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

[Liu, Padhy, Ren, Lin, Wen, Jerfel, Nado, Snoek, Tran, and Lakshminarayanan, 2022]

Javier Sáez

March 22, 2023

University of Granada
Visual Information Processing Group

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.

Model: "I am 90% sure that is a dog"

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 (same class probability for all classes) over output labels if the input is far.

Example

Suppose that in our data we have images of cats and dogs.

$p(dog) = 1$
$p(cat) = 0$
$OOD = False$

$p(dog) = 0$
$p(cat) = 1$
$OOD = False$

Example

Suppose that in our data we have images of cats and dogs.

$p(dog) = 1$
$p(cat) = 0$
$OOD = False$

$p(dog) = 0$
$p(cat) = 1$
$OOD = False$

$p(dog) = \frac{1}{2}$
$p(cat) = \frac{1}{2}$
$OOD = True$

Notation

- Consider the data distribution $p^\star(y \mid \mathbf{x})$, where $y \in \{1, \ldots, K\}$ and $x \in \mathcal{X} \subset \mathbb{R}^d$.

Notation

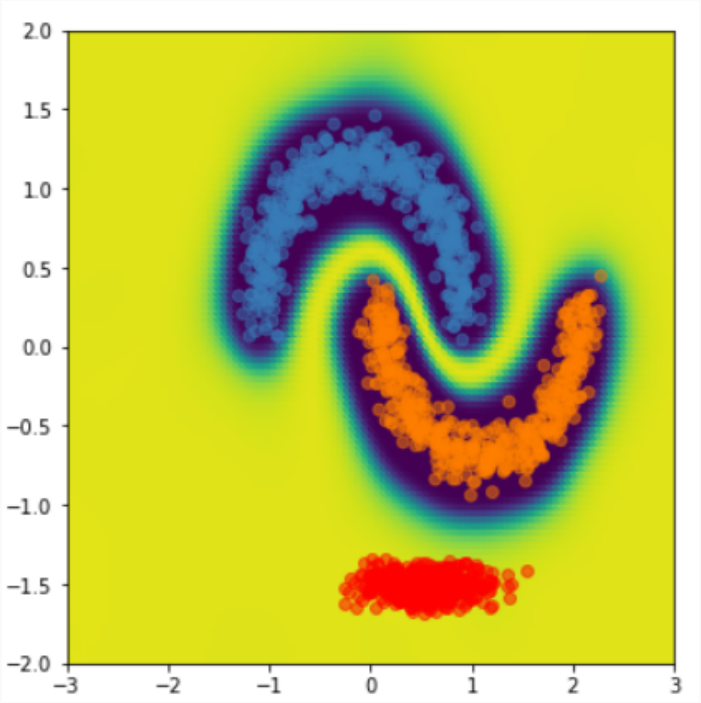
- Consider the data distribution $p^\star(y \mid \mathbf{x})$, where $y \in \{1, \ldots, K\}$ and $x \in \mathcal{X} \subset \mathbb{R}^d$.
- In practise, data is collected from a subset $\mathcal{X}_{IND} \subset \mathcal{X}$. As a result,

$$p^\star(y \mid \mathbf{x}) = \overbrace{p^\star(y \mid \mathbf{x}, \mathbf{x} \in \mathcal{X}_{IND}) \cdot p^\star(\mathbf{x} \in \mathcal{X}_{IND} \mid \mathbf{x})}^{\mathbf{x} \text{ in distribution}} + \underbrace{p^\star(y \mid \mathbf{x}, \mathbf{x} \notin \mathcal{X}_{IND}) \cdot p^\star(\mathbf{x} \notin \mathcal{X}_{IND} \mid \mathbf{x})}_{\mathbf{x} \text{ out of distribution}}$$

