Sketching Curvature for Efficient Out-of-Distribution Detection for Deep Neural Networks

Apoorva Sharma[1] **Navid Azizan**[1,2] **Marco Pavone**[1]

[1]Stanford University, Stanford, California, USA,
[2]Massachusetts Institute of Technology, Cambridge, Massachusetts, USA,

Abstract

In order to safely deploy Deep Neural Networks (DNNs) within the perception pipelines of real-time decision making systems, there is a need for safeguards that can detect out-of-training-distribution (OoD) inputs both efficiently and accurately. Building on recent work leveraging the local curvature of DNNs to reason about epistemic uncertainty, we propose *Sketching Curvature for OoD Detection (SCOD)*, an architecture-agnostic framework for equipping any trained DNN with a task-relevant epistemic uncertainty estimate. Offline, given a trained model and its training data, SCOD employs tools from matrix sketching to tractably compute a low-rank approximation of the Fisher information matrix, which characterizes which directions in the weight space are most influential on the predictions over the training data. Online, we estimate uncertainty by measuring how much perturbations orthogonal to these directions can alter predictions at a new test input. We apply SCOD to pre-trained networks of varying architectures on several tasks, ranging from regression to classification. We demonstrate that SCOD achieves comparable or better OoD detection performance with lower computational burden relative to existing baselines.

1 INTRODUCTION

Deep Neural Networks (DNNs) have enabled breakthroughs in extracting actionable information from high-dimensional input streams, such as videos or images. However, a key limitation of these black-box models is that their performance can be erratic when queried with inputs that are significantly different from those seen during training. To alleviate this, there has been a growing field of literature aimed at equipping pre-trained DNN models with a measure of their uncertainty. These approaches aim to characterize what a given DNN model has learned from the dataset it was trained on, so as to detect at test time whether a new input is inconsistent.

One appealing direction to this end has leveraged the curvature of a pre-trained DNN about its optimized weights [Madras et al., 2020, Ritter et al., 2018]. The second-order analysis of the local sensitivity of the network to its weights can offer a post-hoc approximation of the intractable Bayesian posterior on the network weights. Known as the Laplace approximation, this is an attractive approach in its promise of adding a principled measure of epistemic uncertainty to any pre-trained network. However, computing the required curvature matrix is quadratic in the number of weights of the network, and is still intractable for today's DNN models with millions of weights. Thus, the literature has focused on methods to approximate this curvature to yield scalable approaches, yet these approaches have typically relied on imposing sparsity structures on the curvature matrix, e.g., by ignoring cross terms between layers of the network. Furthermore, these approaches tend to focus on estimating the *posterior predictive* distribution by marginalizing over the approximate posterior over the weights, which often requires the computationally intensive process of sampling weights and estimating the posterior predictive through Monte-Carlo integration.

Contributions. In this work, we build from the framework of the Laplace approximation to propose a novel, architecture-agnostic method for equipping any trained DNN with estimates of task-relevant input atypicality. Rather than incorporate epistemic uncertainty into the probabilistic prediction of the network, we propose augmenting the network output with a scalar uncertainty measure that quantifies the degree of atypicality for a given input. Our specific contributions are as follows: (1) We leverage information geometry to translate curvature-informed weight uncertainty in DNNs to a task-relevant measure of input

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

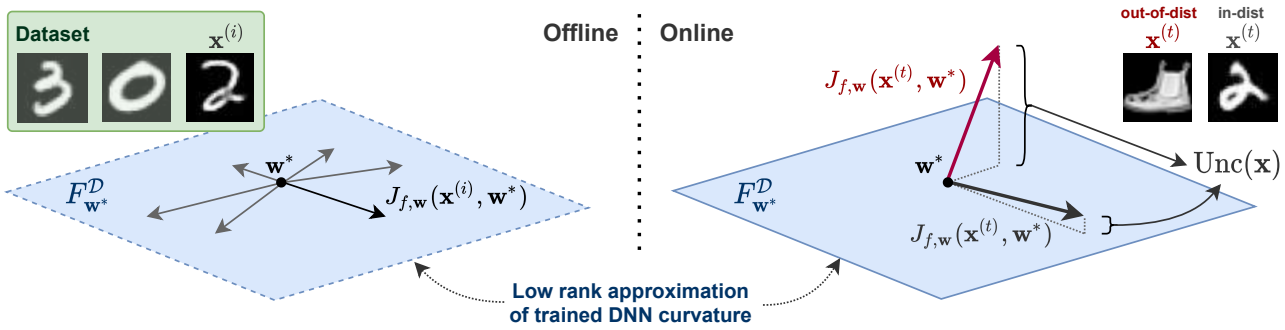

Figure 1: Offline, SCOD uses gradients $\boldsymbol{J}_{f,\boldsymbol{w}}$ of the DNN on training data to build a low-rank estimate of the network curvature, as defined by the Fisher information matrix, employing tools from matrix sketching to make this estimation tractable and scalable. Online, SCOD produces a metric of atypicality Unc using gradient information at a test datapoint $\boldsymbol{x}^{(t)}$, and comparing this to the low-rank approximation of the Fisher. Derived from an approximation of the Bayesian posterior over the weights, this metric can be viewed, for scalar regression DNNs, as measuring how orthogonal a new test point gradient is from the gradients seen on training inputs.

atypicality; (2) we show how this measure can be computed efficiently by leveraging backpropagation and a characterization of the top eigenspace of the Fisher matrix; (3) we develop a technique based on matrix sketching to tractably approximate this top eigenspace, even for large models and large datasets; (4) we empirically demonstrate that this atypicality measure matches or exceeds the OoD detection performance relative to baselines on a diverse set of problems and architectures from regression to classification.

