Efficient Parametric Approximations of Neural Network Function Space Distance

Nikita Dhawan [1 2] **Sicong Huang** [1 2] **Juhan Bae** [1 2] **Roger Grosse** [1 2 3]

Abstract

It is often useful to compactly summarize important properties of model parameters and training data so that they can be used later without storing and/or iterating over the entire dataset. As a specific case, we consider estimating the Function Space Distance (FSD) over a training set, i.e. the average discrepancy between the outputs of two neural networks. We propose a Linearized Activation Function TRick (LAFTR) and derive an efficient approximation to FSD for ReLU neural networks. The key idea is to approximate the architecture as a linear network with stochastic gating. Despite requiring only one parameter per unit of the network, our approach outcompetes other parametric approximations with larger memory requirements. Applied to continual learning, our parametric approximation is competitive with state-of-the-art nonparametric approximations, which require storing many training examples. Furthermore, we show its efficacy in estimating influence functions accurately and detecting mislabeled examples without expensive iterations over the entire dataset.

1. Introduction

As machine learning models are trained on increasingly large quantities of data or experience, it can be useful to compactly summarize information contained in a training set. In continual learning, an agent continues interacting with its environment over a long time period — longer than it is able to store explicitly. A natural goal is to avoid overwriting previously learned knowledge as it learns new tasks (Goodfellow et al., 2013) while controlling storage costs. Even in cases where it is possible to store the entire training set, a compact representation circumvents the need for expensive iterative procedures over the full data.

We focus on the problem of estimating *Function Space Distance (FSD)* for neural networks: the amount by which the outputs of two networks differ, in expectation over the training distribution. Benjamin et al. (2018) observed that regularizing FSD over the previous task data is an effective way to prevent catastrophic forgetting. Other tasks such as influence estimation (Bae et al., 2022a), model editing (Mitchell et al., 2021), unlearning (Bourtoule et al., 2021) and second-order optimization (Amari, 1998; Bae et al., 2022b) have also been formulated in terms of FSD regularization or similar locality constraints.

Methods for summarizing the training data can be categorized as parametric or nonparametric. In the context of preventing catastrophic forgetting, parametric approaches typically store the parameters of a previously trained network, along with additional information about the importance of different directions in parameter space for preserving past knowledge. The canonical example is Elastic Weight Consolidation (Kirkpatrick et al., 2017, EWC), which uses a diagonal approximation to the Fisher information matrix. Nonparametric approaches explicitly store, in addition to network parameters, a collection (coreset) of training examples, often optimized directly to be the most important or memorable ones (Rudner et al., 2022; Pan et al., 2020; Titsias et al., 2019). Currently, the most effective approaches to prevent catastrophic forgetting are nonparametric due to the lack of sufficiently accurate parametric models. However, this advantage comes at the expense of high storage requirements.

In this paper, we formally formulate neural network FSD estimation and propose novel parametric approximations. To motivate our approach, notice that several parametric approximations, like EWC, can be interpreted as a second-order Taylor approximation to the FSD. This leads to a quadratic form involving the Fisher information matrix $\boldsymbol{F_\theta}$ or some other metric matrix $\boldsymbol{G_\theta}$, where $\boldsymbol{\theta}$ denotes the network parameters. Second-order approximations are practically useful because one can estimate $\boldsymbol{F_\theta}$ or $\boldsymbol{G}_\theta$ by sampling vectors from a distribution with these matrices as the covariance (Martens & Grosse, 2015). Then, tractable probabilistic models can be fit to these samples to approximate the

[1]University of Toronto [2]Vector Institute [3]Anthropic. Correspondence to: Nikita Dhawan <nikita@cs.toronto.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

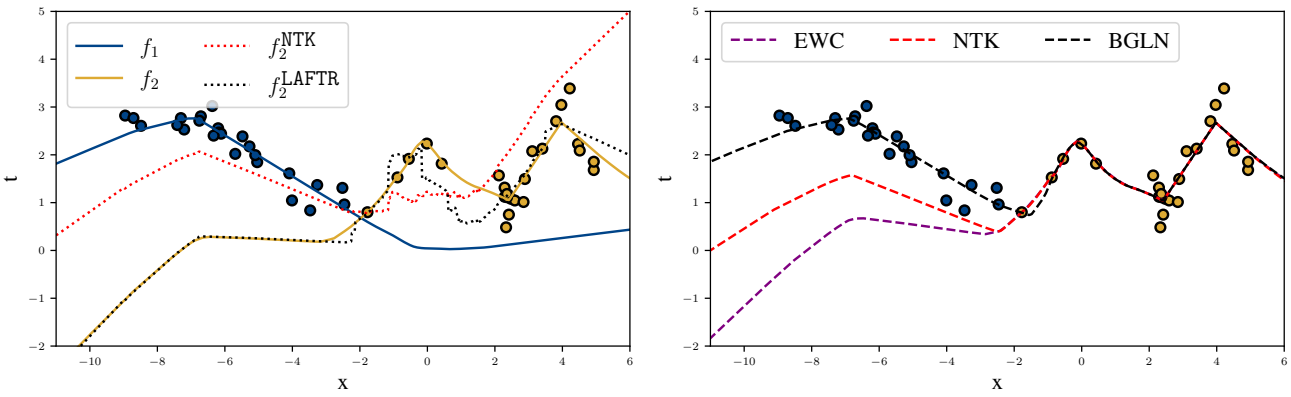

Figure 1. Comparison of FSD regularization on a 1-D regression task. **(Left)** Training sequentially on two tasks (blue yields f_1, then yellow yields f_2) results in catastrophic forgetting. The LAFTR approximation more closely matches the true function f_2 than its NTK approximation does. **(Right)** BGLN retains performance on task 1 after training on task 2 more accurately than EWC and NTK.

corresponding distribution. Unfortunately, these tend to be inaccurate for continual learning compared to nonparametric approaches. We believe the culprit is the second-order Taylor approximation: we show in several examples that even the exact second-order Taylor approximation can perform poorly in terms of average classification accuracy and backward transfer in sequentially learned tasks. Since such an approximation can be interpreted as network linearization (Grosse, 2021), this finding is consistent with a recent line of results that find linearized approximations of neural networks to be an inaccurate model of their behavior (Seleznova & Kutyniok, 2022a;b; Hanin & Nica, 2019; Bai et al., 2020; Huang & Yau, 2020). In Section 2, we present this network linearization perspective of some existing approaches for regularization in function space.

Our method, based on a *Linearized Activation Function TRick (LAFTR)*, does *not* make a second-order Taylor approximation in the parameter space, and hence is able to capture nonlinear interactions between parameters of the network. Specifically, it linearizes each step of the network's forward pass with respect to its inputs. In the case of ReLU networks, our approximation yields a linear network with stochastic gating, which we refer to as the *Bernoulli Gated Linear Network (BGLN)*. We derive a stochastic and a deterministic estimate of FSD, both of which rely only on the first two moments of the data. This allows the application of our methods in different scenarios where stochasticity is or isn't desirable.

We evaluate our BGLN approximation in the contexts of continual learning and influence function estimation. Our method significantly outperforms previous parametric approximations despite being much more memory-efficient. For continual learning tasks, our method is competitive with nonparametric approaches. For influence function estimation tasks, it closely matches an oracle estimate of a network's loss after a data point is removed, but without having to iterate over the whole dataset.

The key contributions and findings of this work are:

- We introduce LAFTR, an idealized FSD approximation, which improves over parameter space linearization by capturing nonlinear interactions between weights in different layers.
- We propose the Bernoulli Gated Linear Network (BGLN), an efficient parametric FSD approximation for ReLU networks based on LAFTR which stores only aggregate statistics of the data and the activations.
- In continual learning, BGLN outcompetes state-of-the-art methods on sequential MNIST and CIFAR100 tasks, with significantly lower memory requirements than nonparametric methods.
- For influence function estimation, BGLN efficiently approximates the effect of removing a single data point without iterating over or storing the full dataset.

2. Background

Let $\boldsymbol{z} = f(\mathbf{x}, \boldsymbol{\theta})$ denote the function computed by a neural network, which takes in inputs $\mathbf{x}$ and parameters $\boldsymbol{\theta}$. Consistent with prior works, we use FSD to refer to the expected output space distance[1] ρ between the outputs of two neural networks (Benjamin et al., 2018; Grosse, 2021; Bae et al., 2022b) with respect to the training distribution, as defined in equation 1. When the population distribution is inaccessible, the empirical distribution is often used as a proxy:

$$D(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, p_{\text{data}}) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\rho(f(\mathbf{x}, \boldsymbol{\theta}_0), f(\mathbf{x}, \boldsymbol{\theta}_1))] \quad (1)$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} \rho(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}_0), f(\mathbf{x}^{(i)}, \boldsymbol{\theta}_1)), \quad (2)$$

where p_{data} is the data-generating distribution. Constraining the FSD term has been successful in preventing catas-

[1]Note that we use the term *distance* throughout since we focus on Euclidean distance in our derivation. However, other metrics like KL divergence can also be used, as shown in Section 5.

trophic forgetting (Benjamin et al., 2018), computing influence functions (Bae et al., 2022a), training teacher-student models (Hinton et al., 2015), and fine-tuning pre-trained networks (Jiang et al., 2019; Mitchell et al., 2021). Natural choices for ρ are Euclidean distance for networks trained using mean-squared error (e.g. regression) and KL divergence for those trained with cross-entropy loss (e.g. classification).

