clusters slowly drifting away from each other.

7.3. Instantaneous and average vector fields

With the addition of the nugget, the permanent removal of close points from the pool of candidates is no longer required for avoiding singular matrices. In Figs. 11, 12 and 13 (see [25] for videos) we consider the $N = 100$ version of the Swiss Roll Cheesecake. The kernel is $K_1(x,x') = e^{-\gamma|x-x'|^2} + \delta(x-x')e^{-\gamma 6^2}$ ($\gamma^{-1} = 4$). ϵ is chosen at each step n so that the absolute perturbation (maximum translation) of each data point x_i of the batch is no greater than 0.2. Only a nugget is added and close points are not removed from the pool of candidates (which eliminates the instabilities observed in Fig. 9). Fig. 11 shows $F_n(x_i)$ vs n for $1 \leq n \leq 180000$ and the decision boundary between the two classes vs n. The two intertwined spirals unroll into stable clusters. Fig. 12 shows the instantaneous velocity field $F_{n+1}(x) - F_n(x)$. Fig. 13 shows the average velocity field $10(F_{n+300}(x) - F_n(x))/300$. The difference between the instantaneous and average velocity fields is and indication of the presence of multiple time scales caused by the randomization process and the stiffness of the underlying flow.

7.4. The continuous flow

The intriguing behavior of these flows, calls for their investigation from the numerical integration perspective (note that an ODE perspective emerges as $\epsilon \downarrow 0$, an SDE perspective is relevant when ϵ is non-null due to the randomization of the

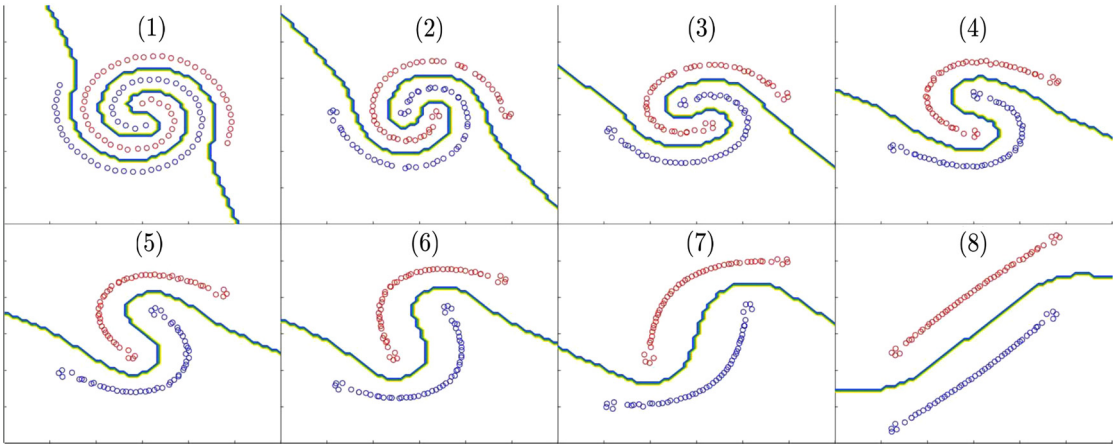

Fig. 11. $F_n(x_i)$ and decision boundary for 8 different values of n.

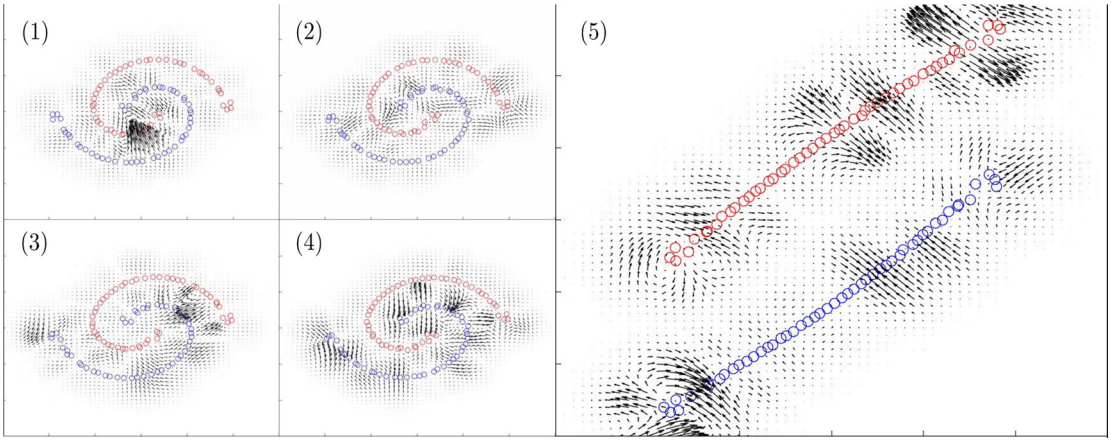

Fig. 12. Instantaneous velocity field $F_{n+1}(x) - F_n(x)$. (1-4) show 4 successive values. (5) shows the instantaneous velocity field for the final configuration.

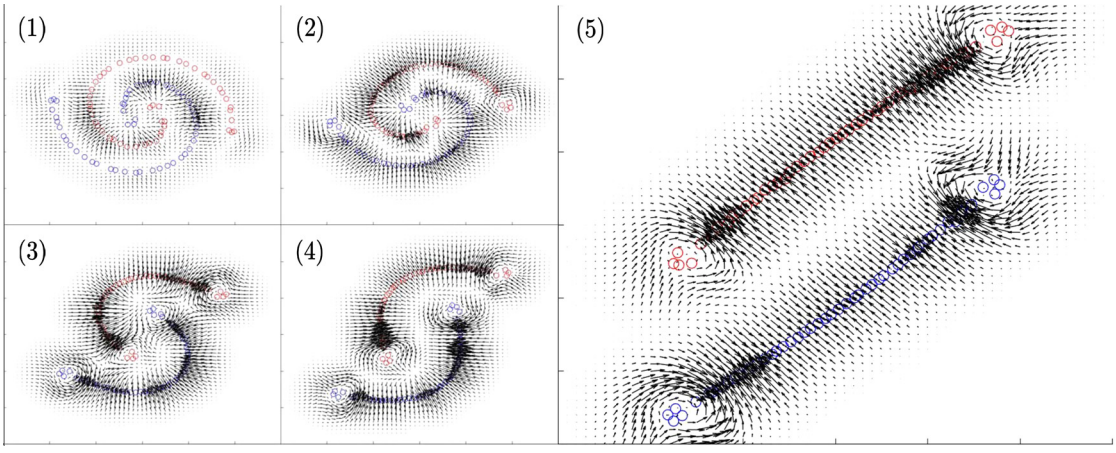

Fig. 13. Average velocity field $10(F_{n+300}(x) - F_n(x))/300$ for 5 different values of n.

batches, a PDE perspective emerges as $\epsilon \downarrow 0$ and $N \to \infty$, and an SPDE approximation perspective is relevant when ϵ is non-null and N is finite).

Note that the KF flow $F_n(x)$ can be seen (as $\epsilon \downarrow 0$) as the numerical approximation of a continuous flow $F(t, x)$. Identified as the solution of the dynamical system

Table 1
MNIST test errors using N_I interpolation points.

N_I	Average error	Min error	Max error	Standard deviation
6000	0.014	0.0136	0.0143	1.44×10^{-4}
600	0.014	0.0137	0.0142	9.79×10^{-5}
60	0.0141	0.0136	0.0146	2.03×10^{-4}
10	0.015	0.0136	0.0177	7.13×10^{-4}

$$\frac{\partial F(t,x)}{\partial t} = -\mathbb{E}_{X,\pi}\Big[K_1\big(F(t,x),Z\big)\big(K_1(Z,Z)\big)^{-1}\big(\nabla_Z \rho(X,Z,\pi)\big)\big|_{Z=F(X,t)}\Big], \tag{7.2}$$

with initial condition $F(0,x)=x$ and where the elements of (7.2) are defined as follows. X is a random vector of $\mathcal{X}^{N_f}$ representing the random sampling of the training data in a batch size N_f. Writing $u(X)\in\mathcal{Y}^{N_f}$ for the vector whose entries are the labels of the entries of $X\in\mathcal{X}^{N_f}$ and π for a random $N_c\times N_f$ matrix corresponding to the selection of N_c elements at out N_f (at random, uniformly, without replacement), ρ, in (7.2), is defined as follows

$$\rho(X,Z,\pi) = 1 - \frac{u(X)^T\pi^T(K_1(\pi Z,\pi Z))^{-1}\pi u(X)}{u(X)^T(K_1(Z,Z))^{-1}u(X)}. \tag{7.3}$$

The average vector field in Fig. 13 is an approximation of the right hand side of (7.2) and the convergence of $F_{\text{round}(t/\epsilon)}(x)$ towards $F(t,x)$ as $\downarrow 0$ is in the sense of two-scale flow convergence described in [33].

