A Neural Tangent Kernel Perspective on Function-Space Regularization in Neural Networks

Zonghao Chen
Tsinghua University

Xupeng Shi
Northeastern Univerisity

Tim G. J. Rudner
University of Oxford

Qixuan Feng
University of Oxford

Weizhong Zhang
HKUST

Tong Zhang
HKUST

Abstract

Regularization can help reduce the gap between train and test error by systematically limiting model complexity. Popular regularization techniques like weight ℓ_2-regularization act directly on the network parameters, but do not explicitly take into account how the interplay between the parameters and the network architecture may affect the induced predictive functions. To address this shortcoming, we propose a simple technique for effective function-space regularization. Drawing on the result that fully-trained wide multilayer perceptrons is equivalent to kernel regression under the Neural Tangent Kernel (NTK), we hypothesize that the functions induced by a given neural network architecture can be well-approximated by functions in a *reproducing kernel Hilbert space* (RKHS) induced by the network's NTK. We show that, under this assumption, it is possible to approximate the RKHS norm on the function space defined by the neural network and use it as a function-space regularizer. We demonstrate that this regularization technique can result in improved generalization and state-of-the-art performance on continual learning tasks where effective regularization on the induced space of functions is particularly important.

1 Introduction

Over-parameterized deep neural networks (DNNs) have been shown to succeed at learning complex tasks when a large number of training data is available. However, in areas where data is scarce or data labeling is expensive, DNNs may overfit and require effective regularization techniques to systematically control the complexity of the functions encoded by the neural networks and to improve generalization (Vapnik, 1998).

Unfortunately, commonly used regularization techniques in deep learning are developed for regularizing less expressive models rather than highly overparameterized DNNs used in modern machine learning. Recent work on double decent in deep learning with over-parameterized DNNs (Belkin et al., 2019) has demonstrated that notions of regularization and generalization developed for smaller models may not transfer to highly expressive over-parameterized neural networks.

This raises the question of why parameter space regularization techniques fall short and to what extent over-parameterized neural networks may benefit from alternative regularization techniques that do not explicitly regularize the network parameters but rather regularize a principled measure of network complexity.

Recent work has shown that weight penalization cannot regularize the complexity of functions (Triki et al., 2018) and that network parameters are a poor proxy for the functions induced by the DNN (Benjamin et al., 2019). It is also demonstrated that regularizing the parameter norm is unnecessary for better generalization (Zhang et al., 2017; Kawaguchi et al., 2020).

In this paper, we propose to directly regularize the norm of function mappings induced by the neural network to more effectively control the complexity of the learned function and improve generalization in over-parameterized models. Theoretical work by Bietti and Mairal (2018) has proved that functions encoded by a DNN lie in a *reproducing kernel Hilbert space* (RKHS) under smooth approximation for ReLU activations, therefore a natural candidate for regularization technique is the RKHS norm. Unfortunately, computing the RKHS norm is challenging since the characteristics of the RKHS defined by the DNN is not well-understood.

Preprint (November 21, 2021).

Prior work on function space regularization in DNNs has explored using the L_2 norm (Benjamin et al., 2019) or using the Jacobian norm as a lower bound of RKHS norm (Bietti et al., 2019). However, both methods fall short, as L_2 space is a trivial approximation and loses many topological information of the space of functions and regularizing the lower bound has no direct relation to regularizing the RKHS norm.

Drawing on a result in Arora et al. (2019) that a fully-trained wide multi-layer perceptron is equivalent to a kernel regression predictor under the Neural Tangent Kernel (NTK), we argue that the functions encoded by a given DNN can be well approximated by functions in a RKHS associated with the network's NTK. Under this assumption, we derive the RKHS norm associated with NTK from the perspective of kernel regression and use it as a function-space regularizer. Interestingly, we can further validate our approach by deriving the same regularizer from the perspective of function-space maximum a posteriori inference (FS-MAP).

To evaluate whether the proposed function-space regularization technique leads to improved generalization, we follow prior work (Bietti et al., 2019) and train on MNIST and CIFAR-10 with small number of training samples. We scale our method to MLP-Mixer (Tolstikhin et al., 2021) which is prone to overfit (Yu et al., 2021) and show that our function-space regularizer can effectively alleviate overfitting and achieve higher accuracy as well as lower test negative log-likelihood. Lastly, we evaluate the proposed regularization technique on practically-relevant downstream continual learning tasks where effective regularization is particularly important. We show that our regularization method leads to state-of-the-art predictive accuracy, outperforming related parameter- and function-space regularization techniques (Kirkpatrick et al., 2017; Nguyen et al., 2018; Titsias et al., 2020; Pan et al., 2021).

Contributions. We propose to explicitly regularize in the space of functions induced by highly over-parameterized deep neural networks through regularizing the norm of functions in the RKHS associated with the Neural Tangent Kernel. Then we develop the same regularizer from the perspective of function-space maximum a posteriori, which provides an alternative understanding of the proposed regularizer. Finally, we empirically verify that our regularization technique results in improved generalization performance. We also extend our approach to continual learning and achieve state-of-the-art performance on commonly used continual learning benchmarks.

2 Related Work

Function-space regularization. Bietti and Mairal (2018) has proved that functions encoded by deep neural networks lie in a RKHS under smooth approximations for ReLU activation, but the RKHS norm is difficult to compute. Benjamin et al. (2019) uses function L_2 norm as a trivial approximation to the RKHS norm and Bietti et al. (2019) gives a lower bound of the RKHS norm with Jacobian norm.

The concept of function space regularization has also been employed in other areas like variational inference (Sun et al., 2019; Rudner et al., 2021; Burt et al., 2020; Khan et al., 2020; Blei et al., 2017) and continual learning (Titsias et al., 2020; Pan et al., 2021). These function-space regularization techniques directly constrain the space of functions and as such allow for a more explicit incorporation of prior information about the functions to be learned.

For variational inference, function-space approaches avoid the limitations in parameter-space variational inference (Blundell et al., 2015; Farquhar et al., 2021) and achieve better uncertainty quantification as well as calibration, but function-space approaches is usually not scalable. Sun et al. (2019) uses an expensive gradient estimator which limits the application to low-dimensional data, and Rudner et al. (2021) requires backpropagation through the NTK-induced covariance matrix which limits its application under ResNet18 (He et al., 2015a).

For continual learning, function-space approaches usually adopt a variational inference framework. The regularization term is KL divergence from the variational posterior to the posterior of the previous task which is used as prior in the current task (Titsias et al., 2020). Benjamin et al. (2019) uses function L_2 norm as a function-space regularizer; FROMP (Pan et al., 2021) first constructs a GP based on DNN2GP (Khan et al., 2020) and then uses the GP posterior in KL divergence, which deviates from the original neural network. We adapt our function-space regularizer to continual learning and propose to regularize the RKHS norm of the relative function change from the previous task to the current task.

Generalization in neural networks. Classical learning theory attributes generalization ability to low-capacity class of hypotheses space (Vapnik, 1998; Mohri et al., 2012; Bartlett and Mendelson, 2002), but neural networks have contradicted traditional concept by demonstrating that good generalization can be achieved despite over-parameterization (Zhang et al., 2017; Kawaguchi et al., 2020). Keskar et al. (2017) argues that flat minima lead to better generalization,

but Dinh et al. (2017) shows that sharp minima can also generalize well. Although it remains an open problem why over-parameterized DNNs can generalize well, DNNs will definitely overfit when the number of training samples is small and we propose to improve DNN generalization when training with limited number of samples.

Kernel methods. Kernel methods, especially support vector machines (Schölkopf et al., 2002; Vapnik, 1998), are popular statistical learning methods. Classical choices of kernels include Gaussian kernel, polynomial kernels etc. Some of the previous works also consider families of Gaussian kernels (Micchelli et al., 2005), hyperkernels (Ong et al., 2005). The generalization performance of kernel methods have been verified in theory (Lanckriet et al., 2004; Cortes et al., 2010). Despite the theoretical success of kernel methods, its high computational costs limits its application, so many approximation methods have been proposed to scale up the kernel machines, such as random Fourier features (Rahimi et al., 2007) and the Nyström method (Williams and Seeger, 2001).

3 Function-Space Regularization

In this section, we introduce the necessary theoretical details and include a brief introduction of kernel ridge regression, the Neural Tangent Kernel (NTK) and function-space maximum a posteriori inference (FS-MAP).

