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
N	Number of data points
L	Number of layers in a network
T	Number of tasks
$\boldsymbol{\theta}$	Parameters of a neural network
$\boldsymbol{\theta}_t$	Parameters obtained after training on task t
$\boldsymbol{z} = f(\mathbf{x}; \boldsymbol{\theta})$	Prediction of the network f on $\mathbf{x}$ parameterized by $\boldsymbol{\theta}$
ρ	Output space distance, for instance Euclidean distance
$D(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$	Function space divergence between networks parameterized by $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$
$\boldsymbol{J} = \nabla_{\boldsymbol{\theta}} f$	Network Jacobian
$\boldsymbol{G}_{\boldsymbol{\theta}}$	Weight space metric matrix
$\boldsymbol{F}_{\boldsymbol{\theta}}$	Fisher information matrix
$\mathbf{m}$	Bernoulli random variable
μ	Bernoulli mean parameter

B RECURSION EQUATIONS FOR BGLN-D

We derive the determinstic version of our algorithm by taking expectations and covariances for the quantities in equations 5 to 12. We use linearity of expectations and our assumptions of independence between the Bernoulli random variables. We also assume $\text{Cov}(a_0, \Delta a)$ 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 Figure 3a). 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 connected layers. At each ReLU, we use the Bernoulli mean parameters to sample activation signs and obtain the difference of activations, Δa. We treat any dropout layers as they are treated at evaluation time, i.e. as the identity function. Finally, the Euclidean distance between the final layer outputs leads to the stochastic estimate of the FSD. The complete procedure is shown in Algorithm 3.

Algorithm 2 BGLN-D

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

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 if `layer` is `Dropout` **then**
    pass
  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 $\frac{1}{2}||\Delta \boldsymbol{z}||^2$

D Continual Learning Experiments

D.1 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).

Table 4: CL tasks training epochs used in CL experiments.

Method	Split MNIST	Permuted MNIST	Split CIFAR100
NTK (coreset)	15	5	80
BGLN-S	15	15	45
BGLN-D	15	15	-
BGLN-S (CW)	15	15	85
BGLN-D (CW)	15	15	-

Table 5: FSD scale used in CL experiments.

Method	Split MNIST	Permuted MNIST	Split CIFAR100
NTK (coreset)	1	1	0.005
BGLN-S	5	1	0.0005
BGLN-D	0.1	0.005	-
BGLN-S (CW)	2	1	0.0000001
BGLN-D (CW)	0.1	0.005	-

Table 6: 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	-

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. Both Split MNIST and Split CIFAR100 models have a multiheaded final layer.

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 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 4. 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.

For each method and dataset, the scaling factor for FSD penalty, λ_{FSD}, is reported in Table 5. Similarly, batch size is reported in Table 6.

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,

Table 7: Ablation of backward transfer on continual learning benchmarks (higher is better).

Method	Split MNIST	Permuted MNIST	Split CIFAR100
BGLN-D-Var	-0.18 ± 0.09	-3.90 ± 0.49	-
BGLN-S-Var	-0.13 ± 0.11	-0.41 ± 0.08	-9.25 ± 0.06
BGLN-S (coreset)	-0.13 ± 0.11	-0.41 ± 0.08	-9.26 ± 0.06

