




# Accelerated Linearized Laplace Approximation for Bayesian Deep Learning

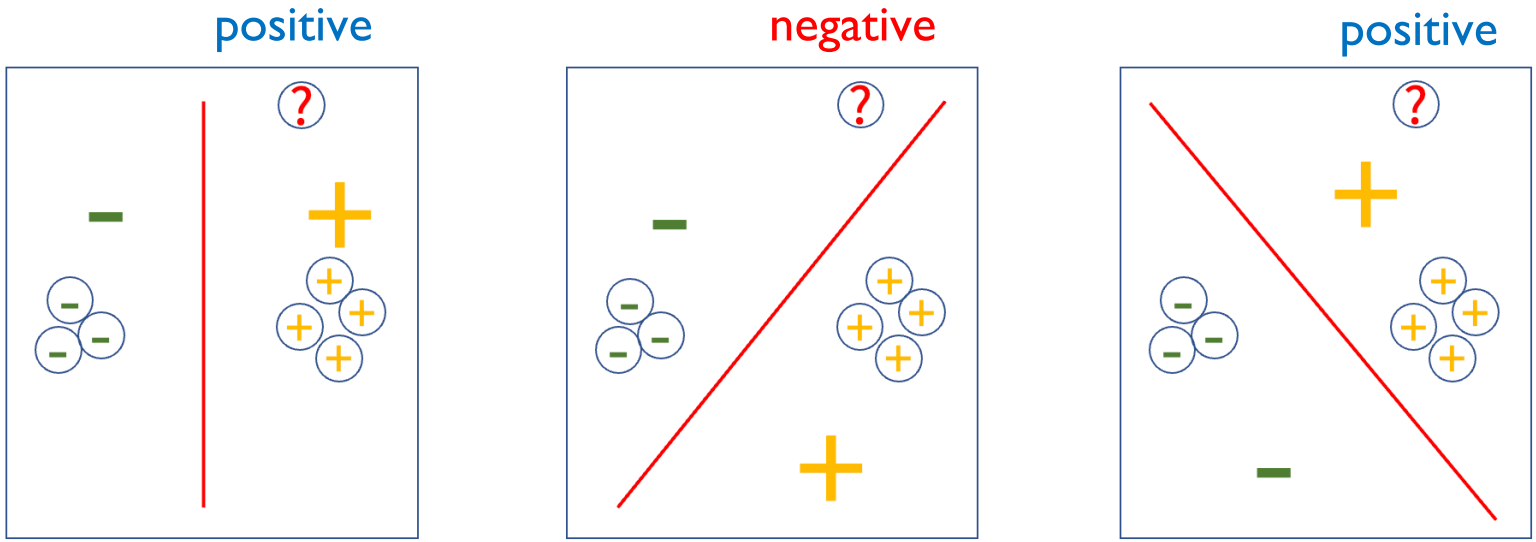
Zhijie Deng

Shanghai Jiao Tong University

Contact: zhijied@sjtu.edu.cn

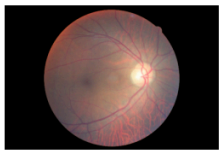
*Joint work with: Feng Zhou and Jun Zhu*

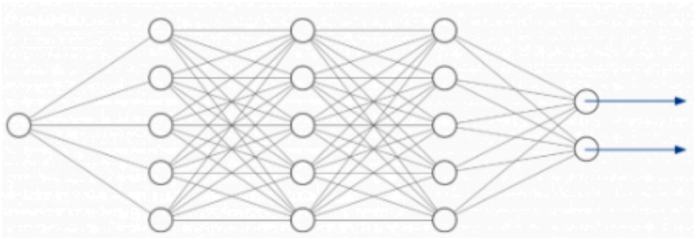
# The issues of deterministic neural networks




Which model to select?

# The issues of deterministic neural networks



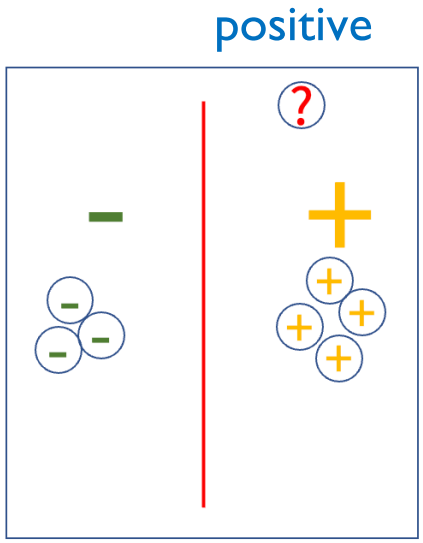


| confidence | prediction | correct |
|---|---|---|
| 99.5% | healthy | ✗ |
| 0.5% | | |

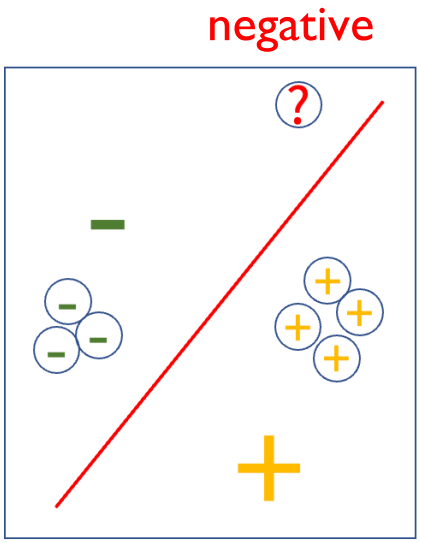
["Benchmarking Bayesian Deep Learning with Diabetic Retinopathy Diagnosis" by Angelos Filos et al.]

Can we trust the prediction with such confidence?