Notation. In the following sections, we denote the training set as $\mathcal{D} \doteq \left\{\left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right)\right\}_{n=1}^{N} = (\mathbf{X}, \mathbf{y})$ with inputs $\mathbf{x}^{(n)} \in \mathcal{X} \subseteq \mathbb{R}^d$ and targets $\mathbf{y}^{(n)} \in \mathcal{Y}$. We use different subscripts under $\mathbf{X}$ to indicate different subset of training inputs. We consider the function $f_{\boldsymbol{\theta}}(\cdot)$ encoded by a neural network with parameters $\boldsymbol{\theta} \in \mathbb{R}^P$. We also denote the Jacobian of $f_{\boldsymbol{\theta}}(\cdot)$ with respect to $\boldsymbol{\theta}$ by $\mathcal{J}_{\boldsymbol{\theta}}(\cdot)$.

3.1 Function-Space Regularization From a Kernel Ridge Regression Perspective

Kernel ridge regression. Consider a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is symmetric and positive definite. That is to say $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$ for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ and

$$\int_{\mathcal{X}\times\mathcal{X}} f(\mathbf{x})K(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')\mathrm{d}\mu(\mathbf{x})\mathrm{d}\mu(\mathbf{x}') > 0 \quad (1)$$

for any $f \in L_2(\mathcal{X}, \mu)$, where L_2 is the squared integrable function space over $\mathcal{X}$ with respect to a probability measure μ on $\mathcal{X}$.

According to Moore–Aronszajn theorem (Aronszajn, 1950), for any symmetric, positive definite kernel K on $\mathcal{X}$ we have a unique *reproducing kernel Hilbert space* (RKHS) $\mathcal{H}_K$, which is the completion of the pre-Hilbert space consisting of all linear span of $\{K_{\mathbf{x}}(\cdot) : \mathbf{x} \in \mathcal{X}\}$, where $K_{\mathbf{x}}(\cdot) = K(\cdot, \mathbf{x})$. To be precise, we have

$$\mathcal{H}_K = \Bigg\{ f(\mathbf{x}) = \sum_{j=1}^{\infty} \alpha_j K(\mathbf{x}^{(j)}, \mathbf{x}) : (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots) \subset \mathcal{X} \\ \|f\|_{\mathcal{H}_K}^2 = \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) < \infty \Bigg\} \quad (2)$$

The kernel function K is called *reproducing kernel* since for any $f \in \mathcal{H}_K$, we have the following reproducing property:

$$f(\mathbf{x}) = \langle f, K_{\mathbf{x}} \rangle_{\mathcal{H}_K} \quad (3)$$

Classical kernel ridge regression consider the following non-parametric optimization problem:

$$f^* = \arg\min_{f \in \mathcal{H}_K} \mathop{\mathbb{E}}_{\mathbf{x} \sim p(\mathbf{x})} \ell(f(\mathbf{x})) + \lambda \|f\|_{\mathcal{H}_K}^2 \quad (4)$$

where $p(\mathbf{x})$ is the density function of $\mu(\mathbf{x})$. Kernel ridge regression seeks an optimal solution in the RKHS $\mathcal{H}_K$ associated with a given reproducing kernel K, and the function norm regularizer $\|f\|_{\mathcal{H}_K}$ ensures a most smooth solution.

In practice, we usually consider the case when $p(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N} \delta_{\mathbf{x}^{(j)}}$ is the empirical data distribution with respect to the training set $\mathbf{X}$, then the kernel matrix $K(\mathbf{X}, \mathbf{X}) = [K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})]_{1 \le i,j \le N}$ is symmetric and positive definite. Hence the RKHS norm $\|f\|_{\mathcal{H}_K}$ in Equation (2) reads as

$$\|f\|_{\mathcal{H}_K}^2 = \boldsymbol{\alpha}^\top K(\mathbf{X}, \mathbf{X})\boldsymbol{\alpha} \quad (5)$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_N]^\top$ represents the coordinate of f in $\mathcal{H}_K$ with respect to the basis $\Phi = [K_{\mathbf{x}^{(1)}}, \cdots, K_{\mathbf{x}^{(N)}}]^\top$ given by the feature map. Equation (5) implies that $\mathcal{H}_K$ can be viewed as dual space of $\mathcal{X}$ equipped with the metric induced by K so we refer to Equation (5) as the dual form of $\|f\|_{\mathcal{H}_K}$.

The RKHS norm $\|f\|_{\mathcal{H}_K}$ can also be computed directly in the primal space with respect to function f, which we refer to as the primal form of $\|f\|_{\mathcal{H}_K}$.

Proposition 1. *Suppose K is a positive definite kernel function and $p(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N} \delta_{\mathbf{x}^{(j)}}$ is the empirical data distribution. Then for any $f \in \mathcal{H}_K$, we have*

$$\|f\|_{\mathcal{H}_K}^2 = f(\mathbf{X})^\top K(\mathbf{X}, \mathbf{X})^{-1} f(\mathbf{X}) \quad (6)$$

where $f(\mathbf{X}) = [f(\mathbf{x}^{(1)}), \cdots, f(\mathbf{x}^{N})]^\top$.

The proof of this Proposition (1) is a simple two lines of calculation so we give it below:

$$\begin{aligned}\|f\|^2_{\mathcal{H}_K} &= \boldsymbol{\alpha}^\top K(\mathbf{X},\mathbf{X})\boldsymbol{\alpha} \\ &= f(\mathbf{X})^\top K(\mathbf{X},\mathbf{X})^{-1}K(\mathbf{X},\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}f(\mathbf{X}) \\ &= f(\mathbf{X})^\top K(\mathbf{X},\mathbf{X})^{-1}f(\mathbf{X})\end{aligned}$$

The calculation in Equation (6) is valid for all positive definite kernels. This form has been mentioned in many works such as Rasmussen and Williams (2006) but has not been widely used in practice because kernel ridge regression is usually solved in dual form. To the best of our knowledge, this is the first try to use such form in practice, at least to use it as a function-space regularizer in deep learning.

Kernel methods in deep learning. Jacot et al. (2018) has studies the training dynamics of neural networks, and has proposed the definition of the Neural Tangent Kernel (NTK) as:

$$\Theta(\mathbf{x},\mathbf{x}') = \langle \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{x}), \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{x}')\rangle \tag{7}$$

Under the assumption that the number of neurons in each hidden layer goes to infinity, Jacot et al. (2018) has proved that NTK Θ converges to a deterministic limiting kernel Θ_{ntk}. We refer to Θ_{ntk} as the analytic NTK to distinguish from the empirical NTK Θ.

Arora et al. (2019) has proved the equivalence between a fully-trained sufficiently wide neural net and the kernel regression solution under analytic NTK Θ_{ntk}. Roughly speaking (Arora et al. (2019) Theorem 3.2), for any fully-trained network function $f_{\boldsymbol{\theta}}$, if f_{ntk} is the optimal solution of kernel regression problem under Θ_{ntk}, then with high probability, for any precision ε, we have

$$|f_{\boldsymbol{\theta}}(\mathbf{x}) - f_{ntk}(\mathbf{x})| \leq \varepsilon \tag{8}$$

Since f_{ntk} belongs to the RKHS $\mathcal{H}_{\Theta_{ntk}}$ induced by NTK, Equation (8) indicates that any function defined by the neural network can be well approximated by functions in $\mathcal{H}_{\Theta_{ntk}}$. Inspired by this observation, it is natural to consider regularizing deep neural networks using the norm of functions in the RKHS associated with NTK. However, given that the ultra-wide assumption is not valid in practice, we use the empirical kernel Θ instead of the analytic kernel Θ_{ntk} in computation and propose to use the following NTK-based RKHS norm as the function-space regularizer:

$$\|f\|^2_{\mathcal{H}_\Theta} = f(\mathbf{X})^\top \Theta^{-1}(\mathbf{X},\mathbf{X})f(\mathbf{X}) \tag{9}$$

3.2 Function-Space Regularization From a Maximum A Posteriori Perspective

The function-space regularizer in Equation (9) can also be derived from the perspective of function-space maximum a posteriori inference (FS-MAP). First we briefly revisit parameter-space maximum a posteriori inference (MAP).