Solution using Brier's Score

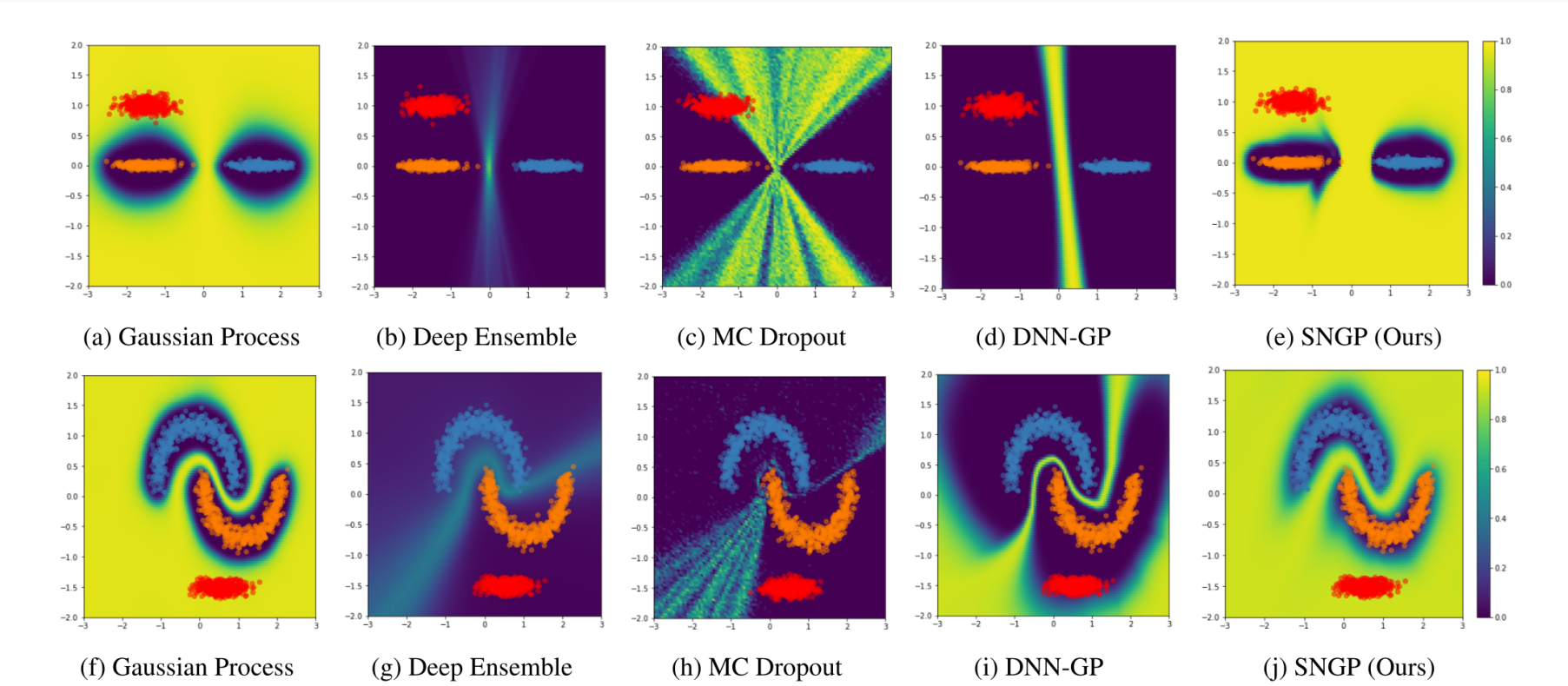
Using our **need**, we would like our model to estimate the previous probability as

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



Experiments

Synthetic example: Uncertainty for OOD detection



(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)

Real example

Model Usage:

1. Train the model in the **original dataset**.
2. Compute an uncertainty metric in **that** dataset.

Model Evaluation:

1. Train the model in the **original dataset**.
2. Create a **new** dataset.
3. Compute an uncertainty metric in the **new** dataset.