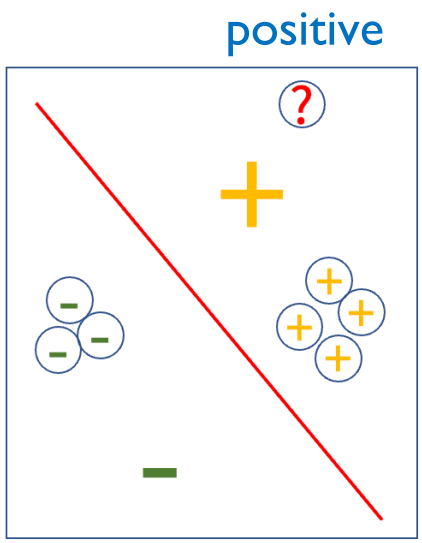
# Bayesian deep learning is helpful here




With probability 1/3




With probability 1/3



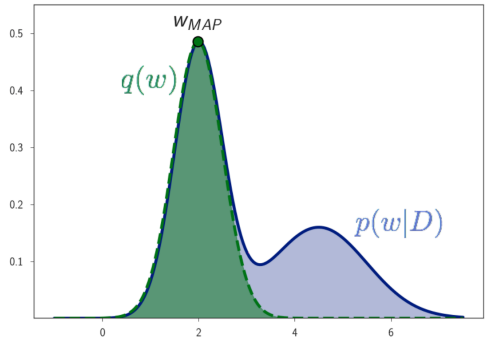

With probability 1/3

A posterior distribution over solutions: $p(\mathbf{w}|\mathcal{D}) \propto p(\mathbf{w}) \prod_{n=1}^{N} p(y_n|\mathbf{x}_n, \mathbf{w})$

Marginalization for prediction $p(y|x_*, \mathcal{D}) = \int p(y|x_*, w)p(w|\mathcal{D})dw$

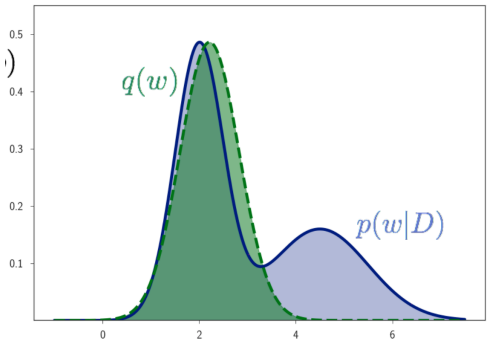
# Bayesian neural networks

## Approximate Bayesian inference



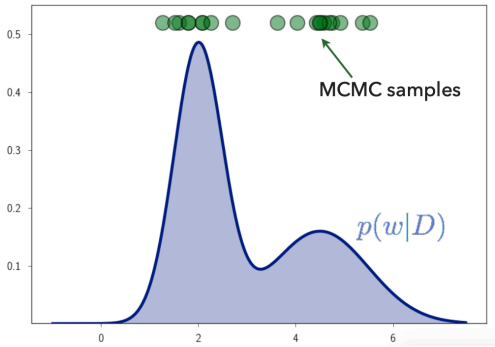

Laplace approx.

Simple yet less flexible




Variational inference

Efficient yet without the guarantee of asymptotic consistency

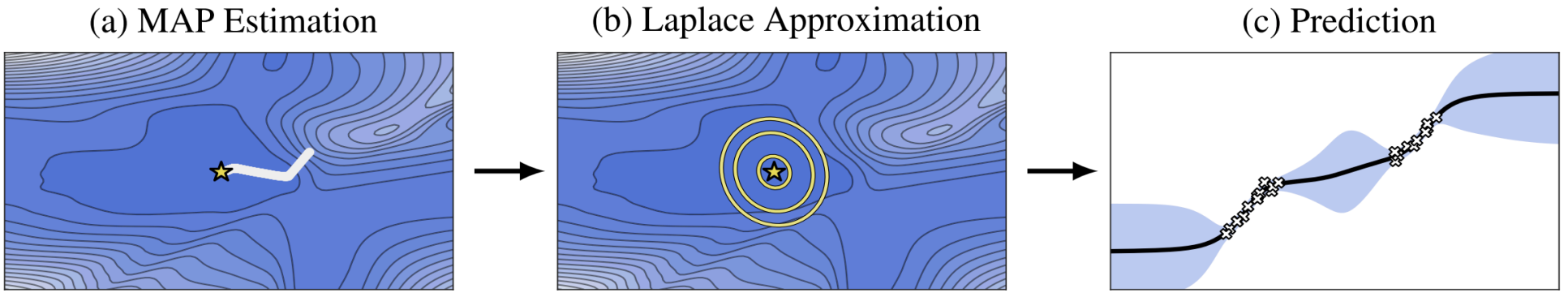



MCMC

Non-parametric and asymptotically exact yet typically with low convergence rate

# Laplace approximation

## Adapts a pre-trained DNN to BNN




["Laplace Redux – Effortless Bayesian Deep Learning" by Daxberger et al.]

1. Find the maximum a posteriori (MAP) solution (the mode of Bayesian posterior)

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

2. Construct a Gaussian approximation with the Hessian of log posterior

$$q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, \boldsymbol{\Sigma}) \qquad \boldsymbol{\Sigma}^{-1} = -\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} (\log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}))|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}$$

3. Make prediction by marginalization

# The generalized Gauss-Newton (GGN) approximation and linearized LA (LLA)

Let $g_\theta(x)$ denote the model and set *an isotropic Gaussian prior*. GGN approx. sets

$$\boldsymbol{\Sigma}^{-1} = \sum_i J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_i)^\top \Lambda(\boldsymbol{x}_i, \boldsymbol{y}_i) J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_i) + \mathbf{I}_P / \sigma_0^2,$$

where $J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) \triangleq \nabla_{\boldsymbol{\theta}} g_{\boldsymbol{\theta}}(\boldsymbol{x})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}$ and $\Lambda(\boldsymbol{x}, \boldsymbol{y}) \triangleq -\nabla^2_{\boldsymbol{gg}} \log p(\boldsymbol{y}|\boldsymbol{g})|_{\boldsymbol{g}=g_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})}$

