NeuralEF: Deconstructing Kernels by Deep Neural Networks

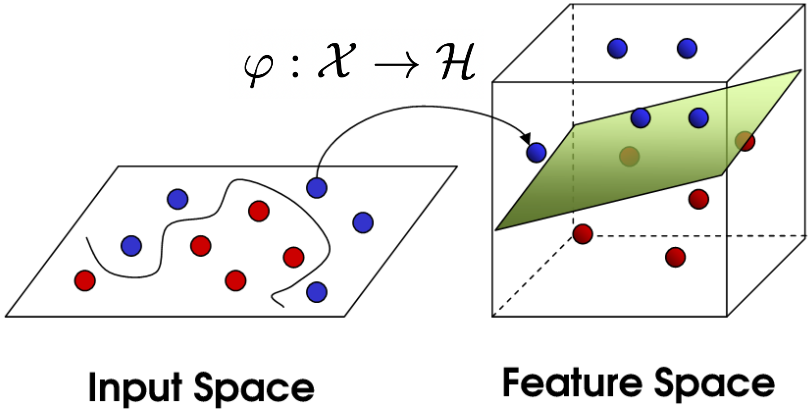
Zhijie Deng

Tsinghua University

Contact: dengzhijiethu@gmail.com

Joint work with: Jiaxin Shi and Jun Zhu

Kernel methods



- $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{x}') \rangle_{\mathcal{H}}$

- Pros: non-parametric flexibility & analytical inference

- Cons: limited scalability – at least O(N^2) complexity, typically O(N^3); inefficiency issue in the test phase

Kernel approximation

- Approximate the kernel with the inner product of some explicit vector representations of the data:

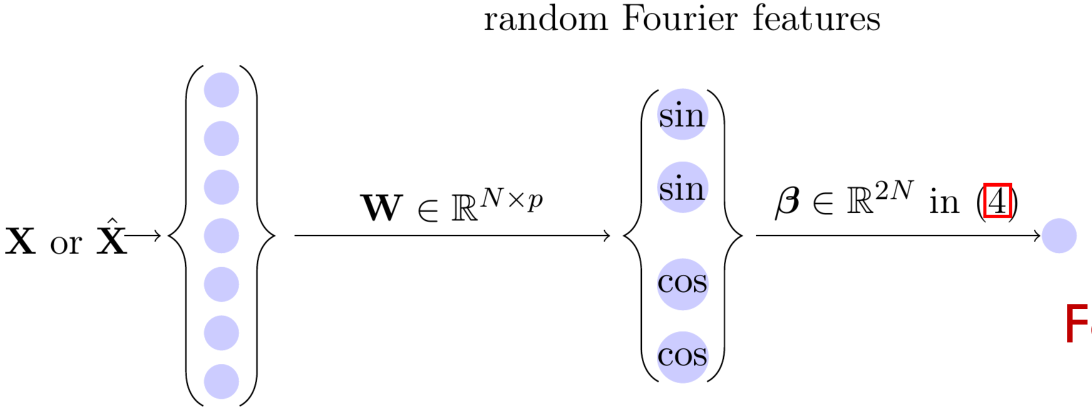
$$\kappa(\boldsymbol{x}, \boldsymbol{x}') \approx \nu(\boldsymbol{x})^\top \nu(\boldsymbol{x}') \qquad \nu : \mathcal{X} \to \mathbb{R}^k$$

- A small k is desired for scalability while the approximation is low-rank

- Popular approaches:
    1. Random Fourier features [Rahimi & Recht, 2007; 2008]
    2. Nystrom method [Nystrom, 1930; Williams & Seeger, 2001]
    3. …

Kernel approximation

Random features (RFs)



$\mathbf{X} \in \mathbb{R}^{p \times n}$

$\hat{\mathbf{X}} \in \mathbb{R}^{p \times \hat{n}}$

$$\boldsymbol{\Sigma}_{\mathbf{X}}^{\top} = [\cos(\mathbf{W}\mathbf{X})^{\top},\ \sin(\mathbf{W}\mathbf{X})^{\top}]$$

$$\boldsymbol{\Sigma}_{\hat{\mathbf{X}}}^{\top} = [\cos(\mathbf{W}\hat{\mathbf{X}})^{\top},\ \sin(\mathbf{W}\hat{\mathbf{X}})^{\top}]$$

For shift-invariant kernels

[Rahimi et al., 2007]

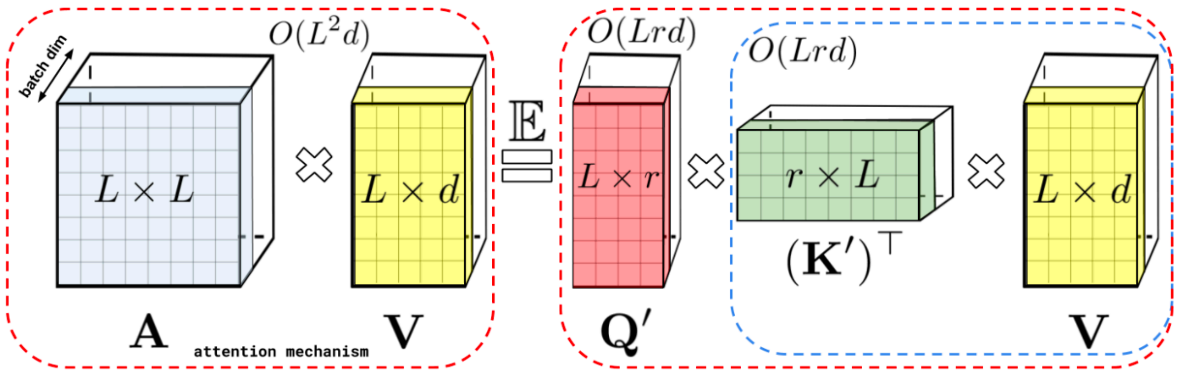


Figure 1: Approximation of the regular attention mechanism $\mathbf{AV}$ (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

Performer (RFs for exp(x, x'))

