Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness

Javier Sáez

March 22, 2023

University of Granada
Visual Information Processing Group

Index

Introduction

Introduction

- Real world applications **need** efficient methods that reliably quantify a deep neural network's (DNN) predictive uncertainty.

Introduction

- Real world applications **need** efficient methods that reliably quantify a deep neural network's (DNN) predictive uncertainty.
- Examples:
  - Object detection in autonomous driving
  - Ad click prediction in websites
  - Understanding in a conversational system (ChatGPT)
  - Domain-specific chatbot services (weather)

Introduction

- **Problem:** The performance of the models can be arbitrarily bad when the new input is far from the training set.

Introduction

- **Problem:** The performance of the models can be arbitrarily bad when the new input is far from the training set.
- **Need:** Methods that return a uniform distribution (max entropy) over output labels if the input is far.

Introduction

- **Problem:** The performance of the models can be arbitrarily bad when the new input is far from the training set.
- **Need:** Methods that return a uniform distribution (max entropy) over output labels if the input is far.

GPs!

GPs need a previous feature extraction. We would like the DNN to be *distance aware*.

Notation

- Consider the data distribution $p^\star(y \mid \mathbf{x})$, where $y \in \{1, \dots, K\}$ and $x \in \mathcal{X} \subset \mathbb{R}^d$, where $\mathcal{X}$ is the data manifold with a suitable metric $\|\cdot\|_X$.

Notation

- Consider the data distribution $p^\star(y \mid \mathbf{x})$, where $y \in \{1, \ldots, K\}$ and $x \in \mathcal{X} \subset \mathbb{R}^d$, where $\mathcal{X}$ is the data manifold with a suitable metric $\|\cdot\|_X$.
- In practise, data is collected from a subset $\mathcal{X}_{IND} \subset \mathcal{X}$. As a result,

$$\begin{aligned} p^\star(y \mid \mathbf{x}) &= p^\star(y, \mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x}) + p^\star(y, \mathbf{x} \notin \mathcal{X}_{IND} \mid \mathbf{x}) \\ &= p^\star(y \mid \mathbf{x}, \mathbf{x} \in \mathcal{X}_{IND}) \cdot p^\star(\mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x}) \\ &\quad + p^\star(y \mid \mathbf{x}, \mathbf{x} \notin \mathcal{X}_{IND}) \cdot p^\star(\mathbf{x} \notin \mathcal{X}_{IND} \mid \mathbf{x}) \end{aligned}$$

  Models learn $p^\star(y \mid \mathbf{x}, \mathbf{x} \in \mathcal{X}_{IND})$.
  However, $p^\star(y \mid \mathbf{x}, \mathbf{x} \notin \mathcal{X}_{IND})$ can be very different.

Notation

- Consider the data distribution $p^\star(y \mid \mathbf{x})$, where $y \in \{1, \ldots, K\}$ and $x \in \mathcal{X} \subset \mathbb{R}^d$, where $\mathcal{X}$ is the data manifold with a suitable metric $\|\cdot\|_X$.
- In practise, data is collected from a subset $\mathcal{X}_{IND} \subset \mathcal{X}$. As a result,

$$\begin{aligned} p^\star(y \mid \mathbf{x}) &= p^\star(y, \mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x}) + p^\star(y, \mathbf{x} \notin \mathcal{X}_{IND} \mid \mathbf{x}) \\ &= p^\star(y \mid \mathbf{x}, \mathbf{x} \in \mathcal{X}_{IND}) \cdot p^\star(\mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x}) \\ &\quad + p^\star(y \mid \mathbf{x}, \mathbf{x} \notin \mathcal{X}_{IND}) \cdot p^\star(\mathbf{x} \notin \mathcal{X}_{IND} \mid \mathbf{x}) \end{aligned}$$

  Models learn $p^\star(y \mid \mathbf{x}, \mathbf{x} \in \mathcal{X}_{IND})$.
  However, $p^\star(y \mid \mathbf{x}, \mathbf{x} \notin \mathcal{X}_{IND})$ can be very different.
- In testing, models use a predictive $p(y \mid \mathbf{x})$ for any $\mathbf{x} \in \mathcal{X} = \mathcal{X}_{IND} \cup \mathcal{X}_{OOD}$. $\mathcal{X}_{OOD}$ can be very different even in a certain topic (user asking for earthquakes in a weather-themed chatbot)!