Consider now the case where X is the vector containing the location of all the training data points and Y is the corresponding vector of labels. Then (7.2) and (7.3) become

$$\frac{\partial F(t,x)}{\partial t} = -\mathbb{E}_{\pi}\Big[K_1\big(F(t,x),Z\big)\big(K_1(Z,Z)\big)^{-1}\big(\nabla_Z \rho(Z,\pi)\big)\big|_{Z=F(X,t)}\Big], \tag{7.4}$$

and

$$\rho(Z,\pi) = 1 - \frac{Y^T\pi^T(K_1(\pi Z,\pi Z))^{-1}\pi Y}{Y^T(K_1(Z,Z))^{-1}Y}. \tag{7.5}$$

Note that $Z := F(t,X)$ solves

$$\frac{dZ}{dt} = -\nabla_Z \mathbb{E}_{\pi}\big[\rho(Z,\pi)\big]. \tag{7.6}$$

Note that $F(t,x)$ is not a function of $Z(t)$ but a function of $(Z(s))_{0\le s\le t}$. Furthermore we have,

$$\partial_t F(t,x) = K_1\big(F(t,x),Z\big)\big(K_1(Z,Z)\big)^{-1}\frac{dZ}{dt}. \tag{7.7}$$

8. Numerical experiments with the MNIST dataset

We will now implement, test and analyze the Sec. 6 KF algorithm on the MNIST dataset [37]. This training set is composed of 60000, 28×28 images of handwritten digits (partitioned into 10 classes) with a corresponding vector of 60000 labels (with values in $\{1,\dots,9,0\}$). The test set is composed of 10000, 28×28 images of handwritten digits with a corresponding vector of 10000 labels.

8.1. Learning with small random batches of the full training dataset

We first implement the Sec. 6 KF algorithm with the full training set, i.e. $N = 60000$. Images are normalized to have L^2 norm one and we use the Gaussian kernel of Corollary 6.3 and set γ^{-1} equal to the mean squared distance between training images. Training is performed in random batches of size $N_f = 600$ and we use $N_c = 300$ to compute the ratio ρ and learn the parameters of the network (we do not use a nugget and we do not exclude points that are too close from those batches). The value of ϵ is chosen at each step n so that the perturbation of each data point x_i of the batch is no greater than 1% ($\epsilon = 0.01\times \max_i \frac{|x_{f,i}^{(n)}|_{L^2}}{|\hat{g}_{f,i}^{(n)}|_{L^2}}$).

Classification of the test data is performed by interpolating a subset of N_I images/labels (x_i, y_i) of the training data with the kernel K_n (the kernel at step/layer n in the Sec. 6.2 Kernel Flow algorithm). Here each x_i is a 28×28 image and each y_i is a unit vector e_j in $\mathbb{R}^{10}$ pointing in the direction of the class of the images (e.g. $y_i = (1,0,0,0,0,0,0,0,0,0) = e_1$ if the class of image x_i is $j=1$ and $y_i = (0,0,0,0,0,0,0,0,0,1) = e_{10}$ if the class of image x_i is $j=0$). The interpolant u_n is a function from $\mathbb{R}^{28\times 28}$ to $\mathbb{R}^{10}$ and the class of an image x is simply identified as $\text{argmax}_j\, e_j^T\cdot u_n(x)$. The Sec. 6.2 Kernel Flow

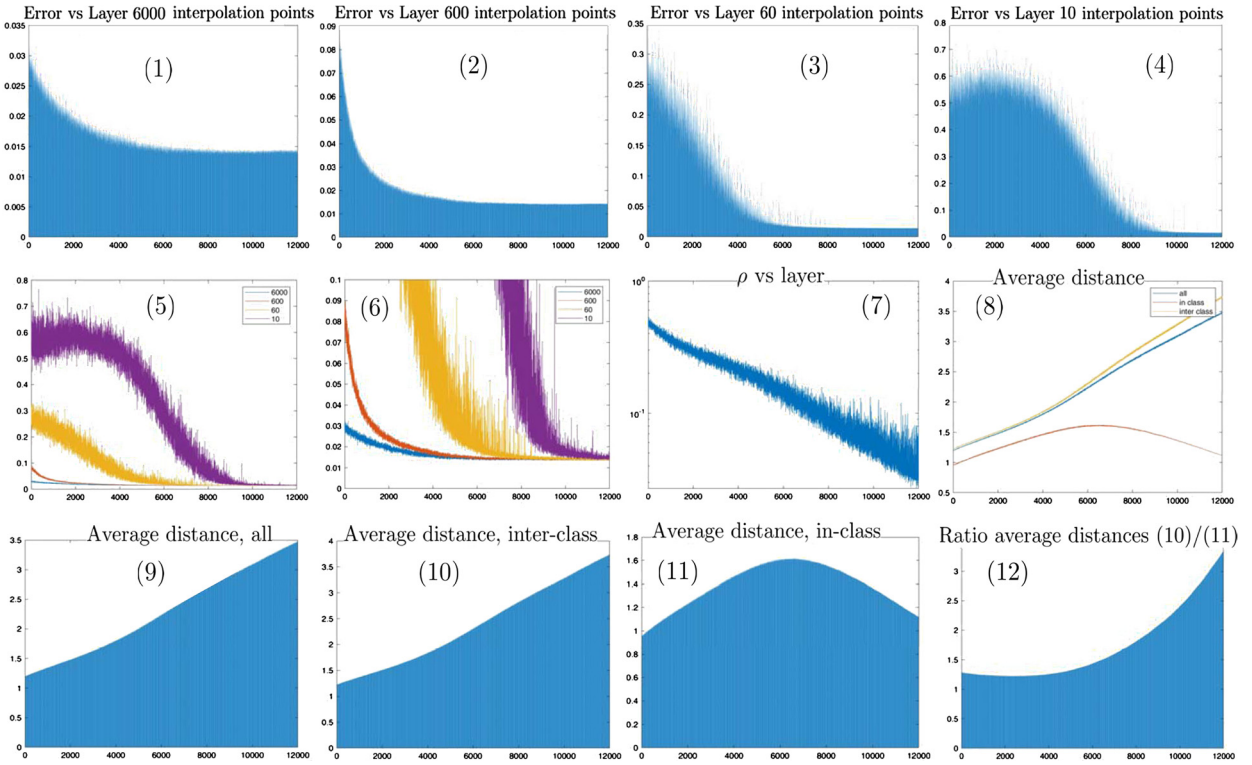

Fig. 14. Results for MNIST. $N = 60000$, $N_f = 600$ and $N_c = 300$. (1) Test error vs depth n with $N_I = 6000$ (2) Test error vs depth n with $N_I = 600$ (3) Test error vs depth n with $N_I = 60$ (4) Test error vs depth n with $N_I = 10$ (5, 6) Test error vs depth n with $N_I = 6000, 600, 60, 10$ (7) ρ vs depth n (8) Mean-squared distances between images $F_n(x_i)$ (all, inter class and in class) vs depth n (9) Mean-squared distances between images (all) vs depth n (10) Mean-squared distances between images (inter class) vs depth n (11) Mean-squared distances between images (in class) vs depth n (12) Ratio (10)/(11).

algorithm is implemented (using Remark 6.1) with $N = 60000$, $N_f = 600$ and $N_c = 300$ and ended for $n = 12000$ (resulting in a network with 12000 layers).

Table 1 shows test errors (with the test data composed of 10000 images) obtained with $N_I = 6000, 600, 60$ and 10 interpolation points (selected at random uniformly without replacement and conditioned on containing an equal number of example from each class to avoid degeneracy for $N_I = 10, 60$). The second column shows errors averaged over the last 100 layers of the network (i.e. obtained by interpolating the N_I data points with K_n for $n = 11901, 11902, \ldots, 12000$). The third, fourth and fifth columns show the min, max and standard deviation of the error over the same last 100 layers of the network. Surprisingly, around layer $n = 12000$, the kernel K_n achieves an average error of about 1.5% with only 10 data points (by using only 1 random example of each digit as an interpolation point). Multiple runs of the algorithm suggest that those results are stable.

Fig. 14 shows test errors vs depth n (with $N_I = 6000, 600, 60, 10$ interpolation points), the value of the ratio ρ vs n (computed with $N_f = 600$ and $N_c = 300$) and the mean squared distances between (all, inter class and in class) images $F_n(x_i)$ vs n. Observe that all mean-squared distances increase until $n \approx 7000$. After $n \approx 7000$ the in class mean-squared distances decreases with n whereas the inter-class mean-squared distances continue increasing. This suggests that after $n \approx 7000$ the algorithm starts clustering the data per class. Note also that while the test errors, with $N_I = 6000, 600$ interpolation points, decrease immediately and sharply, the test errors with $N_I = 10$ interpolation points increase slightly until $n \approx 3000$ towards 60%, after which they drop and seem to stabilize around 1.5% towards $n \approx 10000$.

It is known that iterated random functions typically converge because they are contractive on average [5,6]. Here training appears to create iterated functions that are contractive with each class but expansive between classes.

Fig. 15 shows $10 \times x_i$ and $10 \times F_n(x_i)$ and $20 \times (F_n(x_i) - x_i)$ for $n = 12000$, training images and test images. The algorithm appears to introduce small, archetypical, and class dependent, perturbations in those images.