[Choromanski et al., 2021]

Kernel approximation

- Mercer's theorem

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j \geq 1} \mu_j \psi_j(\boldsymbol{x}) \psi_j(\boldsymbol{x}')$$

where ψ_j denote the eigenfunctions of the kernel κ w.r.t. the probability measure q, and $\mu_j \geq 0$ refer to the corresponding eigenvalues

- By the definition of eigenfunction, we have

$$\int \kappa(\boldsymbol{x}, \boldsymbol{x}') \psi_j(\boldsymbol{x}') q(\boldsymbol{x}') d\boldsymbol{x}' = \mu_j \psi_j(\boldsymbol{x}), \ \forall j \geq 1$$

and

$$\int \psi_i(\boldsymbol{x}) \psi_j(\boldsymbol{x}) q(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{1}[i = j], \ \forall i, j \geq 1$$

Kernel approximation

Nystrom method

- Given $\mathbf{X}_{\text{tr}} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ from q, perform MC integration:

$$\frac{1}{N}\sum_{n'=1}^{N}\kappa(\boldsymbol{x}, \boldsymbol{x}_{n'})\psi_j(\boldsymbol{x}_{n'}) = \mu_j\psi_j(\boldsymbol{x}), \forall j \geq 1$$

- Eigendecompose $\kappa(\mathbf{X}_{\text{tr}}, \mathbf{X}_{\text{tr}})$ and get $\{(\hat{\mu}_j, [\hat{\psi}_j(\boldsymbol{x}_1), ..., \hat{\psi}_j(\boldsymbol{x}_N)]^\top)\}_{j=1}^{k}$

- Kernelized solutions:

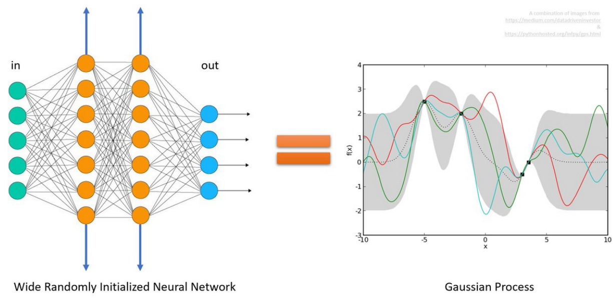
$$\hat{\psi}_j(\boldsymbol{x}) = \frac{1}{N\hat{\mu}_j}\sum_{n'=1}^{N}\kappa(\boldsymbol{x}, \boldsymbol{x}_{n'})\hat{\psi}_j(\boldsymbol{x}_{n'}),\ j = 1, ..., k.$$

- Less scalable; the testing entails extensive computes

The modern kernels
Kernels meet NNs

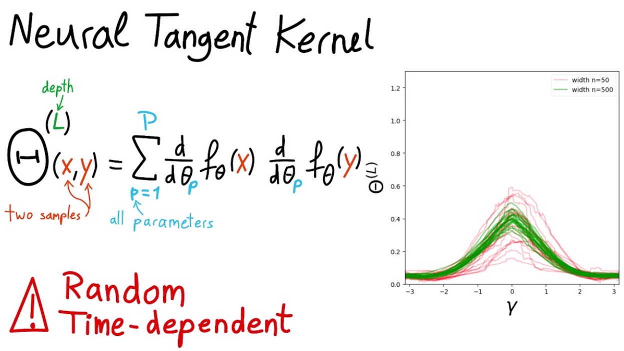
- Classic local kernels suffer from curse of dimensionality [Bengio et al., 2005]

- *Neural network Gaussian process (NNGP) kernels* [Neal, 1995; Lee et al., 2017; Khan et al., 2019]



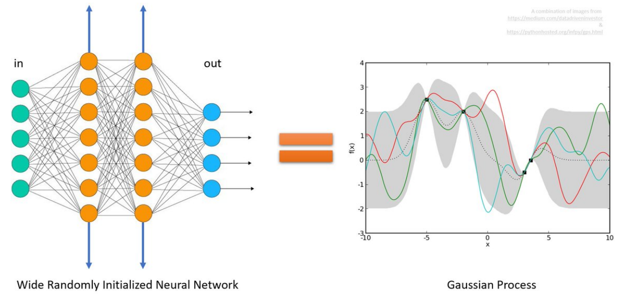
$$\kappa_{\text{NN-GP}}(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta})} g(\boldsymbol{x}, \boldsymbol{\theta}) g(\boldsymbol{x}', \boldsymbol{\theta})^\top$$

- *Neural tangent kernels (NTKs)* [Jacot et al., 2018]

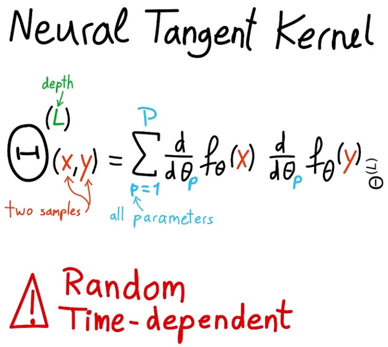


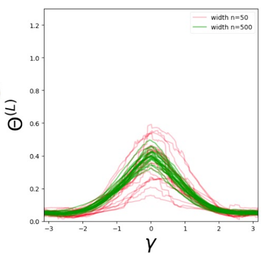
$$\kappa_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') = \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}, \boldsymbol{\theta}) \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}', \boldsymbol{\theta})^\top$$

The modern kernels
Kernels meet NNs



$$\kappa_{\text{NN-GP}}(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta})} g(\boldsymbol{x}, \boldsymbol{\theta}) g(\boldsymbol{x}', \boldsymbol{\theta})^\top$$