- LLA applies LA to the first-order approximation of the NN of concern

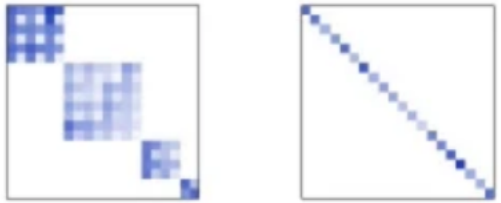
$$g^{\text{lin}}_{\boldsymbol{\theta}}(\boldsymbol{x}) = g_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) + J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})$$

- LLA is more sensible than LA in the presence of GGN approximation [Immer et al., 22]
- LLA can perform on par with or better than popular alternatives on various uncertainty quantification (UQ) tasks
- The Laplace library [Daxberger et al., 21] further substantially advances LLA's applicability

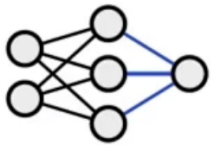
# Further approximations are required in practice!

- The GGN matrix of size P×P is still unamenable in modern DL scenarios (P is the number of parameters)

- Further approximations sparsifying the GGN are introduced:
  - Diagonal and KFAC [Martens & Grosse, 15] approximations



  - Concern only the last-layer inference



- However, these strategies sacrifice the fidelity of the learning outcomes as the approximation errors in these cases can hardly be theoretically measured

# Further approximations are required in practice!

- However, these strategies sacrifice the fidelity of the learning outcomes as the approximation errors in these cases can hardly be theoretically measured

- Our solution: accElerated Linearized Laplace Approximation (ELLA), which scales LLA up to make probabilistic predictions in a more assurable way

# The inherent connections between Neural Tangent Kernels (NTKs) and LLA

- Integrating $q(\theta) = \mathcal{N}(\theta; \hat{\theta}, \Sigma)$ with the linear model $g_\theta^{\text{lin}}(x)$ actually gives rise to a function-space approximate posterior

$$\mathcal{GP}(f|g_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}), \kappa_{\text{LLA}}(\boldsymbol{x}, \boldsymbol{x}')) \quad \text{with} \quad \kappa_{\text{LLA}}(\boldsymbol{x}, \boldsymbol{x}') \triangleq J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) \boldsymbol{\Sigma} J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}')^\top$$

- Doing some simple math, we have

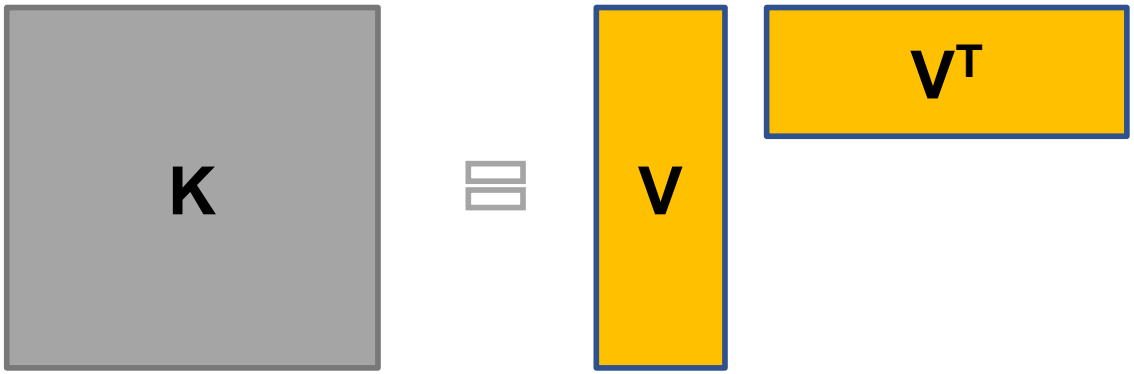
$$\kappa_{\text{LLA}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_0^2 \left( \kappa_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') - \kappa_{\text{NTK}}(\boldsymbol{x}, \mathbf{X}) [\boldsymbol{\Lambda}_{\mathbf{X},\mathbf{Y}}^{-1} / \sigma_0^2 + \kappa_{\text{NTK}}(\mathbf{X}, \mathbf{X})]^{-1} \kappa_{\text{NTK}}(\mathbf{X}, \boldsymbol{x}') \right)$$

where $\kappa_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') \triangleq J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}')^\top$

- The main challenge then turns into the computation and inversion of the gram matrix $\kappa_{\text{NTK}}(X, X)$ of size $NC \times NC$

# Kernel approximation of NTK enables the acceleration of LLA

- If we can approximate $\kappa_{NTK}(x, x')$ with the inner product of some explicit K-dim representations of the data, i.e., $\kappa_{\mathrm{NTK}}(\boldsymbol{x}, \boldsymbol{x}') \approx \varphi(\boldsymbol{x})\varphi(\boldsymbol{x}')^\top$ ,