8.2. Bootstrapping, brittleness and data archetypes

In Sec. 8.1 we used the whole training set of $N = 60000$ images to train our network and Fig. 14. Although the accuracy the network increases (between $n = 1$ and $n = 12000$) when using only $N_I = 6000, 600, 60, 10$ interpolation points, the accuracy of the network with $N_I = 60000$ (the full training set as interpolation points) does not seem to be significantly impacted by the training (the error with $N_I = 60000$ is 0.0128 at $n = 1$ and $n = 0.0144$ at $n = 12000$). In that sense, all the algorithm seems to do is to transfer the information contained in the 60000 data-points to the kernel K_n. Can the accuracy

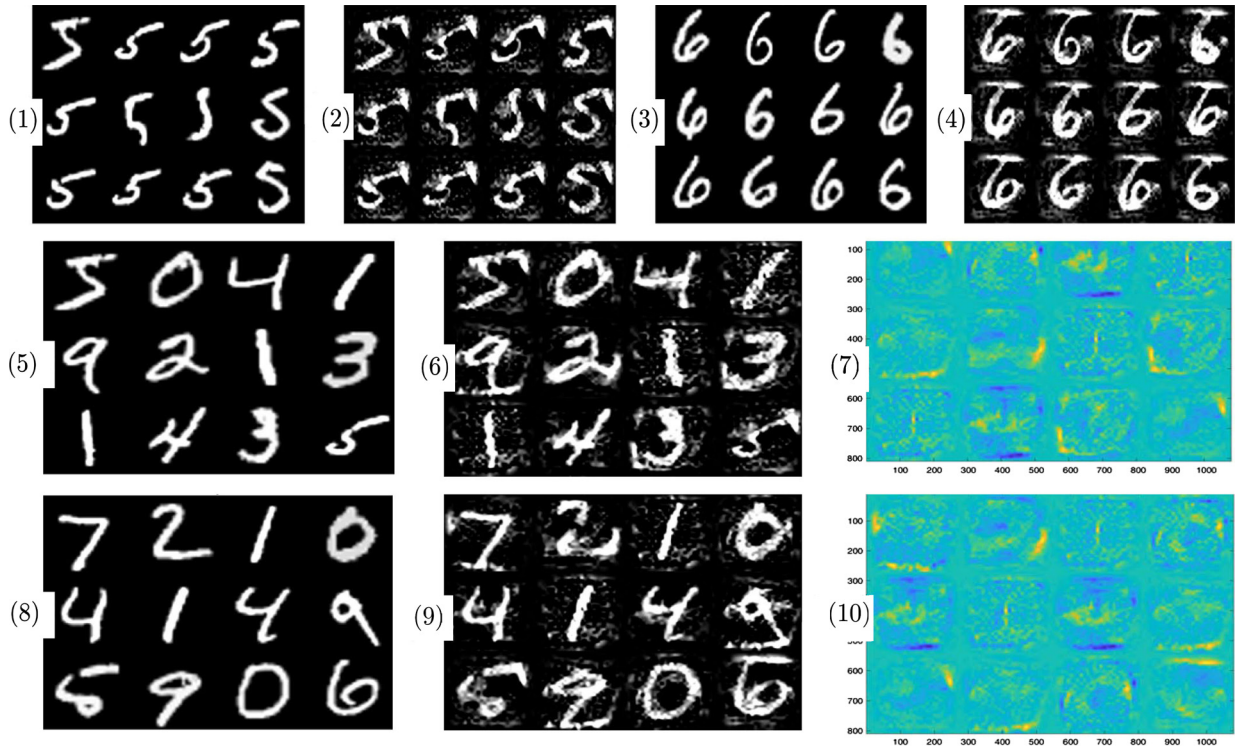

Fig. 15. Results for MNIST. $N = 60000$, $N_f = 600$ and $N_c = 300$. (1, 3, 5) Training data x_i (2, 4, 6) $F_n(x_i)$ for $n = 12000$ (7) $F_n(x_i) - x_i$ for training data and $n = 12000$ (8) Test data x_i (9) $F_n(x_i)$ for test data and $n = 12000$ (10) $F_n(x_i) - x_i$ for test data and $n = 12000$.

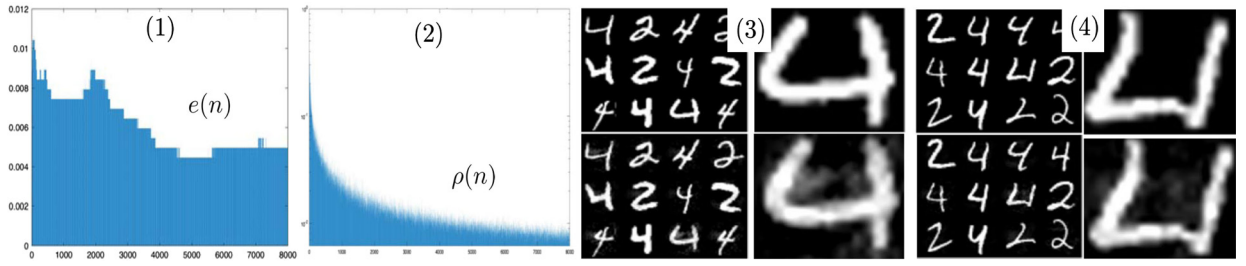

Fig. 16. (1) Test error vs n (2) ρ vs n (3) x_i and $F_n(x_i)$ for $n = 8000$ and i corresponding to the first 12 training images (4) x_i and $F_n(x_i)$ for $n = 8000$ and i corresponding to the first 12 test images. MNIST with classes 2 and 4, 600 training images and 100 test images.

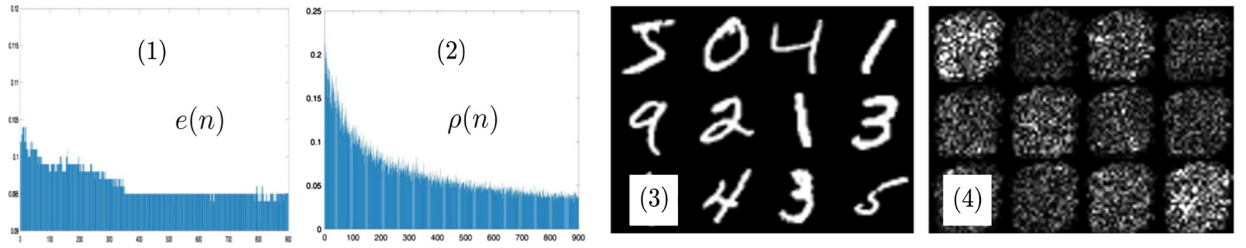

Fig. 17. MNIST with all classes, 1200 training images and 2000 test images. (1) Test error vs n (2) ρ vs n (3) Training images x_i (4) $10 \times \text{abs}(F_n(x_i) - x_i)$ for $n = 900$.

of the interpolation with the full training set be improved? Can the algorithm extract information that cannot already be extracted by performing a simple interpolation (with a simple Gaussian kernel) with the full training dataset? To answer these questions we will now implement the Sec. 6.2 Kernel Flow algorithm with subsets of training and test images and train the network with $N_f = N$ (the size of each batch is equal to the total number of training images), $N_c = N_f/2$ and possibly subsets of the set of all classes. We work with raw images (not normalized to have L^2 norm one). We use the Gaussian kernel of Corollary 6.3 and identify γ^{-1} as the mean squared distance between training images. We first take

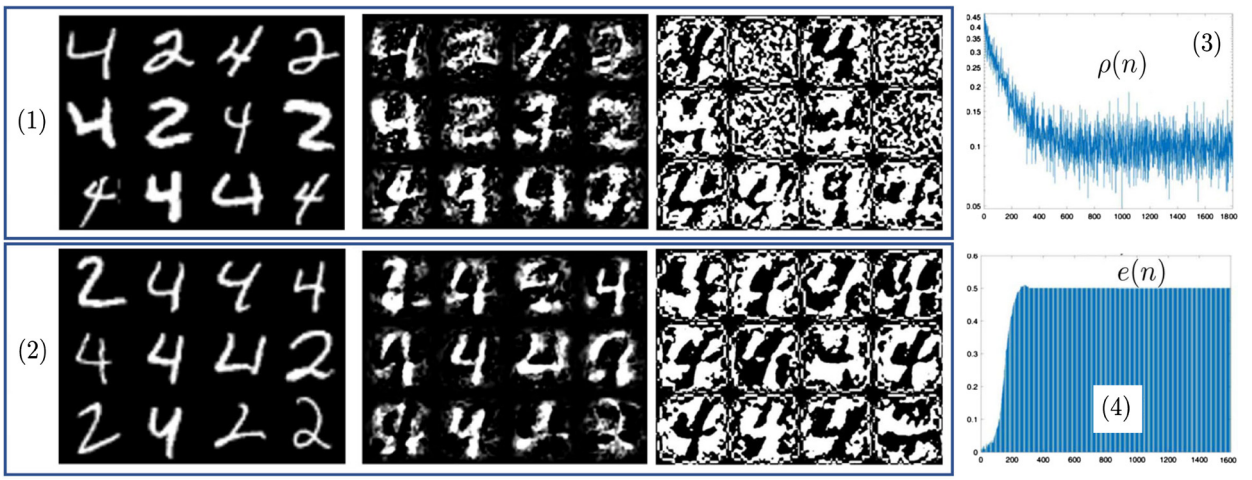

Fig. 18. MNIST with classes 2 and 4, 600 training images and 100 test images. Step sizes are too large, ρ decreases but the modes collapse. (1) x_i and $F_n(x_i)$ for $n = 1, 100, 1800$ and i corresponding to the first 12 training images (2) x_i and $F_n(x_i)$ for $n = 1, 100, 1800$ and i corresponding to the first 12 test images (3) ρ vs n (4) Test error vs n.