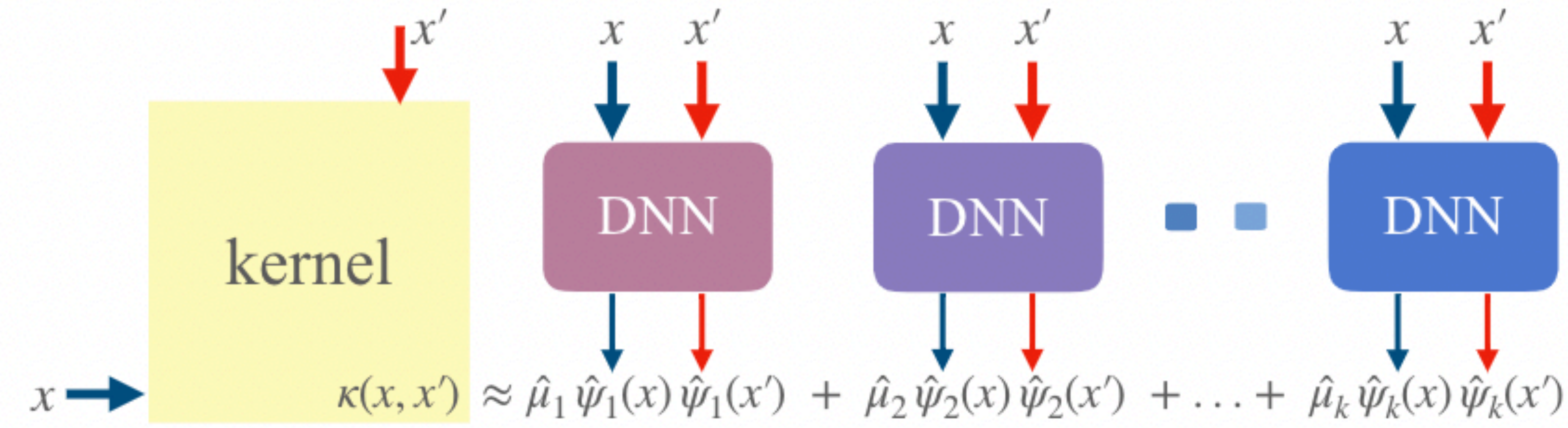
$$\kappa_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') = \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}, \boldsymbol{\theta}) \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}', \boldsymbol{\theta})^\top$$

- Nevertheless, writing down their detailed mathematical formulae is non-trivial [Arora et al., 2019] and evaluating them with recursion is both time and memory consuming.

- They have poor compatibility with standard kernel approximation methods.

NeuralEF: approximate the eigenfunctions of kernels by NNs

Our solution



A closely related work

Spectral Inference Networks (SpIN) [Pfau et al., 2018]

- Recover the top eigenfunctions with NNs due to their universal approximation capability and parametric nature
- Introduce a vector-valued NN function $\Psi(\cdot, \boldsymbol{w}) : \mathcal{X} \to \mathbb{R}^k$ and solve:

$$\max_{\boldsymbol{w}} \operatorname{Tr}\Big(\iint \Psi(\boldsymbol{x}, \boldsymbol{w})\Psi(\boldsymbol{x}', \boldsymbol{w})^\top \kappa(\boldsymbol{x}, \boldsymbol{x}') q(\boldsymbol{x}) q(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{x}' \Big)$$
$$\text{s.t.: } \int \Psi(\boldsymbol{x}, \boldsymbol{w})\Psi(\boldsymbol{x}, \boldsymbol{w})^\top q(\boldsymbol{x}) d\boldsymbol{x} = \mathbf{I}_k, \qquad (9)$$

- However, this objective makes Ψ recover the subspace spanned by the top-k eigenfunctions rather than the top-k eigenfunctions themselves [Pfau et al., 2018].

A closely related work

Spectral Inference Networks (SpIN) [Pfau et al., 2018]

- Recover the top eigenfunctions with NNs due to their universal approximation capability and parametric nature
- Introduce a vector-valued NN function $\Psi(\cdot, \boldsymbol{w}) : \mathcal{X} \to \mathbb{R}^k$ and solve:

$$\max_{\boldsymbol{w}} \operatorname{Tr}\Big(\iint \Psi(\boldsymbol{x}, \boldsymbol{w})\Psi(\boldsymbol{x}', \boldsymbol{w})^\top \kappa(\boldsymbol{x}, \boldsymbol{x}') q(\boldsymbol{x}) q(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{x}' \Big)$$
$$\text{s.t.: } \int \Psi(\boldsymbol{x}, \boldsymbol{w})\Psi(\boldsymbol{x}, \boldsymbol{w})^\top q(\boldsymbol{x}) d\boldsymbol{x} = \mathbf{I}_k, \qquad (9)$$

- To address this issue, SpIN relies on a gradient masking trick which involves a Cholesky decomposition per training iteration.

- SpIN also involves tracking the exponential moving average (EMA) of the Jacobian matrix to debias the stochastic optimization.

Eigendecomposition as asymmetric maximization problems

Our new results

Generalized Rayleigh quotient

Normalization constraint

Orthogonality constraint

Theorem 1 (Proof in Appendix A.1). *The eigenpairs of the kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ can be recovered by simultaneously solving the following series of constrained maximization problems:*

$$\max_{\hat{\psi}_j} \boxed{R_{jj}} \text{ s.t.: } \boxed{C_j = 1,} \boxed{R_{1j} = 0, ..., R_{(j-1)j} = 0,} \forall j \geq 1, \quad (7)$$

where $\hat{\psi}_j \in L^2(\mathcal{X}, q)$ represent the introduced approximate eigenfunctions, and

$$R_{ij} := \iint \hat{\psi}_i(\boldsymbol{x})\kappa(\boldsymbol{x}, \boldsymbol{x}')\hat{\psi}_j(\boldsymbol{x}')q(\boldsymbol{x}')q(\boldsymbol{x})d\boldsymbol{x}'d\boldsymbol{x}, \quad (8)$$

$$C_j := \int \hat{\psi}_j(\boldsymbol{x})\hat{\psi}_j(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}. \quad (9)$$

In particular, $(R_{jj}, \hat{\psi}_j)$ will converge to the eigenpair associated with j-th largest eigenvalue of κ.

