Notes on GVI in FS for Image Data

1. GWI in FS Set Up

Notes taken from Wild et al. (2022).

1.1 Gaussian Measures of GWI in FS

The Reference Gaussian Measure:

$$P \coloneqq \mathbb{P}^F \sim \mathcal{N}(m_{\mathbb{P}}(\cdot), \mathbf{C}_{k(\cdot,\cdot)})$$

where $m_{\mathbb{P}}(\cdot)$ is the reference mean function (generally zero constant mean function) and $k(\cdot,\cdot)$ is the reference kernel function.

Similarly, the Approximation Gaussian Measure:

$$Q \coloneqq \mathbb{Q}^F \sim \mathcal{N}(m_{\mathbb{Q}}(\cdot), \mathbf{C}_{r(\cdot,\cdot)})$$

where $m_{\mathbb{Q}}(\cdot)$ is the approximation mean function, which will be defined with dependence on the reference mean function, and $r(\cdot,\cdot)$ is the approximation kernel function, which will be defined with dependence on the reference kernel function.

1.2 Motivations

Evaluations of Gaussian Measures have computational complexity of order $\mathcal{O}(N^3)$ where N is the number of training points. Thus, despite the nice uncertainty quantification properties that Gaussian Measures provide, they are unable to scale well in large and more complicated data regimes (i.e training on a large number of images).

1.3 Modelling Approach

We will assume that our N data pairs $(\mathbf{x}_n, y_n)$ are generated from:

$$y_n = F(\mathbf{x}_n) + \epsilon_n$$

where $F \sim \mathcal{N}(0, \mathbf{C})$ is a random function, $\epsilon \sim \mathcal{N}(0, \sigma^2)$ for $n = 1, \cdots, N$, and $\mathbf{C} = \mathbf{C}_k$ depends on the reference kernel function k. $\sigma > 0$ is the observation noise or aleatoric uncertainty (irreducible data uncertainty).

The approach of GVI in FS is to define a reference Gaussian Measure P that will have the usual scaling issues. P will be trained on a subset of the data of size $\mathcal{O}\left(N^{1/2}\right)$ such that the evaluation of these points will have computational complexity $\mathcal{O}\left(N^{3/2}\right)$. Then an approximation Gaussian Measure Q is defined with respect to these $\mathcal{O}\left(N^{1/2}\right)$ inducing points such that the evaluation of the full N training points on Q will have computational complexity $\mathcal{O}\left(N^{3/2}\right)$.

GVI in FS involves the process of training Q such that it's behaviour will resemble that of P, our reference measure that is computationally intractable. This will be done in a two step process:

1. Train the parameters of the reference measure P using the inducing points. ($\mathcal{O}(N^{3/2})$)
2. To mimic the behaviour of P, train the parameters of Q by minimising of the Wasserstein metric between P and Q. ($\mathcal{O}(N^{3/2})$)

1.4 Reference Kernels

One choice of a reference kernel k is the ARD kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x_d')^2}{\alpha_d^2}\right)$$

for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$. σ_f^2 is the kernel scaling factor and $\alpha = [\alpha_1 \cdots \alpha_D]^T$ where each $\alpha_d > 0$ is the length-scale for dimension d. Any kernel can be chosen as the reference kernel, another example is the NNGP kernel.

1.4.1 Reference Kernel Hyperparameter Tuning

For GWI, we choose M inducing points where $M \in \{0.5\sqrt{N}, \sqrt{N}, \cdots\}$. M is some multiple of $\sqrt{N}$, which we will see is for control the computational cost of evaluating the approximation kernel is controlled. This provides us with inducing points $z_1, \cdots z_M$.

The reference kernel hyper-parameters are then chosen using these inducing points. One example of hyper-parameter tuning is by maximising the log-likelihood on the inducing points:

$$\log p(\mathbf{y_Z}) = -\frac{1}{2}\log\left(\det(k(\mathbf{X_Z}, \mathbf{X_Z})) + \sigma^2\mathbf{I}_M\right) - \frac{1}{2}\mathbf{y_Z}^T\left(k(\mathbf{X_Z}, \mathbf{X_Z}) + \sigma^2\mathbf{I}_M\right)^{-1}\mathbf{y_Z}$$

where the inducing point pairs of the vectors $\mathbf{X_Z}$ and $\mathbf{y_Z}$ are $(\mathbf{x}_m, y_m)$ for $m = 1, \cdots M$. Recall that $\sigma > 0$ is the observation noise or aleatoric uncertainty (irreducible data uncertainty).

[JW: If we choose the NNGP kernel, I think they have their own method of finding the optimal hyper-parameters.]

1.5 Approximation Mean Functions

There are many different ways we can define $m_{\mathbb{Q}}(\cdot)$, all of which should depend on the reference Gaussian Measure.