Consider the continual learning setting as a motivating example. Common benchmarks (Normandin et al., 2021) involve sequentially learning tasks $t \in \{1, \ldots, T\}$, using loss function $\mathcal{L}$ and a penalty on the FSD between the parameters $\boldsymbol{\theta}$, and the parameters $\{\boldsymbol{\theta}_i\}$ fit to previous tasks. The penalty is computed over the previously seen data distribution p_i and then scaled by a hyperparameter λ_{FSD}:

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) + \lambda_{\text{FSD}} \sum_{i=1}^{t-1} D(\boldsymbol{\theta}, \boldsymbol{\theta}_i, p_i). \quad (3)$$

Continuing with the notation in equation 2, one way to regularize the FSD is to store the training set and explicitly evaluate the network outputs using both $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ (perhaps on random mini-batches). However, this has the drawbacks of having to store and access the entire training set throughout training (precisely the thing continual learning research tries to avoid) and necessarily estimating FSD stochastically. Instead, we would like to compactly summarize information about the training set or distribution.

Many (but not all) practical FSD approximations are based on a second-order Taylor approximation:

$$D(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, p_{\text{data}}) \approx \frac{1}{2}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_0)^\top \boldsymbol{G}_{\boldsymbol{\theta}} (\boldsymbol{\theta}_1 - \boldsymbol{\theta}_0), \quad (4)$$

where $\boldsymbol{G}_{\boldsymbol{\theta}} = \nabla^2_{\boldsymbol{\theta}} D(\boldsymbol{\theta}_0, \boldsymbol{\theta}, p_{\text{data}})$ is the corresponding Hessian. In the case where the network outputs parametrize a probability distribution and ρ corresponds to KL divergence, $\boldsymbol{G}_{\boldsymbol{\theta}}$ reduces to the more familiar Fisher information matrix $F_{\boldsymbol{\theta}} = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}, \mathbf{y}\sim P_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \log p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{x})^\top]$, where $P_{\mathbf{y}|\mathbf{x}}(\boldsymbol{\theta})$ represents the model's predictive distribution over $\mathbf{y}$. It is possible to sample random vectors in parameter space whose covariance is $\boldsymbol{G}_{\boldsymbol{\theta}}$ (Martens et al., 2012; Grosse & Martens, 2016; Grosse, 2021) and some parametric FSD approximations work by fitting simple statistical models to the resulting distribution. For instance, assuming all coordinates are independent gives a diagonal approximation (Kirkpatrick et al., 2017), and more fine-grained independence assumptions between network layers yield a Kronecker-factored approximation (Martens & Grosse, 2015; Ritter et al., 2018). In practice, instead of sampling vectors whose covariance is $\boldsymbol{G}_{\boldsymbol{\theta}}$, many works use the empirical gradients during training, whose covariance is the empirical Fisher matrix. We caution the reader that the empirical Fisher matrix is less well motivated theoretically and can result in different behavior (Kunstner et al., 2019).

3. A Parametric Estimate with LAFTR

We introduce and apply **LAFTR** (Linearized Activation Function TRick) to linear ReLU networks and propose **BGLN** (Bernoulli Gated Linear Network) which approximates a given model architecture as a linear network with stochastic gating. While it is applicable to different architectures, we first explicitly derive our approximation for multilayer perceptrons (MLPs) with L fully-connected layers and ReLU activation function ϕ. We also discuss its generalization to convolutional networks and empirically evaluate the same in Section 5.

For MLPs with inputs $\mathbf{x}$ drawn from p_{data}, layer l weights and biases $(\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)})$, and outputs $\boldsymbol{z}$, the computation of preactivations and activations at each layer is recursively defined as follows:

$$\boldsymbol{s}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \; \boldsymbol{a}^{(l)} = \phi(\boldsymbol{s}^{(l)}) \quad (5)$$

with $\boldsymbol{a}^{(0)} = \boldsymbol{x}$, and $\boldsymbol{s}^{(L)} = \boldsymbol{z}$. We denote $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$ to be samples of the output distribution obtained with parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$, respectively.

3.1. Linearized Activation Function TRick

Given parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ of two networks, we linearize *each step of the forward pass* around its value under $\boldsymbol{\theta}_0$. For an MLP that alternates between linear layers and nonlinear activation functions, the linear transformations are unmodified while the activation functions are replaced with a first-order Taylor approximation around their inputs. Hence, the network's computation becomes linear in $\mathbf{x}$ (but, importantly, remains nonlinear in $\boldsymbol{\theta}$). Let $(\boldsymbol{W}_i^{(l)}, \boldsymbol{b}_i^{(l)})$ denote the weights and biases of layer l in network i.

$$\boldsymbol{s}_0^{(l)} = \boldsymbol{W}_0^{(l)} \boldsymbol{a}_0^{(l-1)} + \boldsymbol{b}_0^{(l)} \quad (6)$$

$$\boldsymbol{a}_0^{(l)} = \phi(\boldsymbol{s}_0^{(l)}) \quad (7)$$

$$\boldsymbol{s}_1^{(l)} = \boldsymbol{W}_1^{(l)} \boldsymbol{a}_1^{(l-1)} + \boldsymbol{b}_1^{(l)} \quad (8)$$

$$\boldsymbol{a}_1^{(l)} = \phi(\boldsymbol{s}_0^{(l)}) + \phi'(\boldsymbol{s}_0^{(l)}) \odot (\boldsymbol{s}_1^{(l)} - \boldsymbol{s}_0^{(l)}), \quad (9)$$

where ϕ' is the derivative of the activation function.

We define some additional notation for differences between preactivations and activations. For $\Delta\boldsymbol{s}^{(l)} = \boldsymbol{s}_1^{(l)} - \boldsymbol{s}_0^{(l)}$, $\Delta\boldsymbol{a}^{(l)} = \boldsymbol{a}_1^{(l)} - \boldsymbol{a}_0^{(l)}$, $\Delta\boldsymbol{W}^{(l)} = \boldsymbol{W}_1^{(l)} - \boldsymbol{W}_0^{(l)}$, and $\Delta\boldsymbol{b}^{(l)} = \boldsymbol{b}_1^{(l)} - \boldsymbol{b}_0^{(l)}$, we have the following formulae:

$$\Delta\boldsymbol{s}^{(l)} = \Delta\boldsymbol{W}^{(l)} \boldsymbol{a}_0^{(l-1)} + \boldsymbol{W}_1^{(l)} \Delta\boldsymbol{a}^{(l-1)} + \Delta\boldsymbol{b}^{(l)} \quad (10)$$

$$\Delta\boldsymbol{a}^{(l)} = \phi'(\boldsymbol{s}_0^{(l)}) \odot \Delta\boldsymbol{s}^{(l)}. \quad (11)$$

Here, base cases are $\boldsymbol{a}_0^{(0)} = \mathbf{x}$ and $\Delta\boldsymbol{a}^{(0)} = 0$. Written this way, the Δ terms at each step of computation rely linearly on corresponding terms from the immediately preceding

step, making LAFTR conducive to implementation via propagation of the base cases through the network. Observe that with $\boldsymbol{W}_0$ held fixed, the model parametrized using $\boldsymbol{W}_1$ is a linear network, i.e. the network's exact outputs are a linear function of its inputs.

There are two significant differences between LAFTR and parameter space linearization (referred to as NTK (Jacot et al., 2018) henceforth), which confer an advantage to the former. First, our linearization is with respect to inputs instead of parameters (Lee et al., 2019), hence capturing nonlinear interactions between the parameters in different layers. Second, the only computations that introduce linearization errors into our approximation are those involving activation functions; hence, our method is exact for linear networks, whereas NTK is only approximate. We note that linear networks are commonly used to model nonlinear training dynamics of neural networks (Saxe et al., 2013). Hence, we regard our weaker form of linearity as a significant advantage over Taylor approximations in parameter space. We note that LAFTR can be extended directly to other types of linear layer, such as convolution layers.

In the following two sections, we use the intuition above to motivate two probabilistic approximations which enable a memory-efficient implementation of this algorithm. One is to approximate preactivation signs and the other is for the input data distribution.

3.2. Bernoulli Gating

In the specific case of ReLU networks, observe that LAFTR depends on the training data only through the signs of preactivations, which we denote as a mask $\mathbf{m} = \mathbb{1}\{\boldsymbol{s} > 0\}$, since $\phi(\boldsymbol{s}) = \mathbf{m} \odot \boldsymbol{s}$ and $\phi'(\boldsymbol{s}) \odot \Delta\boldsymbol{s} = \mathbf{m} \odot \Delta\boldsymbol{s}$. Here, ϕ' is the derivative of the ReLU function, given by $\phi'(\boldsymbol{s}) = \mathbb{1}\{\boldsymbol{s} > 0\}$. We approximate $\mathbf{m}$ as a vector of independent Bernoulli random variables and fit its mean vector $\boldsymbol{\mu}$ using maximum likelihood estimation (i.e., computing the fraction of times the unit is activated). We can accordingly rewrite equations 7 and 11 as $\boldsymbol{a}_0^{(l)} = \mathbf{m}^{(l)} \odot \boldsymbol{s}_0^{(l)}$ and $\Delta\boldsymbol{a}^{(l)} = \mathbf{m}^{(l)} \odot \Delta\boldsymbol{s}^{(l)}$, respectively, where $\odot$ denotes element-wise multiplication and $\mathbf{m}^{(l)} \sim Ber(\boldsymbol{\mu}^{(l)})$. For efficiency, we compute the activation statistics over the last epoch of training. We call this approximation the Bernoulli Gated Linear Network (BGLN).

Note that this gating technique is not specific only to MLPs. It can be implemented in ReLU convolutional networks by replacing activations with Bernoulli random variables.