Towards the loss function

If the model predicts class y with confidence $\hat{p}$, we would like the model to be correct $\hat{p}$ of the times. In that case, the model is said to be **callibrated**. Formally, the model is calibrated if

$$\mathbb{P}(\hat{y} = y_{true} \mid \hat{p} = p) = p$$

We can take the expectation in the possible predicted confidences $\hat{p}$, leading to the Expected Calibration Error (ECE) [Guo et al., 2017]:

$$ECE = \mathbb{E}_{\hat{p}}\left[|P(\hat{y} = y_{true} \mid \hat{p} = p) - p|\right]$$

However, this loss function is not suitable for this case, since it is not uniquely minimized.

Towards the loss function

Definition

A scoring rule[1] is any extended real-valued function $\mathbf{S} : \mathcal{P} \times \Omega \to \mathbb{R}$ such that $\mathbf{S}(P, \cdot)$ is $\mathcal{P}$-quasi-integrable for all $P \in \mathcal{P}$.

$\mathbf{S}(P, y)$ represents the loss or penalty when the prediction $P \in \mathcal{P}$ is issued and the observation $y \in \Omega$ materializes.

[1] Wikipedia's link

Example: Brier's Score

An example of a strictly proper scoring rule is the **Brier's score**, defined as:

$$B(p, y) = \sum_{j=1}^{C} (y_j - p_j)^2,$$

where

- $y_j = 1$ if the the input belongs to class j-th and 0 otherwise,
- C is the number of classes and
- p_j is the probability assigned to class j.

Towards the loss function

If the label y is sampled from a distribution Q, the expected score under $Q \in \mathcal{P}$ can be written as:

$$\mathbf{S}(P, Q) = \int \mathbf{s}(P, y) \; dQ(y).$$

Towards the loss function

If the label y is sampled from a distribution Q, the expected score under $Q \in \mathcal{P}$ can be written as:

$$\mathbf{S}(P, Q) = \int \mathbf{s}(P, y)\ dQ(y).$$

We can define a **divergence** of P from Q as:

$$d(P, Q) \triangleq \mathbf{S}(P, Q) - \mathbf{S}(Q, Q).$$

Towards the loss function

Definition

A scoring rule **S** is proper *relative to* $\mathcal{P}$ if

$$d(P, Q) \geq 0, \quad \forall P, Q \in \mathcal{P}.$$

Also, it is **strictly proper** [Gneiting and Raftery, 2007] if the above equation holds with equality if, and only if, $P = Q$.

Towards the loss function

Consider the vector of calibrated probabilities $\mathbf{C} = (C_1, \ldots, C_k)$ and let P be the output probabilities of a model. Using $\mathbf{C}$, the expected loss according a P.S.R. can be expressed [Kull and Flach, 2015] as:

$$\mathbb{E}\left[d(P, Y)\right] = \mathbb{E}[d(P, \mathbf{C})] + \mathbb{E}[d(\mathbf{C}, Y)]$$

Consequence: Each PSR is an upper bound of the calibration error, so minimizing a PSR implies minimizing the calibration error.

Towards the loss function

The problem of uncertainty quantification turns into constructing an optimal p to minimize the expected risk over $\mathcal{X}$:

$$\inf_{p \in \mathcal{P}} S(p, p^*) = \inf_{p \in \mathcal{P}} \mathop{\mathbb{E}}_{\mathbf{x} \in \mathcal{X}} \left[s(p, p^* | \mathbf{x}) \right]. \tag{1}$$

Towards the loss function

The problem of uncertainty quantification turns into constructing an optimal p to minimize the expected risk over $\mathcal{X}$:

$$\inf_{p \in \mathcal{P}} S(p, p^*) = \inf_{p \in \mathcal{P}} \mathop{\mathbb{E}}_{\mathbf{x} \in \mathcal{X}} \left[s(p, p^* | \mathbf{x}) \right]. \tag{1}$$

However, this is not possible to solve since data only comes from $\mathcal{X}_{IND}$, OOD distribution is never learned, so generalization is not guaranteed. We then take a *Minimax* strategy (minimize the worst-case risk):

Towards the loss function

The problem of uncertainty quantification turns into constructing an optimal p to minimize the expected risk over $\mathcal{X}$:

$$\inf_{p \in \mathcal{P}} S(p, p^*) = \inf_{p \in \mathcal{P}} \mathop{\mathbb{E}}_{\mathbf{x} \in \mathcal{X}} \left[s(p, p^* | \mathbf{x}) \right]. \tag{1}$$