Real example: Evaluation dataset creation

1. Choose 2 datasets, one will be the **OUT** of distribution and the other one will be the **IN** distribution.
2. Create a new label:
    - 1 if the image belongs to the **OUT** of distribution dataset
    - 0 if the image belongs to the **IN** distribution dataset
3. Combine all the images with their new labels into a single dataset.

Real example: Uncertainty metric

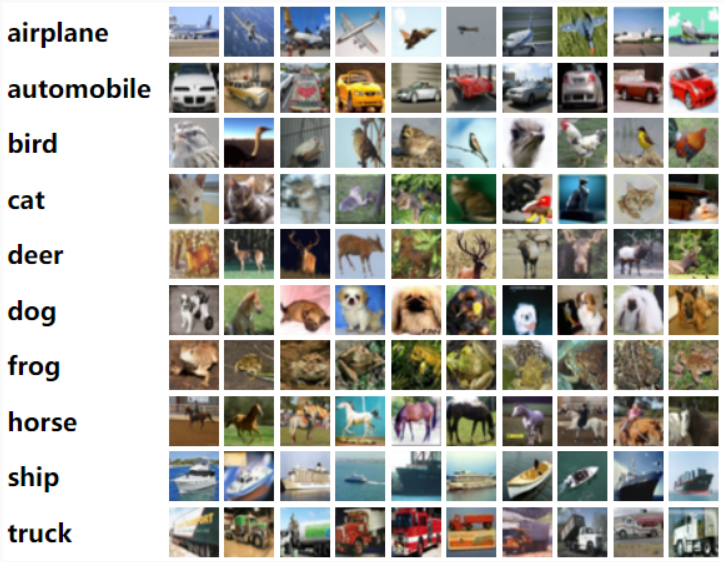
Consider the **output** of a Deep Learning model $logit(\mathbf{x}) = g \circ h(\mathbf{x})$.

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

Interpretation:

- $\uparrow\uparrow$ logits implies $u(\mathbf{x}) \to 0$, datapoint is **IN** distribution.
- $\downarrow\downarrow$ logits implies $u(\mathbf{x}) \to \frac{1}{2}$, datapoint is **OUT** of distribution.

Experiments: Example of databases



CIFAR-10

Street View House Numbers (SVHN)

- We train on CIFAR-10
- When creating the new dataset, SVHN is given label 1 (OOD).

Results on CIFAR-10

Method	Accuracy (↑) Clean	Corrupted
Deterministic	96.0 ± 0.01	72.9 ± 0.01
MC Dropout	96.0 ± 0.01	70.0 ± 0.02
Deep Ensembles	**96.6 ± 0.01**	**77.9 ± 0.01**
MCD-GP	95.5 ± 0.02	70.0 ± 0.01
DUQ	94.7 ± 0.02	71.6 ± 0.02
DNN-SN	96.0 ± 0.01	72.5 ± 0.01
DNN-GP	95.9 ± 0.01	71.7 ± 0.01
SNGP (Ours)	95.9 ± 0.01	74.6 ± 0.01

Accuracy

OOD AUPR (↑) SVHN	CIFAR-100
0.781 ± 0.01	0.835 ± 0.01
0.971 ± 0.01	0.832 ± 0.01
0.964 ± 0.01	0.888 ± 0.01
0.960 ± 0.01	0.863 ± 0.01
0.973 ± 0.01	0.854 ± 0.01
0.974 ± 0.01	0.859 ± 0.01
0.976 ± 0.01	0.887 ± 0.01
0.990 ± 0.01	**0.905 ± 0.01**

OOD performance

Thank you for your attention

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.