$N = 600$ (600 training images) showing only twos and fours and attempt to classify 100 test images (with only twos and fours). Fig. 16 corresponds to a successful outcome (with small adapted step sizes ϵ) and shows the test error vs depth n, ρ vs n and x_i and $F_n(x_i)$ for $n = 8000$. These illustrations suggest that the network can bootstrap data and improve accuracy by introducing small (nearly imperceptible to the naked eye) perturbations to the dataset. Fig. 17 shows another successful run with $N = 1200$ training images, 1200 test images and the full set of 10 classes.

Mode collapse from going too deep, too fast, with $N_f = N$ Fig. 18 shows a failed outcome (with very large step sizes ϵ, the other parameters are the same as in Fig. 16). Although the ratio ρ decreases during training the error blows up towards 50% and the $F_n(x_i)$ seem to collapse towards two images: a four and a random blur. We will explain and analyze this mode collapse below.

8.3. Mode collapse, brittleness of deep learning

The mode collapse observed in Fig. 18 is interesting for several reasons. First it shows that a decreasing ρ is not sufficient to ensure generalization and learning. Indeed writing w for a function exactly interpolating (fitting) the training data the kernel $K_w(x, y) = w(x)w(y)$ would lead to a perfect fit (and hence a value $\rho = 0$) of the training set with any number of interpolation points (and in particular one). Although this K_w is positive but degenerate, if the space of kernels explored by the algorithm is large enough, then, unless $N_f \ll N$, it is not clear what would prevent the algorithm from over-fitting and converging towards those pathological kernels.

The brittleness of deep learning [32] is a well known phenomenon predicted [15] from the brittleness of doing inference in large dimensional spaces [22,19,23,26]. The mechanisms at play in [23] suggest that those instabilities may be unavoidable if the inference space is too large and could to some degree be alleviated through a compromise between accuracy and robustness [26]. From that perspective the learning of the Green's function in Sec. 5 appears to be stable because of strong constraints imposed on the space of kernels (by the structure of the underlying PDE). In Sec. 8.1 the difference in size between the training dataset (N) and that of the batches (N_f) seems to have a stabilizing effect on the algorithm. The pathologies observed in Fig. 18, and mechanisms leading to brittleness [22,19,23,26,17] seem to suggest that will small data sets and a large space of admissible kernels instabilities may occur and could be alleviated by introducing further constraints on the space of kernels.

8.4. Classification archetypes

The brittleness of deep learning [22,32] has lead to the construction of libraries of adversarial examples whose persistence in the physical work [12] be exploited by an adversary [10]. These adversarial examples are constructed via small (near-undetectable to the naked eye) data-dependent (non-random) perturbations of the original images. The Kernel Flow algorithm seems to exploit the brittleness of deep learning in the opposite direction (towards improved performance), i.e. as suggested in Fig. 15 and 16 the Kernel Flow algorithm seems to improve performance through the construction of residual maps introducing small (data-dependent) perturbations to the original images. The resulting images $F_n(x_i)$ at profound depths could be interpreted as archetypes of the classes being learned.

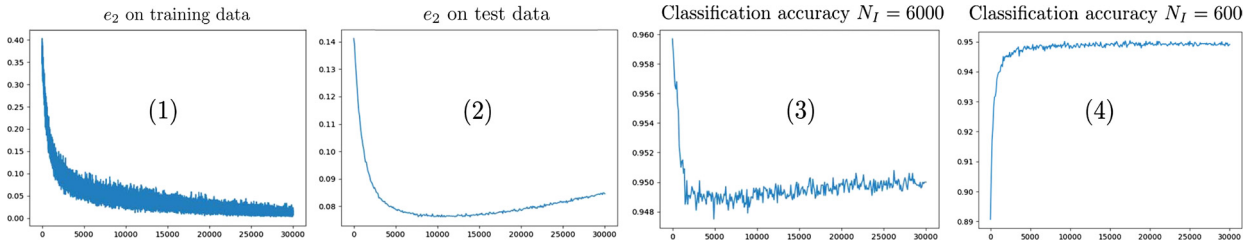

Fig. 19. Minimizing mean squared interpolation error rather than ρ may not lead to generalization for KF. (1) Mean squared interpolation error $e_2(n)$ calculated with random subsets of the training data ($N_f = 600$ and $N_c = 300$) (2) Mean squared interpolation error $e_2(n)$ calculated with the test data ($N_I = 6000$) (3) Classification accuracy (using $N_I = 6000$ interpolation points and all 10000 test data points) vs n (4) Classification accuracy (using $N_I = 600$ interpolation points and all 10000 test data points) vs n.

Table 2
Fashion-MNIST test errors (between layers 15000 and 25000) using N_I interpolation points.

N_I	Average error	Min error	Max error	Standard deviation
6000	0.0969	0.0944	0.1	7.56×10^{-4}
600	0.0977	0.0951	0.101	8.57×10^{-4}
60	0.114	0.0958	0.22	0.0169
10	0.444	0.15	0.722	0.096

Table 3
Fashion-MNIST test errors (between layers 49901 and 50000) using N_I interpolation points.

N_I	Average error	Min error	Max error	Standard deviation
6000	0.10023	0.0999	0.1006	1.6316×10^{-4}
600	0.10013	0.0999	0.1004	1.1671×10^{-4}
60	0.10018	0.0999	0.1005	1.445×10^{-4}
10	0.10018	0.0996	0.1009	2.2941×10^{-4}

8.5. On generalization

Why the KF algorithm does not seem to overfit the data? Why is it capable of generalization? From an initial perspective the KF algorithm appears to promote generalization by grouping data points into clusters according to their classes. However the reason for its generalization properties appears to be more subtle and defining ρ through the RKHS norm seems to also play a role (minimizing ρ by aligning the eigensubspace corresponding to the lowest eigenvalues of the kernel with the training data.).

Indeed, using the notations of Sec. 6, let $v(x_{f,i}^{(n)})$ be the predicted labels of the N_f points $x_{f,i}^{(n)}$ obtained by interpolating a random subset of $N_c = N_f/2$ points (x_i, y_i) with the kernel K_1 and write $e_2 := \sum_{i=1}^{N_f} |y_{f,i}^{(n)} - v(x_{f,i}^{(n)})|^2$ for the mean squared error between training labels and predicted labels. Then minimizing e_2 instead of ρ may lead to a decreasing test classification accuracy rather than an increasing one as shown in Fig. 19 (using the MNIST dataset with $N = 60000$ training points, 10000 test points, $N_f = 600$, $N_c = 300$ for the random batches and $N_I = 600, 6000$ interpolation points for calculating classification accuracies). Note that although the mean squared interpolation error decreases for the training and the test data, the classification error (on the 10000 test data points of MNIST using 6000 interpolation points) increases.

9. Numerical experiments with the fashion-MNIST dataset

We now implement and test the Sec. 6 KF algorithm with the Fashion-MNIST dataset [36]. As with the MNIST dataset [37], the Fashion-MNIST dataset is composed of 60000, 28×28 images portioned into 10 classes (T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot) with a corresponding vector of 60000 labels (with values in $\{0, \ldots, 9\}$). The test set is composed of 10000, 28×28 images of handwritten digits with a corresponding vector of 10000 labels.