2 PROBLEM STATEMENT

We take a probabilistic interpretation, and define a DNN model as a mapping from inputs $\boldsymbol{x} \in \mathcal{X}$ to probability distributions over targets $\boldsymbol{y} \in \mathcal{Y}$. Typically, this is broken down into the composition of a neural network architecture f mapping inputs $\boldsymbol{x}$ and weights $\boldsymbol{w} \in \mathbb{R}^N$ to the distribution parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, and a distributional family $\mathcal{P}$ mapping these parameters to a distribution over the targets:

$$\boldsymbol{\theta} = f(\boldsymbol{x}, \boldsymbol{w}), \qquad p(\boldsymbol{y}) = \mathcal{P}(\boldsymbol{\theta}).$$

We use $p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x}) := \mathcal{P} \circ f$ as a convenient shorthand for the conditional distribution that the DNN model defines. This model is trained on a dataset of examples $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^M$, where we assume the inputs are drawn i.i.d. from some training distribution $p_{\text{train}}(x, y)$, to minimize the Kullback-Leibler (KL) divergence from the empirical distribution in the dataset to $p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x})$. [1]

Our goal is to take a trained DNN, and equip it with an OoD monitor which can detect if a given input is far from the data seen at train time, and thus likely to result in incorrect and overconfident predictions. Specifically, we assume we are given the functional forms f and $\mathcal{P}$, the trained weights $\boldsymbol{w}^*$, as well as a set of training data $\mathcal{D}$, and wish to construct an uncertainty measure $\mathrm{Unc}(\cdot) : \mathcal{X} \to \mathbb{R}$. Intuitively, we wish for this measure to be low when queried on inputs drawn from $p_{\text{train}}(x)$, but high for inputs far from this data manifold. This can be thought of as providing a measure of *epistemic uncertainty* due to an input not being covered by the training set, or a measure of *novelty* with respect to the training data, useful for detecting *anomalies* or *out-of-distribution* inputs.

We wish to design a monitor that can provide a real-time uncertainty signal alongside the DNN's predictions. Thus, we desire a function $\mathrm{Unc}(\cdot)$ that is both computationally efficient (to run at test time) as well as informative, providing a meaningful anomaly signal that can effectively separate held-out validation inputs that are known to be in-distribution, from inputs which we know to be out-of-distribution and outside the DNN's domain of competency.

3 BACKGROUND: CURVATURE AND LAPLACE APPROXIMATION

Key to our approach is the curvature of DNN models: a second-order characterization of how perturbations to the weights of a DNN influence its probabilistic output. In this section, we review tools to characterize DNN curvature, and discuss the connections between curvature and epistemic uncertainty estimates through the Laplace approximation.

[1] This general formulation covers almost all applications of DNNs; for example, choosing $\mathcal{P}$ to be a categorical distribution parameterized by logits $\boldsymbol{\theta}$ recovers the standard softmax training objective for classification, while choosing $\mathcal{P}$ to be unit variance Gaussian with mean $\boldsymbol{\theta}$ recovers the typical mean-squared error objective for regression.

3.1 FISHER INFORMATION

A crucial aspect of our approach relies on understanding how the parameters of the model influence its output distribution. To do so, we leverage tools from information geometry. We can view the family of distributions defined by the model $\mathcal{P}(\boldsymbol{\theta})$ as defining a statistical manifold with coordinates $\boldsymbol{\theta}$. The Riemannian metric for this manifold is the Fisher information matrix

$$F_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{y}}\left[(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}))(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}))^{\mathsf{T}}\right] \quad (1)$$

$$= \mathbb{E}_{\boldsymbol{y}}\left[-\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \log p_{\boldsymbol{\theta}}(\boldsymbol{y})\right], \quad \text{(see note)}^2 \quad (2)$$

where $p_{\boldsymbol{\theta}}(\cdot) = \mathcal{P}(\boldsymbol{\theta})$, the pdf of the probability distribution on $\mathcal{Y}$ defined by parameters $\boldsymbol{\theta}$. The Fisher information matrix (henceforth referred to as the Fisher) represents the second-order approximation of the local KL divergence, describing how the output distribution of a model changes with small perturbations to the distribution parameters $\boldsymbol{\theta}$:

$$D_{\mathsf{KL}}(\mathcal{P}(\boldsymbol{\theta})||\mathcal{P}(\boldsymbol{\theta} + d\boldsymbol{\theta})) \approx d\boldsymbol{\theta}^{\mathsf{T}} F_{\boldsymbol{\theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta} + O(d\boldsymbol{\theta}^3). \quad (3)$$

The subscript on $F_{\boldsymbol{\theta}}$ serves to make explicit the Fisher's dependence on the model's parameterization.

For many common parametric distributions, the Fisher can be computed analytically. For example, for the family of Gaussian distributions with fixed covariance Σ, parameterized by the mean vector $\boldsymbol{\theta} = \boldsymbol{\mu}$, the Fisher information is simply the constant $F_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \Sigma^{-1}$. For a categorical distribution parameterized such that $\boldsymbol{\theta}_i$ represents the probability assigned to class i, $F(\boldsymbol{\theta}) = (\mathsf{diag}(\boldsymbol{\theta}))^{-1}$. In cases where this analytic computation is not possible or difficult, one can compute a Monte-Carlo approximation of the Fisher by sampling $\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot)$ to estimate the expectation in (1).

3.2 FISHER FOR DEEP NEURAL NETWORKS

For DNNs, we can also consider the Fisher in terms of the network weights $\boldsymbol{w}$ using a change of variables. Since $\boldsymbol{\theta} = f(x, \boldsymbol{w})$, we have

$$F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{J}_{f,\boldsymbol{w}}^{\mathsf{T}} F_{\boldsymbol{\theta}}(f(\boldsymbol{x}, \boldsymbol{w})) \boldsymbol{J}_{f,\boldsymbol{w}}, \quad (4)$$

where $\boldsymbol{J}_{f,\boldsymbol{w}}$ is the Jacobian matrix with $[\boldsymbol{J}_{f,\boldsymbol{w}}]_{ij} = \frac{\partial f_i}{\partial w_j}$, evaluated at $(\boldsymbol{x}, \boldsymbol{w})$. Note that $F_{\boldsymbol{w}}$ is a function of both $\boldsymbol{w}$ and the input $\boldsymbol{x}$. We will henceforth use the shorthand $F^{(t)}_{\boldsymbol{w}^*} := F_{\boldsymbol{w}}(\boldsymbol{x}^{(t)}, \boldsymbol{w}^*)$ to denote this weight-space Fisher evaluated for a particular input $\boldsymbol{x}^{(t)}$ and the trained weights $\boldsymbol{w}^*$.