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

Extra slides: mathematical formulation

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:
It 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. $\|\cdot\|_{\mathcal{X}}$

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

Previous results

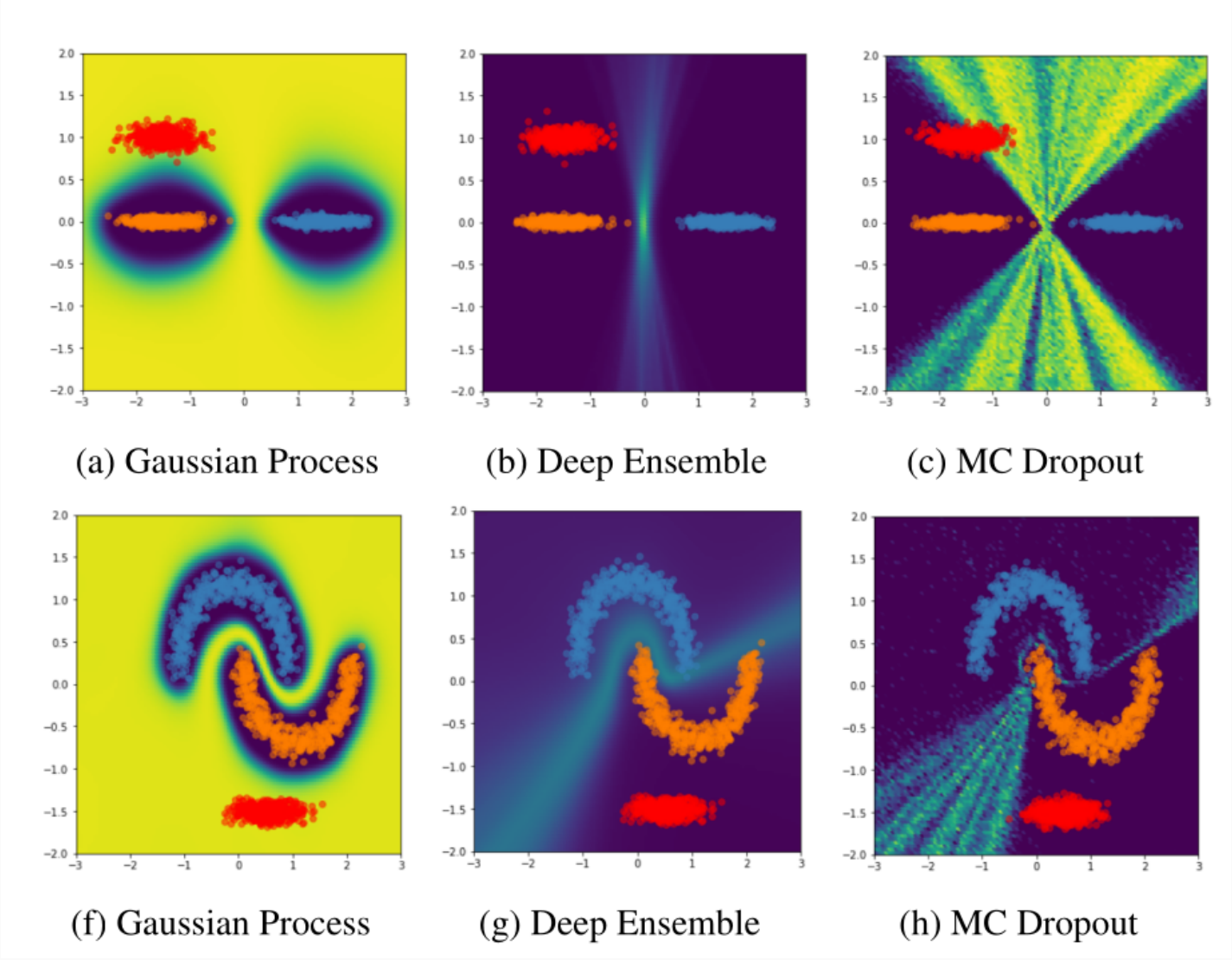


Figure 4: Results of preceeding models.

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 between points.