9.1. Network trained to depth $n = 50000$

The KF algorithm is implemented with the exact same parameters as for the MNIST dataset (Sec. 8.1, in particular it does not require any manual tuning of hyperparameters nor a laborious process of guessing an architecture for the network). In particular, images are normalized to have L^2 norm one we use the Gaussian kernel of Corollary 6.3 and set γ^{-1} equal to

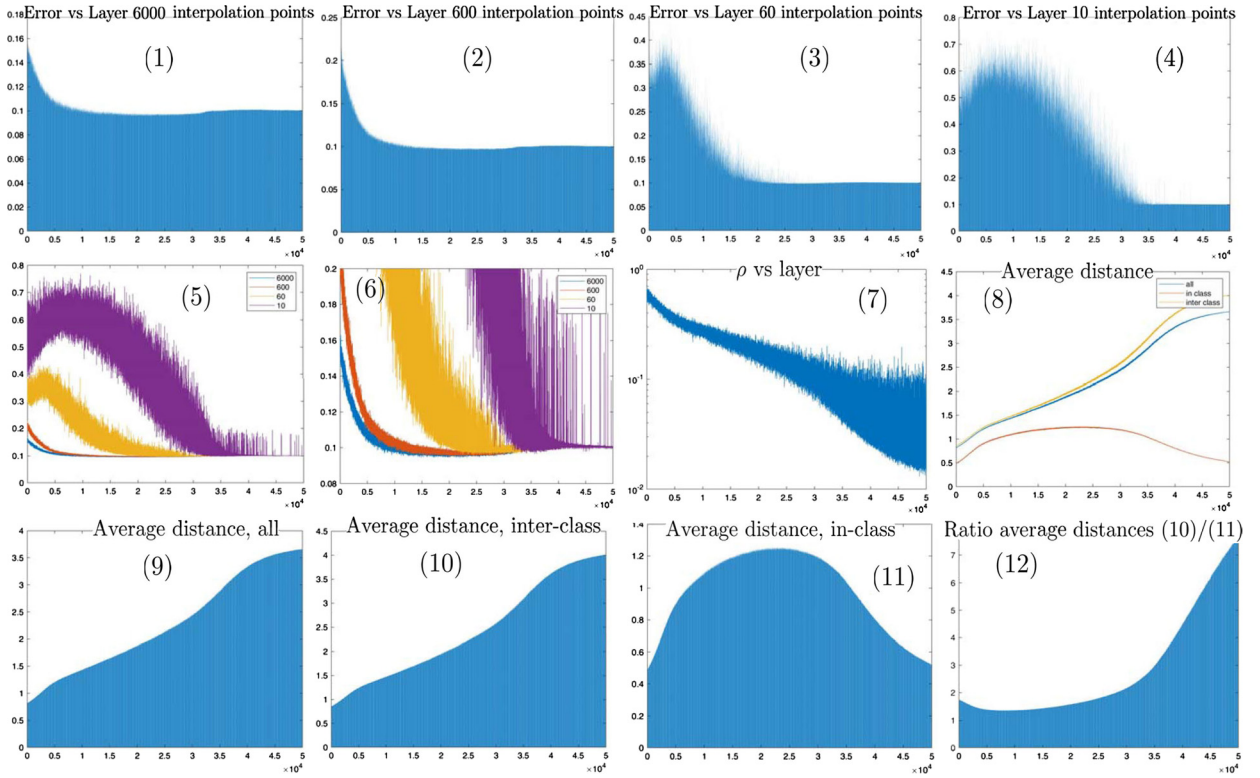

Fig. 20. Results for Fashion-MNIST. $N = 60000$, $N_f = 600$ and $N_c = 300$. (1) Test error vs depth n with $N_I = 6000$ (2) Test error vs depth n with $N_I = 600$ (3) Test error vs depth n with $N_I = 60$ (4) Test error vs depth n with $N_I = 10$ (5,6) Test error vs depth n with $N_I = 6000, 600, 60, 10$ (7) ρ vs depth n (8) Mean-squared distances between images $F_n(x_i)$ (all, inter class and in class) vs depth n (9) Mean-squared distances between images (all) vs depth n (10) Mean-squared distances between images (inter class) vs depth n (11) Mean-squared distances between images (in class) vs depth n (12) Ratio (10)/(11).

the mean squared distance between training images. Training is performed in random batches of size $N_f = 600$ and we use $N_c = 300$ to compute the ratio ρ and learn the parameters of the network (we do not use a nugget and we do not exclude points that are too close from those batches). The value of ϵ is chosen at each step n so that the perturbation of each data point x_i of the batch is no greater than 1% ($\epsilon = 0.01 \times \max_i \frac{|x_{f,i}^{(n)}|_{L^2}}{|\hat{g}_{f,i}^{(n)}|_{L^2}}$).

The network is trained to depth $n = 50000$. Table 2 shows test error statistics (on the full test dataset) using the kernel K_n for $15000 \leq n \leq 25000$ and $N_I = 6000, 600, 60, 10$ interpolation points. Table 3 shows test error statistics (on the full test dataset) using the kernel K_n for $49901 \leq n \leq 50000$ and $N_I = 6000, 600, 60, 10$ interpolation points. Fig. 20 plots test errors vs n using $N_I = 6000, 600, 60, 10$ interpolation points and shows average distances between $F_n(x_i)$ vs n (for x_i selected uniformly at random amongst all training images, within the same or in different classes).

Note that although the network achieves an average test error of 9.7% between layers 15000 and 25000 with $N_I = 600$, average test errors for $N_I = 60, 10$ interpolation points require a depth of more than 37000 layers to achieve comparable accuracies. Note that the average error around layer 50000 with $N_I = 10$ interpolation points is 10% and does not seem to significantly depend on N_I. The average error ($\approx 9.7\%$) of the classifier with $N_I = 600, 6000$ interpolation points between layers 15000 and 25000 and the slight increase of average test errors with $N_I = 600, 6000$ interpolation points between layers 25000 and 50000 (from $\approx 9.7\%$ to $\approx 10\%$) seem to decrease with the value of ϵ. (6.1) could be interpreted an underlying stochastic differential equation with an explicit scheme with time steps ϵ and the efficiency of the resulting classifier seems to improve as $\epsilon \downarrow 0$.

Note from Fig. 20.(8-12), 21 and 4 that $F_n(x_i)$ converges towards an archetype of the class of x_i and that (after layer $n \approx 25000$) the KF algorithm contracts distances within each class while continuing to expand distances between different classes.

Interpolation with K_1 and all $N_I = N = 60000$ training points used as interpolation points results in 12.75% test error and interpolation with K_n with $n = 50000$ and all $N_I = N = 60000$ training points used as interpolation points results in 10% test error. Therefore the KF algorithm appears to bootstrap information contained in the training data in the sense discussed in Sec. 8.2.

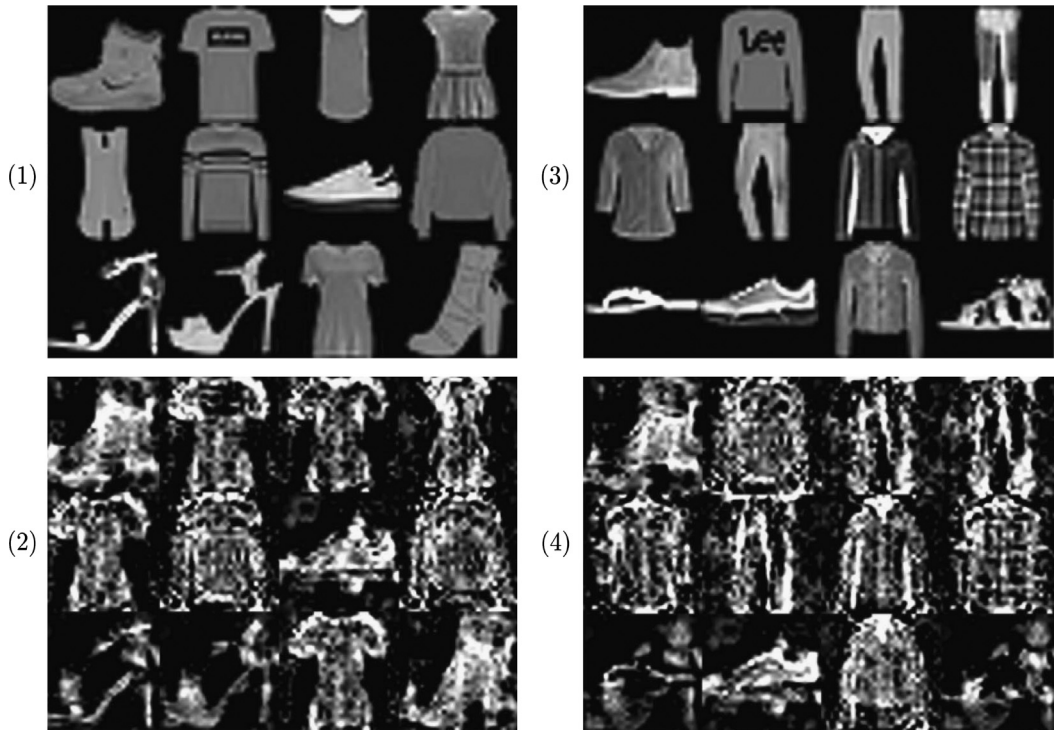

Fig. 21. Results for Fashion-MNIST. $N = 60000$, $N_f = 600$ and $N_c = 300$. (1) Training data x_i (2) $F_n(x_i)$ training data and $n = 50000$ (3) Test data x_i (9) $F_n(x_i)$ for test data and $n = 50000$.