1.5.1 SVGP Mean Function

To recover the stochastic variational Gaussian Process from Titsias (2009), we can define the approximation mean function as:

$$m_{\mathbb{Q}}(\mathbf{x}) \coloneqq m_{\mathbb{P}}(\mathbf{x}) + \sum_{m=1}^{M}\beta_m k_m(\mathbf{x})$$

where $m_{\mathbb{P}}(\mathbf{x})$ is the evaluation of the reference mean function and $k_m(\mathbf{x})$ is the evaluation of the reference kernel function at the inducing point $\mathbf{x}_m$. With this mean function and the SVGP Approximation Kernel, we recover the SVGP from Titsias (2009).

1.5.2 DNN Mean Function

Another choice of mean function is by incorporating a neural network such that:

$$m_{\mathbb{Q}}(\mathbf{x}) \coloneqq m_{\mathbb{P}}(\mathbf{x}) + g(\mathbf{x})$$

where $m_{\mathbb{P}}(\mathbf{x})$ is the evaluation of the reference mean function and $g(\mathbf{x})$ a neural network.

1.6 Approximation Kernels

There are many different ways we can define $r(\cdot, \cdot)$, all of which should depend on the reference Gaussian Measure.

1.6.1 SVGP Kernel

One choice of approximation kernel is by defining a variational kernel:

$$r(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{X}_{\mathbf{Z}_M}) k(\mathbf{X}_{\mathbf{Z}_M}, \mathbf{X}_{\mathbf{Z}_M})^{-1} k(\mathbf{X}_{\mathbf{Z}_M}, \mathbf{x}') + k(\mathbf{x}, \mathbf{X}_{\mathbf{Z}_M}) \mathbf{\Sigma} k(\mathbf{X}_{\mathbf{Z}_M}, \mathbf{x}')$$

where k is the reference kernel and $\mathbf{\Sigma} \in \mathbb{R}^{m \times m}$ defined with respect to a Cholesky decomposition:

$$\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$$

and

$$\mathbf{L} = \text{Chol}\left(\left[k(\mathbf{X}_{\mathbf{Z}_M}, \mathbf{X}_{\mathbf{Z}_M}) + \frac{1}{\sigma^2} k(\mathbf{X}_{\mathbf{Z}_M}, \mathbf{X}_{\mathbf{Z}_N}) k(\mathbf{X}_{\mathbf{Z}_N}, \mathbf{X}_{\mathbf{Z}_M}) + \lambda \mathbf{I}_M \right]^{-1} \right)$$

where $\mathbf{Z}_M$ are the inducing points, $\mathbf{Z}_N$ are the training points and λ is a regulariser to ensure numerical stability.

2. GWI for Regression

Notes taken from Wild et al. (2022).

2.1 The Objective Function

The generalised objective to minimise:

$$\mathcal{L} = -\mathbb{E}_{\mathbb{Q}} \left[\log p(y|F) \right] + \mathbb{D}(\mathbb{Q}^F, \mathbb{P}^F)$$

the negative expected log-likelihood and the dissimilarity measure between the two distributions.

Assuming:

$$p(y|f) = \prod n = 1^N p(y_n|f)$$

and

$$p(y_n|f) \coloneqq \mathcal{N}(y_n|f(x_n), \sigma^2)$$

we can write the expected log-likelihood as:

$$\mathbb{E}_{\mathbb{Q}}\left[\log p(y|F)\right] = \sum_{n=1}^{N} \mathbb{E}_{\mathbb{Q}}\left[\mathcal{N}(y_n|f(x_n), \sigma^2)\right]$$

Choosing the Wasserstein metric as the dissimilarity measure:

$$\mathbb{D}(\mathbb{Q}^F, \mathbb{P}^F) = W_2^2(P, Q)$$

Combining:

$$\mathcal{L} = -\sum_{n=1}^{N} \mathbb{E}_{\mathbb{Q}}\left[\log \mathcal{N}(y_n|F(x_n), \sigma^2)\right] + W_2^2(P, Q)$$

our objective for Gaussian Wasserstein Inference.

2.1.1 The Negative Expected Log-Likelihood

Assuming Gaussian measures for P and Q, the negative expected log likelihood can be expressed as:

$$-\sum_{n=1}^{N} \mathbb{E}_{\mathbb{Q}}\left[\log \mathcal{N}(y_n|F(x_n), \sigma^2)\right] = -\frac{N}{2}\log(2\pi\sigma^2) - \sum_{n=1}^{N} \frac{(y_n - m_Q(x_n))^2 + r(x_n, x_n)}{2\sigma^2}$$

2.1.2 The Wasserstein Metric

[JK: Explain why this is a natural estimator/what the exact Wasserstein distance would look like and how this estimate relates to that. Also, you haven't covered the regression case here so it's not clear how this differs/is similar to the regression case.] [JW: Using Veit's paper I've followed the same reasoning. I've tried elaborating on the last trace term to fill in the steps that Veit skipped for my own understanding.]