Eigendecomposition as asymmetric maximization problems

Proof scratch--the first problem

- The ground-truth eigenfunctions form a set of orthonormal bases of the L2(X , q) space
- Represent the approximations in such a new axis system $\hat{\psi}_1 = \sum_{i \geq 1} w_i \psi_i$
- The the maximization objective reduces to

$$R_{11} = \sum_{j \geq 1} \mu_j \langle \hat{\psi}_1, \psi_j \rangle^2 = \sum_{j \geq 1} \mu_j \langle \sum_{i \geq 1} w_i \psi_i, \psi_j \rangle^2 = \sum_{j \geq 1} \mu_j w_j^2.$$

- And the constraint reduces to

$$\langle \hat{\psi}_1, \hat{\psi}_1 \rangle = \langle \sum_{i \geq 1} w_i \psi_i, \sum_{j \geq 1} w_j \psi_j \rangle = \sum_{i,j \geq 1} w_i w_j \langle \psi_i, \psi_j \rangle = \sum_{j \geq 1} w_j^2 = 1$$

- It is straight-forward to see the maxima

Eigendecomposition as asymmetric maximization problems

Proof scratch--the second problem

- Given $\hat{\psi}_1 = \psi_1$

$$
\begin{aligned}
& R_{12} = 0 \\
\Rightarrow \quad & \iint \hat{\psi}_1(\boldsymbol{x})\kappa(\boldsymbol{x},\boldsymbol{x}')\hat{\psi}_2(\boldsymbol{x}')q(\boldsymbol{x}')q(\boldsymbol{x})d\boldsymbol{x}'d\boldsymbol{x} = 0 \\
\Rightarrow \quad & \int \hat{\psi}_2(\boldsymbol{x}')q(\boldsymbol{x}') \int \hat{\psi}_1(\boldsymbol{x})\kappa(\boldsymbol{x},\boldsymbol{x}')q(\boldsymbol{x})d\boldsymbol{x}d\boldsymbol{x}' = 0 \\
\Rightarrow \quad & \int \hat{\psi}_2(\boldsymbol{x}')q(\boldsymbol{x}') \int \psi_1(\boldsymbol{x})\kappa(\boldsymbol{x},\boldsymbol{x}')q(\boldsymbol{x})d\boldsymbol{x}d\boldsymbol{x}' = 0 \\
\Rightarrow \quad & \int \hat{\psi}_2(\boldsymbol{x}')q(\boldsymbol{x}')\mu_1\psi_1(\boldsymbol{x}')d\boldsymbol{x}' = 0 \\
\Rightarrow \quad & \langle \psi_1, \hat{\psi}_2 \rangle = 0.
\end{aligned}
$$

- $\hat{\psi}_2$ is constrained in the orthogonal complement of the subspace spanned by ψ_1
- Then we can apply an analysis similar to that for the first problem
- Applying this procedure incrementally to the additional problems then finishes the proof

Eigendecomposition as asymmetric maximization problems

- Slack the constraints on orthogonality as penalties and solve the first k optimization problems

$$\max_{\hat{\psi}_j} R_{jj} - \sum_{i=1}^{j-1} \frac{R_{ij}^2}{R_{ii}} \quad \text{s.t.: } C_j = 1, \text{for } j = 1, ..., k,$$

- Our objective forms a function-space generalization of that in EigenGame [Gemp et al., 2020]

DNNs as eigenfunctions

Use an ensemble of k DNNs to approximate the top-k eigenfunctions

- Mini-batch training -- by MC estimation:

$$\tilde{R}_{ij} = \sum_{b=1}^{B}\sum_{b'=1}^{B}\frac{1}{B^2}\hat{\psi}_i(\boldsymbol{x}_b)\kappa(\boldsymbol{x}_b, \boldsymbol{x}_{b'})\hat{\psi}_j(\boldsymbol{x}_{b'})$$
$$= \frac{1}{B^2}\hat{\boldsymbol{\psi}}_i^{\mathbf{X}\top}\boldsymbol{\kappa}^{\mathbf{X},\mathbf{X}}\hat{\boldsymbol{\psi}}_j^{\mathbf{X}},$$

- L2 Batch normalization (L2BN) to absorb the normalization constraints:

$$h_b^{\text{out}} = \frac{h_b^{\text{in}}}{\sigma}, \text{ with } \sigma = \sqrt{\frac{1}{B}\sum_{b=1}^{B} {h_b^{\text{in}}}^2},\ b = 1, ..., B.$$

- The gradients: $\nabla_{\boldsymbol{w}_j}\ell = -\frac{2}{B^2}\boldsymbol{\kappa}^{\mathbf{X},\mathbf{X}}\left(\hat{\boldsymbol{\psi}}_j^{\mathbf{X}} - \sum_{i=1}^{j-1}\frac{\hat{\boldsymbol{\psi}}_i^{\mathbf{X}\top}\boldsymbol{\kappa}^{\mathbf{X},\mathbf{X}}\hat{\boldsymbol{\psi}}_j^{\mathbf{X}}}{\hat{\boldsymbol{\psi}}_i^{\mathbf{X}\top}\boldsymbol{\kappa}^{\mathbf{X},\mathbf{X}}\hat{\boldsymbol{\psi}}_i^{\mathbf{X}}}\hat{\boldsymbol{\psi}}_i^{\mathbf{X}}\right)\cdot\partial_{\boldsymbol{w}_j}\hat{\boldsymbol{\psi}}_j^{\mathbf{X}}.$

- Extension to *matrix-valued* kernels (e.g., NTKs):
  strategy 1: use multi-output DNNs
  strategy 2: make a factorization assumption