From (3), we see that the Fisher defines a second-order approximation of how perturbations in the weight space influence the DNN's probabilistic predictions:

$$\delta_{\mathsf{KL}}(\boldsymbol{x}^{(t)}) := D_{\mathsf{KL}}\left(p_{\boldsymbol{w}^*}(\cdot \mid \boldsymbol{x}^{(t)})||p_{\boldsymbol{w}^*+d\boldsymbol{w}}(\cdot \mid \boldsymbol{x}^{(t)})\right)$$
$$\approx d\boldsymbol{w}^{\mathsf{T}} F^{(t)}_{\boldsymbol{w}^*} d\boldsymbol{w}.$$

We can also use the Fisher to consider the impact of weight perturbations on the predictions over the entire dataset as

$$\delta_{\mathsf{KL}}(\mathcal{D}) = \frac{1}{M}\sum_{i=1}^{M} \delta_{\mathsf{KL}}(\boldsymbol{x}^{(i)}) = \frac{1}{M}\sum_{i=1}^{M} d\boldsymbol{w}^{\mathsf{T}} F^{(i)}_{\boldsymbol{w}^*} d\boldsymbol{w}$$
$$= d\boldsymbol{w}^{\mathsf{T}} \underbrace{\left(\frac{1}{M}\sum_{i=1}^{M} F^{(i)}_{\boldsymbol{w}^*}\right)}_{:=F^{\mathcal{D}}_{\boldsymbol{w}^*}} d\boldsymbol{w}.$$

3.3 CONNECTION TO THE HESSIAN

As is evident from (1), there are strong connections between the dataset Fisher $F^{\mathcal{D}}_{\boldsymbol{w}^*}$ and the Hessian with respect to $\boldsymbol{w}$ of the log likelihood of the training data. If we define $L(\boldsymbol{w}) = \sum_{i=1}^{M} \log p_{\boldsymbol{w}^*}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)})$, the Hessian of L evaluated at $\boldsymbol{w}^*$ can be well approximated by the dataset Fisher[3] [Ritter et al., 2018, Martens and Grosse, 2015]:

$$H_L = \left.\frac{\partial^2 L}{\partial \boldsymbol{w}^2}\right|_{\boldsymbol{w}=\boldsymbol{w}^*} \approx M F^{\mathcal{D}}_{\boldsymbol{w}^*}.$$

Unlike the Hessian, the Fisher is always guaranteed to be positive semidefinite. For this reason, this approximation is common in the field of second-order optimization, where preconditioning gradient steps with the inverse Fisher tends to be more efficient and numerically stable than using the Hessian [Kunstner et al., 2019].

3.4 THE LAPLACE APPROXIMATION OF EPISTEMIC UNCERTAINTY

The Hessian of the log-likelihood has strong connections to Bayesian ideas of *epistemic uncertainty*, the uncertainty due to lack of data. From a Bayesian perspective, one can choose a prior over the weights of a DNN, $p(\boldsymbol{w})$, and then reason about the posterior distribution on these weights given the dataset, $p(\boldsymbol{w} \mid \mathcal{D})$. Often, due to the overparameterized nature of DNNs, many values of $\boldsymbol{w}$ are likely under this dataset,

[2]This equality holds under mild regularity conditions on $\mathcal{P}(\boldsymbol{\theta})$

[3]By application of the chain rule, $H_L = \hat{F}^{\mathcal{D}}_{\boldsymbol{w}^*} + C(\boldsymbol{w}^*)$, where $\hat{F}^{\mathcal{D}}_{\boldsymbol{w}^*}$ is the empirical Fisher where the expectation in (1) is replaced by an empirical expectation over the dataset, and $C(\boldsymbol{w}^*)$ is a term involving first derivatives of the log likelihood and second derivatives of the network f. For trained networks, we expect the Fisher and the empirical Fisher to be closely aligned, and $C(\boldsymbol{w}^*) \approx 0$ since the first derivatives of the log likelihood are near zero at the end of training. Furthermore, for piece-wise linear networks (e.g., with ReLU activations), the second deriviatves of f are 0, and so, $C(\boldsymbol{w}^*) = 0$.

corresponding to different ways the DNN can fit the training data [Azizan et al., 2019]. By characterizing the posterior, and then marginalizing over it to produce a posterior predictive distribution $p(\boldsymbol{y} \mid \boldsymbol{x}) = \int p(\boldsymbol{w} \mid \mathcal{D}) p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x}) d\boldsymbol{w}$, one can hope to detect atypical data by incorporating uncertainty due to lack of data into the network's probabilistic predictions.

While computing this posterior is intractable for DNNs, due to their nonlinearity and high-dimensional weight space, many approximations exist, leveraging Monte-Carlo sampling, or by assuming a distributional form and carrying out variational inference. One approximation is the Laplace approximation [MacKay, 1992], which involves a second-order approximation of the log posterior $\log p(\boldsymbol{w} \mid \mathcal{D})$ about a point estimate $\boldsymbol{w}^*$. This quadratic form yields a Gaussian posterior over the weights. If the prior on the weights is $p(\boldsymbol{w}) = \mathcal{N}(\cdot; 0, \epsilon^2 I_N)$, the Laplace posterior is given by $\Sigma^* = \left(H_L + \epsilon^{-2} I_N\right)^{-1}$. The Laplace approximation is attractive as it uses local second-order information (the Hessian H_L) to produce an estimate of the Bayesian posterior for any pretrained model. However, even computing this approximation to the exact Bayesian posterior can be challenging for large models, where estimating and inverting an $N \times N$ matrix to compute Σ^* is demanding.

4 PROPOSED METHOD: SKETCHING CURVATURE FOR OOD DETECTION

In this section, we describe the main steps in our OoD detection method. We build upon ideas from the Laplace approximation, but using the dataset Fisher to approximate the Hessian. Thus, our approximate posterior is given by

$$\Sigma^* = \left(M F^{\mathcal{D}}_{\boldsymbol{w}^*} + \epsilon^{-2} I\right)^{-1}. \tag{5}$$

As we are only focused on the downstream-task OoD detection, we avoid the computationally expensive step of marginalizing over this uncertainty to form a posterior predictive distribution, and instead, directly quantify how this posterior uncertainty on the weights impacts the networks probabilistic predictions. A natural metric for that purpose is the expected change in the output distribution (measured by the KL divergence) when the weights are perturbed according to the Laplace posterior distribution, i.e.,

$$\begin{aligned} \mathrm{Unc}(\boldsymbol{x}^{(t)}) &= \mathop{\mathbb{E}}_{d\boldsymbol{w} \sim \mathcal{N}(0, \Sigma^*)} \left[\delta_{\mathsf{KL}}(\boldsymbol{x}^{(t)}) \right] \\ &\approx \mathop{\mathbb{E}}_{d\boldsymbol{w} \sim \mathcal{N}(0, \Sigma^*)} \left[d\boldsymbol{w}^{\mathsf{T}} F^{(t)}_{\boldsymbol{w}^*} d\boldsymbol{w} \right] \\ &= \mathrm{Tr}\left(F^{(t)}_{\boldsymbol{w}^*} \Sigma^* \right). \end{aligned} \tag{6}$$

Crucially, by using the Fisher to estimate the change in the output distribution due to weight perturbation, we obtain a quadratic form whose expectation we can compute analytically. Note that this metric is the Frobenius inner product between the test input's Fisher and a regularized inverse of the dataset Fisher. Thus, we can view this metric as measuring novelty in the terms of DNN's curvature, i.e., measuring the abnormality of a test input $\boldsymbol{x}$ by comparing the curvature at this input against the curvature on training inputs.