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:

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

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:

1. Make the output layer g distance aware, so it outputs an uncertainty metric reflecting distance between points.
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

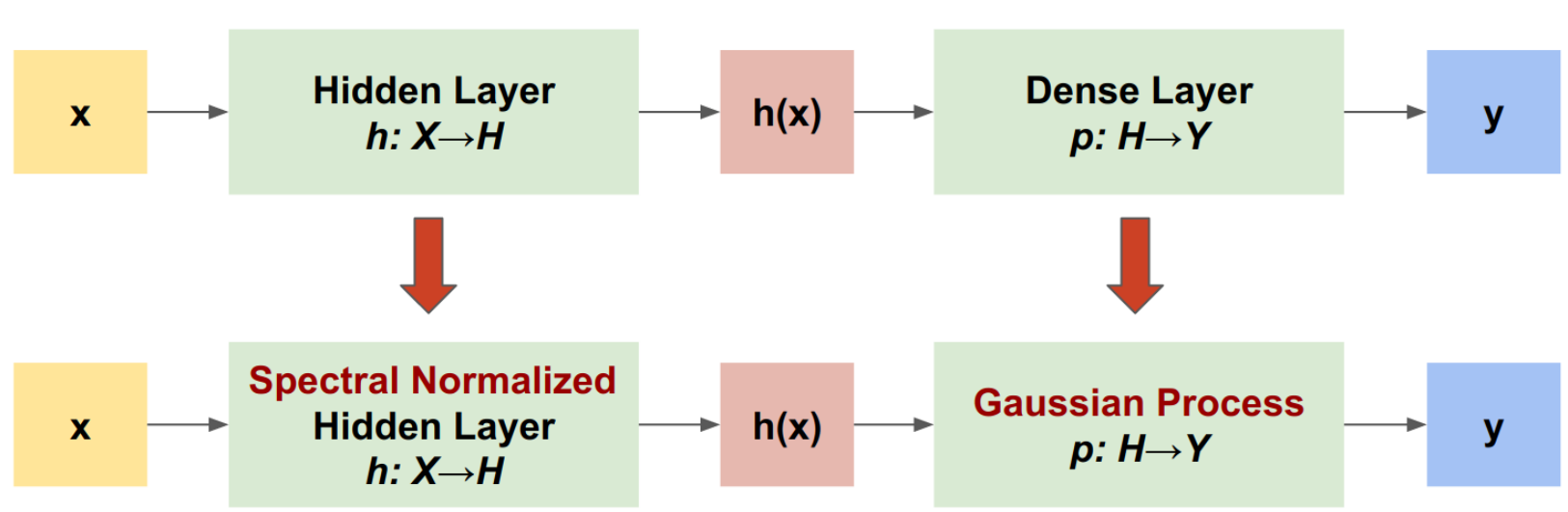


Figure 5: Visual description from the provided tutorial.

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)

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

Distance-aware Output Layer: Approximating GP Prior

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

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

Then, the k^{th} logit can be written as a NN layer:

$$g_k(h_i) = \sqrt{2/D_L} * \cos(-\mathbf{W}_L h_i + \mathbf{b}_L)^\top \underbrace{\beta_k}_{\text{learnable}} .$$

Distance-aware Output Layer: Approximating GP Prior

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

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

Then, the k^{th} logit can be written as a NN layer:

$$g_k(h_i) = \sqrt{2/D_L} * \cos(-\mathbf{W}_L h_i + \mathbf{b}_L)^\top \underbrace{\beta_k}_{\text{learnable}} .$$

The **GP posterior** is also approximated using a **Laplace approximation**, that is, using a Gaussian distribution centered at the MAP estimate:

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

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

$$h(\mathbf{x}) = h_{L-1} \circ \cdots \circ h_1(\mathbf{x}), \quad h_l(\mathbf{x}) = \mathbf{x} + f_l(\mathbf{x}). \tag{2}$$

Distance-preserving Hidden Mapping

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

$$\mathbf{W}_l = \begin{cases} c \cdot \mathbf{W}_l/\lambda & \text{if } c < \lambda \\ \mathbf{W}_l & \text{otherwise} \end{cases} \tag{3}$$

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

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