NeuralEF

The algorithm

Algorithm 1 Find the top-k eigenpairs of a kernel by NeuralEF

Input: Training data $\mathbf{X}_{\text{tr}}$, kernel κ, batch size B.
Initialize NNs $\hat{\psi}_j(\cdot) = \hat{\psi}(\cdot, \boldsymbol{w}_j)$ and scalars $\hat{\mu}_j, j \in [k]$;
Compute the kernel matrix $\boldsymbol{\kappa}^{\mathbf{X}_{\text{tr}}, \mathbf{X}_{\text{tr}}} = \kappa(\mathbf{X}_{\text{tr}}, \mathbf{X}_{\text{tr}})$;
for *iteration* **do**
    Draw a mini-batch $\mathbf{X} \subseteq \mathbf{X}_{\text{tr}}$; retrieve $\boldsymbol{\kappa}^{\mathbf{X}, \mathbf{X}}$ from $\boldsymbol{\kappa}^{\mathbf{X}_{\text{tr}}, \mathbf{X}_{\text{tr}}}$;
    Do forward propagation $\hat{\boldsymbol{\psi}}_{\boldsymbol{j}}^{\mathbf{X}} = \hat{\psi}(\mathbf{X}, \boldsymbol{w}_j), j \in [k]$;
    $\hat{\mu}_j \leftarrow \text{EMA}(\hat{\mu}_j, \frac{1}{B^2} \hat{\boldsymbol{\psi}}_{\boldsymbol{j}}^{\mathbf{X}\top} \boldsymbol{\kappa}^{\mathbf{X}, \mathbf{X}} \hat{\boldsymbol{\psi}}_{\boldsymbol{j}}^{\mathbf{X}}), j \in [k]$;
    Compute $\nabla_{\boldsymbol{w}_j} \ell, j \in [k]$ by Equation (18) and do SGD;
end for

Enable the learning of NN-GP kernels and NTKs

Based on thousands of random features

$$\underbrace{\mathbf{H}(\mathbf{w}_0)}_{n \times n} = \underbrace{\nabla_w \mathbf{y}(\mathbf{w}_0)^T}_{n \times p} \underbrace{\nabla_w \mathbf{y}(\mathbf{w}_0)}_{p \times n}$$

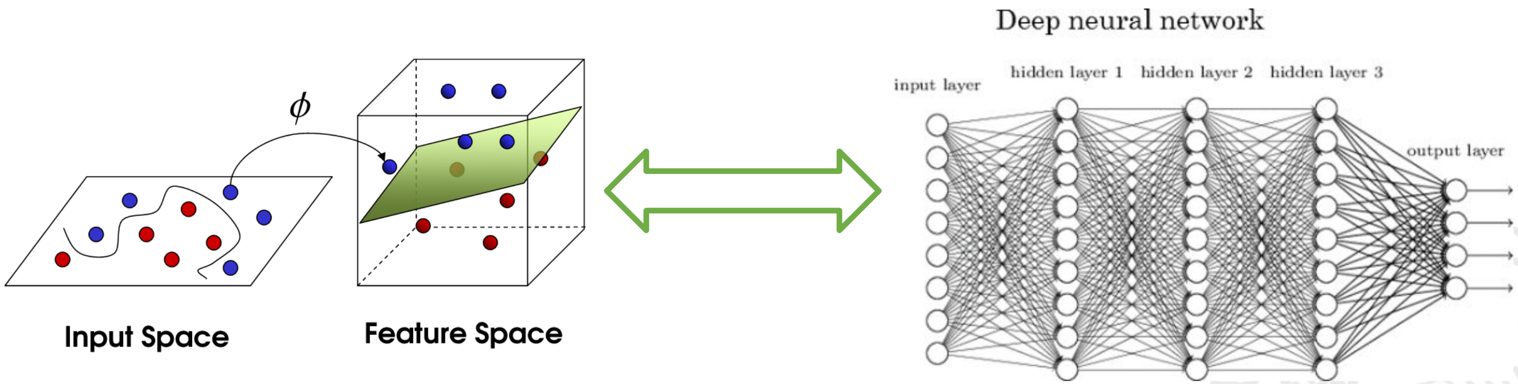
NTK

$$= \begin{bmatrix} — \phi(\bar{x}_1)^T — \\ \vdots \\ — \phi(\bar{x}_n)^T — \end{bmatrix} \begin{bmatrix} | & & | \\ \phi(\bar{x}_1) & \cdots & \phi(\bar{x}_n) \\ | & & | \end{bmatrix}$$

- Computing the training kernel matrices by MC estimation given a distribution $p(v)$ satisfying $\mathbb{E}_{p(\boldsymbol{v})}[\boldsymbol{v}\boldsymbol{v}^\top] = \mathbf{I}_{\dim(\boldsymbol{\theta})}$, then

$$\begin{aligned} \boldsymbol{\kappa}_{\text{NTK}}^{\mathbf{X}_{\text{tr}},\mathbf{X}_{\text{tr}}} &= \mathbb{E}_{\boldsymbol{v}\sim p(\boldsymbol{v})}\left[\partial_{\boldsymbol{\theta}} g(\mathbf{X}_{\text{tr}},\boldsymbol{\theta})\boldsymbol{v}\right]\left[\partial_{\boldsymbol{\theta}} g(\mathbf{X}_{\text{tr}},\boldsymbol{\theta})\boldsymbol{v}\right]^\top \\ &\approx \frac{1}{S}\sum_{s=1}^{S}\left[\frac{g(\mathbf{X}_{\text{tr}},\boldsymbol{\theta}+\epsilon\boldsymbol{v}_s)-g(\mathbf{X}_{\text{tr}},\boldsymbol{\theta})}{\epsilon}\right]\left[\frac{g(\mathbf{X}_{\text{tr}},\boldsymbol{\theta}+\epsilon\boldsymbol{v}_s)-g(\mathbf{X}_{\text{tr}},\boldsymbol{\theta})}{\epsilon}\right]^\top \end{aligned}$$