MAP. Assume $\boldsymbol{\theta}$ follow some prior distribution $p(\boldsymbol{\theta})$, then the rule of maximum a posteriori states that the optimal $\boldsymbol{\theta}$ is the best guess that maximizes the value of posteriori distribution. According to the Bayes's theorem, the posteriori distribution is proportional to the product of prior and likelihood, so we have the following formula:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}}\{\log p(\boldsymbol{\theta}|\mathcal{D})\} \\ &= \arg\max_{\boldsymbol{\theta}}\{\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\} \\ &= \arg\max_{\boldsymbol{\theta}}\{p(\mathcal{D}|\boldsymbol{\theta}) + \tau||\boldsymbol{\theta}||\},\end{aligned}$$

where the norm depends on the choice of prior distribution over parameters and τ is a positive scaling factor.

FS-MAP. Instead of expressing the optimization objective *directly* in terms of the model parameters $\boldsymbol{\theta}$, we suggest a function-space objective, which is expressed in terms of the functions $f_{\boldsymbol{\theta}}(\cdot)$ induced by the parameters $\boldsymbol{\theta}$. To keep notation tight in this section, we restrict function $f_{\boldsymbol{\theta}}(\cdot)$ to be a scalar function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}$. Similar to MAP, if we assume $f_{\boldsymbol{\theta}}(\cdot)$ follows some prior distribution $p(f_{\boldsymbol{\theta}}(\cdot))$, then we have:

$$f_{\boldsymbol{\theta}}{}^* = \arg\max_{\boldsymbol{\theta}}\{\log p(f_{\boldsymbol{\theta}}(\mathbf{X})|\mathcal{D}) + \log p(f_{\boldsymbol{\theta}}(\cdot))\} \tag{10}$$

Remark. Although we hope to find the optimal function that maximizes this objective, the optimization is performed on parameters given that the function is parametrized by $\boldsymbol{\theta}$. Note that there is an abuse of notation in the above formula. The function-space prior distribution $p(f_{\boldsymbol{\theta}}(\cdot))$ is a distribution over functions, i.e stochastic process (Lamperti, 1977). The density function $p(f_{\boldsymbol{\theta}}(\cdot))$ is ill-defined and doesn't even exist because there is no infinite-dimensional Lebesgue measure (Eldredge, 2016). For a more rigorous definition of Equation (10) from the perspective of measure theory, please refer to Appendix A.2

Following prior work (Sun et al., 2019; Rudner et al., 2021), in order to avoid the pathology in infinite-dimensional probability density, we only evaluate the log density of function-space prior distribution on finite number of points $\mathbf{X}_{\mathcal{I}}$ sampled from some distribution $q(\mathbf{x})$. Under this assumption, Equation (10) becomes:

$$f^*_{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\{\log p(f_{\boldsymbol{\theta}}(\mathbf{X})|\mathcal{D}) + \mathop{\mathbb{E}}_{\mathbf{X}_{\mathcal{I}}\sim q(\mathbf{x})} \log p(f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}}))\} \tag{11}$$

Algorithm 1 Function-space regularization

Initialize $\boldsymbol{\theta}$
for training epoch $m = 1, 2 \dots M$ **do**
  for each training iteration **do**
    Sample a mini batch of data $\mathcal{B} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \dots, \left(\mathbf{x}^{(B)}, \mathbf{y}^{(B)}\right) \right\}$
    Sample a sub batch of data $\mathcal{I} \subset \mathcal{B}$
    Compute NTK $\Theta(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) = \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top$
    Compute Loss $\mathcal{L}_\mathcal{B} = -\sum_{i=1}^{B} \log p(\mathbf{y}^{(i)} | f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + \tau f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})$
    Update $\boldsymbol{\theta}$: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_\mathcal{B}$
  end for
end for
return $\boldsymbol{\theta}$

Function-space prior. Now we consider the choice of function-space prior distribution evaluated at a finite number of samples $p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}))$. We propose to use the function-space prior distribution induced by the prior distribution on network's parameters. Although many other function-space priors are also widely used such as RBF Gaussian prior (Rasmussen and Williams, 2006), our choice of function-space prior distribution best captures the correlations between function values induced by the neural network. A similar function-space prior has been used in Gaussian process (Khan et al., 2020).

We assume all the network's parameters follow isotropic Gaussian prior distribution $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_P)$. The same assumption has been used in mean-field variational inference in Bayesian neural networks (Blundell et al., 2015). However, the derivation of $p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}))$ from $p(\boldsymbol{\theta})$ is intractable and is usually estimated via Monte Carlo sampling. To solve the problem, we propose to use a local linearization of the function mapping $f_{\boldsymbol{\theta}}(\cdot)$ about the mean of the prior distribution over parameters.

Proposition 2 (Rudner et al. (2021))**.** *Suppose we have a neural network whose parameters $\boldsymbol{\theta}$ follow a prior isotropic Gaussian distribution $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_P)$. Under local linearization around the prior mean, the prior distribution over function values evaluated at a finite number of samples $p(f_{\boldsymbol{\theta}}(X_\mathcal{I}))$ can be approximated by a multivariate Gaussian distribution $\tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(X_\mathcal{I}))$:*

$$p(f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})) \approx \tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})) = \mathcal{N}\left(\mathbf{0}, \sigma^2 \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \mathcal{J}_{\boldsymbol{\theta}}\left(\mathbf{X}_\mathcal{I}\right)^\top\right).$$

The intuition behind proposition 2 is that linearization turns $f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})$ into a linear function $\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})$ with respect to $\boldsymbol{\theta}$ and linear transformation of Gaussian distribution is still Gaussian. Full proof in Appendix A.1.

So the log prior term of FS-MAP in Equation (11) evaluated at a finite number of samples $X_\mathcal{I}$ can be approximate by

$$\log \tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})) = c\ f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}),$$

for $\Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I})$ as defined in Equation (7), which agrees with Equation (9) up to a constant c. The expectation in Equation (11) can be estimated by sampling $\mathbf{X}_\mathcal{I}$ uniformly from $\mathbf{X}$ at every iteration. Therefore, we arrive at the same regularization technique using a different approach.

4 Method

In this section, we formalize our approach with some practical techniques and extend it to continual learning.

4.1 Regularization in Neural Networks

Based on the discussion in Section 3, we compute the RKHS norm induced by the Neural Tangent Kernel (NTK) and use it as a function-space regularizer. Unfortunately, the naive way to compute the NTK $\Theta(\mathbf{X}, \mathbf{X})$ has $\mathcal{O}(P^2)$ space complexity and $\mathcal{O}(N^2 P)$ time complexity, which is too expensive for large networks.

So we propose to use an implicit way to compute NTK provided by the *neural tangents* library (Novak et al., 2020a) available in JAX (Bradbury et al., 2018) and we also propose to use batch estimation to avoid using full training set $\mathbf{X}$ to evaluate NTK.

Approximation 1 (Batch estimation)**.** *Suppose $\mathbf{X}_\mathcal{I}$ is sampled from the training set $\mathbf{X}$ under some distribution $q(\mathbf{x})$, then we have:*

$$\begin{aligned} \|f\|^2_{\mathcal{H}_\Theta} &= f_{\boldsymbol{\theta}}(\mathbf{X})^\top \Theta^{-1}(\mathbf{X}, \mathbf{X}) f_{\boldsymbol{\theta}}(\mathbf{X}) \\ &\approx \frac{|\mathbf{X}|}{|\mathbf{X}_\mathcal{I}|} \mathop{\mathbb{E}}_{\mathbf{X}_\mathcal{I} \sim q(\mathbf{x})} \left[f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I})^\top \Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I}) f_{\boldsymbol{\theta}}(\mathbf{X}_\mathcal{I}) \right] \end{aligned}$$

At every gradient step, we sample $\mathbf{X}_\mathcal{I}$ from $\mathbf{X}$ to compute the RKHS norm and scale it up accordingly. Note that the inverse batch NTK matrix $\Theta^{-1}(\mathbf{X}_\mathcal{I}, \mathbf{X}_\mathcal{I})$ is only a block inversion of the full NTK matrix, so it is a biased estimation of the whole inverse NTK matrix $\Theta^{-1}(\mathbf{X}, \mathbf{X})$.