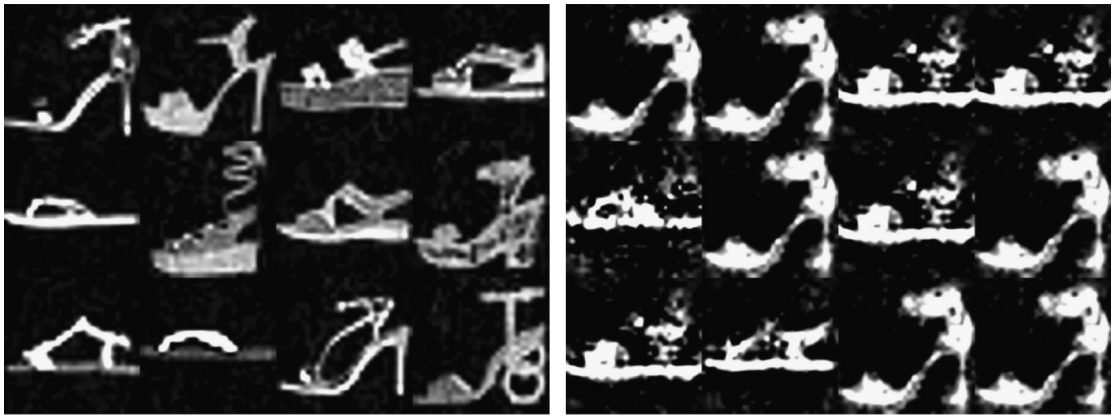

Fig. 22. Results for Fashion-MNIST. $N = 60000$, $N_f = 600$ and $N_c = 300$. Left: Training data x_i for class 5. Right: $F_n(x_i)$ training data and $n = 11000$.

9.2. Sign of unsupervised learning?

Fig. 22 shows x_i and $F_n(x_i)$ for a group of images in the class 5 (sandal). The network is trained to depth $n = 11000$ and the value of ϵ is chosen at each step n so that the perturbation of each data point x_i of the batch is no greater than 10% ($\epsilon = 0.1 \times \max_i \frac{|x_{f,i}^{(n)}|_{L^2}}{|g_{f,i}^{(n)}|_{L^2}}$). Note that this value of ϵ is 10 times larger than the one of Sec. 9.1. Surprisingly the flow F_n accurately clusters that class (sandal) into 2 sub-classes: (1) high heels (2) flat bottom. This is surprising because the training labels contain no information about such sub-classes: KF has created those clusters/sub-classes without supervision.

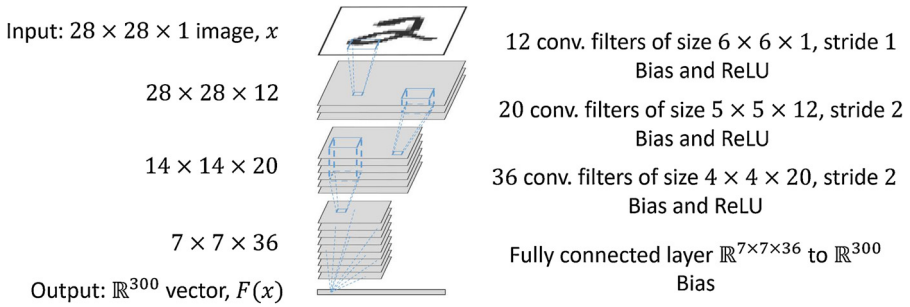

Fig. 23. Convolutional filters used for MNIST classification [1].

Table 4
Test error statistics using $N_I = 12000$ interpolation points at iteration 10000 over 5 runs.

Algorithm	Average error	Min error	Max error	Standard deviation
Minimizing e_2	0.596%	0.55%	0.63%	0.032%
Minimizing ρ	0.640%	0.60%	0.70%	0.034%

10. Kernel Flows and convolutional neural network

10.1. MNIST

The proposed approach can also be applied to families of kernels parameterized by the weights of a Convolutional Neural Network (CNN) [13]. Such networks are known to achieve superior performance by, to some degree, encoding (i.e. providing prior information about) known invariants (e.g. to translations) and the hierarchical structure of data generating distribution into the architecture of the network.

We will first consider an application the MNIST dataset [37] with L^2 normalized test and training images. The structure of the CNN is the one presented in [1] and its first layers are illustrated in Fig. 23. Given an input/image x, the last layer produces a vector $F(x) \in \mathbb{R}^{300}$ used for SVN classification [3] with the Gaussian kernel $K(x,x') = K_1(F(x), F(x')) = e^{-\gamma|F(x)-F(x')|^2}$ where $4/\gamma$ is the mean squared distance between the $F(x_i)$ (writing x_i for the training images).

The training of the filters (weights of the network) is done as described in sections 3 and 4 using random batches of $N_f = 500$ images (sampled uniformly without replacement out of $N = 60000$ training images) and sub-batches of $N_c = 250$ images (sampled uniformly without replacement out of the batch of $N_f = 500$ images). As in Sec. 3, write $v^\dagger$ and v^s for optimal recoveries using the kernel K, and respectively, the batch of N_f and the sub-batch of N_c interpolation points.

Writing $y_i \in \mathbb{R}^{10}$ for the label of the image x_i, the relative approximation error (in the RKHS norm associated with K) caused by halving the number of points is (using, for simplicity, the notations of Sec. 3 to describe the computation of ρ for one batch)

$$\rho = 1 - \frac{\mathrm{Tr}\,(y^T \tilde{A} y)}{\mathrm{Tr}\,(y^T A y)}\,. \tag{10.1}$$

where $y \in \mathbb{R}^{N_f \times 10}$, $\tilde{A}, A \in \mathbb{R}^{N_f \times N_f}$. We will also consider the mean squared error

$$e_2 = \frac{2}{N_f} \sum_{i=1}^{N_f} \left| y_i - v^s(x_i) \right|^2, \tag{10.2}$$

where the sum is taken over the N_f elements of the batch (note that $y_i - v^s(x_i) = 0$ when i is in the sub-batch of N_c elements used as interpolation points for v^s).

To train the network we simply let the Adam optimizer [11] in TensorFlow minimize ρ or e_2 (used as cost functions, which does not require the manual identification of their Fréchet derivatives with respect to the weights of the network).

Table 4 shows statistics of the corresponding test errors using the kernel K learned above (using all $N = 60000$ images in batches of size $N_f = 500$) and five randomly selected subsets of $N_I = 12000$ training images as interpolation points. Each run consisted of 10000 iterations and test errors were calculated on the final iteration. Fig. 24 shows the values of ρ and e_2 evaluated at every 100 iterations for both algorithms (minimizing ρ and e_2). When trained with relative entropy and dropout [30] Gorner reports [1] a minimum classification error of 0.65% testing every 100 iterations over 5 runs. Since we are using the same CNN architecture, this appears to suggest that the proposed approach (of minimizing ρ or e_2) might lead to better test accuracies than training with relative entropy and dropout.

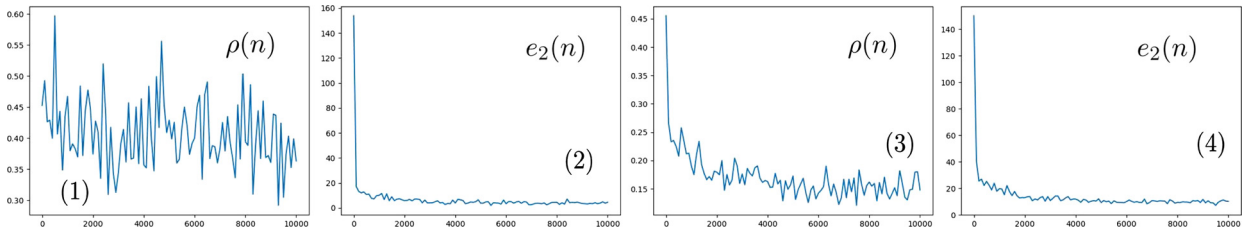

Fig. 24. (1) and (2) show ρ and e_2 respectively evaluated at the n-th batch using the e_2 minimizing network. (3) and (4) show analogous plots for the ρ minimizing network.

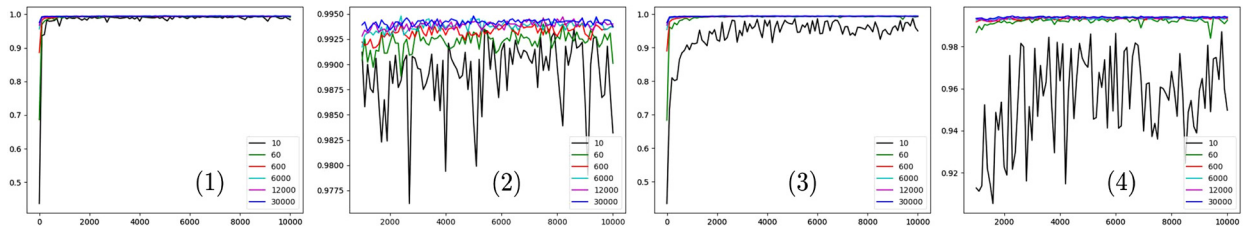

Fig. 25. (1) Classification test errors for $N_I = 10, 60, 600, 6000, 12000, 30000$ evaluated at the n-th batch for $0 \leq n \leq 10000$ using the network minimizing e_2. (2) same as (1) with $1000 \leq n \leq 10000$. (3), (4) same as (1), (2) for the network minimizing ρ.