However, this is not possible to solve since data only comes from $\mathcal{X}_{IND}$, OOD distribution is never learned, so generalization is not guaranteed. We then take a *Minimax* strategy (minimize the worst-case risk):

$$\inf_{p \in \mathcal{P}} \left[\sup_{p^* \in \mathcal{P}^*} S(p, p^*) \right]. \tag{2}$$

Solution using Brier's Score

Under the classification task and using Brier's score, the solution to Eq. (2) is:

$$p(y|\mathbf{x}) = p(y|\mathbf{x}, \mathbf{x} \in \mathcal{X}_{IND}) * p^*(\mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x}) + p_{\texttt{uniform}}(y|\mathbf{x}, \mathbf{x} \notin \mathcal{X}_{IND}) * p^*(\mathbf{x} \notin \mathcal{X}_{IND} \mid \mathbf{x}).$$

Advantages:

- Intuitive!
- Verifies that there is the unique optimal solution to the problem in Eq. (2)

Disadvantage:

- We have to assume that the model can quantify $p^\star(\mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x})$ well.

Input Distance Awareness

We need a notion of distance between a new example and $\mathcal{X}_{IND}$!

Input Distance Awareness

We need a notion of distance between a new example and $\mathcal{X}_{IND}$!

Definition (Input Distance Awareness)

Consider a predictive distribution $p(y \mid \mathbf{x})$ trained on $\mathcal{X}_{IND}$. We say that it is **input distance aware** if:
Exists $u(\mathbf{x})$ a summary statistic of $p(y \mid \mathbf{x})$ such that:

1. quantifies model uncertainty (eg: predictive variance) and
2. reflects the distance between $\mathbf{x}$ and training data w.r.t. $\|\cdot\|_{\mathcal{X}}$

$$u(\mathbf{x}) = v(d(\mathbf{x}, \mathcal{X}_{IND})),$$

where v is a monotonic function and

$$d(\mathbf{x}, \mathcal{X}_{IND}) = \mathbb{E}_{x' \sim \mathcal{X}_{IND}} \|\mathbf{x} - \mathbf{x}'\|_X^2$$

Example: Gaussian Processes

GPs with RBF kernel are input distance aware models:

- Predictive $p(y \mid \mathbf{x}) = softmax(g(\mathbf{x}))$
- Exists $u(\mathbf{x}^\star) = var(g(\mathbf{x}^\star)) = 1 - \mathbf{k}^{\star^T}\mathbf{V}\mathbf{k}^\star$, with

$$\mathbf{k}_i^\star = \exp\left(-\gamma \|\mathbf{x}^\star - \mathbf{x}_i\|_X^2\right)$$

Example: Gaussian Processes

GPs with RBF kernel are input distance aware models:

- Predictive $p(y \mid \mathbf{x}) = \mathit{softmax}(g(\mathbf{x}))$
- Exists $u(\mathbf{x}^\star) = \mathit{var}(g(\mathbf{x}^\star)) = 1 - \mathbf{k}^{\star T}\mathbf{V}\mathbf{k}^\star$, with

$$\mathbf{k}_i^\star = \exp\left(-\gamma \|\mathbf{x}^\star - \mathbf{x}_i\|_X^2\right)$$

In a typical DL model, the confidence in a class is computed in a dense layer as:

$$\text{logit}_k(\mathbf{x}) = h(\mathbf{x})^\top \beta_k.$$

The model decission is not based on its distance to training data $\mathcal{X}_{IND}$, but on its distance to the decission boundaries.

Conditions for IDA in Deep Learning

Consider a DL model $\mathit{logit}(\mathbf{x}) = g \circ h(\mathbf{x})$ where

- h creates a representation of $\mathbf{x}$ and
- g maps $h(\mathbf{x})$ to the label space.

To make this model distance aware we need:

Conditions for IDA in Deep Learning

Consider a DL model $logit(\mathbf{x}) = g \circ h(\mathbf{x})$ where

- h creates a representation of $\mathbf{x}$ and
- g maps $h(\mathbf{x})$ to the label space.