3.3. Propagating Moments of the Activations

A key insight enabling efficient computation is that when the output distance ρ is chosen to be (squared) Euclidean distance, the FSD depends only on the first and second moments of the output difference $\Delta\boldsymbol{z} := \boldsymbol{z}_1 - \boldsymbol{z}_0$:

$$\mathbb{E}\left[\tfrac{1}{2}||\Delta\boldsymbol{z}||^2\right] = \tfrac{1}{2}||\mathbb{E}[\Delta\mathbf{z}]||^2 + \tfrac{1}{2}tr\left(\mathrm{Cov}(\Delta\mathbf{z})\right). \quad (12)$$

We compute these terms recursively by propagating the first two moments of $\boldsymbol{a}_0^{(l)}$ and $\Delta\boldsymbol{a}^{(l)}$ through the network. Using equations 10 and 11, we obtain the following equations for the first moments (see Appendix B for analogous equations for second moments):

$$\mathbb{E}[\boldsymbol{s}_0^{(l)}] = \boldsymbol{W}_0^{(l)}\mathbb{E}[\boldsymbol{a}_0^{(l-1)}] + \boldsymbol{b}_0^{(l)}$$
$$\mathbb{E}[\boldsymbol{a}_0^{(l)}] = \boldsymbol{\mu}^{(l)} \odot \mathbb{E}[\boldsymbol{s}_0^{(l)}]$$
$$\mathbb{E}[\Delta\boldsymbol{s}^{(l)}] = \Delta\boldsymbol{W}^{(l)}\mathbb{E}[\boldsymbol{a}_0^{(l-1)}] + \boldsymbol{W}_1^{(l)}\mathbb{E}[\Delta\boldsymbol{a}^{(l-1)}] + \Delta\boldsymbol{b}^{(l)}$$
$$\mathbb{E}[\Delta\boldsymbol{a}^{(l)}] = \boldsymbol{\mu}^{(l)} \odot \mathbb{E}[\Delta\boldsymbol{s}^{(l)}]$$

Hence, LAFTR-based FSD approximation depends only on the first two moments of the data. This view leads to two insights: (1) we can store the first two moments of the data instead of a coreset and compute a deterministic FSD estimate and (2) we can approximate the data distribution as a multivariate Gaussian parameterized by its moments and compute an unbiased stochastic estimate of the LAFTR-based FSD by sampling from it.

3.4. BGLN-D and BGLN-S

The most straightforward way to use the BGLN approximation is to draw Monte Carlo samples of the random variables. When only the first and second moments matter, we are free to assume Gaussianity of the inputs. We denote this method BGLN-S (S for "stochastic"). This is sufficient in situations where FSD is used as a regularization term in stochastic gradient-based optimization (as we do in our continual learning experiments). In other situations, it is advantageous to have a deterministic computation; for instance, optimization with nonlinear conjugate gradient requires a deterministic objective. In the case of Euclidean distance as the output space metric, we can exactly compute the BGLN approximation by propagating the first and second moments of all random variables through the forward pass, as described in Section 3.3. We call this deterministic estimator BGLN-D. **BGLN-S** is outlined in Algorithm 1 and **BGLN-D** in Algorithm 2. We describe the analogous BGLN-S computations for convolutional networks in Appendix C.

We present the above algorithms such that they depend on storing the mean and covariance of the data. Note that storing moments of the data requires less memory than storing sufficient subsets of the data itself in several practical settings demonstrated in Table 4. We can further reduce the memory requirement by approximating the covariance matrix as a diagonal matrix, i.e., using only the variance of each dimension of the inputs. This is equivalent in cost to storing two data points per task. We empirically investigate

Algorithm 1 A stochastic version of BGLN (BGLN-S)

Require: $\mathbb{E}[\mathbf{x}], \text{Cov}(\mathbf{x}), \{\boldsymbol{W}, \boldsymbol{b}\}_1^L, \{\boldsymbol{\mu}\}_1^{L-1}$
$\mathbb{E}[\boldsymbol{a}_0], \mathbb{E}[\boldsymbol{a}_1] \leftarrow \mathbb{E}[\mathbf{x}]$
$\text{Cov}(\boldsymbol{a}_0), \text{Cov}(\boldsymbol{a}_1) \leftarrow \text{Cov}(\mathbf{x})$
$\boldsymbol{a}_0, \boldsymbol{a}_1 \sim \mathcal{N}(\mathbb{E}[\boldsymbol{a}_0], \text{Cov}(\boldsymbol{a}_0))$ {sample inputs}
$\Delta \boldsymbol{a} \leftarrow 0$
for $l \leftarrow 1$ to $L-1$ **do**
  $\mathbf{m} \sim Ber(\boldsymbol{\mu}^{(l)})$ {sample Bernoullis}
  $\boldsymbol{s}_0 \leftarrow \boldsymbol{W}_0^{(l)} \boldsymbol{a}_0 + \boldsymbol{b}_0^{(l)}$
  $\Delta \boldsymbol{s} \leftarrow \Delta \boldsymbol{W}^{(l)} \boldsymbol{a}_0 + \boldsymbol{W}_1^{(l)} \Delta \boldsymbol{a} + \Delta \boldsymbol{b}^{(l)}$
  $\boldsymbol{a}_0 \leftarrow \mathbf{m} \odot \boldsymbol{s}_0$ {stochastic gating}
  $\Delta \boldsymbol{a} \leftarrow \mathbf{m} \odot \Delta \boldsymbol{s}$
  $\boldsymbol{s}_1 \leftarrow \boldsymbol{W}_1^{(l)} \boldsymbol{a}_1 + \boldsymbol{b}_1$
  $\boldsymbol{a}_1 \leftarrow \boldsymbol{a}_0 + \Delta \boldsymbol{a}$ {linearized activations}
end for
$\Delta \boldsymbol{z} \leftarrow \Delta \boldsymbol{W}^{(L)} \boldsymbol{a}_0 + \boldsymbol{W}_1^{(L)} \Delta \boldsymbol{a} + \Delta \boldsymbol{b}^{(L)}$
return $\frac{1}{2} ||\Delta \boldsymbol{z}||^2$

the effect of this approximation on continual learning benchmarks. Furthermore, capturing the expected FSD over the training distribution in a single term, via a single forward pass, is far less computationally expensive than iterative alternatives.

3.5. Class-conditional Estimates

In the classification setting, it is also possible to a extend BGLN to a more fine-grained, class-conditional approximation. In particular, we can fit a mixture model for our probabilistic approximations, with one component per class. In this case, each class has its own associated input moments and Bernoulli mean parameters. The lower memory cost of our method allows for this when number of classes is not too large. We refer to theis variant as BGLN-CW. As expected, it boosts performance in our continual learning experiments at the expense of a slightly higher memory requirement, as shown in Tables 1, 2 and 3.

4. Related Works

Several works (Benjamin et al., 2018; Bernstein et al., 2020; Bae et al., 2022b) have highlighted the importance of measuring meaningful distances between neural networks. Benjamin et al. (2018) contrast training dynamics in parameter space and function space and observe that function space distances are often more useful than, and not always correlated with, parameter space distances. Bae et al. (2022b) propose an Amortized Proximal Optimization (APO) scheme that regularizes an FSD estimate to the previous iterate for second-order optimization. Natural gradient descent (Amari et al., 1995; Amari, 1998) can also be interpreted as a steepest descent method, using a second-order Taylor approximation to the FSD (Pascanu & Bengio, 2014).

Parisi et al. (2019); De Lange et al. (2021); Ramasesh et al. (2020); Normandin et al. (2021) have reviewed and surveyed the challenge of catastrophic forgetting in continual learning, along with benchmarks and metrics to evaluate different methods. Parametric methods focus on different approximations to the weight space metric matrix, like diagonal (Kirkpatrick et al., 2017, EWC) or Kronecker-factored (Ritter et al., 2018, OSLA). As described in Section 2, we interpret these as second-order Taylor approximations to the FSD with further structured approximations to the Hessian. Several methods are motivated as approximations to a posterior Gaussian distribution in a Bayesian setting (Ebrahimi et al., 2019), for instance through a variational lower bound (Nguyen et al., 2017) or via Gaussian process inducing points (Kapoor et al., 2021). Non-parametric methods (Kapoor et al., 2021; Titsias et al., 2019; Pan et al., 2020; Rudner et al.; 2022; Kirichenko et al., 2021) usually employ some form of experience replay of stored or optimized data points. Some of these methods (Pan et al., 2020) can also be related to the Neural Tangent Kernel (Jacot et al., 2018, NTK), or in other words, network linearization. Doan et al. (2021) directly study forgetting in continual learning in the infinite width NTK regime. Mirzadeh et al. (2022) further study the impact of network widths on forgetting.

In this paper, we also examine influence functions (Cook, 1979; Hampel, 1974) which is another application that involves the FSD between networks. Influence functions are a classical robust statistics technique that has since been used in machine learning (Koh & Liang, 2017). Bae et al. (2022a) formally study influence functions in neural networks and show that they approximate an objective called the proximal Bregman response function (PBRF). This approximation depends on a FSD term that is typically computed by iterating through the full training dataset.