4.1 EFFICIENTLY COMPUTING THE UNCERTAINTY METRIC

While (6) is a clean expression for an uncertainty metric, its naïve computation is intractable for typical DNNs, as both Σ^* and $F^{(t)}_{\mathcal{D}}$ are $N \times N$ matrices. To make this computation amenable for real-time operation, we exploit the fact that both matrices are low-rank. From their definitions, it follows that $\mathsf{rank}(F^{(t)}_{\boldsymbol{w}^*}) \leq d$, and thus $\mathsf{rank}(F^{\mathcal{D}}_{\boldsymbol{w}^*}) \leq Md$. The number of weights in a neural network N is always greater than the output dimension d, and for large models and typical dataset sizes, often also greater than Md. Thus, we choose instead to express both Fisher matrices in factored forms

$$F^{(t)}_{\boldsymbol{w}^*} = L^{(t)}_{\boldsymbol{w}^*} L^{(t)\mathsf{T}}_{\boldsymbol{w}^*}, \qquad F^{\mathcal{D}}_{\boldsymbol{w}^*} = U \mathsf{diag}(\boldsymbol{\lambda}) U^T, \tag{7}$$

where $L^{(t)}_{\boldsymbol{w}^*} \in \mathbb{R}^{N \times d}, U \in \mathbb{R}^{N \times Md}$, and $\boldsymbol{\lambda} \in \mathbb{R}^{Md}$. Note that if we write $F_{\boldsymbol{\theta}}(f(\boldsymbol{x}^{(t)}, \boldsymbol{w}^*)) = L^{(t)}_{\boldsymbol{\theta}^*} L^{(t)\mathsf{T}}_{\boldsymbol{\theta}^*}$, then from (4), we can see that $L^{(t)}_{\boldsymbol{w}^*} = \boldsymbol{J}^{\mathsf{T}}_{f,\boldsymbol{w}} L^{(t)}_{\boldsymbol{\theta}^*}$. For many common choices of parametric distributions, $L^{(t)}_{\boldsymbol{\theta}^*}$ can be computed analytically. Furthermore, leveraging the linearity of the derivative, we can compute each row of $L^{(t)}_{\boldsymbol{w}^*}$ efficiently via backpropagation (see Appendix A for details and examples for common distributions).

Given these factored forms, an application of the Woodbury matrix identity allows us to simplify the computation to

$$\begin{aligned} &\mathrm{Unc}(\boldsymbol{x}^{(t)}) = \\ &\epsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}} - \epsilon^2 \left\| \mathsf{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}}{\boldsymbol{\lambda} + 1/(M\epsilon^2)}} \right) U^{\mathsf{T}} L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\mathrm{F}}. \end{aligned} \tag{8}$$

A derivation is provided in Appendix B. In this new form, the computation is split into computing the factor $L^{(t)}_{\boldsymbol{w}^*}$, carrying out the matrix product with U^{T}, and then computing Frobenius norms.

The main bottleneck in this procedure, both in terms of memory and computation, is the matrix multiplication with the $N \times Md$ matrix U. To address this, we note that several empirical analyses of neural network curvature have found that the Hessian and dataset Fisher $F^{\mathcal{D}}_{\boldsymbol{w}^*}$ exhibit rapid spectral decay [Sagun et al., 2017, Madras et al., 2020], meaning that there are only a small number of significant eigenvalues and eigenvectors. Inspecting (8), we see that if $\lambda_i << (M\epsilon^2)^{-1}$, the corresponding element in the diagonal matrix tends to 0. This suggests that we can effectively

approximate this uncertainty metric *without* the tail eigenvalues. Using $\boldsymbol{\lambda}_{\text{top}}$ and U_{top} to denote the top k eigenvalues and corresponding eigenvectors of $F^{\mathcal{D}}_{\boldsymbol{w}^*}$, we have

$$\widetilde{\text{Unc}}(\boldsymbol{x}^{(t)}) = \epsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\text{F}} - \epsilon^2 \left\| \text{diag}\left(\sqrt{\frac{\boldsymbol{\lambda}_{\text{top}}}{\boldsymbol{\lambda}_{\text{top}} + 1/(M\epsilon^2)}} \right) U^{\mathsf{T}}_{\text{top}} L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{\text{F}}. \tag{9}$$

Working with the much smaller $N \times k$ matrix U_{top} drastically improves both memory and computational complexity. Notably, making this low-rank approximation gives us a strict *over-estimate* of the exact quantity, which is well-suited for safety-critical settings, where being under-confident is more desirable than being over-confident. The approximation error relates to the magnitude of the tail eigenvalues, and can be bounded using the final obtained eigenvalue

$$\begin{aligned} &\widetilde{\text{Unc}}(\boldsymbol{x}^{(t)}) - \text{Unc}(\boldsymbol{x}^{(t)}) \\ &\quad \leq \epsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{F} \sum_{j=k+1}^{\mathsf{rank}\left(F^{\mathcal{D}}_{\boldsymbol{w}^*}\right)} \frac{\lambda_j}{\lambda_j + 1/(M\epsilon^2)} \qquad (10) \\ &\quad \leq \epsilon^2 \left\| L^{(t)}_{\boldsymbol{w}^*} \right\|^2_{F} (\min(Md, N) - k) \ln(1 + M\epsilon^2 \lambda_k) \qquad (11) \end{aligned}$$

A derivation of this bound is provided in Appendix B.