To make this model distance aware we need:

1. Make the output layer g distance aware, so it outputs an uncertainty metric reflecting distance in the representation space

Conditions for IDA in Deep Learning

Consider a DL model $logit(\mathbf{x}) = g \circ h(\mathbf{x})$ where

- h creates a representation of $\mathbf{x}$ and
- g maps $h(\mathbf{x})$ to the label space.

To make this model distance aware we need:

1. Make the output layer g distance aware, so it outputs an uncertainty metric reflecting distance in the representation space
2. Make the hidden mapping h *distance preserving*.

Conditions for IDA in Deep Learning

Consider a DL model $logit(\mathbf{x}) = g \circ h(\mathbf{x})$ where

- h creates a representation of $\mathbf{x}$ and
- g maps $h(\mathbf{x})$ to the label space.

To make this model distance aware we need:

1. Make the output layer g distance aware, so it outputs an uncertainty metric reflecting distance in the representation space
2. Make the hidden mapping h *distance preserving*. Mathematically, that is fullfilling the bi-Lipschitz condition:

$$\underbrace{L_1 \, \|\mathbf{x}_1 - \mathbf{x}_2\|_X}_{\text{not unnecessarily invariant}} \leq \|h(\mathbf{x}_1) - h(\mathbf{x}_2)\|_H \leq \underbrace{L_2 \, \|\mathbf{x}_1 - \mathbf{x}_2\|_X}_{\text{robustness}},$$

where $0 < L_1 < 1 < L_2$. This condition encourages h to be approximately isometric.

SNGP: Spectral-normalized Neural Gaussian Process

Visual description

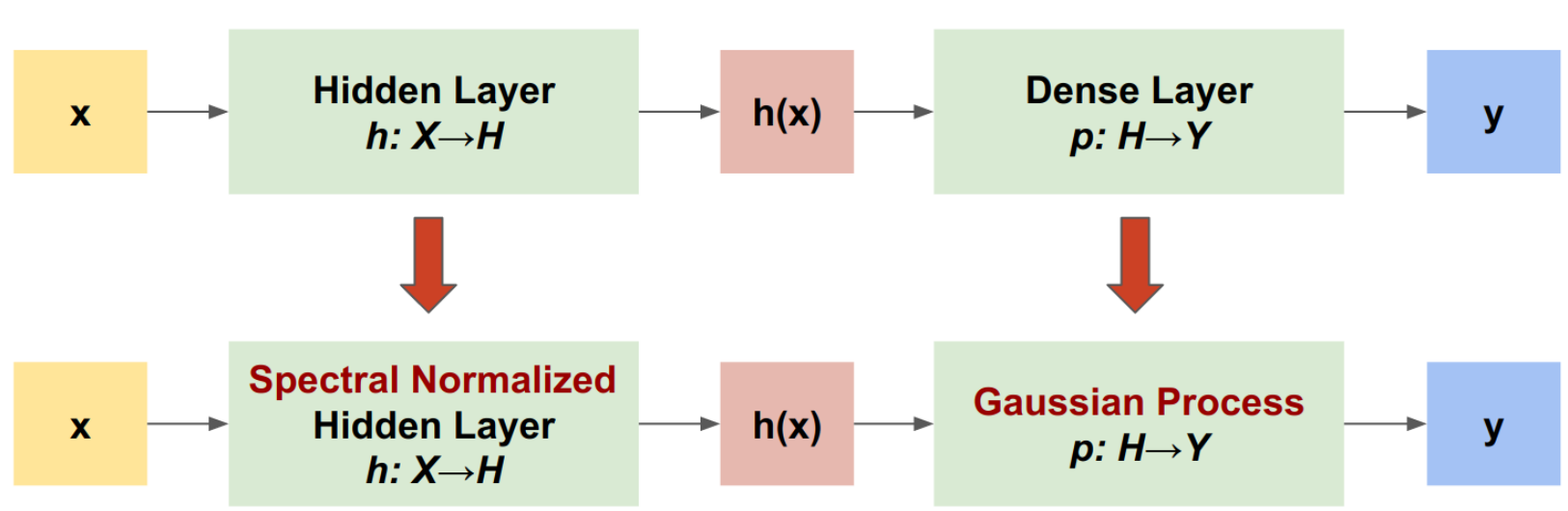


Figure 1: Visual description from the provided tutorial.

Distance-aware Output Layer

To make the output layer g distance aware, the model replaces the typical dense layer with a **GP with RBF Kernel**.