Table 5
Test error statistics using N_I interpolation points between iterations 9900 and 10000 over 5 runs of optimizing e_2.

N_I	Average error	Min error	Max error	Standard deviation
6000	0.575%	0.42%	0.72%	0.052%
600	0.628%	0.48%	0.83%	0.062%
60	0.728%	0.51%	1.23%	0.103%
10	1.05%	0.58%	4.81%	0.375%

Table 6
Test error statistics using N_I interpolation points between iterations 9900 and 10000 over 5 runs of optimizing ρ.

N_I	Average error	Min error	Max error	Standard deviation
6000	0.646%	0.51%	0.78%	0.046%
600	0.676%	0.56%	0.82%	0.047%
60	0.850%	0.58%	3.98%	0.357%
10	4.434%	0.97%	18.91%	2.320%

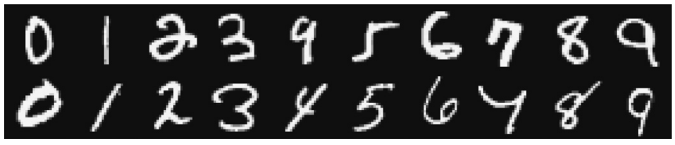

Fig. 26. A "bad" (top) and "good" (bottom) selection of 10 interpolation points.

10.1.1. Interpolation with small subsets of the training set

Fig. 25 shows test errors using the kernel K learned above (using all $N = 60000$ images in batches of size $N_f = 500$) and randomly selected subsets of $N_I = 30000, 12000, 6000, 600, 60, 10$ training images as interpolation points.

Tables 5 and 6 show test errors statistics using the kernel K (learned above with $N_f = 500$) with $N_I = 6000, 600, 60, 10$ interpolation points sampled at random (all use the same convolutional filters obtained in a single optimization run). Averages, min, max and STD are computed over iterations between iterations 9900 to 10000 using 5 independent runs of the Adam optimizer [11] with ρ and e_2 as objective functions.

Observe that, although as with Kernel Flow, using only a small fraction of the training data as interpolation points is sufficient to achieve low classification errors (the minimum error with 10 interpolation points is 0.58%), interpolation with only one image per class appears to be more sensitive to the particular selection of 10 interpolation points. Fig. 26 shows an example of a "good" and a "bad" selection for the interpolation with 10 points.

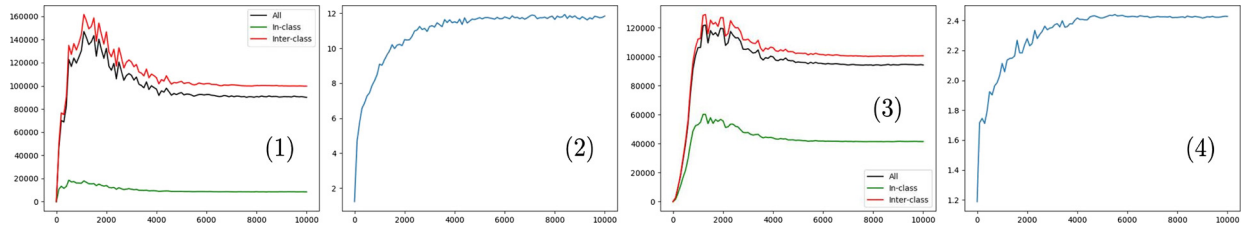

Fig. 27. (1) Mean-squared distance between $F(x_i)$ (all, in-class, and inter-class) vs iteration n for the network optimizing e_2 (2) Ratio between inter-class and in-class mean-squared distance for the network optimizing e_2. (3) and (4) are identical except for the network which optimizes ρ.

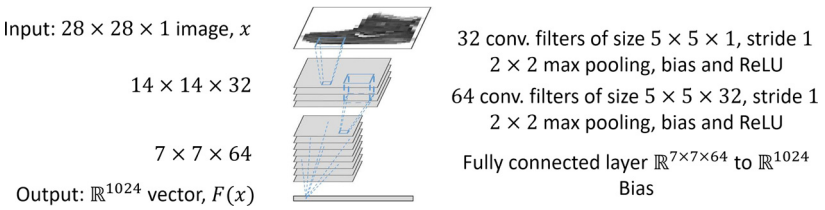

Fig. 28. Convolutional filters used for Fashion MNIST classification [2].

Table 7
Test error statistics using $N_I = 12000$ interpolation points at iteration 10000 over 5 runs.

Algorithm	Average error	Min error	Max error	Standard deviation
Optimizing e_2	8.474%	8.24%	8.70%	0.147%
Optimizing ρ	8.412%	8.29%	8.56%	0.091%

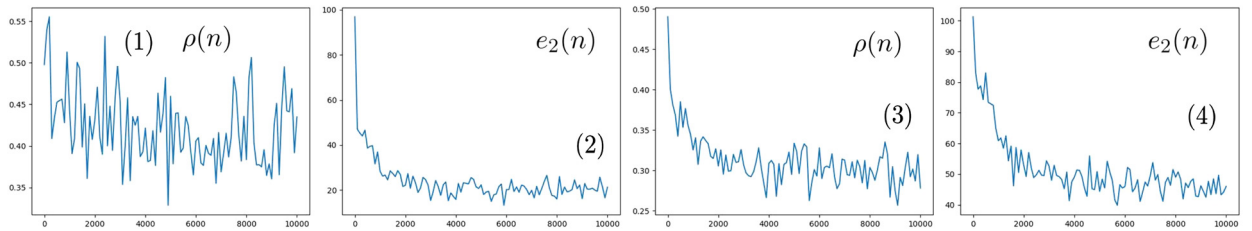

Fig. 29. (1) and (2) show ρ and e_2 respectively evaluated at the n-th batch using the e_2 minimizing network. (3) and (4) show analogous plots for the ρ minimizing network.

The clustering of the $F(x_i)$ (x_i are training images and $F(x) \in \mathbb{R}^{300}$ is the output of the last layer of the CNN with input x) is a possible explanation for this extreme generalization. Fig. 27 shows the average mean squared Euclidean distance between $F(x_i)$ in the same class and in distinct classes. Note that the ratio between average square distances between two inter-class and two in-class points approaches 12 (for the network optimizing e_2), suggesting that the map F clusters of images per class.

10.2. Fashion MNIST

We now apply the proposed approach to the Fashion-MNIST database. The architecture of the CNN is derived from [2] and the first layers of the network are shown in Fig. 28.

The classification of test images is done as in Sec. 10.1.

Table 7 shows test errors statistics (over 5 runs) after training (using 10000 iterations and computing test errors at the final iteration) by minimizing ρ or e_2 (as defined in (10.1) and (10.2)) using $N_I = 12000$ interpolation points Mahajan [2] reports a testing error of 8.6% when using the validation set to obtain the convolutional filters. As above, this suggests that the proposed approach could lead to better test accuracies than training with relative entropy and dropout. Finally, Fig. 29 shows ρ and e_2 evaluated at every 100 iterations for both algorithms.

10.2.1. Interpolation with small subsets of the training set

Fig. 30 shows test errors using the kernel K learned above (using all $N = 60000$ images in batches of size $N_f = 500$) and randomly selected subsets of $N_I = 30000, 12000, 6000, 600, 60, 10$ training images as interpolation points.

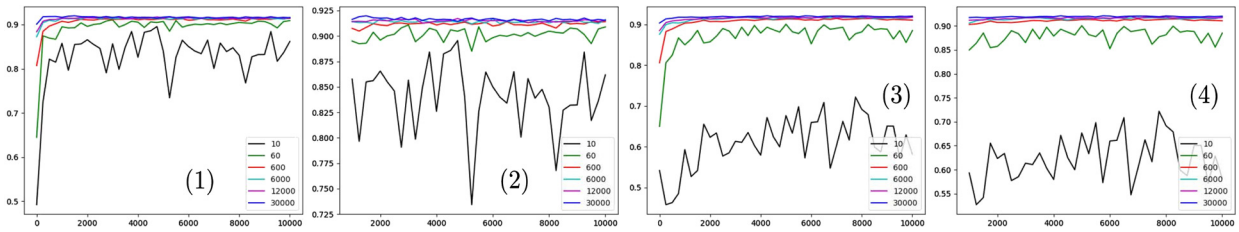

Fig. 30. (1) Classification test errors for $N_I = 10, 60, 600, 6000, 12000, 30000$ evaluated at the n-th batch for $0 \le n \le 10000$ using the network minimizing e_2. (2) same as (1) with $1000 \le n \le 10000$. (3), (4) same as (1), (2) for the network minimizing ρ.

Table 8
Test error statistics using N_I interpolation points between iterations 9900 and 10000 over 5 runs of optimizing e_2.

N_I	Average error	Min error	Max error	Standard deviation
6000	8.561%	8.23%	8.97%	0.135%
600	8.724%	8.31%	9.26%	0.161%
60	9.677%	8.77%	11.48%	0.486%
10	15.261%	10.00%	32.69%	3.196%

Table 9
Test error statistics using N_I interpolation points between iterations 9900 and 10000 over 5 runs of optimizing ρ.