5. Experiments

We empirically assess the effectiveness of LAFTR (our idealized method) and the BGLN (our practical algorithm) in approximating FSD as well as their usefulness for downstream tasks: continual learning and influence function estimation. The experiments investigate the following questions:

- Can LAFTR outperform NTK in approximating FSD?
- Does LAFTR improve performance and memory cost on continual learning benchmarks relative to existing methods?
- How do choice of output space metric ρ, or the use of the Gaussian input and the Bernoulli activation approximations, impact empirical performance?
- Can the BGLN perform competitively with iteration-based influence function estimators without requiring

Method	Split MNIST	Permuted MNIST
Nonparametric		
VCL (coreset)	98.40	95.50
VAR-GP (coreset)	90.57 ± 1.06	$\mathbf{97.20 \pm 0.08}$
FROMP (coreset)	99.00 ± 0.04	94.90 ± 0.04
S-FSVI (coreset)	99.54 ± 0.04	95.76 ± 0.02
NTK (coreset)	99.50 ± 0.09	96.46 ± 0.11
BGLN-S (coreset)	99.50 ± 0.03	96.36 ± 0.13
Parametric		
EWC	63.10	84.00
OSLA	80.56	95.73
VCL	97.00	87.50 ± 0.61
BGLN-D	99.72 ± 0.03	96.03 ± 0.20
BGLN-D-CW	$\mathbf{99.78 \pm 0.02}$	96.85 ± 0.02
BGLN-S	99.64 ± 0.04	96.36 ± 0.12
BGLN-S-CW	99.77 ± 0.05	96.99 ± 0.07
BGLN-D-Var	99.64 ± 0.04	94.98 ± 0.18
BGLN-S-Var	99.50 ± 0.03	96.36 ± 0.13

Table 1. Average accuracies of nonparametric and parametric approaches on Split and Permuted MNIST datasets.

Method	Split MNIST	Permuted MNIST
FROMP	-0.50 ± 0.20	-1.00 ± 0.10
S-FSVI	-0.21 ± 0.06	-0.65 ± 0.21
BGLN-S	$\mathbf{-0.04 \pm 0.03}$	-0.41 ± 0.08
BGLN-D	-0.09 ± 0.04	-0.56 ± 0.04
BGLN-S-CW	-0.18 ± 0.06	$\mathbf{-0.37 \pm 0.14}$
BGLN-D-CW	-0.07 ± 0.07	-1.17 ± 0.07

Table 2. Backward transfer on Split and Permuted MNIST. Higher is better.

iteration over the dataset?

5.1. Comparing FSD Estimators

To conduct further empirical analysis of our methods' estimation and minimization of the true (empirical) FSD, we use tasks and models from standard continual learning settings which are prone to forgetting. These include Split MNIST, Permuted MNIST and Split CIFAR100 (Pan et al., 2020; Rudner et al., 2022). In addition to directly evaluating the continual learning performance, we also use a collection of networks trained in the course of this experiment (with varying hyperparameter settings) to directly evaluate the accuracy of the FSD estimates. Specifically, we vary the learning rate and the number of training iterations, and consider the set of trained networks that result; on each pair of these networks, we compare the FSD estimates against the true empirical FSD computed using the full training set.

Figure 2 (Left) shows that BGLN-S and BGLN-D consistently estimate the true FSD more accurately than NTK. Analogous analysis using CIFAR100 shows a similar trend in Figure 4. We can also measure how the true FSD changes when different FSD estimates are optimized during training, as in Figure 2 (Middle), where BGLN methods more effectively minimize true FSD as new task accuracy increases. Finally, we train networks of varying depths on CIFAR100 tasks and measure correlation (Spearman rank-order (Spearman, 1961) and Kendall's Tau (Kendall, 1938)) with true FSD. Figure 2 (Right) shows that LAFTR has a higher correlation using both metrics, and its advantage over NTK increases with network depth, and hence with a number of nonlinear interactions between parameters. This corroborates our intuition that LAFTR captures nonlinearities that NTK is unable to account for.

5.2. Continual Learning

Recall the formulation of continual learning in terms of FSD, as described in equation 3. We visualize our method's comparative performance on 1-D regression with two sequential tasks shown in Figure 1. More realistically, we test our methods on standard benchmarks used in prior works (Pan et al., 2020; Rudner et al., 2022), with standard architectures for a fair comparison. See Appendix E.2 for details on the datasets, architectures, and hyperparameters. We evaluate average final accuracy across tasks, backward transfer (Lopez-Paz & Ranzato, 2017) and memory cost.

Toy Regression. Figure 1 shows the functions learned by different methods when sequentially trained on two one-dimensional regression tasks. LAFTR gives a better approximation of the learned function than NTK. When used to regularize the network, BGLN retains good predictions on both tasks, while EWC and exact parameter space linearization (NTK) suffer catastrophic forgetting. We hypothesize that this difference in performance is due to important nonlinearities between network parameters that EWC and NTK approximations are unable to capture.

Split and Permuted MNIST. As shown in Tables 1 and 2, our LAFTR-based methods outperform other parametric methods (EWC, OSLA, and VCL) on Split and Permuted MNIST tasks and are competitive with the state-of-the-art (SOTA) nonparametric methods, in terms of average accuracy. Class-conditional approximations further boost performance and the diagonal approximation to input covariance (BGLN-S-Var, BGLN-D-Var) does not harm it significantly. With respect to the backward transfer, a more direct measure of forgetting, BGLN methods significantly outperform the SOTA. Finally, they are also amenable to successful adaptation to the nonparametric setting when a coreset is available for use in place of Gaussian samples.

Split CIFAR100. We consider the much more challenging Split CIFAR100 task to compare our method to existing approaches and tease apart the effects of our algorithmic choices and approximations. Table 3 summarizes these results and analytically compares the memory costs associated with each method (see Appendix E.1 for details). **Coreset** refers to a coreset of real inputs vs. Gaussian samples,

Method	ρ	Coreset	Bernoulli	CW	Average Accuracy ↑	Backward Transfer ↑	Memory Cost
Nonparametric							
VCL (coreset)		–			67.40 ± 0.60	–	$2P + Nd$
VAR-GP (coreset)		–			–	–	$2P + Nd + C^2N^2$
FROMP (coreset)		–			76.20 ± 0.20	-2.60 ± 0.90	$2P + Nd + C^2N^2$
S-FSVI (coreset)		–			77.60 ± 0.20	-2.50 ± 0.20	$2P + Nd + C^2N^2$
NTK (coreset)	KL		–		77.61 ± 0.20	-2.03 ± 0.04	$2P + Nd$
LAFTR (coreset)	KL	✓	✗	✗	$\mathbf{78.33 \pm 0.01}$	$\mathbf{-0.73 \pm 0.10}$	$P + Nd$
BGLN-S (coreset)	KL	✓	✓	✗	73.27 ± 0.01	-4.99 ± 0.28	$P + A + Nd$
LAFTR (coreset)	Euclidean	✓	✗	✗	76.22 ± 0.01	-2.64 ± 0.40	$P + Nd$
Parametric							
EWC		–			71.60 ± 0.40	–	$2P$
OSLA		–			72.61	–	$P + \sum_{l=1}^{L} p_l^2$
VCL		–			–	–	$2P$
LAFTR	KL	✗	✗	✗	75.61 ± 0.01	-1.93 ± 0.59	$P + d + d^2$
LAFTR-CW	KL	✗	✗	✓	76.22 ± 0.01	-1.45 ± 0.63	$P + C(d + d^2)$
BGLN-S	KL	✗	✓	✗	72.37 ± 0.01	-8.20 ± 0.04	$P + A + d + d^2$
BGLN-S-CW	KL	✗	✓	✓	74.02 ± 0.01	-2.44 ± 0.15	$P + C(A + d + d^2)$
LAFTR	Euclidean	✗	✗	✗	75.51 ± 0.01	-3.12 ± 0.44	$P + d + d^2$
BGLN-S	Euclidean	✗	✓	✗	74.29 ± 0.01	-5.49 ± 0.05	$P + A + d + d^2$
BGLN-S-CW	Euclidean	✗	✓	✓	77.78 ± 0.01	-1.75 ± 0.50	$P + C(A + d + d^2)$

Table 3. Split CIFAR100: Average Accuracy and Backward Transfer. Notation for memory cost: p_l = # parameters in layer l, P = # parameters $= \sum_{l=1}^{L} p_l$, A = # activations $< P$, d = data dimension, N = coreset size, C = # classes.

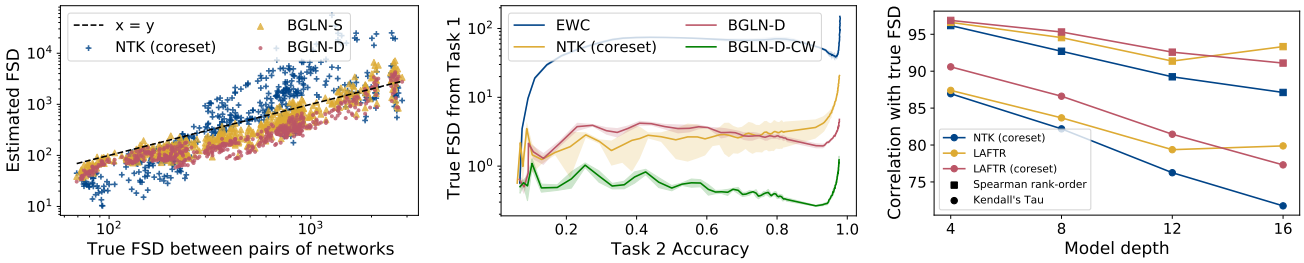

Figure 2. Comparison of different FSD estimators. **(Left)** Compared to NTK, BGLN-S and BGLN-D consistently give closer FSD values to the true empirical FSD. **(Middle)** While training on task 2, FSD from the optimal task 1 parameters increases with task 2 accuracy. Optimizing BGLN-D and class-conditioned BGLN-D-CW effectively minimizes the true FSD. **(Right)** LAFTR has a higher correlation with true FSD than NTK, with a more significant advantage as network depth (and hence a number of nonlinear interactions) increases.

Bernoulli refers to Bernoulli activations vs. simply passing preactivations through the ReLU function (termed LAFTR here) and **CW** refers to the class-conditonal estimate. Our empirical analysis shows that given a random coreset of the same size as comparable methods, LAFTR outperforms the SOTA as well as the NTK baseline on average accuracy and backward transfer significantly. Other LAFTR and BGLN variants remain competitive with prior methods while incurring lower memory costs. We also observe that performance is hurt to some extent by the Gaussian and Bernoulli modeling assumptions, while it is improved by class-conditioning. We present a more fine-grained task-wise comparison of accuracies in Figure 5 and results on a longer task sequence in Table 10 of Appendix E.3.