Specifically, given $\{\mathbf{x}_i, y_i\}_{i=1}^N$, we place the usual prior to the output layer $g_{N\times 1} = [g(h_1), \ldots, g(h_N)]^T$:

$$g_{N\times 1} \sim GP(\mathbf{0}, \mathbf{K}_{NN}), \quad \mathbf{K}_{i,j} = \exp(-\|h_i - h_j\|_2^2/2).$$

Distance-aware Output Layer

To make the output layer g distance aware, the model replaces the typical dense layer with a **GP with RBF Kernel**.

Specifically, given $\{\mathbf{x}_i, y_i\}_{i=1}^N$, we place the usual prior to the output layer $g_{N\times 1} = [g(h_1), \ldots, g(h_N)]^T$:

$$g_{N\times 1} \sim GP(\mathbf{0}, \mathbf{K}_{NN}), \quad \mathbf{K}_{i,j} = \exp(-\|h_i - h_j\|_2^2/2).$$

As per usual with GPs, computing the exact posterior distribution is intractable.

Distance-aware Output Layer: Approximating GP Prior

The GP prior is approximated using **Random Fourier Features** (RFF)

$$\mathbf{g}_{N\times 1} \sim GP(\mathbf{0}, \mathbf{\Phi}\mathbf{\Phi}^\top), \qquad \mathbf{\Phi}_{i,D_L\times 1} = \sqrt{2/D_L} * \cos(-\mathbf{W}_L h_i + \mathbf{b}_L),$$

where $w_{ij} \sim \mathcal{N}(0,1)$ and $b_k \sim Uniform(0, 2\pi)$.

Distance-aware Output Layer: Approximating GP Prior

The GP prior is approximated using **Random Fourier Features** (RFF)

$$\mathbf{g}_{N\times 1} \sim GP(\mathbf{0}, \mathbf{\Phi}\mathbf{\Phi}^\top), \qquad \mathbf{\Phi}_{i,D_L\times 1} = \sqrt{2/D_L} * \cos(-\mathbf{W}_L h_i + \mathbf{b}_L),$$

where $w_{ij} \sim \mathcal{N}(0,1)$ and $b_k \sim Uniform(0, 2\pi)$.
As a result, for the k^{th} logit, this approximation can be written as a NN layer:

$$g_k(h_i) = \sqrt{2/D_L} * \cos(-\mathbf{W}_L h_i + \mathbf{b}_L)^\top \boldsymbol{\beta}_k,$$

where $\mathbf{W}$ is fixed and the prior of β_k is$\boldsymbol{\beta}^k_{D_L\times 1} \sim \mathcal{N}(0, \mathbf{I}_{D_L\times D_L})$.

Distance-aware Output Layer: Posterior Approximation

Consequence: We have reduced the infinite-dimensional GP to a Bayesian linear model, so we can apply one of the known posterior approximation methods.

Distance-aware Output Layer: Posterior Approximation

Consequence: We have reduced the infinite-dimensional GP to a Bayesian linear model, so we can apply one of the known posterior approximation methods.

Definition (Laplace approximation)

Laplace method approximates the posterior distribution $p(\beta \mid \mathcal{D})$ using a Gaussian distribution centered at the MAP estimate $\hat{\beta} = \arg\max_\beta p(\beta \mid \mathcal{D})$ and covariance matrix the inverse of the Hessian of the log posterior likelihood evaluated at $\hat{\beta}$, that is:

$$p(\beta_k \mid \mathcal{D}) \approx \mathcal{N}(\hat{\beta}_k, \hat{\mathbf{H}}_k^{-1}),$$

with $\hat{\mathbf{H}}_{k,(i,j)} = \frac{\partial^2}{\partial \beta_i \partial \beta_j} \log\, p(\beta_k|\mathcal{D})|_{\beta_k=\hat{\beta}_k}$

Distance-aware Output Layer: Results

The GP posterior can be learned:

- scalably,
- in closed form,
- with minimal modification to the training of a deterministic DNN,
- by the Bernstein-von-Mises theorem [Dehaene, 2019], the Laplace approximation ot the RFF posterior is asymptotically exact.

Distance-preserving Hidden Mapping

The last goal is to ensure that the hidden mapping h is *distance preserving* so that the distance in the hidden space $\|h(\mathbf{x}) - h(\mathbf{x}')\|_H$ has a meaningful correspondence to the distance in the input space $\|\mathbf{x} - \mathbf{x}'\|_X$.