The impact of NeuralEF



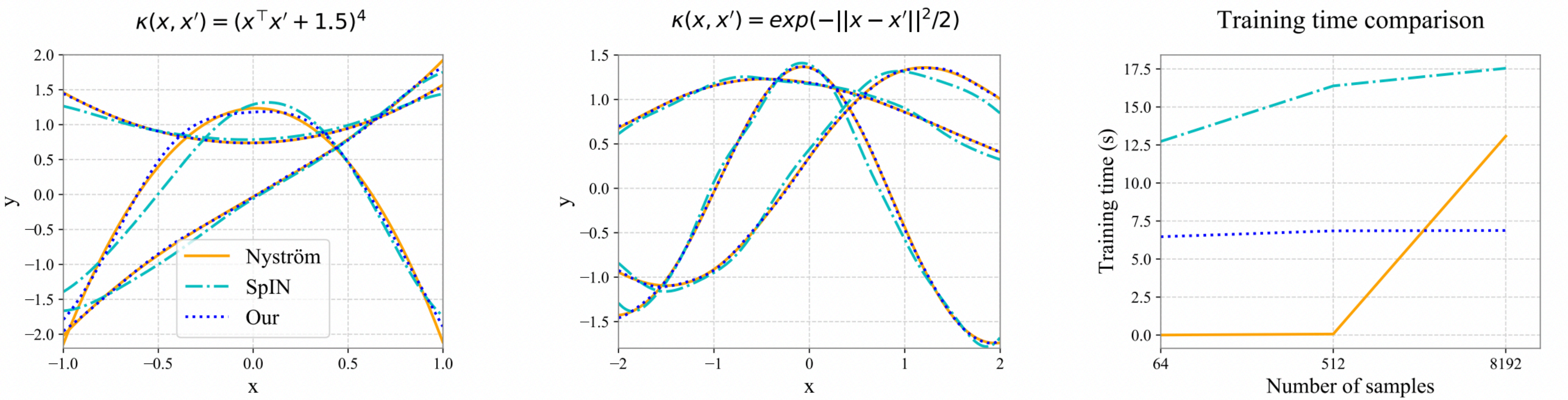
- NeuralEF approximate NTKs and NN-GP kernels with less NN forward passes than RFs

- It gives rise to an unsupervised representation learning paradigm, where the pairwise similarity captured by kernels is embedded into NNs

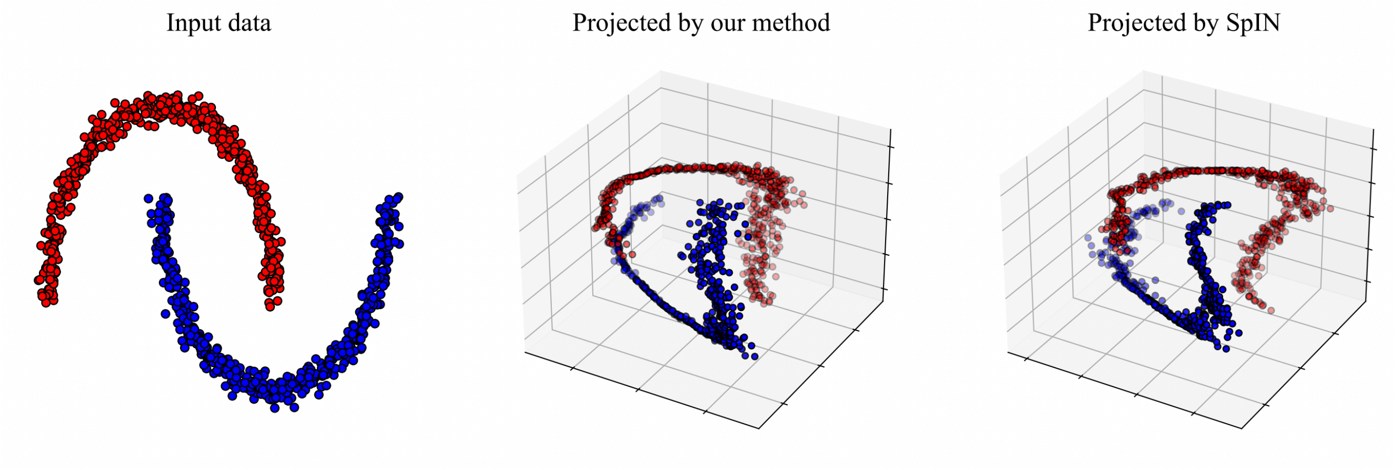
- It *relates two fields of research*

The applications

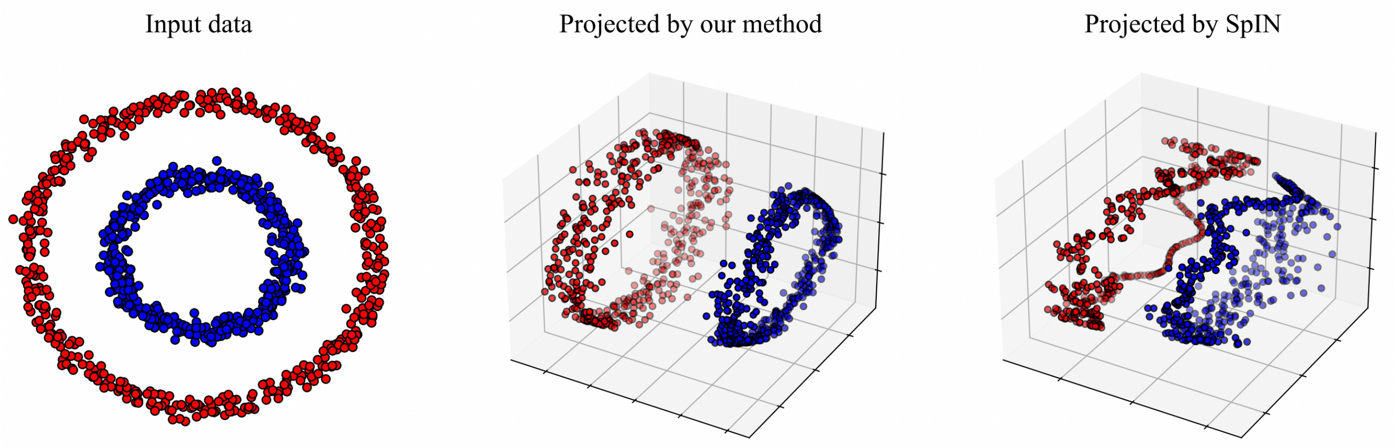
Find the eigenfunctions of classic kernels