Memory Gains. To demonstrate the memory gains for our LAFTR and LAFTR (coreset) methods in practical scenarios, we compute the percentage reduction in memory costs relative to those of typical nonparametric methods. Specifically, we fix the network architecture to be a convolutional network used for training Split CIFAR100, and select reasonable values of coreset size, number of classes per task and data dimension from a range encountered in practice. We then use equations in Table 3 to compute $100 \times \frac{B-L}{B}$, where B and L are the memory costs of a nonparametric method and a LAFTR variant, respectively. As summarized in Table 4, the gains in memory complexity enjoyed by LAFTR methods increase as coreset size is increased, number of classes are increased or data dimension is decreased. Further results on Split CIFAR100 performance as amount of stored information (for example, coreset size) is varied are included in Appendix E.4.

5.3. Influence Function Estimation

To further assess BGLN's applicability to other settings involving FSD estimation and regularization, we consider influence function estimation (Cook, 1979; Hampel, 1974;

C \ d	1000	2000	3000	N
10	66.27	24.08	−43.71	**200**
	74.84	43.26	−7.60	**250**
20	87.86	72.15	46.33	**200**
	91.81	81.18	63.68	**250**
50	97.78	94.87	90.03	**200**
	98.57	96.69	93.56	**250**

C \ d	1000	2000	3000	N
10	78.14	75.89	73.77	**200**
	83.14	80.90	78.79	**250**
20	92.13	91.15	90.20	**200**
	94.51	93.67	92.84	**250**
50	98.56	98.37	98.18	**200**
	99.04	98.88	98.73	**250**

Table 4. Percentage reduction in memory cost of LAFTR (**Left**) and LAFTR (coreset) (**Right**) relative to typical nonparametric methods for a fixed network architecture and varying values of C (number of classes), d (data dimension) and N (coreset size). Higher percentages are better. In both cases, gains in memory complexity increase as N increases, C increases or d decreases.

Dataset	EWC		CG		BGLN-D	
	P	S	P	S	P	S
Concrete	0.78	0.57	0.92	0.94	**0.96**	**0.97**
Energy	0.68	0.39	0.97	0.98	**0.99**	**0.98**
Housing	0.86	0.33	0.92	**0.89**	**0.95**	0.83
Kinetics	0.36	0.30	0.88	0.86	**0.99**	**0.99**
Wine	0.97	0.70	0.99	**0.94**	**0.99**	0.90

Table 5. Comparison of training loss differences computed by EWC, CG and BGLN-D. We show Pearson (P) and Spearman rank-order (S) correlations with the PBRF estimates.

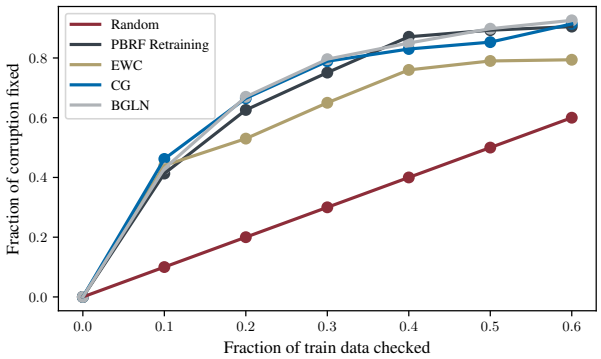

Figure 3. Effectiveness of BGLN in detecting mislabeled examples. BGLN can approximate the FSD term in the PBRF objective accurately and be used in applications involving influence functions without explicitly storing or iterating over the dataset.

Koh & Liang, 2017). Given parameters $\boldsymbol{\theta}_0$ trained on dataset $\mathcal{D}_{\text{train}}$ of size N, influence functions approximate the parameters $\boldsymbol{\theta}_-$ that would be obtained by training without a particular point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}_{\text{train}}$. The difference in loss between $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_-$ is an indicator of the influence of $(\boldsymbol{x}, \boldsymbol{y})$ on the trained network.

Bae et al. (2022a) show that influence functions in neural networks can be formulated as solving for the proximal Bregman response function (PBRF):

$$\boldsymbol{\theta}_- = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} -\frac{1}{N}\mathcal{L}(f(\boldsymbol{x},\boldsymbol{\theta}),\boldsymbol{y}) + D_B(\boldsymbol{\theta},\boldsymbol{\theta}_0,p_{\text{train}}) + \frac{\lambda}{2}\|\boldsymbol{\theta}-\boldsymbol{\theta}_0\|^2. \tag{13}$$

Here, the first term maximizes the loss of the data point we are interested in removing. The second term is the Bregman divergence defined on network outputs and measures the FSD between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_0$ over training distribution p_{train} similar to the FSD term as defined in equation 1. For standard loss functions like squared error and cross-entropy, the Bregman divergence term is equivalent to the soft training error where the original targets are replaced with soft targets produced by $\boldsymbol{\theta}_0$. Finally, the last term is a proximity term with strength $\lambda > 0$, which prevents large changes in weight space. Intuitively, the PBRF maximizes the loss of data we would like to remove while constraining the network in both function and weight space so that the predictions and losses of other training examples remain unaffected.

Existing approaches for this optimization face two key challenges: (1) the entire training dataset must be stored and iterated over, requiring as many forward passes as there are mini-batches and (2) techniques like nonlinear Conjugate Gradient (CG) (Hager & Zhang, 2006) do not work well with the stochastic gradients produced by sampling batches of data. LAFTR enables estimating the PBRF (or FSD) by storing only the first two data moments, requires just a single forward pass to compute it and provides a deterministic function to optimize, implemented as BGLN-D.

Regression. We first train a MLP with two hidden layers and ReLU activations for 200 epochs on regression datasets from the UCI benchmark (Dua & Graff, 2017). Then, we randomly select 50 independent data points to be removed. For each removed point, we sample batches and use a Stochastic Gradient Descent (SGD) optimizer to minimize the PBRF objective and compute the difference in loss after removing that data point, commonly referred to as the self-influence score (Koh & Liang, 2017; Schioppa et al., 2022). Next, we follow the same procedure as above but approximate the FSD term in the PBRF objective with EWC, CG (Koh & Liang, 2017) and BGLN-D. Since the direct minimization of PBRF can be considered as the ground truth for influence estimation, we compare the alignment

of these methods' estimates with that of PBRF via Pearson correlation (Sedgwick, 2012) and Spearman rank-order correlation (Spearman, 1961). The results are shown in Table 5. Without having to iterate over or store the entire dataset, BGLN-D correlates with PBRF more strongly than EWC and CG (Koh & Liang, 2017), which can be seen as minimizing a linearized version of the PBRF objective.

Mislabeled Example Detection. Influence function estimators are commonly evaluated in terms of their ability to identify mislabeled examples. Intuitively, if some fraction of the training labels is corrupted, they would behave as outliers and have a more significant influence on the training loss (self-influence score). One approach to efficiently detect and correct these examples is to prioritize and examine training inputs with higher self-influence scores. Following the evaluation setup from Bae et al. (2022a), we use 10% of the MNIST dataset and corrupt 10% of it by assigning random labels to it. We train a two layer MLP with 1024 hidden units and ReLU activations using SGD with a batch size of 128. Then, we use EWC, CG and BGLN-D to approximate the FSD term in equation 13 and compute individual self-influence scores. We also compare these methods against a baseline of randomly sampling data points to check for corruption. The results are summarized in Figure 3. BGLN-D significantly outperforms the random baseline and EWC and closely matches the oracle PBRF and CG, while being much faster, cheaper and more memory-efficient.

6. Conclusions

In this work, we addressed the problem of compactly summarizing a model's predictions on a given dataset, and formulated it as approximating neural network FSD. We developed the Linearized Activation Function TRick as an improvement over network linearization in parameter space and proposed novel parametric methods, BGLN, to estimate FSD. Our methods capture nonlinearities between network parameters, are much more memory-efficient than prior works and are amenable to adaptation to the nonparametric setting when a coreset of data is available.

We empirically show that LAFTR-based estimates are highly correlated with the true FSD across several settings. In continual learning, our methods outcompete existing methods without storing any data samples. Further, in influence function estimation, they estimate influence-scores with high correlation and can efficiently detect mislabeled examples without expensive iteration over the whole dataset.

Extending the formulation of FSD approximation to other applications like model editing or unlearning are exciting research avenues. We hope that our work inspires methods to further enhance memory and computational efficiency in settings where estimating or constraining FSD is relevant.

Acknowledgements

We would like to thank Florian Shkurti for useful discussions, Cem Anil for feedback on the draft, and Gerald Shen for assistance with the compute environment. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute (www.vectorinstitute.ai/partners).

References

Amari, S. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, 1998. doi: 10.1162/089976698300017746. URL https://doi.org/10.1162/089976698300017746.

Amari, S., Cichocki, A., and Yang, H. H. A new learning algorithm for blind signal separation. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, pp. 757–763, Cambridge, MA, USA, 1995. MIT Press.

Bae, J., Ng, N., Lo, A., Ghassemi, M., and Grosse, R. B. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967, 2022a.

Bae, J., Vicol, P., HaoChen, J. Z., and Grosse, R. B. Amortized proximal optimization. *Advances in Neural Information Processing Systems*, 35:8982–8997, 2022b.

Bai, Y., Krause, B., Wang, H., Xiong, C., and Socher, R. Taylorized training: Towards better approximation of neural network training at finite width. *arXiv preprint arXiv:2002.04010*, 2020.