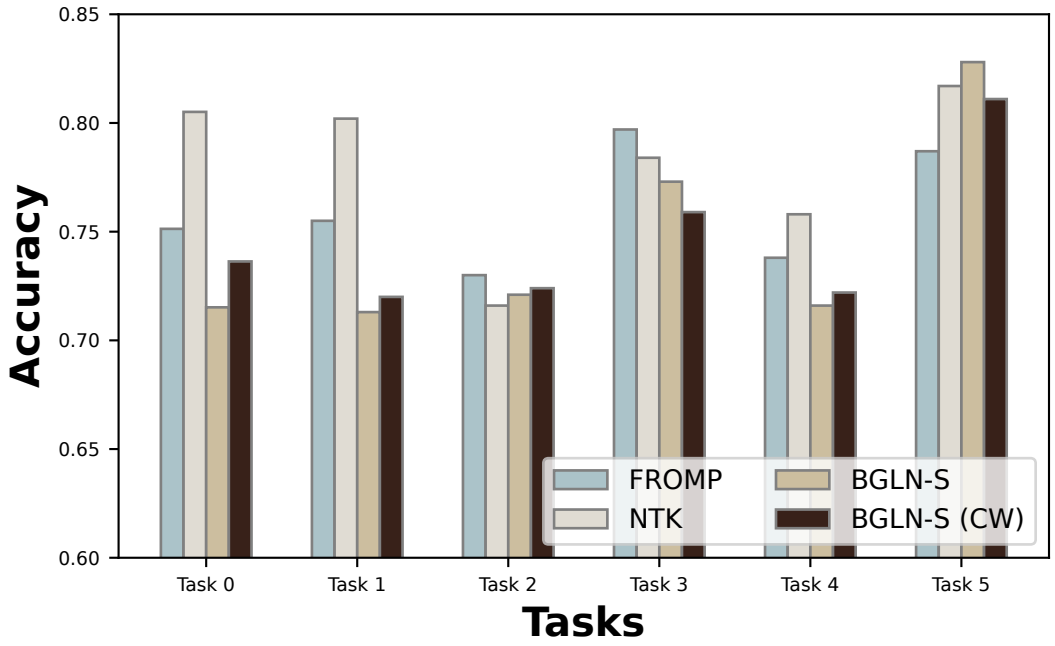

Figure 4: Comparison of task-wise accuracies on the Split CIFAR100 benchmark after training on all tasks is complete, for our methods, NTK and a nonparametric state-of-the-art method, FROMP.

backward transfer is given by the following formula.

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

D.2 Task-wise classification results

We show in Figure 4 the task-wise accuracies on the Split CIFAR100 benchmark after training on all tasks is complete, drawing a comparison between our methods (BGLN-S and BGLN-S (CW)), NTK (with coreset) and a nonparametric state-of-the-art method, FROMP.

D.3 Ablation Study - Backward Transfer

We include further results on the performance of our ablations on the backward transfer metric in Table 7.

E Influence Function Experiment

E.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

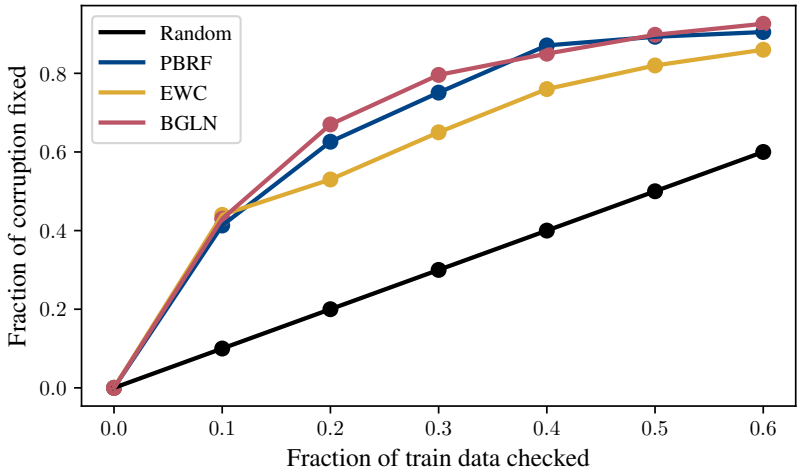

Figure 5: Effectiveness of BGLN in detecting mislabeled examples. BGLN can approximate the FSD term in the PBRF objective accurately and be used in applications involving influence functions without explicitly storing or iterating over the dataset.

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.

E.2 Mislabeled Example Identification

A common application of the influence function is the detection of influential or mislabeled examples. Intuitively, if some fraction of the training data labels is corrupted, they would behave as outliers and have a greater influence on the training loss. One approach to efficiently detect and fix these examples is to prioritize and examine training inputs with higher self-influence scores.

We use 10% of the MNIST dataset and corrupt 10% of the data by assigning random labels to it. We train a two-layer MLP with 1024 hidden units and ReLU activation using SGD with a batch size of 128. Then, we use EWC and BFLN to approximate the FSD term in the PBRF objective in equation 14 for each data point. We also compare these methods against randomly sampling datapoints to check for corruptions. The results are summarized in Figure 5. Both PBRF and BGLN show significantly improved performance compared to the random baseline.