The objective can be formalized as:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + \tau \mathop{\mathbb{E}}_{\mathbf{X}_{\mathcal{I}}\sim q(\mathbf{x})} \left[f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})^{\top} \Theta^{-1}(\mathbf{X}_{\mathcal{I}}, \mathbf{X}_{\mathcal{I}}) f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}}) \right], \tag{12}$$

where τ is a positive scaling factor[1]. The expectation in Equation (12) can estimated by iteratively sampling $\mathbf{X}_{\mathcal{I}}$ at every gradient step. For a detailed description, see algorithm 1.

Remark on the empirical NTK. Note that the empirical NTK in Equation (7) depends on the network parameters $\boldsymbol{\theta}$ which are constantly being updated during training, so we denote Θ_t as the empirical NTK obtained at iteration t and Θ_0 as the empirical NTK obtained at initialization.

Lee et al. (2019) has showed that for fully-connected neural networks, the empirical NTK is stable during training under infinite width assumption:

$$\|\Theta_t - \Theta_0\|_F = \mathcal{O}(n^{-1/2}), \quad n \to \infty \tag{13}$$

where $\|\cdot\|_F$ is the Frobenius norm of matrices and n is the number of neurons for each hidden layer. Arora et al. (2019) has proved that if n is sufficiently large, then for any precision ε, we have

$$\|\Theta_0 - \Theta_{ntk}\|_F < (L+1)\varepsilon \tag{14}$$

where L is the number of hidden layers. Equation (13) and Equation (14) together imply that for networks of infinite width, the empirical NTK Θ_t is the same as analytic NTK Θ_{ntk} and is constant during training.

Intuition. For infinitely wide neural networks under least squared error, our method using full $\Theta(\mathbf{X}, \mathbf{X})$ would be equivalent to kernel ridge regression under Θ_{ntk}. Therefore, our method can be viewed as parameterizing the solutions of kernel ridge regression with a neural network so that we can use stochastic gradient descent and thus avoid the expensive computation required in kernel regression. Given that we combine the expressiveness of neural networks along with the regularization technique in kernel ridge regression, it is not surprising that we achieve better generalization performance in Section 5.

4.2 Continual Learning

Continual learning is the setting where a model receives K distinct datasets sequentially one at a time and the model is required to memorize previously trained tasks when learning a new one (Kirkpatrick et al., 2017).

We denote S sequential datasets as $\mathcal{D}_s = \{\mathbf{x}_s^{(n)}, \mathbf{y}_s^{(n)}\}_{n=1}^{N_s}, \quad s = 0, 1, \ldots, S-1$. When learning the current task, the model has no access to previous datasets except for a small number of samples contained in the coreset[2] C_s. We use subscript s to indicate the parameters $\boldsymbol{\theta}_s$ and the corresponding functions $f_{\boldsymbol{\theta}_s}(\cdot)$ at task s. We also denote the NTK obtained at task s as Θ_s.

At task 0, we use maximum log-likelihood as optimization objective without any regularization. When learning task s, in order to maintain the memory on the previous task, we penalize the changes of the input-output function encoded by the neural networks. Therefore, we propose to regularize the distance between the function learned at task $s-1$ and the function to be learned at task s. In Benjamin et al. (2019), this distance is trivially measured by L_2 norm as $\|f_s - f_{s-1}\|_{L_2}$. Based on the discussion in Section 3 where we argue that the function norm should consider the Hilbert structure of the space of functions induced by the neural network, we diverge from Benjamin et al. (2019) and use the norm of $f_s - f_{s-1}$ in the RKHS $\mathcal{H}_{\Theta_{s-1}}$ after task $s-1$ has been learned, i.e. $\|f_s - f_{s-1}\|_{\mathcal{H}_{\Theta_{s-1}}}$.

In continual learning, we do not need to explicitly sample $\mathbf{X}_{\mathcal{I}}$ from $\mathbf{X}$ as we can directly evaluate function values on coresets C_s. So we write the objective for continual learning at task s as follows:

$$\mathcal{L}_s(\boldsymbol{\theta}_s) = \sum_{i=1}^{N_s} \log p(\mathbf{y}^{(i)}|f_{\boldsymbol{\theta}_s}(\mathbf{x}^{(i)})) + \tau \Delta f(C_s)^{\top} \Theta_{s-1}^{-1}(C_s, C_s) \Delta f(C_s) \tag{15}$$

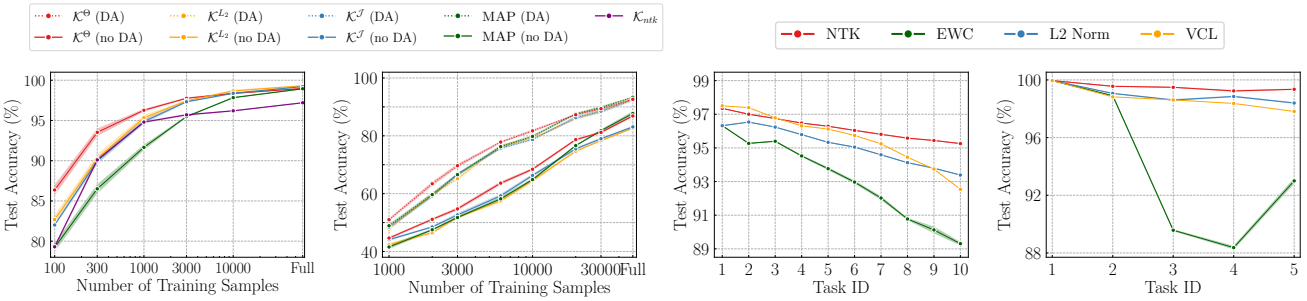
where $\Delta f(C_s) = f_{\boldsymbol{\theta}_s}(C_s) - f_{\boldsymbol{\theta}_{s-1}}(C_s)$ denotes the relative change of function values evaluated on coreset C_s from task $s-1$ to task s. Intuitively, the objective trades off fitting the data at task s while also maintaining the previous memory at task $s-1$. And similar to Benjamin et al. (2019), our approach only needs to keep a working memory of function values at coresets and the NTK matrix, which is more memory efficient than storing a snapshot of all parameters.

5 Experiments

We argue in Section 4 that we propose a better regularization technique for neural networks, so we evaluate the performance of our method in three different scenarios where regularization is particularly important.

[1] τ is a hyperparameter.

[2] Coreset is a commonly-used concept in continual learning and is usually a small set of samples (several hundred) randomly selected at every task.



(a) Accuracy on MNIST **(b)** Accuracy on CIFAR10 **(c)** Permuted MNIST **(d)** Split MNIST

Figure 1: The left two plots show test accuracy on MNIST and CIFAR-10 datasets. The right two plots show results of continual learning experiments. For all the plots, the shaded area denotes one standard error (across 10 random seeds).

Firstly, we follow Bietti et al. (2019) and evaluate the generalization performance of our approach on MNIST and CIFAR-10 by training with a small number of samples where regularization is crucial to prevent the model from overfitting. Secondly, MLP-Mixer (Tolstikhin et al., 2021) has recently drawn much attention as it has demonstrated competitive performance on large-scale datasets with much higher throughput compared to convolution networks or self-attention models. However, the application of MLP-Mixer is limited because MLP-Mixer is very easy to overfit (Tolstikhin et al., 2021). We propose to use our function-space regularizer on MLP-Mixer and show that our regularization technique alleviates overfitting and consequently leads to higher accuracy as well as lower test negative log-likelihood. Finally, based on the discussion in Section 4.2, we evaluate our approach in continual learning experiments where regularization is particularly effective in keeping memory on previously learned tasks (Nguyen et al., 2018; Kirkpatrick et al., 2017; Benjamin et al., 2019; Titsias et al., 2020; Pan et al., 2021). Note that we are not doing few-shot learning (FSL) experiments although FSL is proposed to tackle the problem of generalizing from a few examples. The reason is that existing studies (Wang et al., 2020) show that FSL requires specific techniques such as extracting prior knowledge from other related tasks, instead of simply introducing a general regularizer into training. For a comprehensive description of the experiment setups and details, see Appendix B.

Notation. In this section, we refer to our method as $\mathcal{K}^{\Theta}$, function L_2 norm in Benjamin et al. (2019) as $\mathcal{K}^{L_2}$, Jacobian norm in Bietti et al. (2019) as $\mathcal{K}^{\mathcal{J}}$. We also refer to weight decay as MAP and kernel regression under NTK as $\mathcal{K}_{ntk}$.