N_I	Average error	Min error	Max error	Standard deviation
6000	8.526%	8.17%	8.96%	0.120%
600	8.810%	8.36%	9.29%	0.140%
60	11.677%	9.32%	18.03%	1.437%
10	36.642%	23.44%	53.56%	4.900%

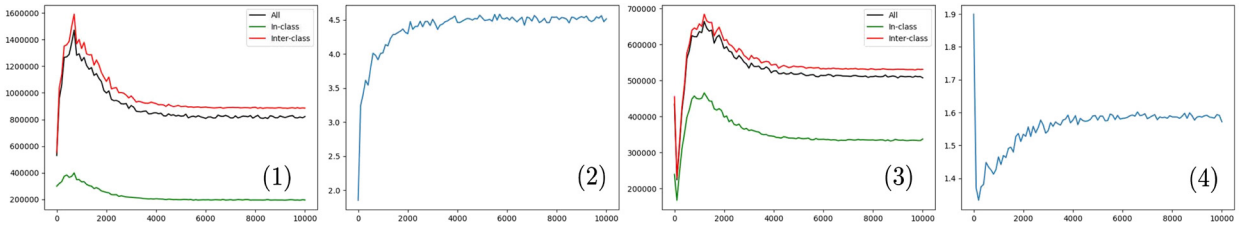

Fig. 31. (1) Mean-squared distance between $F(x_i)$ (all, in-class, and inter-class) vs iteration n for the network optimizing e_2 (2) Ratio between inter-class and in-class mean-squared distance for the network optimizing e_2. (3) and (4) are identical except for the network which optimizes ρ.

Further, the minimum errors in Fig. 30.1, 3 are observed to be 8.02% and 7.89% respectively, where both minima used $N_I = 30000$."

Tables 8 and 9 show test errors statistics using the kernel K (learned above with $N_f = 500$) with $N_I = 6000, 600, 60, 10$ interpolation points sampled at random (all use the same convolutional filters obtained in a single optimization run). Averages, min, max and STD are computed over iterations between 9900 and 10000 using 5 independent runs of the Adam optimizer [11] with ρ and e_2 as objective functions.

It can again be observed that using only a small fraction of the training data as interpolation points yields relatively low classification errors. The instability of test errors with $N_I = 10$ interpolation points, compared to the Kernel Flow algorithm proposed in Sec. 6 seem to suggest that deep architectures might be required to achieve stable results with only one interpolation point per class.

The clustering of the $F(x_i)$ (x_i are training images and $F(x) \in \mathbb{R}^{300}$ is the output of the last layer of the CNN with input x) using the network optimizing e_2 is shown in Fig. 31. The figure shows the average mean squared Euclidean distance between $F(x_i)$ in the same class and in distinct classes. Note that the ratio between average square distances between two inter-class and two in-class points approaches 4.5, suggesting that the map F aggregates images per class.

Acknowledgements

The authors gratefully acknowledge this work supported by the Air Force Office of Scientific Research and the DARPA EQUiPS Program under award number FA9550-16-1-0054 (Computational Information Games) and the Air Force Office of

Scientific Research under award number FA9550-18-1-0271 (Games for Computation and Learning). We also thank Andrew Stuart and Yifan Chen for helpful discussions for the clarification of Section 7.4.

References

[1] Learn tensorflow and deep learning, without a ph.d, https://cloud.google.com/blog/big-data/2017/01/learn-tensorflow-and-deep-learning-without-a-phd.
[2] Playing with fashion mnist, https://pravarmahajan.github.io/fashion/.
[3] Corinna Cortes, Vladimir Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[4] Nello Cristianini, John Shawe-Taylor, et al., An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000.
[5] Diaconis Persi, David Freedman, Iterated random functions, SIAM Rev. 41 (1) (1999) 45–76.
[6] Matthew M. Dunlop, Mark Girolami, Andrew M. Stuart, Aretha L. Teckentrup, How deep are deep gaussian processes?, arXiv preprint: arXiv:1711.11280, 2017.
[7] Robert A. Gingold, Joseph J. Monaghan, Smoothed particle hydrodynamics: theory and application to non-spherical stars, Mon. Not. R. Astron. Soc. 181 (3) (1977) 375–389.
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, Kilian Q. Weinberger, Densely connected convolutional networks, in: CVPR, vol. 1, 2017, p. 3.
[10] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, J.D. Tygar, Adversarial machine learning, in: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, ACM, 2011, pp. 43–58.
[11] D.P. Kingma, J.L. Ba Adam, A method for stochastic optimization, https://arxiv.org/abs/1412.6980, 2014.
[12] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Adversarial examples in the physical world, arXiv preprint: arXiv:1607.02533, 2016.
[13] Yann LeCun, Yoshua Bengio, et al., Convolutional networks for images, speech, and time series, in: The Handbook of Brain Theory and Neural Networks, vol. 3361(10), 1995, p. 1995.
[14] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[15] M. McKerns, Mystic: a framework for predictive science: scipy 2013 presentation, https://youtu.be/o-nwSnLC6DU?t=96, June 2013.
[16] C.A. Micchelli, T.J. Rivlin, A survey of optimal recovery, in: Optimal Estimation in Approximation Theory, Springer, 1977, pp. 1–54.
[17] Jeffrey W. Miller, David B. Dunson, Robust Bayesian inference via coarsening, J. Am. Stat. Assoc. (2018) 1–13, https://doi.org/10.1080/01621459.2018.1469995.
[18] H. Owhadi, Multigrid with rough coefficients and multiresolution operator decomposition from hierarchical information games, SIAM Rev. 59 (1) (2017) 99–149.
[19] H. Owhadi, C. Scovel, Brittleness of Bayesian inference and new Seiberg formulas, Commun. Math. Sci. 14 (1) (2016) 83–145.
[20] H. Owhadi, C. Scovel, Universal scalable robust solvers from computational information games and fast eigenspace adapted multiresolution analysis, arXiv:1703.10761, 2017.
[21] H. Owhadi, C. Scovel, Operator Adapted Wavelets, Fast Solvers, and Numerical Homogenization From a Game Theoretic Approach to Numerical Approximation and Algorithm Design, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2020.
[22] H. Owhadi, C. Scovel, T.J. Sullivan, Brittleness of Bayesian inference under finite information in a continuous world, Electron. J. Stat. 9 (2015) 1–79, arXiv:1304.6772.
[23] H. Owhadi, C. Scovel, T.J. Sullivan, On the brittleness of Bayesian inference, SIAM Rev. (Res. Spotlights) (2015).
[24] H. Owhadi, L. Zhang, L. Berlyand, Polyharmonic homogenization, rough polyharmonic splines and sparse super-localization, ESAIM Math. Model. Numer. Anal. 48 (2) (2014) 517–552.
[25] Owhadi Houman, Kernel flows, Youtube, https://www.youtube.com/watch?v=h9wB8FVH7YM&list=PLdWd7x7FVuLphAODzEvj2KRNws7z7Sv87.
[26] Houman Owhadi, Clint Scovel, Qualitative robustness in Bayesian inference, ESAIM Probab. Stat. 21 (2017) 251–274.
[27] Tomaso Poggio, Steve Smale, The mathematics of learning: dealing with data, Not. Am. Math. Soc. 50 (5) (2003) 537–544.
[28] F. Schäfer, T.J. Sullivan, H. Owhadi, Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity, arXiv:1706.02205, 2017.
[29] Anton Schwaighofer, Volker Tresp, Kai Yu, Learning gaussian process kernels via hierarchical bayes, in: Advances in Neural Information Processing Systems, 2005, pp. 1209–1216.
[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[31] Ingo Steinwart, Philipp Thomann, Nico Schmid, Learning with hierarchical gaussian kernels, arXiv preprint: arXiv:1612.00824, 2016.
[32] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus, Intriguing properties of neural networks, arXiv preprint: arXiv:1312.6199, 2013.
[33] Molei Tao, Houman Owhadi, Jerrold E. Marsden, Nonintrusive and structure preserving multiscale integration of stiff odes, sdes, and hamiltonian systems with hidden slow dynamics via flow averaging, Multiscale Model. Simul. 8 (4) (2010) 1269–1324.
[34] Christopher K.I. Williams, Carl Edward Rasmussen, Gaussian processes for regression, in: Advances in Neural Information Processing Systems, 1996, pp. 514–520.
[35] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, Eric P. Xing, Deep kernel learning, in: Artificial Intelligence and Statistics, 2016, pp. 370–378.
[36] Han Xiao, Kashif Rasul, Roland Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint: arXiv:1708.07747, 2017.
[37] LeCun Yann, Cortes Corinna, J.B. Christopher, The mnist database of handwritten digits, http://yhann.lecun.com/exdb/mnist, 1998.
[38] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, Oriol Vinyals, Understanding deep learning requires rethinking generalization, arXiv preprint: arXiv:1611.03530, 2016.