4.2 TRACTABLY APPROXIMATING THE DATASET FISHER VIA MATRIX SKETCHING

In order to compute $\widetilde{\text{Unc}}(\boldsymbol{x}^{(t)})$ online, we require the top k eigenvalues and eigenvectors of $F^{\mathcal{D}}_{\boldsymbol{w}^*}$. While this computation can happen offline, it is still intractable to carry out exactly for common DNN models and large datasets, since just representing $F^{\mathcal{D}}_{\boldsymbol{w}^*}$ exactly requires storing an $Md \times N$ factor, which can easily grow beyond the capacity of common GPU memories for large perception networks and datasets with tens of thousands of parameters. To alleviate this issue, prior work has considered imposing sparsity patterns on the Fisher, e.g., diagonal (which ignores correlations between weights), or layer-wise block-diagonal [Ritter et al., 2018] (which ignores correlations between layers). Instead, we note that only the top eigenvectors of the datastet Fisher are important to the computation of our uncertainty metric, and thus we turn to tools from matrix sketching to tractably estimate a low-rank approximation of the Fisher *without* imposing any sparsity structure on the matrix.

The key idea in matrix sketching, to avoid working with a large matrix directly, is to apply a randomized linear map S to the matrix of interest [Tropp et al., 2017]. By appropriately randomizing this map (the *sketching operator*), we obtain high-probability guarantees that the image of the original matrix produced by the map (the *sketch*) encodes sufficient information about the original matrix. Given a bound on the desired approximation error, the size required for the sketch depends on the desired rank of the approximation k, and not the original size of the large matrix. Thus, this can enable our technique to be applied to arbitrarily large datasets.

The linearity of the sketching operator allows us to form this sketch iteratively, using a single pass over the dataset, without storing the full dataset Fisher in memory. Specifically, from the definition of $F^{\mathcal{D}}_{\boldsymbol{w}^*}$, we can compute its sketch as the sum of smaller sketches:

$$\mathsf{S}\left(F^{\mathcal{D}}_{\boldsymbol{w}^*}\right) = \frac{1}{M} \sum_{i=1}^{M} \mathsf{S}\left(F^{(i)}_{\boldsymbol{w}^*}\right) = \frac{1}{M} \sum_{i=1}^{M} \mathsf{S}\left(L^{(i)}_{\boldsymbol{w}^*} L^{(i)\mathsf{T}}_{\boldsymbol{w}^*}\right). \tag{12}$$

Following Tropp et al. [2017], we choose S to independently left- and right-multiply the Fisher by random sketching matrices. Specifically, the sketch of each component is computed as $Y^{(i)}, W^{(i)} \leftarrow \mathsf{S}(L^{(i)}_{\boldsymbol{w}^*} L^{(i)\mathsf{T}}_{\boldsymbol{w}^*})$, where

$$Y^{(i)} = \left(\left(\Omega L^{(i)}_{\boldsymbol{w}^*} \right) L^{(i)\mathsf{T}}_{\boldsymbol{w}^*} \right)^{\mathsf{T}}, \quad W = \left(\Psi L^{(i)}_{\boldsymbol{w}^*} \right) L^{(i)\mathsf{T}}_{\boldsymbol{w}^*}, \tag{13}$$

where $\Omega \in \mathbb{R}^{r \times N}$ and $\Psi \in \mathbb{R}^{s \times N}$ are random sketching matrices, with $T = r + s$ defining the total size of the sketch. Note that following the operation order indicated by the parentheses avoids instantiating any $N \times N$ matrix. The memory complexity of this operation is $O(T(N + d))$. Tropp et al. [2017] suggest splitting the budget to $r = (T - 1)/3$, $s = T - r$, and suggest choosing $T = 6k + 4$ as a minimal value of T for a given k to minimize a high-probability bound on the approximation error of the sketch. We refer the reader to [Tropp et al., 2017] for a discussion of these theoretical results.

These sketching matrices can be as simple as matrices with i.i.d. standard Gaussian entries. However, to further reduce the memory and computation overhead of sketching, in this work, we use Subsampled Randomized Fourier Transform (SRFT) sketching matrices [Woolfe et al., 2008]

$$\Omega = P_1 F_N \text{diag}(\boldsymbol{d}_1), \qquad \Psi = P_2 F_N \text{diag}(\boldsymbol{d}_2), \tag{14}$$

where $\boldsymbol{d}_1, \boldsymbol{d}_2 \in \mathbb{R}^N$ are vectors with entries sampled from independent Rademacher random variables[4], F_N is the linear operator which applies the discrete cosine transform on each N-length column, and P_1, P_2 are matrices which each select a random subset of r and s rows respectively. The SRFT sketching matrices offer similar approximation performance when compared to Gaussian matrices, but adding only a $(T + 2N)$ parameter overhead, as opposed to TN of the Gaussian case [Tropp et al., 2017].

[4] A Rademacher random variable has a value of $+1$ or -1 with equal probability.

Algorithm 1 SCOD Offline

Require: Dataset $\mathcal{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{M}$, DNN architecture $f, \mathcal{P}$, trained weights $\boldsymbol{w}^*$

function SKETCHCURVATURE($f, \mathcal{P}, \boldsymbol{w}^*, \mathcal{D}$)
  Sample Ω, Ψ as in (14). ▷ construct sketching map
  $Y, W \leftarrow 0, 0$ ▷ initialize sketch
  for $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ **in** $\mathcal{D}$ **do**
    $\boldsymbol{\theta}^{(i)} \leftarrow f(\boldsymbol{x}^{(i)}, \boldsymbol{w}^*)$ ▷ forward pass
    Compute $L^{(i)}_{\boldsymbol{w}^*}$ from $\boldsymbol{\theta}^{(i)}$ ▷ d backward passes
    $Y \leftarrow Y + \frac{1}{M}\left(\left(\Omega L^{(i)}_{\boldsymbol{w}^*}\right) L^{(i)\mathsf{T}}_{\boldsymbol{w}^*}\right)^{\mathsf{T}}$ ▷ update sketch
    $W \leftarrow W + \frac{1}{M}\left(\Psi L^{(i)}_{\boldsymbol{w}^*}\right) L^{(i)\mathsf{T}}_{\boldsymbol{w}^*}$
  end for
  $U_{\text{top}}, \boldsymbol{\lambda}_{\text{top}} \leftarrow$ FIXEDRANKSYM(Ω, Ψ, Y, W)
  return $U_{\text{top}}, \boldsymbol{\lambda}_{\text{top}}$
end function

Armed with these tools, we can now follow the procedure detailed in Algorithm 1 to produce a low-rank approximation of the dataset Fisher. Our approach uses one pass over the dataset $\mathcal{D}$, incrementally building the sum in (12) by applying the SRFT sketching matrices to the $L^{(i)}_{\boldsymbol{w}^*}$, the factor for the Fisher for a single input, which can be computed with d backward passes for each input. Having constructed the sketch, we can use this much lower-dimensional, $T \times N$ representation of the dataset Fisher to extract a low-rank, diagonalized representation $F^{\mathcal{D}}_{\boldsymbol{w}^*} = U_{\text{top}}\mathsf{diag}(\boldsymbol{\lambda}_{\text{top}})U^{\mathsf{T}}_{\text{top}}$ by following the `FixedRankSymmetric` algorithm detailed in Tropp et al. [2017].