Benjamin, A. S., Rolnick, D., and Kording, K. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

Bernstein, J., Vahdat, A., Yue, Y., and Liu, M.-Y. On the distance between two neural networks and the stability of learning. *Advances in Neural Information Processing Systems*, 33:21370–21381, 2020.

Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C. A., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 141–159. IEEE, 2021.

Cook, R. D. Influential observations in linear regression. *Journal of the American Statistical Association*, 74(365): 169–174, 1979.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A

continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Doan, T., Bennani, M. A., Mazoure, B., Rabusseau, G., and Alquier, P. A theoretical analysis of catastrophic forgetting through the ntk overlap matrix. In *International Conference on Artificial Intelligence and Statistics*, pp. 1072–1080. PMLR, 2021.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Ebrahimi, S., Elhoseiny, M., Darrell, T., and Rohrbach, M. Uncertainty-guided continual learning with bayesian neural networks. *arXiv preprint arXiv:1906.02425*, 2019.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Grosse, R. University of toronto CSC2541, topics in machine learning: Neural net training dynamics, chapter 4: Second-Order optimization. Lecture Notes, 2021. URL https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2021/readings/L04_second_order.pdf.

Grosse, R. and Martens, J. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2016.

Hager, W. W. and Zhang, H. A survey of nonlinear conjugate gradient methods. *Pacific journal of Optimization*, 2(1): 35–58, 2006.

Hampel, F. R. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974. doi: 10.1080/01621459.1974.10482962. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1974.10482962.

Hanin, B. and Nica, M. Finite depth and width corrections to the neural tangent kernel. In *International Conference on Learning Representations*, 2019.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Huang, J. and Yau, H.-T. Dynamics of deep neural networks and neural tangent hierarchy. In *International conference on machine learning*, pp. 4542–4551. PMLR, 2020.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Zhao, T. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

Kapoor, S., Karaletsos, T., and Bui, T. D. Variational autoregressive gaussian processes for continual learning. In *International Conference on Machine Learning*, pp. 5290–5300. PMLR, 2021.

Kendall, M. G. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. ISSN 00063444. URL http://www.jstor.org/stable/2332226.

Kirichenko, P., Farajtabar, M., Rao, D., Lakshminarayanan, B., Levine, N., Li, A., Hu, H., Wilson, A. G., and Pascanu, R. Task-agnostic continual learning with hybrid probabilistic models. *arXiv preprint arXiv:2106.12772*, 2021.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Krizhevsky, A., Nair, V., and Hinton, G. CIFAR-10 (canadian institute for advanced research). a. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Krizhevsky, A., Nair, V., and Hinton, G. CIFAR-100 (canadian institute for advanced research). b. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Kunstner, F., Hennig, P., and Balles, L. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in neural information processing systems*, 32, 2019.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Martens, J., Sutskever, I., and Swersky, K. Estimating the hessian by back-propagating curvature. *arXiv preprint arXiv:1206.6464*, 2012.

Mirzadeh, S. I., Chaudhry, A., Yin, D., Hu, H., Pascanu, R., Gorur, D., and Farajtabar, M. Wide neural networks forget less catastrophically. In *International Conference on Machine Learning*, pp. 15699–15717. PMLR, 2022.

Mitchell, E., Lin, C., Bosselut, A., Finn, C., and Manning, C. D. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

Normandin, F., Golemo, F., Ostapenko, O., Rodriguez, P., Riemer, M. D., Hurtado, J., Khetarpal, K., Zhao, D., Lindeborg, R., Lesort, T., et al. Sequoia: A software framework to unify continual learning research. *arXiv preprint arXiv:2108.01005*, 2021.

Pan, P., Swaroop, S., Immer, A., Eschenhagen, R., Turner, R., and Khan, M. E. E. Continual deep learning by functional regularisation of memorable past. *Advances in Neural Information Processing Systems*, 33:4453–4464, 2020.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Netw.*, 113(C):54–71, may 2019. ISSN 0893-6080. doi: 10.1016/j.neunet.2019.01.012. URL https://doi.org/10.1016/j.neunet.2019.01.012.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. *CoRR*, abs/1301.3584, 2014.

Ramasesh, V. V., Dyer, E., and Raghu, M. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020.

Ritter, H., Botev, A., and Barber, D. Online structured laplace approximations for overcoming catastrophic forgetting. *Advances in Neural Information Processing Systems*, 31, 2018.

Rudner, T. G., Smith, F. B., Feng, Q., Teh, Y. W., and Gal, Y. Continual learning via function-space variational inference.

Rudner, T. G. J., Smith, F. B., Feng, Q., Teh, Y. W., and Gal, Y. Continual Learning via Sequential Function-Space Variational Inference. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2022.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Schioppa, A., Zablotskaia, P., Vilar, D., and Sokolov, A. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8179–8186, 2022.

Sedgwick, P. Pearson's correlation coefficient. *Bmj*, 345, 2012.

Seleznova, M. and Kutyniok, G. Analyzing finite neural networks: Can we trust neural tangent kernel theory? In *Mathematical and Scientific Machine Learning*, pp. 868–895. PMLR, 2022a.

Seleznova, M. and Kutyniok, G. Neural tangent kernel beyond the infinite-width limit: Effects of depth and initialization. *arXiv preprint arXiv:2202.00553*, 2022b.

Spearman, C. The proof and measurement of association between two things. 1961.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. Functional regularisation for continual learning with gaussian processes. *arXiv preprint arXiv:1901.11356*, 2019.

Appendix

A. Notation

f	Function corresponding to a neural network
ϕ	ReLU activation function
ϕ'	Derivative of a function ϕ
$\boldsymbol{s}_l$	Preactivations at layer l
$\boldsymbol{a}_l = \phi(\boldsymbol{s}_l)$	Activations at layer l
$\mathbf{x}$	Input data
$\boldsymbol{y}$	Target data
$\mathbf{x}^{(i)}$	Input data point i
p_{data}	Data distribution from which $\mathbf{x}$ is sampled
d	Data dimension
N	Number of data points in a coreset
L	Number of layers in a network
C	Number of classes in a classification task
T	Number of tasks
$\boldsymbol{\theta}$	Parameters of a neural network
$\boldsymbol{\theta}_t$	Parameters obtained after training on task t
p_l	Number of parameters in layer l
P	Total number of parameters in a network $= \sum_{l=1}^{L} p_l$
$\boldsymbol{z} = f(\mathbf{x}; \boldsymbol{\theta})$	Prediction of the network f on $\mathbf{x}$ parameterized by $\boldsymbol{\theta}$
ρ	Output space distance, for instance Euclidean distance
$D(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, p_{\text{data}})$	FSD between networks parameterized by $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ over data distribution p_{data}
$\boldsymbol{G}_{\boldsymbol{\theta}}$	Weight space metric matrix
$\boldsymbol{F}_{\boldsymbol{\theta}}$	Fisher information matrix
$\mathbf{m}$	Mask vector or Bernoulli random variable
$\boldsymbol{\mu}$	Bernoulli mean parameter

B. Recursion Equations for BGLN-D

We derive the deterministic version of our algorithm by taking expectations and covariances for the quantities in equations 6 to 11 (rewritten using Bernoulli variables). We use linearity of expectations and our Bernoulli modeling approximation. We also assume $\text{Cov}(\boldsymbol{a}_0, \Delta\boldsymbol{a})$ is close to 0 and ignore it in our computations. This assumption is tested empirically in our experiments and we find that it does not severely move the FSD estimate away from the true empirical FSD (see Figures 2 and 4). The complete steps for BGLN-D computations are given in Algorithm 2.

C. Generalization to Convolutional Networks

The generalization of BGLN-S to convolutional networks involves passing the inputs sampled using the data moments through the network. In convolutional networks, ReLU activation is usually applied after the convolutional and the fully

Algorithm 2 BGLN-D

Require: $\mathbb{E}[\mathbf{x}], \mathrm{Cov}(\mathbf{x}), \{\boldsymbol{W}, \boldsymbol{b}\}_1^L, \{\boldsymbol{\mu}\}_1^{L-1}$
  $\mathbb{E}[\boldsymbol{a}_0], \mathbb{E}[\boldsymbol{a}_1] \leftarrow \mathbb{E}[\mathbf{x}]$
  $\mathrm{Cov}(\boldsymbol{a}_0), \mathrm{Cov}(\boldsymbol{a}_1) \leftarrow \mathrm{Cov}(\mathbf{x})$
  $\mathbb{E}[\Delta\boldsymbol{a}], \mathrm{Cov}(\Delta\boldsymbol{a}) \leftarrow 0$
  for $l \leftarrow 1$ to $L-1$ **do**
    $\mathbb{E}[\boldsymbol{s}_0] \leftarrow \boldsymbol{W}_0^{(l)}\mathbb{E}[\boldsymbol{a}_0] + \boldsymbol{b}_0^{(l)}$
    $\mathbb{E}[\Delta\boldsymbol{s}] \leftarrow \Delta\boldsymbol{W}^{(l)}\mathbb{E}[\boldsymbol{a}_0] + \boldsymbol{W}_1^{(l)}\mathbb{E}[\Delta\boldsymbol{a}] + \Delta\boldsymbol{b}^{(l)}$
    $\mathbb{E}[\boldsymbol{a}_0] \leftarrow \boldsymbol{\mu}^{(l)} \odot \mathbb{E}[\boldsymbol{s}_0]$
    $\mathbb{E}[\Delta\boldsymbol{a}] \leftarrow \boldsymbol{\mu}^{(l)} \odot \mathbb{E}[\Delta\boldsymbol{s}]$
    $\mathrm{Cov}(\boldsymbol{s}_0) \leftarrow \boldsymbol{W}_0^{(l)}\mathrm{Cov}(\boldsymbol{a}_0)\boldsymbol{W}_0^{(l)T}$
    $\mathrm{Cov}(\Delta\boldsymbol{s}) \leftarrow \Delta\boldsymbol{W}^{(l)}\mathrm{Cov}(\boldsymbol{a}_0)\Delta\boldsymbol{W}^{(l)T} + \boldsymbol{W}_1^{(l)}\mathrm{Cov}(\Delta\boldsymbol{a})\boldsymbol{W}_1^{(l)T}$
    $\mathrm{Cov}(\boldsymbol{a}_0) \leftarrow (\boldsymbol{\mu}^{(l)}\boldsymbol{\mu}^{(l)T}) \odot \mathrm{Cov}(\boldsymbol{s}_0)$
    $\mathrm{Cov}(\Delta\boldsymbol{a}) \leftarrow (\boldsymbol{\mu}^{(l)}\boldsymbol{\mu}^{(l)T}) \odot \mathrm{Cov}(\Delta\boldsymbol{s})$
  end for
  $\mathbb{E}[\Delta\boldsymbol{z}] \leftarrow \Delta\boldsymbol{W}^{(L)}\mathbb{E}[\boldsymbol{a}_0] + \boldsymbol{W}_1^{(L)}\mathbb{E}[\Delta\boldsymbol{a}] + \Delta\boldsymbol{b}^{(L)}$
  $\mathrm{Cov}(\Delta\boldsymbol{z}) \leftarrow \Delta\boldsymbol{W}^{(L)}\mathrm{Cov}(\boldsymbol{a}_0)\Delta\boldsymbol{W}^{(L)T} + \boldsymbol{W}_1^{(L)}\mathrm{Cov}(\Delta\boldsymbol{a})\boldsymbol{W}_1^{(L)T}$
  return $\frac{1}{2}||\mathbb{E}[\Delta\boldsymbol{z}]||^2 + \frac{1}{2}tr\left(\mathrm{Cov}(\Delta\boldsymbol{z})\right)$