5.1 Generalization on Small Datasets

Following Bietti et al. (2019), we train with a small number of samples on MNIST and CIFAR-10 to demonstrate generalization performance. We also gradually increase the number of training samples to show that our method does not lead to underfitting when the number of samples is large. We use a LeNet-style (Lecun et al., 1998) network for MNIST and ResNet18 (He et al., 2015a) for CIFAR-10. We compare our method with MAP, $\mathcal{K}^{L_2}$ and $\mathcal{K}^{\mathcal{J}}$. The comparison against $\mathcal{K}_{ntk}$ is only implemented on MNIST because the kernel regression under Θ_{ntk} is too expensive on ResNet18 even with Nyström approximation.

Figure 1a and 1b show the test accuracy on MNIST and CIFAR-10 as we increase the number of training samples. On small datasets like MNIST or CIFAR-10, our networks can easily reach 100% accuracy on training set, therefore higher test accuracy indicates better generalization performance. We can see that $\mathcal{K}^{\Theta}$ has the best generalization performance when the number of training samples are small, but MAP gradually catches up as the number of training samples grows. We believe that higher test accuracy on a small number of training samples is caused by stronger regularization in that the RKHS norm $\|f_\theta\|_{\mathcal{H}_\Theta}$ enforces the neural network to find the smoothest function among all the functions in $\mathcal{H}_\Theta$ that can perfectly fit the training data, and hence achieves the best generalization performance. Moreover, our method also outperforms $\mathcal{K}_{ntk}$ due to the expressiveness of neural networks. However, as the number of training samples grow, $\mathcal{K}^{\Theta}$ loses its advantage because the large number of training samples makes the networks harder to overfit. We also note that in Figure 1b, $\mathcal{K}^{\Theta}$ is more effective when there is no data augmentation, because data augmentation is also a strong regularization technique that offsets the regularization effect of $\mathcal{K}^{\Theta}$.

5.2 Regularization on MLP-Mixer

Based on the pratical techniques discussed in Section 4.1, we are able to train an MLP-Mixer of B/16 architecture on CIFAR-10 and CIFAR-100 with objective

(12). We load MLP-Mixer B/16 parameters that are pretrained on ImageNet (Deng et al., 2009) at initialization from public resource online.

Method	CIFAR-10		CIFAR-100	
	NLL[2]	Acc (%)	NLL	Acc (%)
ML[1]	0.10±0.01	96.5±0.1	0.56±0.01	84.5±0.1
ours	0.10±0.00	**97.0**±0.1	**0.54**±0.00	**85.2**±0.1

Table 1: MLP-Mixer performance on CIFAR-10 and CIFAR-100. The reported mean and standard error are computed over five runs with different random seeds. [1]Maximum log-likelihood. [2]Negative log likelihood.

We can see that $\mathcal{K}^{\Theta}$ has higher accuracy than maximum log-likelihood without regularization. Although the improvement is quite small, the reported values of standard error show that the improvement is statistically significant so that our regularization technique is effective in reducing overfitting for MLP-Mixer. Note that the accuracy on CIFAR-10 almost reaches the upper limit so even 0.5% increase is a huge improvement.

5.3 Continual Learning

We evaluate our approach discussed in Section 4.2 on single-head permuted-MNIST and multi-head split-MNIST, which are standard continual learning benchmarks. Following prior works, we use two-layer fully-connected neural network with 100 hidden units per layer for permuted MNIST and two-layer fully-connected neural network with 256 hidden units per layer for split-MNIST. We use 200 coreset points for permuted-MNIST and 40 coreset points for split-MNIST, in accordance with prior work.

Method	Permuted MNIST	Split MNIST
EWC	84.0%±0.3	63.1%±0.2
SI	86.0%	98.9%
VCL	92.5%±0.1	97.8%±0.0
FROMP	94.9%±0.0	99.0%±0.0
FRCL	94.3%±0.0	97.8%±0.2
L_2	93.5%±0.0	98.4%±0.1
ours	**95.2%**±0.0	**99.3%**±0.0

Table 2: Comparisons of predictive average accuracy computed across all S tasks against a selection of popular continual learning methods. Both the mean and standard error are computed across ten different random seeds.

We can see from Figure 1c 1d and Table 2 that function-space methods provide higher average accuracy across all tasks than parameter-space methods, which shows that function space regularization results in better memorization of past abilities. Among all function space approaches, our method provides the highest average accuracy. We attribute our performance to effective regularization on the relative function change in the RKHS $\mathcal{H}_{\Theta_s}$, which ensures that the function to be learned at the next task $s+1$ still lies in $\mathcal{H}_{\Theta_s}$ and consequently results in less forgetting of the previous tasks.

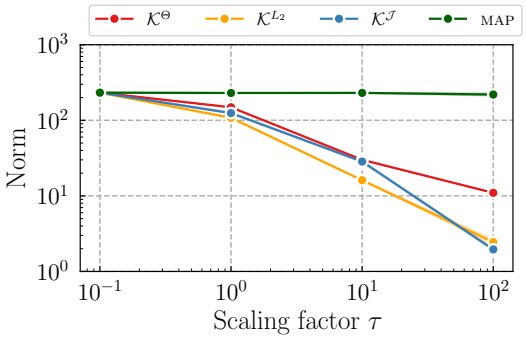


Figure 2: It is an ablation study to show the behaviour of different norms used in all regularization techniques with different scaling factor τ when training on 1000 MNIST images under Equation (12).

5.4 Ablation study

We see from Figure 2 that the $\mathcal{K}^{\Theta}$ norm decreases as scaling factor τ grows, which shows that the regularization term in Equation (12) is taking effect. We observe a discrepancy between the MAP norm and the $\mathcal{K}^{\Theta}$ norm, which shows that weight penalty cannot limit the complexity of functions, which agrees with findings in Benjamin et al. (2019) and Zhang et al. (2017). Furthermore, we find that the $\mathcal{K}^{\mathcal{J}}$ norm and the $\mathcal{K}^{L_2}$ also decreases even if we are not explicitly regularizing these norms. This phenomenon can be explained in that although L_2 norm ignores the Hilbert structure of the function space, it is a better measure of function complexity than MAP, and the $\mathcal{K}^{\mathcal{J}}$ norm is upper bounded by the $\mathcal{K}^{\Theta}$ norm (Bietti et al., 2019).

6 Conclusions

We propose to explicitly regularize in the space of functions induced by highly over-parameterized deep neural networks through regularizing the norm of functions in the RKHS associated with the Neural Tangent Kernel. We show that our approach effectively limits the complexity of functions induced by deep neural networks and results in state-of-the-art performance in continual learning. We believe that our approach could inspire further research on function-space regularization and could provide a theoretical insight towards generalization in neural networks.

References

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950. URL http://dx.doi.org/10.2307/1990404.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8141–8150, 2019.

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, 2002. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlr3.html#BartlettM02.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off, 2019.

Ari S. Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space, 2019.

Alberto Bietti and Julien Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations, 2018.

Alberto Bietti, Grégoire Mialon, Dexiong Chen, and Julien Mairal. A kernel perspective for regularizing deep neural networks, 2019.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518): 859–877, Apr 2017. ISSN 1537-274X. doi: 10.1080/01621459.2017.1285773. URL http://dx.doi.org/10.1080/01621459.2017.1285773.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

David R. Burt, Sebastian W. Ober, Adrià Garriga-Alonso, and Mark van der Wilk. Understanding variational inference in function-space, 2020.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Generalization bounds for learning kernels. 2010.

Alexander G. de G. Matthews, Mark Rowland, Jiri Hron, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks, 2018.

Alexander Graeme de Garis Matthews. *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets, 2017.

Nathaniel Eldredge. Analysis and probability on infinite-dimensional spaces, 2016.

Sebastian Farquhar, Michael Osborne, and Yarin Gal. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015b.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning, 2020.

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

Mohammad Emtiyaz Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into gaussian processes, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks, 2017.

John Lamperti. *Stochastic processes : a survey of the mathematical theory / J. Lamperti.* Applied mathematical sciences (Springer-Verlag New York Inc.); v. 23. Springer-Verlag, New York, 1977. ISBN 0387902759.

Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan):27–72, 2004.

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

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8572–8583. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/0d1a9651497a38d8b1c3871c84528bd4-Paper.pdf.

Charles A Micchelli, Massimiliano Pontil, and Peter Bartlett. Learning the kernel function via regularization. *Journal of machine learning research*, 6(7), 2005.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X.

Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning, 2018.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite

neural networks in python. In *International Conference on Learning Representations*, 2020a. URL `https://github.com/google/neural-tangents`.

Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes, 2020b.

Cheng Soon Ong, Alexander Smola, Robert Williamson, et al. Learning the kernel with hyperkernels. 2005.

Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard E. Turner, and Mohammad Emtiyaz Khan. Continual deep learning by functional regularisation of memorable past, 2021.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos, 2016.

Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

Tim G. J. Rudner, Zonghao Chen, Yee Whye Teh, and Yarin Gal. Rethinking Function-Space Variational Inference in Bayesian Neural Networks. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation, 2017.

Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. Functional variational bayesian neural networks, 2019.

Michalis K. Titsias, Jonathan Schwarz, Alexander G. de G. Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with gaussian processes, 2020.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

Amal Rannen Triki, Maxim Berman, and Matthew B. Blaschko. Function norms and regularization in deep networks, 2018.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3): 1–34, 2020.

Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 14th annual conference on neural information processing systems*, number CONF, pages 682–688, 2001.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks, 2018.

Greg Yang and Samuel S. Schoenholz. Mean field residual networks: On the edge of chaos, 2017.

Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S^2-mlp: Spatial-shift mlp architecture for vision, 2021.

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

Supplementary Materials

Appendix A Derivations and Proofs

A.1 Proof of propostion 2

This proof follows the steps in Rudner et al. (2021). We denote $\boldsymbol{\theta} \in \mathbb{R}^P$ as parameters of a stochastic neural networks that follows a Gaussian distribution $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I_P)$. Unlike in the main text, here we denote functions encoded by the neural network as $f(\cdot; \boldsymbol{\theta})$ to explicitly show that f is parameterized by $\boldsymbol{\theta}$. In proposition 2, the function f is evaluated on finite number of samples $\mathbf{X}_{\mathcal{I}}$, here we get rid of the subscript $\mathcal{I}$ and use $\mathbf{X}$ directly as there is no confusion here.

Note that $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I_P)$, so we can linearize the function f around the parameter mean $\boldsymbol{\mu}$. so the function evaluted at finite number of samples $f(\mathbf{X}; \boldsymbol{\theta})$ can be approximated as $f(\mathbf{X}; \boldsymbol{\theta}) \approx \tilde{f}(\mathbf{X}; \boldsymbol{\theta}) = f(\mathbf{X}; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{X})(\boldsymbol{\theta} - \boldsymbol{\mu})$. Therefore, the distribution of $\tilde{f}(\mathbf{X}; \boldsymbol{\theta})$ is a Gaussian multivariate distribution fully defined by some predictive mean $m(x)$ and predictive covariance $S(x, x')$, for $\forall x, x' \in \mathbf{X}$:

$$m(x) = \mathbb{E}[\tilde{f}(x; \boldsymbol{\theta})] \tag{A.1}$$

and

$$S(x, x') = \mathrm{Cov}\left(\tilde{f}(x; \boldsymbol{\theta}), \tilde{f}(x'; \boldsymbol{\theta})\right) = \mathbb{E}\left[(\tilde{f}(x; \boldsymbol{\theta}) - \mathbb{E}[\tilde{f}(x; \boldsymbol{\theta})])(\tilde{f}(x'; \boldsymbol{\theta}) - \mathbb{E}[\tilde{f}(x'; \boldsymbol{\theta})])^\top\right] \tag{A.2}$$

To see that $m(x) = \mathbb{E}[\tilde{f}(x; \boldsymbol{\theta})] = 0$, note that, by linearity of expectation, we have

$$\begin{aligned} m(x) &= \mathbb{E}[\tilde{f}(x; \boldsymbol{\theta})] = \mathbb{E}\left[f(x; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(x)(\boldsymbol{\theta} - \boldsymbol{\mu})\right] \\ &= f(x; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(x)(\mathbb{E}[\boldsymbol{\theta}] - \boldsymbol{\mu}) = f(x; \boldsymbol{\mu}) \end{aligned} \tag{A.3}$$

To see that $S(x, x') = \mathrm{Cov}\left(\tilde{f}(x; \boldsymbol{\theta}), \tilde{f}(x'; \boldsymbol{\theta})\right) = \sigma^2 \mathcal{J}_{\boldsymbol{\mu}}(x) \mathcal{J}_{\boldsymbol{\mu}}(x')^\top$, note that in general, $\mathrm{Cov}(x, x) = \mathbb{E}[xx^\top] + \mathbb{E}[x]\mathbb{E}[x]^\top$, and hence,

$$\mathrm{Cov}\left(\tilde{f}(x; \boldsymbol{\theta}), \tilde{f}(x'; \boldsymbol{\theta})\right) = \mathbb{E}\left[\tilde{f}(x; \boldsymbol{\theta}) \tilde{f}(x'; \boldsymbol{\theta})^\top\right] - \mathbb{E}[\tilde{f}(x; \boldsymbol{\theta})]\mathbb{E}[\tilde{f}(x'; \boldsymbol{\theta})]^\top \tag{A.4}$$

We already know that $\mathbb{E}[\tilde{f}(x; \boldsymbol{\theta})] = f(x; \boldsymbol{\mu})$, so we only need to find $\mathbb{E}\left[f(x; \boldsymbol{\theta}) f(x'; \boldsymbol{\theta})^\top\right]$:

$$\begin{aligned} &\mathbb{E}_{p(\boldsymbol{\theta})}\left[\tilde{f}(x; \boldsymbol{\theta}) \tilde{f}(x'; \boldsymbol{\theta})^\top\right] \\ &= \mathbb{E}_{p(\boldsymbol{\theta})}\left[\left(f(x; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(x)(\boldsymbol{\theta} - \boldsymbol{\mu})\right)\left(f(x'; \boldsymbol{\mu}) + \mathcal{J}_{\boldsymbol{\mu}}(x')(\boldsymbol{\theta} - \boldsymbol{\mu})\right)^\top\right] \\ &= \mathbb{E}_{p(\boldsymbol{\theta})}\left[f(x; \boldsymbol{\mu}) f(x'; \boldsymbol{\mu})^\top + \left(\mathcal{J}_{\boldsymbol{\mu}}(x)(\boldsymbol{\theta} - \boldsymbol{\mu})\right)\left(\mathcal{J}_{\boldsymbol{\mu}}(x')(\boldsymbol{\theta} - \boldsymbol{\mu})\right)^\top + f(x; \boldsymbol{\mu})\left(\mathcal{J}_{\boldsymbol{\mu}}(x')(\boldsymbol{\theta} - \boldsymbol{\mu})\right)^\top + \mathcal{J}_{\boldsymbol{\mu}}(x)(\boldsymbol{\theta} - \boldsymbol{\mu}) f(x'; \boldsymbol{\mu})^\top\right] \\ &= f(x; \boldsymbol{\mu}) f(x'; \boldsymbol{\mu})^\top + \mathcal{J}_{\boldsymbol{\mu}}(x)\mathbb{E}_{p(\boldsymbol{\theta})}\left[(\boldsymbol{\theta} - \boldsymbol{\mu})(\boldsymbol{\theta} - \boldsymbol{\mu})^\top\right]\mathcal{J}_{\boldsymbol{\mu}}(x')^\top \\ &+ f(x; \boldsymbol{\mu})(\mathcal{J}_{\boldsymbol{\mu}}(x')(\underbrace{\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}] - \boldsymbol{\mu}}_{=0}))^\top + \mathcal{J}_{\boldsymbol{\mu}}(x)(\underbrace{\mathbb{E}_{p(\boldsymbol{\theta})}[\boldsymbol{\theta}] - \boldsymbol{\mu}}_{=0}) f(x'; \boldsymbol{\mu})^\top \\ &= f(\mathbf{x}; \boldsymbol{\mu}) f(\mathbf{x}; \boldsymbol{\mu})^\top + \sigma^2 \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}')^\top \end{aligned} \tag{A.5}$$