5 RELATED WORK

There is a large body of work on characterizing epistemic uncertainty in DNNs. Specific to softmax-based classification models, there has been some effort in using the predictive uncertainty directly as a measure of epistemic uncertainty for OoD detection [Hendrycks and Gimpel, 2017], which can be improved through temperature scaling and input preprocessing [Liang et al., 2018]. Bayesian approaches often employ Monte-Carlo [Neal, 2012, Gal and Ghahramani, 2015] or variational [Graves, 2011, Blundell et al., 2015, Liu and Wang, 2016] methods, but these are not generally applicable to pre-trained networks. In contrast, the Laplace approximation to the Bayesian posterior [MacKay, 1992] is appealing as it can be applied to any pre-trained DNN, yet requires estimating the network curvature.

The most closely-related approaches to ours are the post-training uncertainty methods of Madras et al. [2020] and Ritter et al. [2018]. The main differences between such approaches are (1) how to approximate the curvature (Hessian/Fisher) and (2) how to propagate uncertainty at test time. The true Bayesian posterior is based on the log-likelihood, and a second-order approximation of the log-likelihood would involve the Hessian (the Laplace approximation). However, for DNNs, we often do not optimize to convergence, and thus, the Hessian is not guaranteed to be PSD. Madras et al. [2020] consider only the principal components, but the computation still does not scale well to large datasets. Moreover, stochastic versions of Lanczos algorithm [Lanczos, 1950] or power iteration are difficult to tune. An alternative is to consider the Fisher (or the Gauss–Newton [Botev et al., 2017]) approximation of the full Hessian. For certain cases, such as for the exponential family distributions and piecewise linear networks, the Fisher is the same as the Hessian[5]. An advantage of the Fisher is that it is easier to compute, as it only requires the first-order terms. Further, it is guaranteed to be PSD, and thus is often used in second-order optimization.

As mentioned earlier, Fisher is still intractable to compute, store, and invert for large models, and so, various approximation schemes for that have been proposed. Ritter et al. [2018] use a Kronecker-factored representation of the Fisher, which is easy to store and invert, but it requires certain approximations of the expectations of a Kronecker product, and ignores the cross terms between layers to form a block diagonal structure. We do not enforce this block diagonal structure, and instead use matrix sketching to tractably estimate only the top eigenvectors and eigenvalues of the full Fisher. The KFAC approximation (and its derivatives) are suitable for second-order optimization, when curvature is used to scale the gradient step in the training loop (making the block diagonal approximation enables quick computation of an invertible Fisher). However, in our case, computing the dataset Fisher (and its eigen-decomposition) happens only once, offline, and thus, we can afford to take a slower, more memory-intensive approach.

While proposed for a different purpose, there are similar ideas that have been used for the problem of continual learning, i.e., learning different tasks in a sequential fashion. The challenge there is to represent the information of the previous training data in a compact form and to update the weights in such a way that preserves the previous information as much as possible when training for a new task. Elastic weight consolidation (EWC) of Kirkpatrick et al. [2017] uses the Fisher information matrix to weight each parameter based on its "importance" for the previous task, and uses a regularizer that penalizes changing important parameters more. Farajtabar et al. [2020] proposed orthogonal gradient descent (OGD), which represents the information about the previous data in the form of gradients of the predictions, and then updates the weights in a direction orthogonal to those gradients.

[5] For piecewise linear networks, e.g., those with ReLU activations, the Hessian of $\boldsymbol{\theta}$ with respect to the weights is 0 wherever the network is differentiable, and the nondifferentiable points of such networks form a measure zero set [Singla et al., 2019]

Domain	Training data	OoD data
Wine	Red wine data	White wine data
Rotated MNIST	Rotated '2's	OoD rotations of '2's Rotated '5's
TaxiNet	Runway images Clear weather, 9am	OoD weather OoD time-of-day Diff. runway
Binary MNIST	MNIST '0's and '1's	OoD digits FashionMNIST
MNIST	MNIST (0-5)	OoD digits FashionMNIST
CIFAR10	CIFAR-10 images	SVHN LSUN TinyImageNet

Table 1: Summary of case studies. The first three are regression problems, while the last three are classification problems.

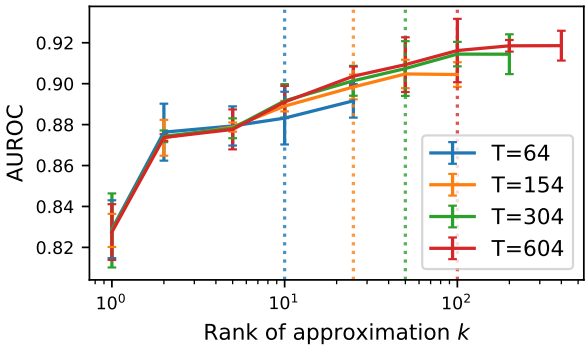

Figure 2: The impact of the sketched approximation on OoD detection (measured by AUROC) for Rotated MNIST. We see that, in general, increasing the rank of the approximation yields a higher AUROC, but with diminishing returns (note the log scale). AUROC is not significantly impacted by the sketch budget T if $k \leq (T-4)/6$, the threshold visualized by the dashed lines.

6 EXPERIMENTAL RESULTS

We are interested in evaluating how *efficiently* SCOD produces uncertainty estimates, and how *useful* these estimates are. We define the utility of the uncertainty estimate in terms of how well it serves to classify atypical inputs from typical ones. Following the literature on OoD detection, we quantify this utility by computing the area under the ROC and precision-recall curves, to produce the AUROC and AUPR metrics, respectively.[6]

Using these metrics, we explore: (1) how choices like the sketch budget T and rank of approximation k impact performance; (2) how performance of SCOD can be improved on large DNNs; and (3) how SCOD compares to baselines on a suite of problem settings, from regression to classification. A summary of these problem settings is provided in table 1, with more details in Appendix C.1.