connected layers. At each ReLU step, we use the Bernoulli mean parameters to sample activation signs and obtain the difference of activations, $\Delta\boldsymbol{a}$. Finally, the Euclidean distance (or KL divergence) between the final layer outputs leads to the stochastic estimate of the FSD. The complete procedure is shown in Algorithm 3.

D. Comparing FSD Estimators

To directly compare the NTK approximation with the linearized activation function trick, we compute the estimated FSD between pairs of networks using these two kinds of linearization, when provided with the same information, and plot them against the true empirical FSD. Hence, both estimators are provided with the same coreset of datapoints. In the case of BGLN-S (coreset), activations are computed using this coreset directly instead of Bernoulli sampling. Figure 4 visualizes this comparison for networks trained using tasks in Split CIFAR100. BGLN-S estimates correlate better with the true FSD than NTK, hence corroborating our intuition about linearizing activation functions. To quantify this difference, we also measure the Spearman rank-order and Kendall's Tau correlation coefficients for each estimator with the true FSD. BGLN-S obtains values 96.36 and 79.19, respectively, outperforming NTK, which obtains 86.42 and 71.71, respectively.

E. Continual Learning

E.1. Memory Cost Analysis

We follow the notation in Table 6 to denote p_l as the number of parameters in layer l, $P = \sum_{l=1}^{L} p_l$ as the total number of parameters in the network, A as the number of activations in the network (note that $A < P$), d as the data dimension, N as the number of samples in a coreset, and C as the number of classes in the continual learning classification setting. We can now write analytic expressions for the memory cost incurred by the different methods considered in the continual learning experiments, as shown in Table 3. Below we arrive at these expressions for each task that the model is continually trained on.

Table 6. Memory cost notation.

p_l	# parameters in layer l
P	# parameters $= \sum_{l=1}^{L} p_l$
A	# activations $< P$
d	data dimension
N	coreset size
C	# classes

- **EWC:** EWC requires storing one value for each parameter of the network and one value for each diagonal element of the $P \times P$ Fisher information matrix, resulting in a cost of $2P$.
- **OSLA:** OSLA approximates the Fisher information matrix as a block diagonal matrix, storing p_l^2 elements for each block corresponding to layer l. This gives a cost of $P + \sum_{l=1}^{L} p_l^2$.

Algorithm 3 BGLN-S (Conv)

Require: $\mathbb{E}[\mathbf{x}], \mathrm{Cov}(\mathbf{x}), \{\texttt{layer}\}_1^{L-1}, \{\mu\}_1^{L-1}$
$\mathbb{E}[\boldsymbol{a}_0], \mathbb{E}[\boldsymbol{a}_1] \leftarrow \mathbb{E}[\mathbf{x}]$
$\mathrm{Cov}(\boldsymbol{a}_0), \mathrm{Cov}(\boldsymbol{a}_1) \leftarrow \mathrm{Cov}(\mathbf{x})$
$\boldsymbol{a}_0, \boldsymbol{a}_1 \sim \mathcal{N}(\mathbb{E}[\boldsymbol{a}_0], \mathrm{Cov}(\boldsymbol{a}_0))$
$\Delta a \leftarrow 0$
for $l \leftarrow 1$ to $L-1$ **do**
  if `layer` is `Conv` or `FC` **then**
    $\boldsymbol{s}_0 \leftarrow \texttt{layer}(\boldsymbol{a}_0, \texttt{grad=False})$
    $\boldsymbol{s}_1 \leftarrow \texttt{layer}(\boldsymbol{a}_1)$
  else if `layer` is `ReLU` **then**
    $\Delta \boldsymbol{s} = \boldsymbol{s}_1 - \boldsymbol{s}_0$
    $\mathbf{m} \sim Ber(\mu^{(l)})$
    $\Delta \boldsymbol{a} \leftarrow \mathbf{m} \odot \Delta \boldsymbol{s}$
  else
    $\boldsymbol{a}_0 \leftarrow \texttt{layer}(\boldsymbol{a}_0)$
    $\boldsymbol{a}_1 \leftarrow \texttt{layer}(\boldsymbol{a}_1)$
  end if
end for
$\Delta \boldsymbol{z} \leftarrow \boldsymbol{s}_1 - \boldsymbol{s}_0$
return $\frac{1}{2}||\Delta \boldsymbol{z}||^2$

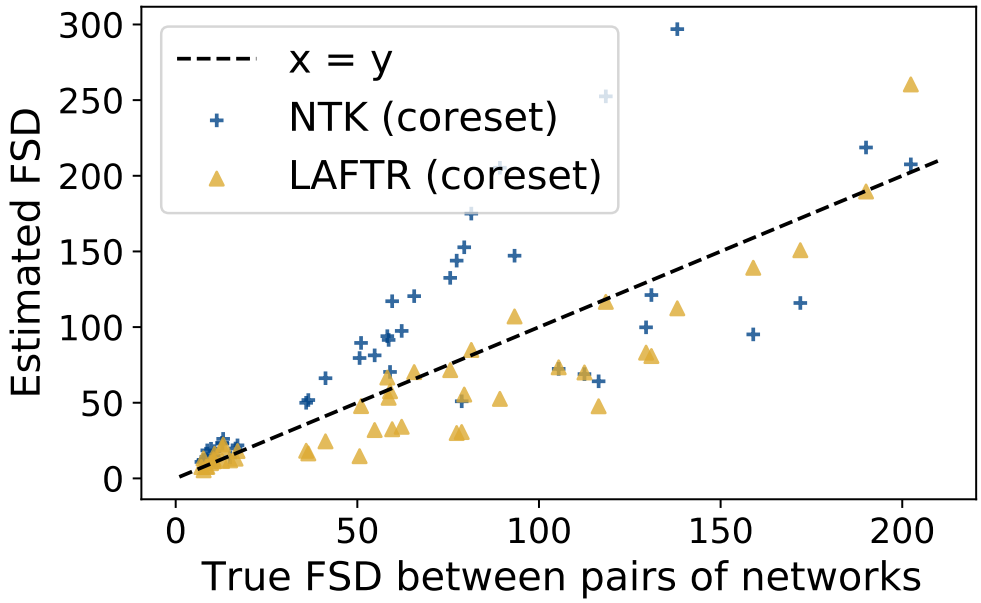

Figure 4. Estimating true FSD with LAFTR and NTK, given the same coreset of inputs, on networks trained using Split CIFAR100.

- **VCL:** VCL stores two pieces of information for the variational distribution for each parameter, one for the mean and one for the variance in the diagonal approximation, incurring a cost equal to $2P$.
- **VCL (coreset):** The "coreset" variant of VCL additionally stores N datapoints, increasing the memory cost by Nd.
- **VAR-GP, FROMP, S-FSVI:** These methods all store two values per parameter, similar to VCL. Further, they require a coreset of datapoints and/or inducing points, as well as a $NC \times NC$ kernel matrix.
- **NTK (coreset):** The NTK approximation stores one value for each parameter of the network and one for the $P-$dimensional Jacobian-vector product used to linearize the network. It further requires a coreset of N datapoints,

result in a $2P + Nd$ cost.

- **LAFTR:** LAFTR methods store each parameter once and the first two moments of the data, incurring a cost of $P + d + d^2$.
- **LAFTR-CW:** The classwise variants of LAFTR store separate data moments for each class, which scales those corresponding memory costs by C, i.e., $P + C(d + d^2)$.
- **LAFTR (coreset):** The "coreset"" ablation of LAFTR requires storing N datapoints instead of the data moments, giving a $P + Nd$ cost.
- **BGLN-S, BGLN-D:** BGLN methods store each parameter once, a Bernoulli mean value for each activation and the first two moments of the data, incurring a cost of $P + A + d + d^2$.
- **BGLN-S-CW, BGLN-D-CW:** The classwise variants of our methods store separate Bernoulli means and data moments for each class, which scales those corresponding memory costs by C, i.e., $P + C(A + d + d^2)$.
- **BGLN-S-Var, BGLN-D-Var:** The "Var" variants of our methods make a diagonal approximation to the data covariance (second moment), hence storing only d values for it. This further reduces memory cost to $P + A2d$.
- **BGLN-S (coreset):** The "coreset" ablation of BGLN-S requires storing N datapoints instead of the data moments, giving a $P + A + Nd$ cost.