Therefore, we obtain the covariance function

$$S(x, x') = \sigma^2 \mathcal{J}_{\boldsymbol{\mu}}(\mathbf{x}) \mathcal{J}_{\boldsymbol{\mu}} (\mathbf{x}')^\top \tag{A.6}$$

So far we have proved a more general form than proposition 2. In deep neural networks, the parameters $\boldsymbol{\theta}$ is usually initialized around $\mathbf{0}$ (He et al., 2015b), which indicates that mean $m(x) = f(x; \boldsymbol{\mu}) = 0$. However, the Jacobian matrix $\mathcal{J}_{\mathbf{0}}$ would also become $\mathbf{0}$, which makes the covariance matrix meaningless. So we compute the Jacobian with respect to an actual realization $\boldsymbol{\theta}$ instead of the mean. So now we have the following:

$$p(f_{\boldsymbol{\theta}}(\mathbf{X})) \approx \tilde{p}(\tilde{f}_{\boldsymbol{\theta}}(\mathbf{X})) = \mathcal{N}\left(\mathbf{0}, \sigma^2 \mathcal{J}_{\boldsymbol{\theta}}(\mathbf{X}) \mathcal{J}_{\boldsymbol{\theta}} (\mathbf{X})^\top\right) \tag{A.7}$$

which concludes the proof.

The linearization might not be accurate because the prior distribution $\mathcal{N}(\mathbf{0}, \sigma^2 I_P)$ usually has large σ and parameters for deep neural networks are high-dimensional so a sample drawn from the prior distribution might be far from the mean. Although it means that $\tilde{p}(\tilde{f}(\mathbf{X}; \boldsymbol{\theta}))$ is only a rough approximation of $p(f)$, $\tilde{p}(\tilde{f})$ still effectively incorporates the information of prior distribution of parameters and the neural network architecture into the prior distribution of functions. Moreover, using other priors over functions does not violate the general structure of FS-MAP and different priors would lead to different forms of function-space regularization for neural networks, which would be an interesting direction for future work.

A.2 Measure theoretical description of FS-MAP

de Garis Matthews (2017) has provided a rigorous definition of FS-MAP in the form of a measure theoretic version of Bayes' theorem. However, the measure theoretic version of FS-MAP does not give a definition for the log prior term in Equation (10). So in this section, we discuss how to define a prior over function spaces and then compute the log prior term in Equation (10).

Notation and background materials A measure space is described by a triple $(\mathcal{X}, \sigma_{\mathcal{X}}, \mu)$, where $\mathcal{X}$ is the underlying space, $\sigma_{\mathcal{X}}$ is the σ-algebra and μ is the measure. μ is a probability measure if $\mu(\mathcal{X}) = 1$, and we will denote $p(\mathbf{x})$ as the probability density function of μ if applicable.

Let $f : \mathcal{X} \to \mathbb{R}$ be any function on $\mathcal{X}$ and $\mathbb{R}^{\mathcal{X}} = \prod_{\mathbf{x} \in \mathcal{X}} \mathbb{R}^{\mathbf{x}}$ be the direct product of $\mathbb{R}$. Note that any function f can be identified by the evaluation on $\mathcal{X}$ via $f \mapsto [f(\mathbf{x})]_{\mathbf{x} \in \mathcal{X}} := f(\mathcal{X})$, hence f can be viewed as a vector of infinite length in $\mathbb{R}^{\mathcal{X}}$.

A note on indexing by an uncountably infinite set If $\mathcal{X}$ is finite or countable, there is no ambiguity in the construction of $\mathbb{R}^{\mathcal{X}}$ and the evaluation map $ev_{\mathcal{X}}(f) = [f(\mathbf{x})]_{\mathbf{x} \in \mathcal{X}}$ is well-defined if we arrange the points in $\mathcal{X}$ with a given order. The axiom of choice ensures we can extend this construction to the case of uncountable $\mathcal{X}$. Roughly speaking, we are able to choose a proper 'order' of uncountable index set so that the target $f(\mathcal{X})$ is determined for any given function f.

Definition 1 (Pushforward measure)**.** *Given measure spaces $(\mathcal{X}, \sigma_{\mathcal{X}}, \mu)$ and $(\mathcal{Y}, \sigma_{\mathcal{Y}}, \nu)$, let $g : \mathcal{X} \to \mathcal{Y}$ be a measurable function. We say the measure ν is pushforward of μ through g if $\nu(A) = \mu(g^{-1}(A))$ for any measurable set $A \in \sigma_{\mathcal{Y}}$.*

Definition 2 (Cylindrical σ-algebra)**.** *Let*

$$\mathbb{R}^{\mathcal{X}} = \{f : f(\mathbf{x}) \in \mathbb{R}, \mathbf{x} \in \mathcal{X}\} \tag{A.8}$$

be the set of real valued functions as above. A cylinder subset is a finitely restricted set defined as

$$C_{\mathbf{x}_1, \cdots, \mathbf{x}_n}(A_1, \cdots, A_n) = \{f \in \mathbb{R}^{\mathcal{X}} : f(\mathbf{x}_i) \in A_i, \forall 1 \le i \le n)\} \tag{A.9}$$

The cylindrical σ-algebra is the smallest σ-algebra generated by all cylinder subsets. To be precise, let

$$\mathcal{G}_{\mathbf{x}_1, \cdots, \mathbf{x}_n} = \{C_{\mathbf{x}_1, \cdots, \mathbf{x}_n}(A_1, \cdots, A_n) : A_i \in \sigma_{\mathbb{R}}, \forall 1 \le i \le n\} \tag{A.10}$$

$$\mathcal{G}_{\mathbb{R}^{\mathcal{X}}} = \bigcup_{n=1}^{\infty} \bigcup_{\mathbf{x}_i \in \mathcal{X}, i \le n} \mathcal{G}_{\mathbf{x}_1, \cdots, \mathbf{x}_n} \tag{A.11}$$

Then the cylindrical σ-algebra of $\mathbb{R}^{\mathcal{X}}$ is the smallest σ-algebra containing $\mathcal{G}_{\mathbb{R}^{\mathcal{X}}}$. Note $\sigma_{\mathbb{R}}$ is commonly chosen to be the σ-algebra of Borel sets.

Note that, if $\mathcal{X}$ is infinite, the cylindrical σ-algebra is in general not the same as product σ-algebra, which is too large to be considered. Intuitively, the generators C of cylindrical σ-algebra define a product σ-algebra on finite dimensional subspaces, which means when considering elements in $\mathbb{R}^{\mathcal{X}}$, we are actually considering the finite realizations.

Definition 3 (Canonical projection)**.** *Suppose $\mathbf{X}_{\mathcal{I}}$ and $\mathbf{X}_{\mathcal{J}}$ are two subsets of $\mathcal{X}$ such that $\mathbf{X}_{\mathcal{J}} \subseteq \mathbf{X}_{\mathcal{I}}$. We define the canonical projection map as*

$$\begin{aligned} \pi_{\mathcal{I}\to\mathcal{J}}: \quad \mathbb{R}^{\mathbf{X}_{\mathcal{I}}} \quad &\longrightarrow \quad \mathbb{R}^{\mathbf{X}_{\mathcal{J}}} \\ w = [w_{\mathbf{x}}]_{\mathbf{x}\in\mathbf{X}_{\mathcal{I}}} &\mapsto w|J = [w_{\mathbf{x}}]_{\mathbf{x}\in\mathbf{X}_{\mathcal{J}}} \end{aligned}$$

Note that $\pi_{\mathcal{I}\to\mathcal{J}}$ is actually induced by the natural projection $\mathbf{X}_{\mathcal{I}} \to \mathbf{X}_{\mathcal{J}}$ with respect to the inclusion $\mathbf{X}_{\mathcal{J}} \subseteq \mathbf{X}_{\mathcal{I}}$. We consider $\mathbf{X}_{\mathcal{I}}$ and $\mathbf{X}_{\mathcal{J}}$ are further indexed by $\mathcal{I}$ and $\mathcal{J}$, hence $\pi_{\mathcal{I}\to\mathcal{J}}$ is induced by $\pi : \mathcal{I} \to \mathcal{J}$ with respect to the inclusion $\mathcal{J} \subseteq \mathcal{I}$. Therefore, we will abuse the notations for objects indexed by $\mathbf{X}_{\mathcal{I}}$ or $\mathcal{I}$ in the following text.