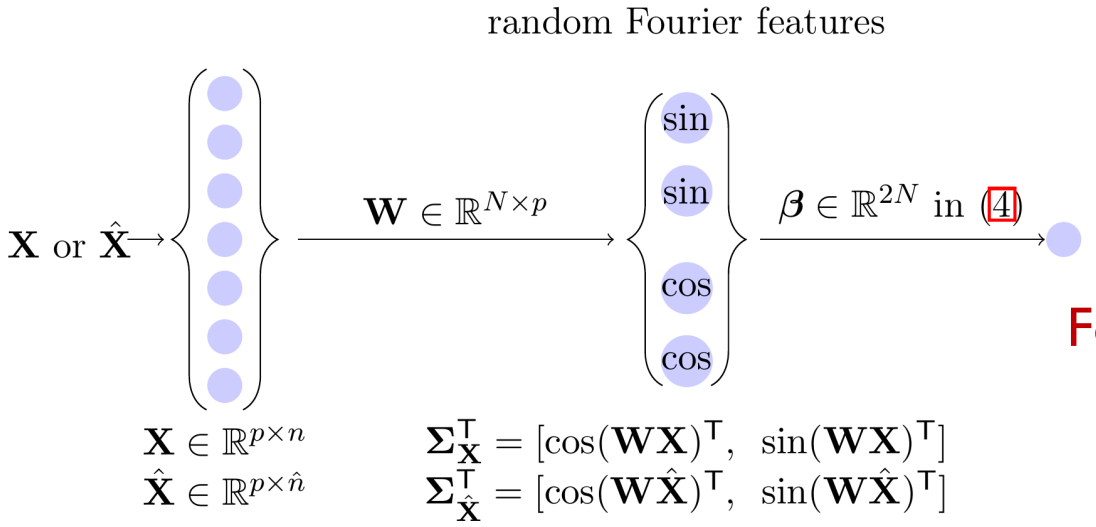



then

$$\begin{aligned}\kappa_{\mathrm{LLA}}(\boldsymbol{x}, \boldsymbol{x}') \approx & \sigma_0^2 \left( \varphi(\boldsymbol{x})\varphi(\boldsymbol{x}')^\top - \varphi(\boldsymbol{x})\boldsymbol{\varphi}_{\mathbf{X}}^\top \left[ \boldsymbol{\Lambda}_{\mathbf{X},\mathbf{Y}}^{-1}/\sigma_0^2 + \boldsymbol{\varphi}_{\mathbf{X}}\boldsymbol{\varphi}_{\mathbf{X}}^\top \right]^{-1} \boldsymbol{\varphi}_{\mathbf{X}}\varphi(\boldsymbol{x}')^\top \right) \\ = & \varphi(\boldsymbol{x}) \underbrace{\left[ \sum_i \varphi(\boldsymbol{x}_i)^\top \Lambda(\boldsymbol{x}_i, \boldsymbol{y}_i)\varphi(\boldsymbol{x}_i) + \mathbf{I}_K/\sigma_0^2 \right]^{-1}}_{\mathbf{G}} \varphi(\boldsymbol{x}')^\top \triangleq \kappa_{\mathrm{ELLA}}(\boldsymbol{x}, \boldsymbol{x}')\end{aligned}$$

# Kernel approximation

## Random features (RFs)




For shift-invariant kernels

[Rahimi et al., 2007]

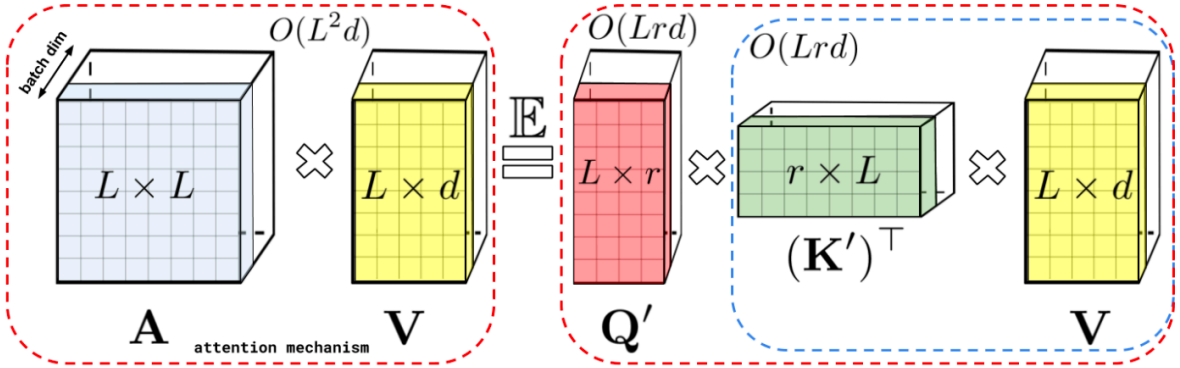



Figure 1: Approximation of the regular attention mechanism $\mathbf{AV}$ (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

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

[Choromanski et al., 2021]

# Kernel approximation
## 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:

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

# Adapt Nyström method to approximate the NTKs of multi-output NNs

- Rewrite $\kappa_{\text{NTK}}(x, x')$ as a scalar-valued kernel

$$\kappa_{\text{NTK}}\left((\boldsymbol{x}, i), (\boldsymbol{x}', i')\right) = J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}, i) J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}', i')^{\top}$$

By definition, the eigenfunctions can represent the spectral information of the kernel:

$$\int \kappa_{\text{NTK}}\left((\boldsymbol{x}, i), (\boldsymbol{x}', i')\right) \psi_k(\boldsymbol{x}', i') q(\boldsymbol{x}', i') = \mu_k \psi_k(\boldsymbol{x}, i), \forall k \geq 1,$$

while being orthonormal under $q$:

$$\int \psi_k(\boldsymbol{x}, i) \psi_{k'}(\boldsymbol{x}, i) q(\boldsymbol{x}, i) = \mathbb{1}[k = k'], \forall k, k' \geq 1.$$

By Monte Carlo estimation

$$\frac{1}{M} \sum_{m=1}^{M} \kappa_{\text{NTK}}\left((\boldsymbol{x}, i), (\boldsymbol{x}_m, i_m)\right) \psi_k(\boldsymbol{x}_m, i_m) = \mu_k \psi_k(\boldsymbol{x}, i), \forall k \in [K]$$