The applications
Process MLP-GP kernels



(a) "Two-moon" data



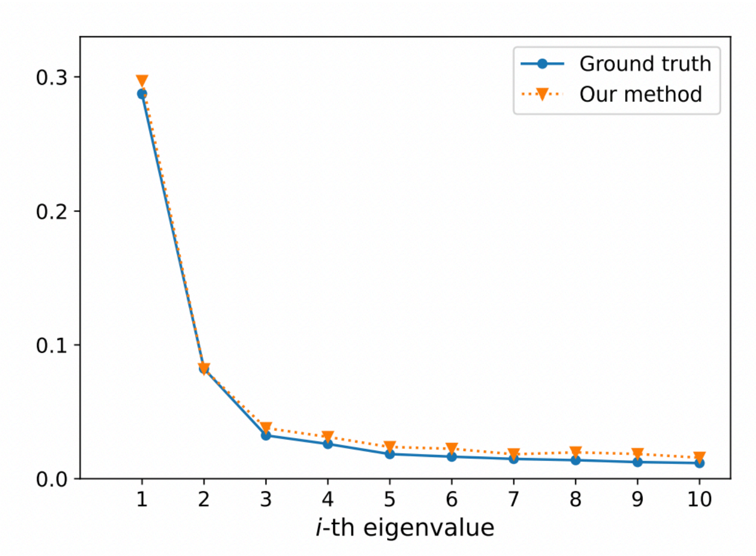
(b) "Circles" data

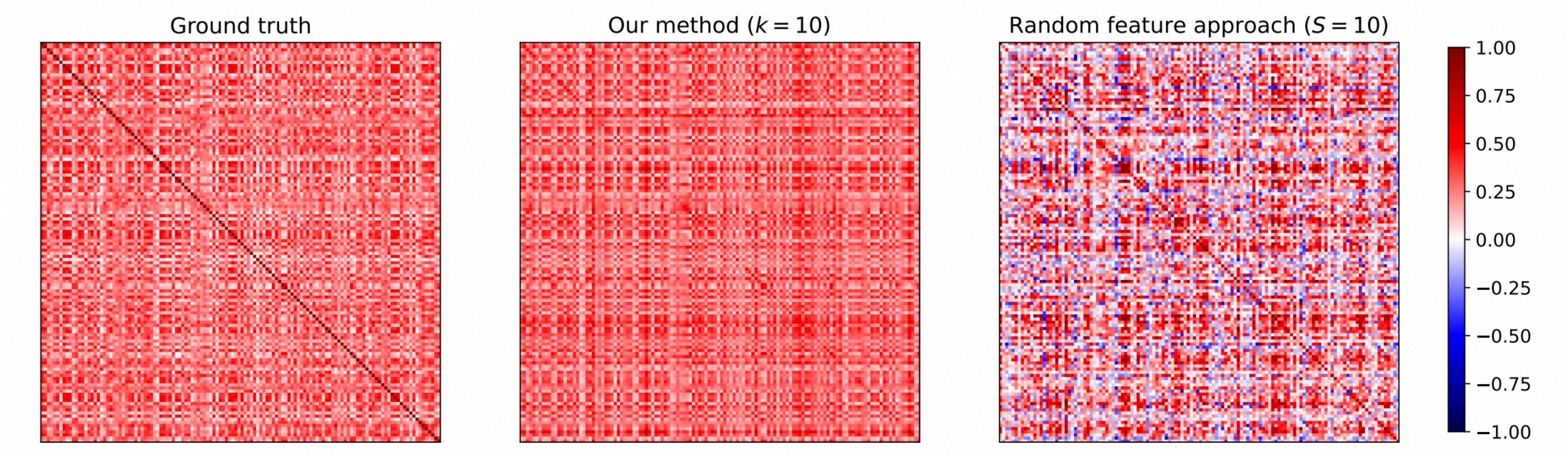
The applications
Process CNN-GP kernels

Method	LR test accuracy
Our method (CNN-GP kernel)	**84.98%**
Nyström (CNN-GP kernel)	N/A
Nyström (polynomial kernel)	78.00%
Nyström (RBF kernel)	77.55%

The applications

Find the eigenfunctions of NTK which itself is hard to compute





The applications

Improve linearised Laplace approximation with NeuralEF

Inverse of the Gauss-Newton of size # of params × # of params

$$\boldsymbol{\Sigma}^{-1} = \sum_i \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}_i, \boldsymbol{\theta}_{\text{MAP}})^\top \boldsymbol{\Lambda}_i \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}_i, \boldsymbol{\theta}_{\text{MAP}}) + 1/\sigma_0^2 \mathbf{I}_{\text{dim}(\boldsymbol{\theta})}$$

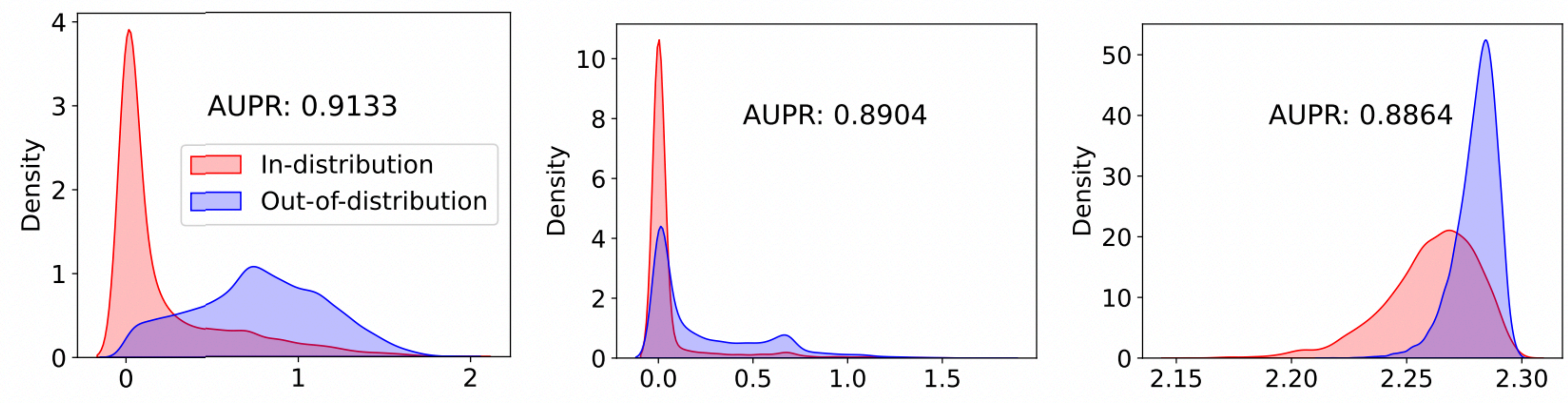
$$\mathcal{GP}(f|g(\boldsymbol{x}, \boldsymbol{\theta}_{\text{MAP}}), \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}, \boldsymbol{\theta}_{\text{MAP}}) \boxed{\boldsymbol{\Sigma}} \partial_{\boldsymbol{\theta}} g(\boldsymbol{x}', \boldsymbol{\theta}_{\text{MAP}})^\top)$$