E.2. Experimental Details

Datasets. Split MNIST consists of five binary prediction tasks to classify non-overlapping pairs of MNIST digits (Deng, 2012). Permuted MNIST is a sequence of ten tasks to classify ten digits, with a different fixed random permutation applied to the pixels of all training images for each task. Finally, Split CIFAR100 consists of six ten-way classification tasks, with the first being CIFAR10 (Krizhevsky et al., a), and subsequent ones containing ten non-overlapping classes each from the CIFAR100 dataset (Krizhevsky et al., b).

Architectures. We use standard architectures used by existing methods for fair comparison. For regression and the MNIST experiments, we use a MLP with two fully connected layers and ReLU activation. For Split CIFAR100, we use a network with four convolutional layers, followed by two fully connected layers, and ReLU activation after each. For MNIST tasks, we compute the FSD with Euclidean output space metric between logits. For CIFAR100, we do the same between softmax outputs. Both Split MNIST and Split CIFAR100 models have a multiheaded final layer. Note that the VAR-GP method included in the results in Section 5 is specific only to singleheaded architectures and does not generalize to the multiheaded networks that our methods and all other comparable methods use.

Hyperparameters. We have performed a grid search over some key hyperparameters and used the ones that resulted in the best final average accuracy across all tasks. All hyperparameter search was done with random seed 42. We then took that best set of hyperparameters, repeated our experiments on seeds 20, 21, 22, and reported the average and standard deviation of our results.

For all nonparametric methods that store and use a coreset of datapoints, we use 40 points for MNIST datasets and 200 points for CIFAR 100, in accordance with standard protocol followed by comparable methods.

For the learning rate, we used 0.001 for all CL experiments except the BGLN-S method for Split MNIST and BGLN-D method for Permuted MNIST, where we used 0.0001 instead.

We used the same number of epochs on each CL task and the exact numbers are reported in Table 7. On the first task, all MNIST experiments used the same number of epochs as the subsequent CL tasks while CIFAR100 experiments used 200 epochs on the first task.

To compute the Bernoulli mean parameters for our stochastic gating implementation, we used simple averaging as the default, but also explored exponential moving averaging. While there was not much difference in performance, we report the momentum values that reproduce our results. All MNIST experiments had a momentum value of $1/\texttt{batch\_size}$. Note that this momentum value of $1/\texttt{batch\_size}$ corresponds to simple moving average. For CIFAR100 experiments, we used 0.99 for BGLN-S (CW) and NTK, and $1/\texttt{batch\_size}$ for BGLN-S.

Table 7. CL tasks training epochs used in CL experiments.

Method	Split MNIST	Permuted MNIST	Split CIFAR100
NTK (coreset)	15	5	80
BGLN-S	15	15	80
BGLN-D	15	15	-
BGLN-S-CW	15	15	50
BGLN-D-CW	15	15	-

Table 8. FSD scale used in CL experiments.

Method	Split MNIST	Permuted MNIST	Split CIFAR100
NTK (coreset)	1	1	0.005
BGLN-S	5	1	1
BGLN-D	0.1	0.005	-
BGLN-S-CW	2	1	10
BGLN-D-CW	0.1	0.005	-

Table 9. Batch size used in CL experiments.

Method	Split MNIST	Permuted MNIST	Split CIFAR100
NTK (coreset)	256	256	512
BGLN-S	32	128	512
BGLN-D	32	128	-
BGLN-S-CW	32	128	512
BGLN-D-CW	32	128	-

For each method and dataset, the scaling factor for FSD penalty, λ_{FSD}, is reported in Table 8. Similarly, batch size is reported in Table 9.

Evaluation Metrics. In addition to average accuracy over tasks, we measure the backward transfer metric. For T tasks, let $R_{i,j}$ be the classification accuracy on task t_j after training on task t_i. Then, backward transfer is given by the following formula.

$$\frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$$

E.3. Task-wise classification and task sequence length

We show in Figure 5 the task-wise accuracies on the Split CIFAR100 benchmark after training on all tasks is complete, for our methods (LAFTR, LAFTR (coreset) and BGLN-S-CW), NTK (coreset) and a nonparametric state-of-the-art method, FROMP. This draws a more fine-grained comparison and depicts the the benefits of LAFTR for each task.

To evaluate learning long sequences of tasks, we test our method's performance on the extended, 11-task version of Split CIFAR100 (longer than the typical 6-task benchmark). The results for both versions of the task are shown in Table 10 for comparison.

We find that performance of our method is maintained even for longer sequences of tasks. In this case, training on more tasks continually with LAFTR actually yields higher average accuracy. As we may expect, we observe a small decrease in the backward transfer performance, which still outperforms existing nonparametric SOTA methods.

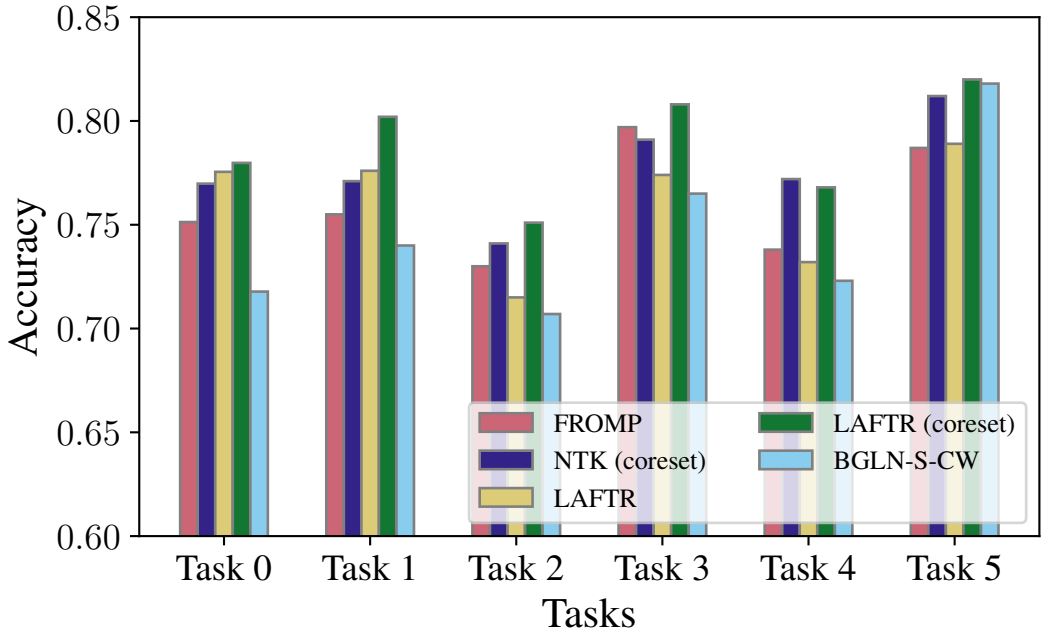

Figure 5. Comparison of task-wise accuracies on the Split CIFAR100 benchmark after training on all tasks is complete, for our methods, NTK and a nonparametric state-of-the-art method, FROMP.

Table 10. LAFTR (coreset) performance for different task sequence lengths.

Number of tasks	Average Accuracy	Backward Transfer
6	78.33 ± 0.01	-0.73 ± 0.10
11	78.75 ± 0.01	-1.59 ± 0.47

E.4. Ablation of information stored

We present further results on the Split CIFAR100 task, with varying amounts of information stored and used in our method. Specifcally, Table 11 summarizes the effect of coreset size on average accuracy and backward transfer for Split CIFAR100. It is also possible to generalize our class-conditional variant, which operates on $k = 10$ clusters, to different number of clusters. We can create clusters by grouping classes and storing the required statistics for each cluster separately. As shown in Table 12, $k = 1$ is sufficient to compete with existing nonparametric methods in this setting. As expected, as k is increased and more dataset-level statistics are used, continual learning performance improves.

Table 11. Effect of coreset size on Split CIFAR100 performance.

Coreset Size	Average Accuracy	Backward Transfer
20	73.26 ± 0.01	-2.51 ± 0.43
50	75.63 ± 0.01	-2.08 ± 0.51
100	76.70 ± 0.01	-1.65 ± 0.09
200	78.33 ± 0.01	-0.73 ± 0.10

Table 12. Effect of number of classwise clusters on Split CIFAR100 performance.

k	Average Accuracy	Backward Transfer
1	75.61 ± 0.01	-1.93 ± 0.59
5	75.74 ± 0.01	-1.58 ± 0.50
10	76.22 ± 0.01	-1.45 ± 0.63

F. Influence Function Estimation

F.1. Experimental Details

We used Concrete, Energy, Housing, Kinetics, and Wine datasets from the UCI collection (Dua & Graff, 2017). For all datasets, we normalized the training dataset to have a mean of 0 and a standard deviation of 1. We used a 2-hidden layer MLP with 128 hidden units and the base network was trained for 200 epochs with SGD and a batch size of 128. We performed hyperparameter searches over the learning rate in the range $\{0.3, 0.1, 0.03, 0.01, 0.003, 0.001\}$ and selected the learning rate based on the validation loss.

For each random data point selected, we optimized the PBRF objective for additional 20 epochs from the base network. The FSD term was computed stochastically with a batch size of 128. Similarly, both EWC and BGLN were trained with the same configuration but with the corresponding approximation to the FSD term.