6.1 CHOOSING THE SKETCH BUDGET AND RANK OF APPROXIMATION

A key aspect of SCOD is using matrix sketching to approximate the dataset Fisher as a low-rank matrix. This presents the practitioner with two key hyperparameters: the memory budget T to allocate for the sketching, and the rank k of the approximation used in online computation. While memory budget T is generally set by hardware constraints, it impacts the quality of low-rank approximation attainable through sketching. Indeed, the sketching procedure produces a rank $2(T-1)/3$ approximation, and theoretical results suggest keeping only the top $(T-4)/6$ eigenvalues and eigenvectors from this approximation [Tropp et al., 2017].

We explore this trade-off empirically on the Rotated MNIST domain. We choose a range of values for the sketching budget T, and sketch the dataset Fisher. For each sketch size, we choose k from a range of values from 1 all the way to the theoretical maximum $2(T-1)/3$, and test the AUROC performance of Unc. Figure 2 shows the results of these experiments. These results show two key trends. First, we see that increasing the rank k improves performance, but with diminishing returns. Second, for a fixed rank k, we see that the performance is insensitive to the sketch budget, especially if $k < (T-4)/6$. For larger k, we start seeing benefits from increasing the sketch budget, consistent with the theory; performance starts to plateau as k increases beyond $(T-4)/6$. Beyond the rank, Unc is also impacted by the scale of the prior ϵ^2. In our experiments, we found performance to be insensitive to this hyperparameter and thus use $\epsilon = 1$ in all the experiments. See Appendix C.3 for more details.

6.2 FURTHER IMPROVING EFFICIENCY FOR LARGE DNNS

These results suggest that the best classification performance is achieved by setting T as high as memory allows, and subsequently choosing $k \geq (T-4)/6$. While sketching enables us to avoid the quadratic dependency on N, the memory footprint of the offline stage of SCOD is still linear, i.e., $O(NT)$. As a result, GPU memory constraints can restrict T, and thus k, substantially for large models. To alleviate this issue, we study how performance is impacted

[6] Code to run experiments and apply SCOD to arbitrary PyTorch models is available at `https://github.com/StanfordASL/SCOD/`.

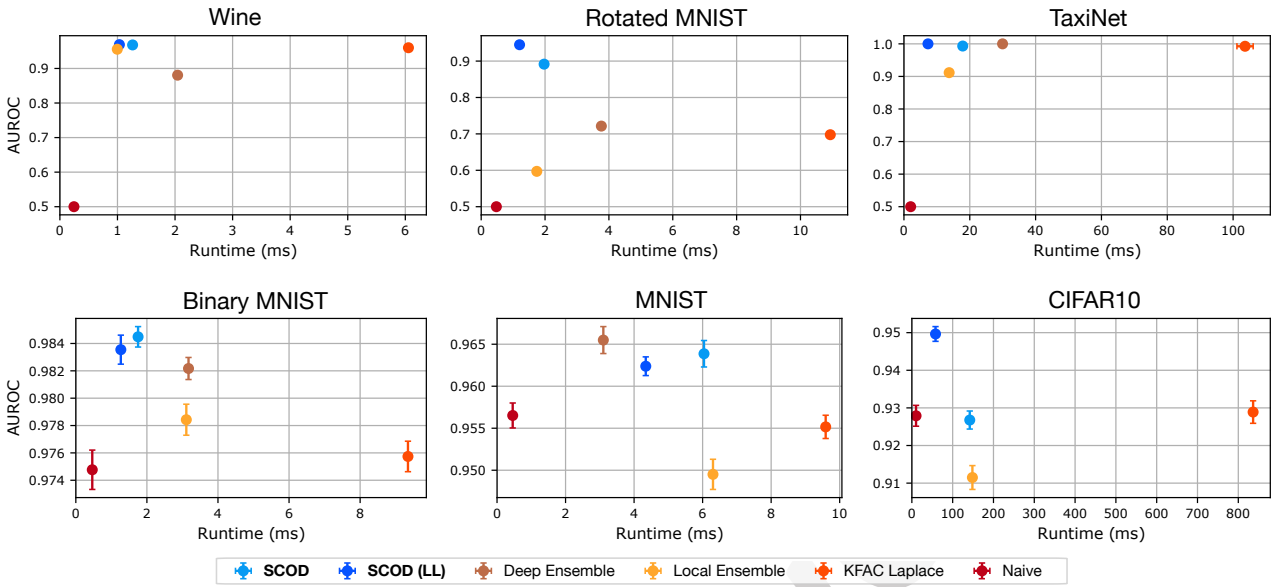

Figure 3: Comparing OoD detection performance and runtime against baselines. We see that SCOD is consistently on a Pareto frontier in terms of the runtime/efficacy tradeoff among post-hoc uncertainty methods (top-left is better). Indeed, SCOD, applied onto a pre-trained model, often matches or exceeds the performance of Deep Ensembles, which is not a post-training approach, and requires retraining multiple models from scratch. Note that the y-axes are independently scaled.

if we simply restrict our analysis to the last few layers of the network, thus lowering the effective value of N. For convolutional networks, the first layers tend to learn generic filter patterns which apply across many domains, while later layers learn task-specific representations [Zeiler and Fergus, 2014]. Thus, it is possible that the curvature on the later-layer parameters is more informative from an OoD detection standpoint.

To test the impact of this, we compare to **SCOD (LL)**, an ablation which applies this analysis only to the last layers of the network. For these experiments, we chose to limit analysis to the last L layers, where we chose L for each network architecture such that a minimum of 2 layers were considered, and at least 1 Conv layer was considered. For the larger models in TaxiNet and CIFAR10, we restricted our analysis to the last 15% of the layers. In these latter two domains, we also increased the sketch budget T and associated rank k up to the capacity of our GPU. Full details of the setup are included in Appendix C.1. Results are included in Figure 3. For an in-depth look into the impact of focusing on the last layer, see Appendix C.4.

6.3 PERFORMANCE RELATIVE TO BASELINES

We compare SCOD against several baselines. First, we compare with the two closely-related approaches, namely, **Local Ensembles** [Madras et al., 2020] and **KFAC Laplace** approximation [Ritter et al., 2018], which use curvature estimation to augment a trained model with uncertainty estimates.