# Adapt Nyström method to approximate the NTKs of multi-output NNs

- Applying this equation to the above samples gives rise to

$$\frac{1}{M}\sum_{m=1}^{M}\kappa_{\mathrm{NTK}}\left((\boldsymbol{x}_{m'},i_{m'}),(\boldsymbol{x}_m,i_m)\right)\psi_k(\boldsymbol{x}_m,i_m)=\mu_k\psi_k(\boldsymbol{x}_{m'},i_{m'}),\forall k\in[K],m'\in[M]$$

then we arrive at

$$\frac{1}{M}\mathbf{K}\boldsymbol{\psi}_k\approx\mu_k\boldsymbol{\psi}_k,k\in[K]$$

- We compute the top-K eigenvalues $\lambda_1$, ..., $\lambda_K$ of matrix K and record the corresponding orthonormal eigenvectors $u_1$, ...,$u_K$. Then it is easy to get the Nyström approximation of the top-K eigenfunctions:

$$\hat{\psi}_k(\boldsymbol{x},i)=\frac{\sqrt{M}}{\lambda_k}\sum_{m=1}^{M}\boldsymbol{u}_k^{(m)}\kappa_{\mathrm{NTK}}\left((\boldsymbol{x},i),(\boldsymbol{x}_m,i_m)\right)=\frac{\sqrt{M}}{\lambda_k}J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x},i)\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top}\boldsymbol{u}_k$$

- Finally, we have

$$\varphi(\boldsymbol{x})=[J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})\boldsymbol{v}_1,\ldots,J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})\boldsymbol{v}_K]\text{ with }\boldsymbol{v}_k=\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top}\boldsymbol{u}_k/\sqrt{\lambda_k}$$

# Implementation

- The estimation of φ on a data point x degenerates as K JVPs, which can be accomplished by invoking forward mode automatic differentiation (fwAD) for K times:

$$\begin{pmatrix} \boldsymbol{x}, & \hat{\boldsymbol{\theta}} \\ & \boldsymbol{v}_k \end{pmatrix} \xrightarrow{\text{forward}} \begin{matrix} g_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) \\ J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})\boldsymbol{v}_k \end{matrix}, \; k \in [K]$$

where the model output $g_{\hat{\theta}}(x)$ and the JVP $J_{\hat{\theta}}(x)v_k$ are simultaneously computed in one single forward pass

- Prevalent DL libraries like PyTorch and Jax have already been armed with the capability for fwAD

# Algorithms

**Algorithm 1:** Build the LLA posterior.

```
# g_θ̂: NN pre-trained by MAP; (X, Y):
# training set; C: number of classes
# M, K, σ_0^2: hyper-parameters
def estimate_G(φ, X, Y, K, σ_0^2):
    G = zeros(K, K)
    for (x, y) in (X, Y):
        g_x, φ_x = φ(x)
        Λ_{x,y} = hessian(nll(g_x, y), g_x)
        G += φ_x^⊤ Λ_{x,y} φ_x
    return G + eye(K)/σ_0^2
def _q_f(φ, G^{-1}, x)
    g_x, φ_x = φ(x)
    κ_{x,x} = φ_x G^{-1} φ_x^⊤
    return N(g_x, κ_{x,x})
φ = build_φ(g_θ̂, X, C, M, K)
G^{-1} = inv(estimate_G(φ, X, Y, K, σ_0^2))
q_f = partial(_q_f, φ, G^{-1})
```

**Algorithm 2:** Build $\varphi$.

```
def build_φ(g_θ̂, X, C, M, K):
    def _φ(g_θ̂, C, {v_k}_{k=1}^K, x):
        φ_x = zeros(C, K)
        for k in range(K):
            with fwAD.enable():
                g_x, jvp = g_{(θ̂, v_k)}(x)
            φ_x[:, k] = jvp
        return g_x, φ_x
    J_{θ̂, X̃} = zeros(M, dim(θ̂))
    for m in range(M):
        x_m = uniform_sample(X)
        i_m = uniform_sample([C])
        J_{θ̂, X̃}[m] = grad(g_θ̂(x_m)[i_m], θ̂)
    {λ_k, u_k}_{k=1}^K = eig(J_{θ̂, X̃} J_{θ̂, X̃}^⊤, top = K)
    for k in range(K):
        v_k = J_{θ̂, X̃}^⊤ u_k / √λ_k
    return partial(_φ, g_θ̂, C, {v_k}_{k=1}^K)
```

# Theoretical Analysis

$\kappa_{\text{ELLA}}$ can be reformulated as follows (in the seek of theoretical analysis)

$$\kappa_{\text{ELLA}}(\boldsymbol{x}, \boldsymbol{x}') = J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}) \underbrace{\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top}\left[\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}^{\top}\boldsymbol{\Lambda}_{\mathbf{X},\mathbf{Y}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top} + \mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top}/\sigma_0^2\right]^{-1}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}}_{\boldsymbol{\Sigma}'} J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x})^{\top}$$

**Theorem 1** (Proof in Appendix A.4)**.** *Let $c_\Lambda$ be a finite constant associated with $\Lambda$, and $\mathcal{E}'$ the error of Nyström approximation $\|\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top}(\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top})^{-1}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}^{\top} - \mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}^{\top}\|$. It holds that*