Theorem 1 (Kolmogorov extension theorem))**.** *Let $\mathbb{R}^{\mathcal{X}}$ be the measurable space with cylindrical σ-algebra defined as above. Suppose for each finite subset $\mathbf{X}_{\mathcal{I}} \subseteq \mathcal{X}$, we have a probability measure $\mu_{\mathcal{I}}$ on $\mathbb{R}^{\mathbf{X}_{\mathcal{I}}}$, which satisfy the following compatibility relationship: for each subset $\mathbf{X}_{\mathcal{J}} \subset \mathbf{X}_{\mathcal{I}}$, we have*

$$\mu_{\mathcal{J}} = \mu_{\mathcal{I}} \circ \pi^{-1}_{\mathcal{I}\to\mathcal{J}} \tag{A.12}$$

Then there exists a unique probability measure μ on $\mathbb{R}^{\mathcal{X}}$ such that for all finite subsets $\mathbf{X}_{\mathcal{I}} \subset X$,

$$\mu_{\mathcal{I}} = \mu \circ \pi^{-1}_{\mathcal{X}\to\mathbf{X}_{\mathcal{I}}} \tag{A.13}$$

Kolmogorov extension theorem allows us to glue probability measures $\mu_{\mathcal{I}}$ defined on finitely dimensional subspaces together to get probability measure μ on the whole space, which is infinitely dimensional, as long as these $\mu_{\mathcal{I}}$s are consistent. Some presentation of the extension theorem may require $\mu_{\mathcal{I}}$s are invariant under permutations of $\mathbf{X}_{\mathcal{I}}$. Since we are considering elements in $\mathbb{R}^{\mathcal{X}}$, we have already assigned a particular 'order'[3] of $\mathcal{X}$. Therefore, permutation on $\mathcal{I}$ will give the same representation of $\mathbf{X}_{\mathcal{I}}$.

Construction of function prior Now consider the function $f_{\boldsymbol{\theta}}$ described by a given neural network at initialization. Let $\mathbf{X}_{\mathcal{I}}$ be an i.i.d. sample from $\mathcal{X}$ sampled according to some probability density $p(x)$. Note that $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ is a random vector, so we can form the joint probability distribution of $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ and obtain a probability measure $\mu_{\mathcal{I}}$ on $\mathbb{R}^{\mathbf{X}_{\mathcal{I}}}$. If $\mu_{\mathcal{I}}$ satisfy the compatibility condition (A.12), then Kolmogorov extension theorem implies there is a probability measure μ on $\mathbb{R}^{\mathcal{X}}$.

Note that the density of a probability measure μ is the Radon-Nikodym derivative of μ with respect to the Lebesgue measure m, i.e $p(\mathbf{x}) = \frac{\mathrm{d}\mu}{\mathrm{d}m}$. However, there is no Lebesgue measure on infinitely dimensional space since any translation invariant measure is either zero or infinite. To tackle this issue, notice that for any measurable subset A of $\mathbb{R}^{\mathcal{X}}$, A can be decomposed into a countable union of cylinder subsets. That is to say,

$$P(f \in A) = \int_A d\mu(f) = \sum_{j=1}^{\infty} \int_{A_j} \mathrm{d}\mu(f) \tag{A.14}$$

where $A_j \in C_{\mathbf{X}_{\mathcal{I}_j}}$. This infinite sum can be well approximated by the partial sum $P_n = \sum_{j=1}^{n} \int_{A_j} \mathrm{d}\mu(f)$ in the sense that if n is sufficiently large, $|P - P_n| \leq \varepsilon$ for any precision ε. Therefore, we are able to find a finite index $\mathcal{D} = \bigcup_{j=1}^{n} \mathcal{I}_j$ such that $\pi^{-1}_{\mathcal{D}\to\mathcal{I}_j}(A_j) \subseteq \mathbf{X}_{\mathcal{D}}$ for all $1 \leq j \leq n$. So calculation in Equation (A.14) can be replaced by calculation in $\mathbb{R}^{\mathbf{X}_{\mathcal{D}}}$. To be precise, we have

$$P(f \in A) \approx \sum_{j=1}^{n} \int_{A_j} \mathrm{d}\mu(f) = \sum_{j=1}^{n} \int_{\pi_{\mathcal{D}\to\mathcal{I}_j}(A_j)} \mathrm{d}\mu_{\mathcal{D}}(f(\mathbf{X}_{\mathcal{D}})) = P_{\mathcal{D}}(\pi_{\mathcal{X}\to\mathbf{X}_{\mathcal{D}}}(A)) \tag{A.15}$$

In this case, we can naturally define

$$p(f) = p(f(\mathbf{X}_{\mathcal{D}})) \tag{A.16}$$

[3]For more concrete description of the order, one may refer any book about set theory. The extension theorem can be viewed as a special case of direct limit, if the reader is familiar with category theory.

One may argue that the above choice of $\mathcal{D}$ depends on A. Note that in statistical learning theory, we often make the assumption that we have enough samples to recover the probability distribution to be learned. Hence a natural choice of a uniform $\mathbf{X}_{\mathcal{D}}$ is just the sample, in which case we consider A to be $\mathcal{X}$.

Now we check the compatibility condition. Fortunately, we have proved in Appendix A.1 that $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ follows a Gaussian distribution under linearization. Moreover, for deep neural networks under the infinite width approximation, if $\boldsymbol{\theta}$ is initialized to be standard normal, $f_{\boldsymbol{\theta}}(\mathbf{X}_{\mathcal{I}})$ follows a Gaussian distribution[4] with mean zero and a deterministic covariance matrix. Therefore, the compatibility condition is satisfied and the log prior term can be computed according to Equation (A.16).

Appendix B Experimental details

We use 10% of the training set as the validation set to conduct a hyperparameter search over the scaling factor τ and learning rate η for all methods. We choose the set of hyperparameter that yielded the lowest validation negative log-likelihood for all experiments. All experiments are implemented in JAX (Bradbury et al., 2018).

B.1 MNIST and CIFAR-10

For MNIST, we use a LeNet style network with three convolutional layers of 6, 16 and 120 5×5 filters and a fully-connected final layer of 84 hidden units. An average pooling operation is placed after each convolutional layer and ReLU activation is used. For CIFAR-10, we use ResNet-18. For both MNIST and CIFAR-10, we use SGD optimizer with a learning rate of 0.03 and momentum of 0.9. The learning rate would decay by a rate of 0.3 at every 10 epochs. During training, batch size is set to 128 for MNIST and 256 for CIFAR-10. At every iteration, 10 input points are randomly sampled from the current batch as $\mathbf{X}_{\mathcal{I}}$ to compute the RKHS norm.

B.2 MLP-Mixer

We use MLP-Mixer of B/16 architecture (Tolstikhin et al., 2021) and load the pretrained parameters on ImageNet from public source online. The training of MLP-Mixer on CIFAR-100 requires 8 GTX 3090 GPUs. We use the SGD optimizer with learning rate 0.01 and momentum 0.9.

B.3 Continual learning

For all continual learning experiments, 60,000 data samples are used for training and 10,000 data samples are used for testing. The input are converted to float values in the range of [0, 1].

Multi-head split MNIST In the multi-head setup, a model receives 5 sequential datasets and uses a different output head for each task. The original 10 classes in MNIST is split into 5 different binary classification tasks. The network is a multilayer perceptron with two layers and 100 hidden units for each layer. 40 coreset points are randomly chosen at each task[5].

Single-head permuted MNIST In the single-head setup, a model receives 10 sequential datasets and uses a same output head for each task. At every task, the MNIST images undergo a random permutation of pixels. The network is a multilayer perceptron with two layers and 256 hidden units for each layer. 200 coreset points are randomly chosen at each task.

For both multi-head split MNIST and single-head permuted MNIST, we use the Adam optimizer of learning rate 10^{-3} with default settings of $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ and we use batch size of 128 in all experiments.

[4] See Yang and Schoenholz (2017); Poole et al. (2016); Schoenholz et al. (2017); Xiao et al. (2018) for more details, and de G. Matthews et al. (2018); Novak et al. (2020b) for a formal treatment.

[5] There are other ways to select coreset such as k-centering, but this is not a continual learning paper so we only choose coreset randomly.