Distance-preserving Hidden Mapping

The last goal is to ensure that the hidden mapping h is *distance preserving* so that the distance in the hidden space $\|h(\mathbf{x}) - h(\mathbf{x}')\|_H$ has a meaningful correspondence to the distance in the input space $\|\mathbf{x} - \mathbf{x}'\|_X$.

Note (Assumption)

The used models will be residual DL models (ResNets, Transformers), composed of residual blocks

$$h(\mathbf{x}) = h_{L-1} \circ \cdots \circ h_1(\mathbf{x}), \quad h_l(\mathbf{x}) = \mathbf{x} + f_l(\mathbf{x}). \tag{3}$$

Distance-preserving Hidden Mapping

Proposition

Let $h : \mathcal{X} \to \mathcal{H}$ be a hidden mapping with residual architecture (3). If for $0 < \alpha \leq 1$, all f_l are $\alpha-$Lipschitz, that is:

$$\|f_l(\mathbf{x}) - f_l(\mathbf{x}')\|_H \leq \alpha \|\mathbf{x} - \mathbf{x}'\|_X \text{ for all } (\mathbf{x}, \mathbf{x}') \in \mathcal{X},$$

then:

$$L_1 \cdot \|\mathbf{x} - \mathbf{x}'\|_X \leq \|h(\mathbf{x}) - h(\mathbf{x}')\|_H \leq L_2 \cdot \|\mathbf{x} - \mathbf{x}'\|_X,$$

where $L_1 = (1 - \alpha)^{L-1}$ and $L_2 = (1 + \alpha)^{L-1}$.

This result implies that if the previous conditions are fulfilled, h is distance preserving.

Distance-preserving Hidden Mapping

If $f_l(\mathbf{x}) = \sigma(\mathbf{W}_l\mathbf{x} + \mathbf{b}_l)$ is the residual block, since

$$\|f_l\|_{Lip} \leq \|\mathbf{W}_l\mathbf{x} + \mathbf{b}_l\|_{Lip} \leq \|\mathbf{W}_l\|_2,$$

it is sufficient to have $\|\mathbf{W}_l\|_2 \leq 1$ to ensure h is distance preserving.

Distance-preserving Hidden Mapping

If $f_l(\mathbf{x}) = \sigma(\mathbf{W}_l\mathbf{x} + \mathbf{b}_l)$ is the residual block, since

$$\|f_l\|_{Lip} \leq \|\mathbf{W}_l\mathbf{x} + \mathbf{b}_l\|_{Lip} \leq \|\mathbf{W}_l\|_2,$$

it is sufficient to have $\|\mathbf{W}_l\|_2 \leq 1$ to ensure h is distance preserving.

Definition (Spectral Normalization)

Let $\lambda = \|\mathbf{W}_l\|_2$ be the spectral norm of $\mathbf{W}_l$. Then, the spectral normalization is performed as:

$$\mathbf{W}_l = \begin{cases} c \cdot \mathbf{W}_l/\lambda & \text{if } c < \lambda \\ \mathbf{W}_l & \text{otherwise} \end{cases} \tag{4}$$

where $c > 0$ is a hyperparameter used as a threshold.

The hyperparameter is useful since it helps with the regularization of the model.

Method summary: Prediction

Algorithm 1 SNGP Prediction

1: **Input:** Testing example $\mathbf{x}$.

2: Compute Feature:

$$\mathbf{\Phi}_{D_L \times 1} = \sqrt{2/D_L} * \cos(\mathbf{W}_L h(\mathbf{x}) + \mathbf{b}_L),$$

3: Compute Posterior Mean:

$$\text{logit}_k(\mathbf{x}) = \Phi^\top \boldsymbol{\beta}_k$$

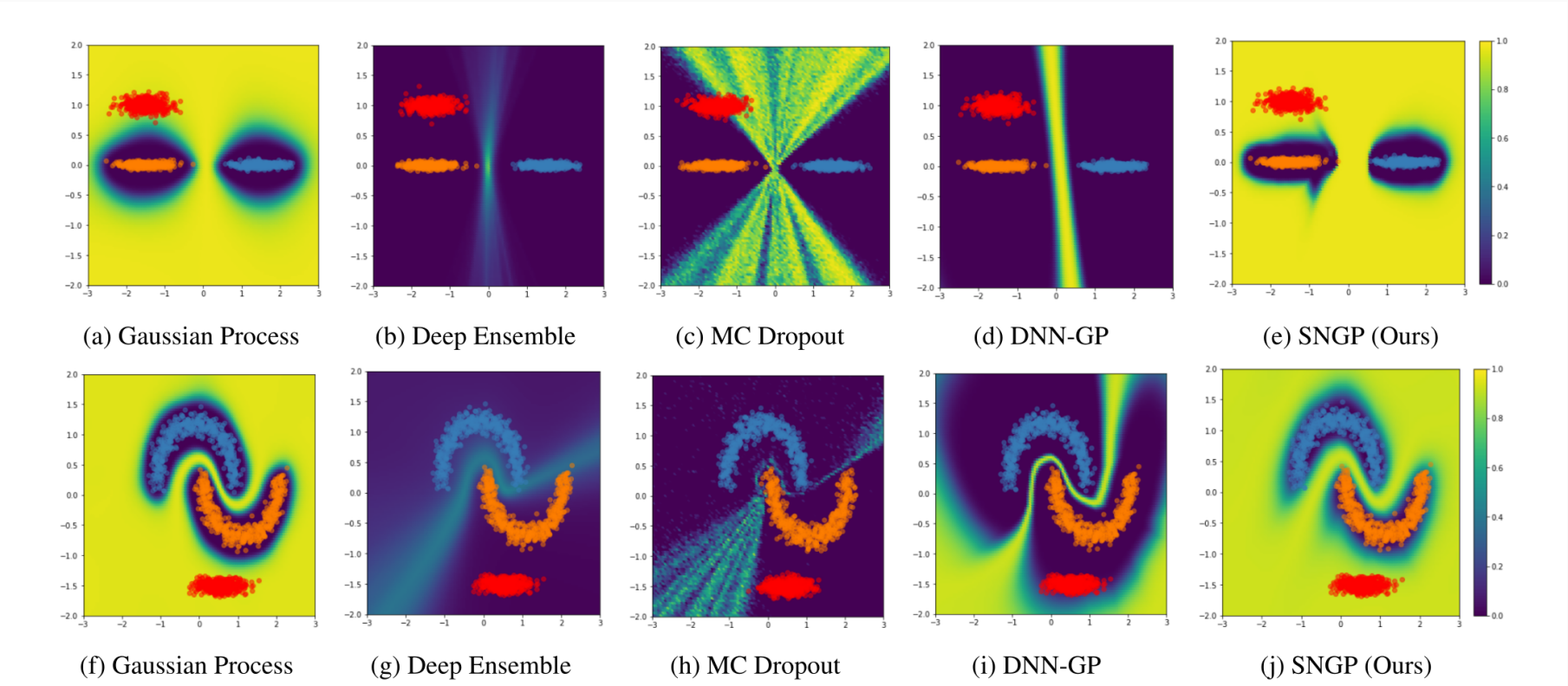
4: Compute Posterior Variance:

$$\text{var}_k(\mathbf{x}) = \Phi^\top \hat{\Sigma}_k \Phi.$$

5: Compute Predictive Distribution:

$$p(y|\mathbf{x}) = \int_{m \sim N(\text{logit}(\mathbf{x}), \text{var}(\mathbf{x}))} \text{softmax}(m)$$

Results



(a) Gaussian Process (b) Deep Ensemble (c) MC Dropout (d) DNN-GP (e) SNGP (Ours)

(f) Gaussian Process (g) Deep Ensemble (h) MC Dropout (i) DNN-GP (j) SNGP (Ours)

Results: OOD metrics

To evaluate the model performance in OOD detection in this task, the **model's uncertainty estimate** is used as a predictive score for OOD classification.

$$u(\mathbf{x}) = \frac{K}{K + \sum_{i=1}^{K} \exp(\text{logit}_i(h(\mathbf{x}))}. \tag{5}$$

Results: OOD Evaluation

1. Generate the dataset adequately: merge two datasets (in-distribution and OOD) in one and add a new label (in/out distribution) to each item.
2. Compute the logits given by the linear model representing the GP.
3. Then, using this logits, we compute an uncertainty metric $u(\mathbf{x})$ for each datapoint as in Equation (5).
4. Create a **new binary classification problem**: OOD class as the positive class (label 1), and we take $u(\mathbf{x})$ to be the *model's* probability of this class.
5. We compute the metrics using this probabilities and the created labels.