$$\mathcal{E} \leq \sigma_0^4 c_\Lambda \mathcal{E}' + \sigma_0^2.$$

**Theorem 2** (Error bound of Nyström approximation)**.** *With probability at least $1-\delta$, it holds that:*

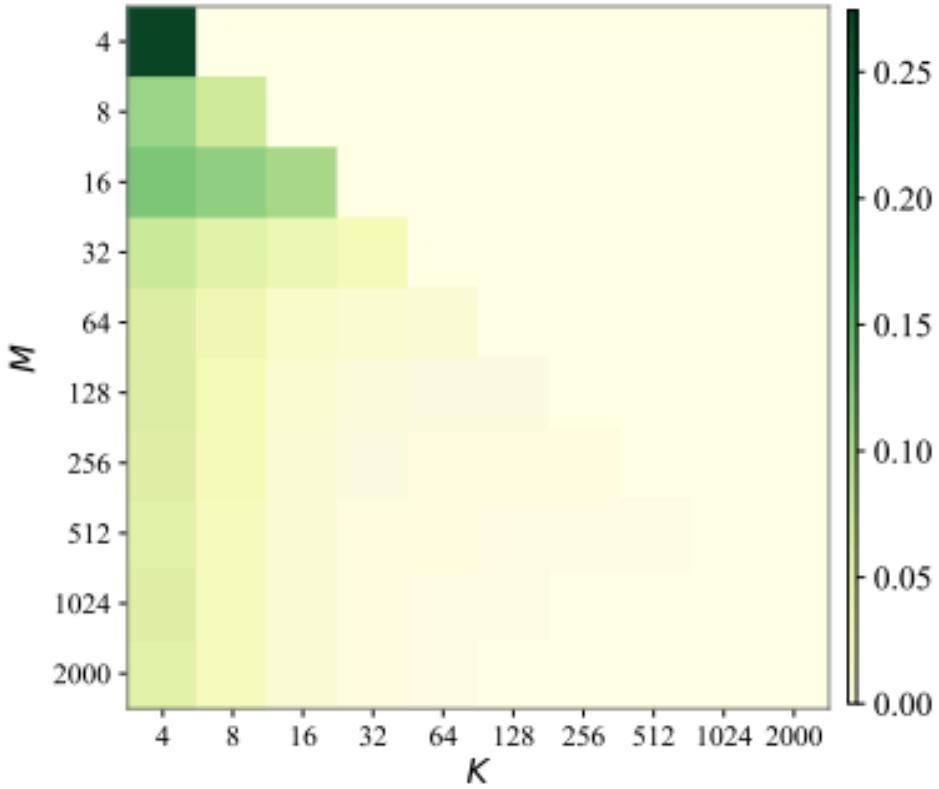
$$\mathcal{E}' \leq \tilde{\lambda}_{M+1} + \frac{NC}{\sqrt{M}} c_\kappa (2 + \log\frac{1}{\delta}).$$

**Corollary 1.** *With probability at least $1-\delta$, the following bound exists:*

$$\mathcal{E} \leq \sigma_0^4 c_\Lambda (\tilde{\lambda}_{M+1} + \frac{NC}{\sqrt{M}} c_\kappa (2 + \log\frac{1}{\delta})) + \sigma_0^2.$$

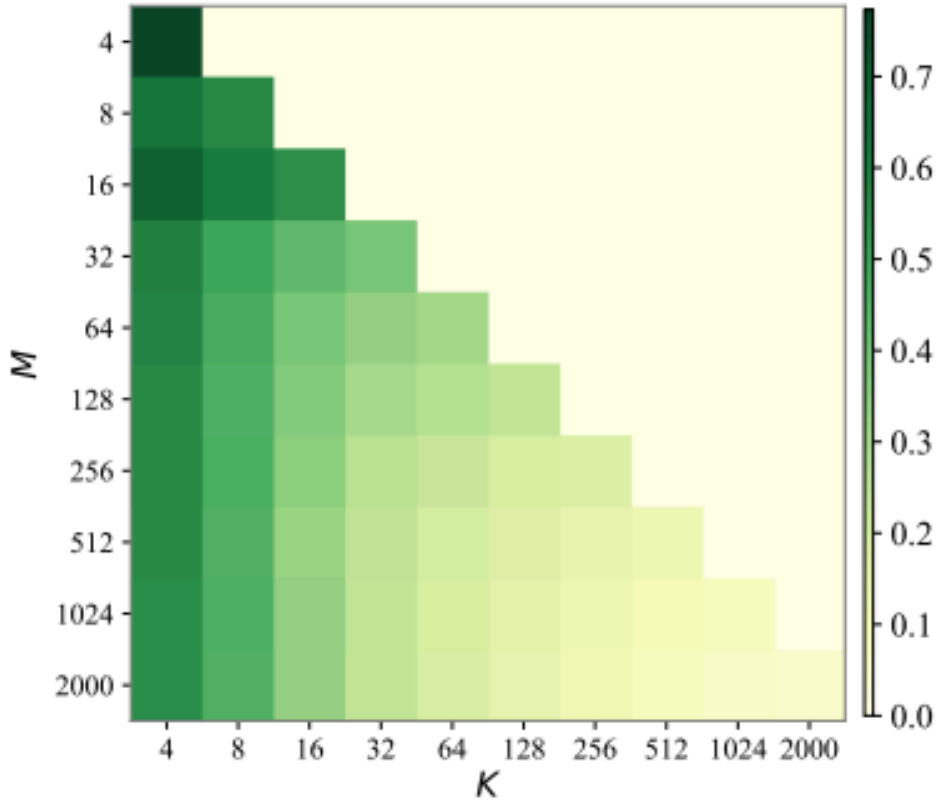
- As desired, the upper bound of approximation error decreases along with the growing of the number of MC samples in Nystrom approximation

# How the approximation errors vary w.r.t. K and M? (MNIST, CNNs)




(a)

$$\epsilon_{\text{Nyström}} \triangleq \|\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top}(\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}^{\top})^{-1}\mathbf{J}_{\hat{\boldsymbol{\theta}},\tilde{\mathbf{X}}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}^{\top} - \mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}^{\top}\|/\|\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}\mathbf{J}_{\hat{\boldsymbol{\theta}},\mathbf{X}}^{\top}\|$$