By Woodbury matrix identity and Mercer's theorem

$$\mathcal{GP}\left(f|g(\boldsymbol{x}, \boldsymbol{\theta}_{\text{MAP}}), \Psi(\boldsymbol{x}) \boxed{\left[\sum_i \Psi(\boldsymbol{x}_i)^\top \boldsymbol{\Lambda}_i \Psi(\boldsymbol{x}_i) + \frac{1}{\sigma_0^2}\mathbf{I}_k\right]}^{-1} \boxed{\Psi}(\boldsymbol{x}')^\top\right),$$

Size: k × k

The concatenation of the k eigenfunctions for the NTK



(a) Ours (b) MAP (c) KFAC LLA

The applications

Bayesian deep learning by modeling SGD trajectory

$$\kappa_{\text{SGD}}(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{M}\sum_{i=1}^{M}\left(g(\boldsymbol{x}, \boldsymbol{\theta}_i) - \bar{g}(\boldsymbol{x})\right)\left(g(\boldsymbol{x}', \boldsymbol{\theta}_i) - \bar{g}(\boldsymbol{x}')\right)^\top$$

$$p(\boldsymbol{x}_{\text{new}}) = \int \mathcal{GP}(f|\bar{g}(\boldsymbol{x}), \kappa_{\text{SGD}}(\boldsymbol{x}, \boldsymbol{x}'))p(\boldsymbol{x}_{\text{new}}|f)df$$

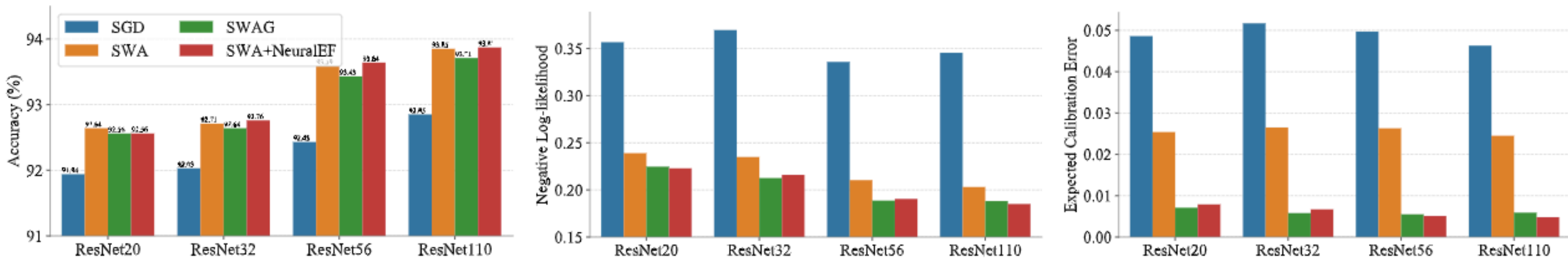
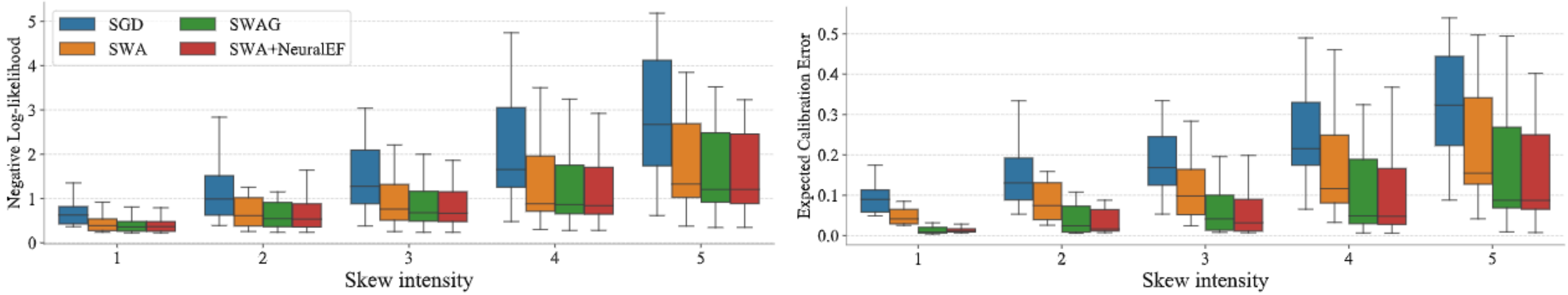


Figure 6: Test accuracy ↑, NLL ↓, and ECE ↓ comparisons among models on CIFAR-10.



Thanks!

Take-away messages

- The relationship between *kernels* and *DNN ensembles* is interesting
- Sometimes, we can replace kernels with DNN ensembles; if necessary, we can also replace DNN ensembles with kernels
- Scaling up kernels with NeuralEF may spark many appealing applications of kernels, e.g., for gradient estimation (SSGE)