References

Jeremiah Zhe Liu, Shreyas Padhy, Jie Ren, Zi Lin, Yeming Wen, Ghassen Jerfel, Zack Nado, Jasper Snoek, Dustin Tran, and Balaji Lakshminarayanan. A simple approach to improve single-model deep uncertainty via distance-awareness, 2022. URL https://arxiv.org/abs/2205.00403.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks, 2017. URL https://arxiv.org/abs/1706.04599.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 2007. URL https://doi.org/10.1198/016214506000001437.

Meelis Kull and Peter Flach. Novel decompositions of proper scoring rules for classification: Score adjustment as precursor to calibration. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I 15*, pages 68–85. Springer, 2015.

Guillaume P. Dehaene. A deterministic and computable bernstein-von mises theorem, 2019. URL https://arxiv.org/abs/1904.02505.

Extra slides

Definition of quasi-integrable

Consider

- a sample space Ω,
- a σ-algebra $\mathcal{A}$ of subsets of Ω and
- a convex class $\mathcal{F}$ of probability measures on $(\Omega, \mathcal{A})$.

Definition

A function defined on Ω and taking values in the extended real line, $\overline{\mathbb{R}} = [-\infty, \infty]$, is $\mathcal{F}$-quasi-integrable if it is measurable with respect to $\mathcal{A}$ and is quasi-integrable with respect to all $F \in \mathcal{F}$.

Method summary: Training

Algorithm 2 SNGP Training

1: **Input:**
Minibatches $\{D_i\}_{i=1}^{N}$ for $D_i = \{y_m, \mathbf{x}_m\}_{m=1}^{M}$.

2: **Initialize:**

$$\hat{\Sigma} = \mathbf{I}, \mathbf{W}_L \overset{iid}{\sim} N(0,1), \mathbf{b}_L \overset{iid}{\sim} U(0, 2\pi)$$

3: **for** train_step $= 1$ **to** max_step **do**
4: SGD update $\left\{\beta, \{\mathbf{W}_l\}_{l=1}^{L-1}, \{\mathbf{b}_l\}_{l=1}^{L-1}\right\}$
5: Spectral Normalization $\{\mathbf{W}_l\}_{l=1}^{L-1}$ (4).
6: **if** final_epoch **then**
7: Update precision matrix $\{\hat{\Sigma}_k^{-1}\}_{k=1}^{K}$ (6).
8: **end if**
9: **end for**
10: Compute posterior covariance $\hat{\Sigma}_k = inv(\hat{\Sigma}_k^{-1})$.

Implementation: Posterior precision matrix

Under the linear-model formulation of the RFF posterior, the precision matrix has the following expression:

$$\hat{\Sigma}_k^{-1} = \mathbf{I} + \sum_{i=1}^{N} \hat{p}_{i,k}(1 - \hat{p}_{i,k})\Phi_i \Phi_i^T, \tag{6}$$

where $\hat{p}_{i,k}$ is the model prediction $softmax(\hat{g}_i)$ under the MAP estimates $\hat{\beta}$.

Implementation: Posterior precision matrix

Under the linear-model formulation of the RFF posterior, the precision matrix has the following expression:

$$\hat{\Sigma}_k^{-1} = \mathbf{I} + \sum_{i=1}^{N} \hat{p}_{i,k}(1 - \hat{p}_{i,k})\Phi_i \Phi_i^T, \tag{6}$$

where $\hat{p}_{i,k}$ is the model prediction $softmax(\hat{g}_i)$ under the MAP estimates $\hat{\beta}$.

When using minibatches of size M, the posterior mean $\hat{\beta}$ is updating via SGD with respect to the log posterior $-\log p(\beta \mid \mathcal{D})$ and the posterior precision matrix is *cheaply* updated as

$$\hat{\Sigma}_{k,t}^{-1} = (1 - m) * \hat{\Sigma}_{k,t-1}^{-1} + m * \sum_{i=1}^{M} \hat{p}_{i,k}(1 - \hat{p}_{i,k})\Phi_i \Phi_i^\top,$$

where m is a small scaling coefficient. This computation is only performed **once at the final epoch**.

Implementation: Spectral Normalization

The spectral norm is approximated by using the **power iteration** method.

Thank you for your attention