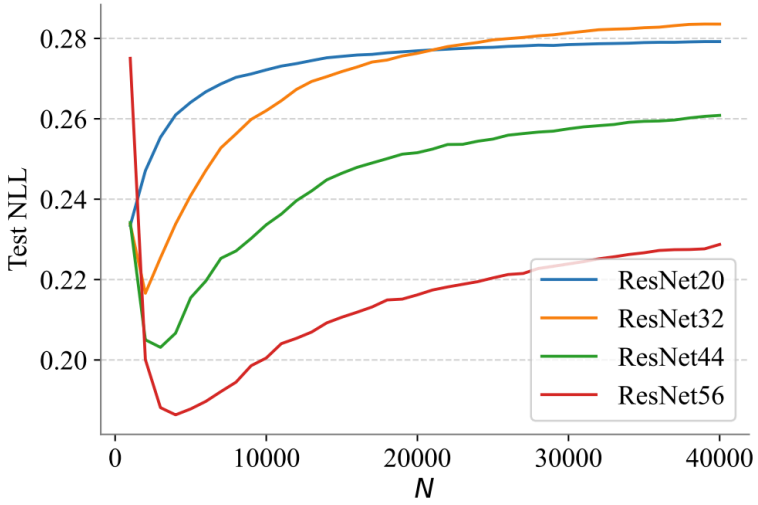

(b)

$$\epsilon_{\text{ELLA}} \triangleq \frac{1}{|\mathbf{X}_{\text{val}}|}\sum_{\boldsymbol{x}\in\mathbf{X}_{\text{val}}}\|\kappa_{\text{ELLA}}(\boldsymbol{x},\boldsymbol{x}) - \kappa_{\text{LLA}}(\boldsymbol{x},\boldsymbol{x})\|/\|\kappa_{\text{LLA}}(\boldsymbol{x},\boldsymbol{x})\|$$

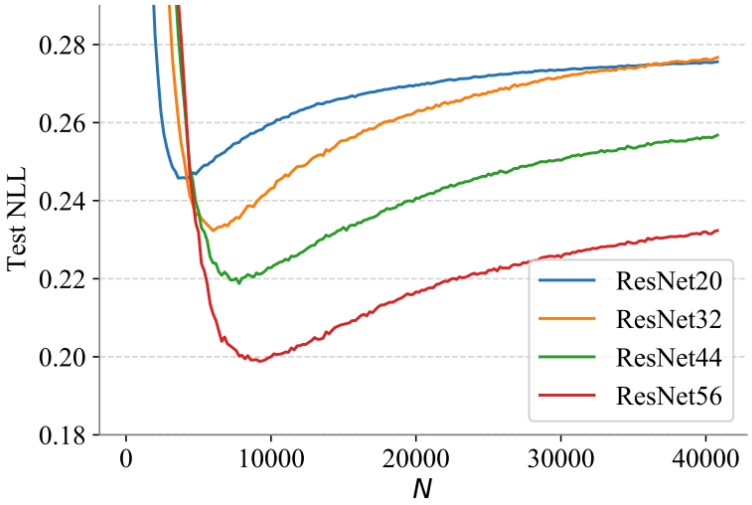
# The overfitting issue of ELLA and general LLA (Cifar-10, ResNets)

$$\boldsymbol{\Sigma}^{-1} = \sum_i J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_i)^\top \Lambda(\boldsymbol{x}_i, \boldsymbol{y}_i) J_{\hat{\boldsymbol{\theta}}}(\boldsymbol{x}_i) + \mathbf{I}_P / \sigma_0^2$$

With more training data involved, the covariance in LA, LLA, and ELLA shrinks and the uncertainty dissipates.




ELLA




Last-layer LLA

Solution: early stopping

# Illustrative Regression

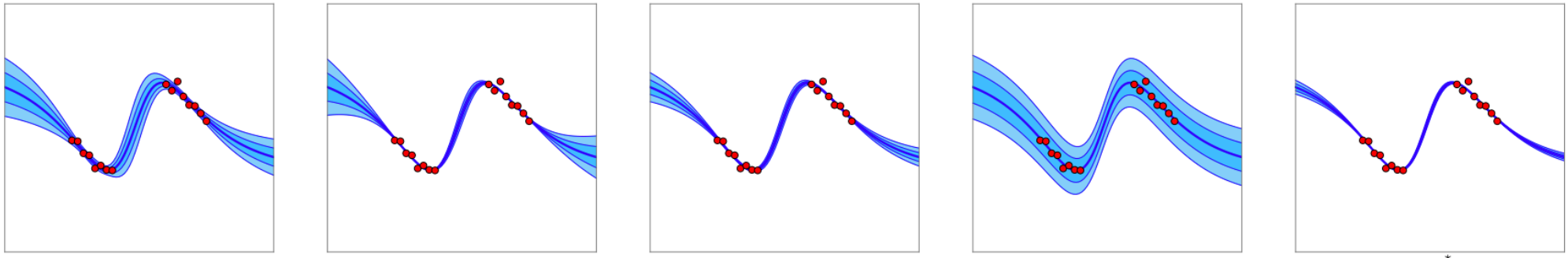



Figure 1: 1-D regression on $y = \sin 2x + \epsilon, \epsilon \sim \mathcal{N}(0, 0.2)$. Red dots, central blue curves, and shaded regions refer to the training data, mean predictions, and uncertainty respectively. The model is a pretrained multilayer perceptron (MLP) with 3 hidden layers. As shown, the predictive uncertainty of ELLA is on par with or better than the competitors such as LLA with KFAC approximation (LLA-KFAC), LLA with diagonal approximation (LLA-Diag), and last-layer LLA (LLA$^*$).