Next, while *not* a method applicable to a pre-trained model, we compare against **Deep Ensembles** [Lakshminarayanan et al., 2017] as a benchmark, as it has shown strong OoD detection performance across regression and classification examples. For this baseline, we retrain $K = 5$ models of the same architecture from different initializations. Both KFAC Laplace and Deep Ensembles output mixture distributions, which we turn into a scalar uncertainty measure by computing the total variance (summed over output dimension) for regression problems, or the predictive entropy for classification, as in [Lakshminarayanan et al., 2017]. Finally, we compare to a **Naive** baseline which produces an uncertainty measure directly from the output of the pre-trained model. For regression models which output only a mean estimate, extracting an uncertainty estimate is not possible, and thus this baseline outputs a constant signal $\mathrm{Unc}(x) = 1$. For classification models, we use the entropy of the output distribution as a measure of uncertainty, as in [Lakshminarayanan et al., 2017].

We compare the performance of these baselines across three regression domains (Wine, Rotated MNIST, and TaxiNet), as well as three classification domains (Binary MNIST, MNIST, and CIFAR10). For each domain, we create a dataset known to be semantically different from the training data. Where possible, we consider OoD data that is realistic — for example, data from white wines as OoD for a model trained on data form red wines; or in TaxiNet, images from a wing-mounted camera in different times of day and different weather conditions for a model trained only on data from the morning with clear weather conditions. Table 1

gives a summary of the domains; more details are provided in Appendix C.1. We also consider using the model's own accuracy to label points as in- or out-of-distribution in Appendix C.5. For all domains except CIFAR10, we train all networks, and then apply the post-training algorithms. For the CIFAR10 domain, we test on a pre-trained DenseNet121 model [Huang et al., 2019, Phan, 2021] to highlight the fact that SCOD can be applied to augment any pre-trained model, independent of training methodology, with uncertainty estimates.

The results are summarized in Figure 3. Overall, we see three key trends. First, SCOD consistently provides the most informative uncertainty measures out of the methods applicable on a pre-trained model. In fact, its AUROC often matches or exceeds that of Deep Ensembles. While KFAC Laplace also matches Ensemble performance in many settings, our approach tends to dominate it on a runtime/AUROC Pareto frontier. Computing uncertainty metrics using KFAC Laplace requires the computationally-intensive repeated evaluations of the DNN with different sampled weights. Furthermore, this sampling process is very sensitive to the regularization hyperparameters. In contrast, our uncertainty metric does not require any sampling, and thus adds very little overhead. Local Ensembles similarly add little overhead, but have a worse AUROC on many domains, especially for large models and datasets, e.g., in the TaxiNet domain, where $M = 50,000$. It is likely that sketching the Fisher yields a better low-rank curvature estimate than using stochastic mini-batching to estimate the top eigenspace of the Hessian.

On regression problems, the output of the network provides no estimate of uncertainty, and, unsurprisingly, the Naive baseline performs poorly. In contrast, on classification problems, the Naive strategy of interpreting the predictive uncertainty as epistemic uncertainty works quite well [Hendrycks and Gimpel, 2017]. Nevertheless, we see that, SCOD is generally able to exceed the Naive performance on these classification domains, and match the performance of Deep Ensembles. Importantly, unlike Deep Ensembles or even KFAC Laplace, we do not directly use the base DNN's output uncertainty as part of our score, and only consider how weight perturbations may change the output. Given that the output uncertainty is a strong baseline for softmax classification problems, connecting these ideas is an interesting avenue for future work.

In general, we see that restricting analysis to the last layers often leads to better AUROC performance as well as faster runtime. This is particularly evident on CIFAR10, where without restricting analysis to the last layers, SCOD performed worse than the Naive baseline (although still achieving AUROC > 0.9). This may be a consequence of the phenomenon in which the first layers tend to represent generic features that are equally suited to all natural images. Therefore, the curvature of the later layers may be more useful for OoD detection. This is supported by a case study on the CIFAR10 model, with results in Appendix C.4.

7 CONCLUSION

We presented *Sketching Curvature for OoD Detection (SCOD)*, a scalable, architecture-agnostic framework for equipping any pre-trained DNN with a task-relevant epistemic uncertainty estimate. Through extensive experiments, we demonstrated that the proposed method achieves comparable or better OoD detection performance with a lower computational burden relative to existing approaches.

There are several important avenues for future work, in addition to what was mentioned earlier. First, while the straightforward single-pass, sketch-based eigenvalue decomposition algorithm we employ was effective in our experiments, performance might be improved using more involved sketching-based algorithms for low-rank matrix approximation [Yu et al., 2017]. Indeed, multiple passes over the data would allow reaping benefits of power iteration and improving approximation quality. The single-pass sketching approach has its own advantages, however — since the sketch is built incrementally, our framework can be potentially extended to continual learning applications. Furthermore, within the context of autonomous systems, another important direction is to tailor OoD detection and uncertainty estimation for downstream control task, by quantifying the impact of posterior parameter uncertainty on the overall system behavior, rather than just on DNN predictions.

Acknowledgements

A. Sharma and M. Pavone were supported in part by DARPA under the Assured Autonomy program. Additionally, the NASA University Leadership initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity. N. Azizan was supported by MIT. The authors wish to thank Robin Brown and Edward Schmerling for helpful input and discussions during the development of these ideas.

References

Navid Azizan, Sahin Lale, and Babak Hassibi. Stochastic mirror descent on overparameterized nonlinear models: Convergence, implicit regularization, and generalization. *arXiv preprint arXiv:1906.03830*, 2019.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*. Pmlr, 2017.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*. Pmlr, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, 2011.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations*, 2017.

Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Weinberger. Convolutional networks with dense connectivity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13), 2017.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 2017.

Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. 1950.

Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1VGkIxRZ.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in Neural Information Processing Systems*, 2016.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3), 1992.

David Madras, James Atwood, and Alexander D'Amour. Detecting extrapolation with local ensembles. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJl6bANtwH.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*. Pmlr, 2015.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Huy Phan. huyvnphan/pytorch_cifar10, January 2021. URL https://doi.org/10.5281/zenodo.4431043.

Hippolyt Ritter, Aleksandar Botev, and D. Barber. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *International Conference on Machine Learning*. Pmlr, 2019.

Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1454–1485, 2017. URL http://arxiv.org/abs/1609.00048.

Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3), 2008.

Wenjian Yu, Yu Gu, Jian Li, Shenghua Liu, and Yaohang Li. Single-pass pca of large high-dimensional data. *arXiv preprint arXiv:1704.07669*, 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 2014.