# Cifar-10 classification

Table 1: Comparison on test accuracy (%) ↑, NLL ↓, and ECE ↓ on CIFAR-10. We report the average results over 5 random runs. As the accuracy values of most methods are close, we do not highlight the best.

| Method | ResNet-20 | | | ResNet-32 | | | ResNet-44 | | | ResNet-56 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | NLL | ECE | Acc. | NLL | ECE | Acc. | NLL | ECE | Acc. | NLL | ECE |
| *ELLA* | 92.5 | 0.233 | **0.009** | 93.5 | **0.215** | **0.008** | 93.9 | **0.204** | **0.007** | 94.4 | **0.187** | **0.007** |
| *MAP* | 92.6 | 0.282 | 0.039 | 93.5 | 0.292 | 0.041 | 94.0 | 0.275 | 0.039 | 94.4 | 0.252 | 0.037 |
| *MFVI-BF* | 92.7 | **0.231** | 0.016 | 93.5 | 0.222 | 0.020 | 93.9 | 0.206 | 0.018 | 94.4 | 0.188 | 0.016 |
| *LLA** | 92.6 | 0.269 | 0.034 | 93.5 | 0.259 | 0.033 | 94.0 | 0.237 | 0.028 | 94.4 | 0.213 | 0.022 |
| *LLA*-KFAC* | 92.6 | 0.271 | 0.035 | 93.5 | 0.260 | 0.033 | 94.0 | 0.232 | 0.028 | 94.4 | 0.202 | 0.024 |
| *LLA-Diag* | 92.2 | 0.728 | 0.404 | 92.7 | 0.755 | 0.430 | 92.8 | 0.778 | 0.445 | 92.9 | 0.843 | 0.480 |
| *LLA-KFAC* | 92.0 | 0.852 | 0.467 | 91.8 | 1.027 | 0.547 | 91.4 | 1.091 | 0.566 | 89.8 | 1.174 | 0.579 |

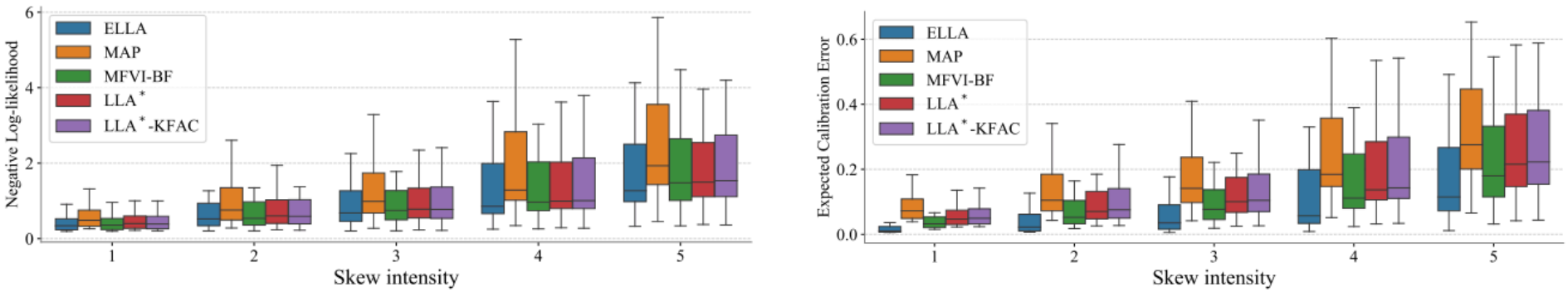



Figure 3: NLL (Left) and ECE (Right) on CIFAR-10 corruptions for models trained with ResNet-56 architecture. Each box corresponds to a summary of the results across 19 types of skew.

# ImageNet classification

Table 2: Comparison on test accuracy (%) ↑, NLL ↓, and ECE ↓ on ImageNet. We report the average results over 3 random runs.

| Method | ResNet-18 | | | ResNet-34 | | | ResNet-50 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Acc. | NLL | ECE | Acc. | NLL | ECE | Acc. | NLL | ECE |
| *ELLA* | 69.8 | 1.243 | **0.015** | 73.3 | 1.072 | **0.018** | **76.2** | 0.948 | **0.018** |
| *MAP* | 69.8 | 1.247 | 0.026 | 73.3 | 1.081 | 0.035 | **76.2** | 0.962 | 0.037 |
| *MFVI-BF* | **70.3** | **1.218** | 0.042 | **73.7** | **1.043** | 0.033 | 76.1 | **0.945** | 0.030 |

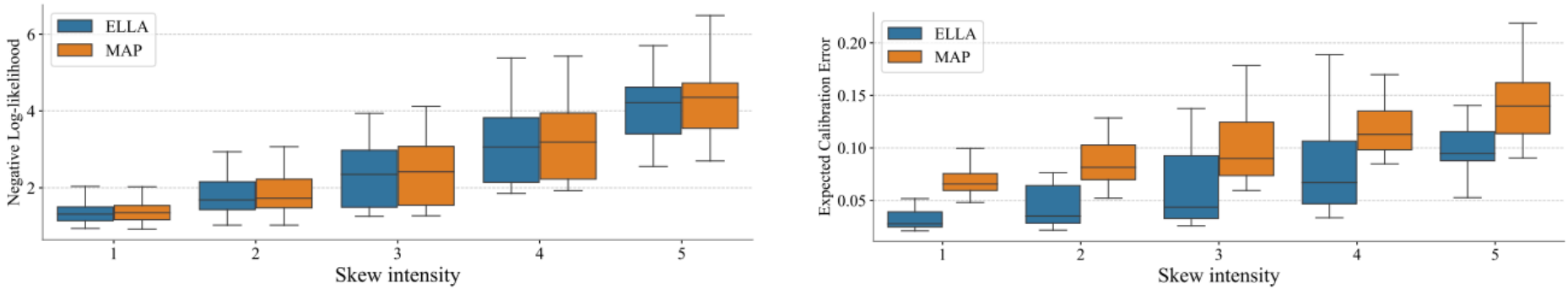



Figure 5: NLL (Left) and ECE (Right) on ImageNet corruptions for models trained with ViT-B architecture. Each box corresponds to a summary of the results across 19 types of skew.

## Results on ViT

| Method | Acc. | NLL | ECE |
|---|---|---|---|
| *ELLA* | 81.6 | 0.695 | 0.022 |
| *MAP* | 81.5 | 0.700 | 0.039 |

# Thanks!