For two Gaussian Measures $\mathbb{P} = \mathcal{N}(m_{\mathbb{P}}, C_{\mathbb{P}})$ and $\mathbb{Q} = \mathcal{N}(m_{\mathbb{Q}}, C_{\mathbb{Q}})$ on the Hilbert space $H = L^2(\mathcal{X}, \rho, \mathbb{R})$ Wasserstein distance $W_2^2(P_j, Q_j)$ the Wasserstein metric is given as:

$$W_2^2(\mathbb{P}, \mathbb{Q}) = \|m_{\mathbb{P}} - m_{\mathbb{Q}}\|_2^2 + tr(C_{\mathbb{P}}) + tr(C_{\mathbb{Q}}) - 2 \cdot tr\left[\left(C_{\mathbb{P}}^{\frac{1}{2}} C_{\mathbb{Q}} C_{\mathbb{P}}^{\frac{1}{2}}\right)^{\frac{1}{2}}\right]$$

which can be approximated with $\hat{W}^2$:

$$\begin{aligned}\hat{W}^2 \coloneqq & \frac{1}{N}\sum_{n=1}^{N}(m_{\mathbb{P}}(x_n) - m_{\mathbb{Q}}(x_n))^2 + \frac{1}{N}\sum_{n=1}^{N} k(x_n, x_n) \\ & + \frac{1}{N}\sum_{n=1}^{N} r(x_n, x_n) - \frac{2}{\sqrt{N N_S}} \sum_{s=1}^{N_S} \sqrt{\lambda_s(r(X_S, X)k(X, X_S))}\end{aligned}$$

where:

- $X_S \coloneqq (x_{S,1}, \ldots, x_{S,N_S})$ with $x_{S,1}, \ldots, x_{S,N_S} \in \mathbb{R}^D$, a set of N_S points sub-sampled from the input data X
- $r(X_S, X) \coloneqq \left(r(x_{S_s,x_n})\right)_{s,n} \in \mathbb{R}^{N_s \times N}$
- $k(X, X_S) \coloneqq \left(k(x_{x_n,S_s})\right)_{n,s} \in \mathbb{R}^{N \times N_s}$
- $\lambda_s(\cdot)$ calculates the s-th eigenvalue

and $n = 1, \ldots, N$, $s = 1, \ldots, N_S$, k is the kernel for $\mathbb{P}$, r is the kernel for $\mathbb{Q}$.

The approximation for each term of W^2 is shown below.
For $\|m_{\mathbb{P}} - m_{\mathbb{Q}}\|_2^2$:

$$\|m_{\mathbb{P}} - m_{\mathbb{Q}}\|_2^2 = \int \left(m_{\mathbb{P}}(x) - m_{\mathbb{Q}}(x)\right)^2 d\rho(x)$$

Approximating $\rho(x)$ with the empirical data distribution:

$$\|m_{\mathbb{P}} - m_{\mathbb{Q}}\|_2^2 \approx \frac{1}{N} \sum_{n=1}^{N} (m_{\mathbb{P}}(x_n) - m_{\mathbb{Q}}(x_n))^2$$

we have our approximation for the first term.
For $tr(C_{\mathbb{P}})$ and $tr(C_{\mathbb{Q}})$:

$$tr(C_{\mathbb{P}}) = \int k(x, x) d\rho(x)$$

Again, approximating $\rho(x)$ with the empirical data distribution, we have our estimate:

$$tr(C_{\mathbb{P}}) \approx \frac{1}{N} \sum_{n=1}^{N} k(x_n, x_n)$$

Similarly:

$$tr(C_{\mathbb{Q}}) \approx \frac{1}{N} \sum_{n=1}^{N} r(x_n, x_n)$$

For $tr\left[\left(C_{\mathbb{P}}^{\frac{1}{2}} C_{\mathbb{Q}} C_{\mathbb{P}}^{\frac{1}{2}}\right)^{\frac{1}{2}}\right]$:

$$\begin{aligned} tr\left[\left(C_{\mathbb{P}}^{\frac{1}{2}} C_{\mathbb{Q}} C_{\mathbb{P}}^{\frac{1}{2}}\right)^{\frac{1}{2}}\right] &= \sum_{n=1}^{\infty} \sqrt{\lambda_n \left(C_{\mathbb{P}}^{\frac{1}{2}} C_{\mathbb{Q}} C_{\mathbb{P}}^{\frac{1}{2}}\right)} \\ &= \sum_{n=1}^{\infty} \sqrt{\lambda_n \left(C_{\mathbb{Q}} C_{\mathbb{P}}\right)} \end{aligned}$$

because $C_{\mathbb{Q}} C_{\mathbb{P}}$ has the same eigenvalues as $C_{\mathbb{P}}^{\frac{1}{2}} C_{\mathbb{Q}} C_{\mathbb{P}}^{\frac{1}{2}}$.

Knowing that $C_{\mathbb{P}}(x) = \int k(x,x')d\rho(x')$, the operator $C_{\mathbb{Q}}C_{\mathbb{P}}$ is given as:

$$\begin{aligned} C_{\mathbb{Q}}C_{\mathbb{P}}f(x) &= \int r(x,x') \left(C_{\mathbb{P}}f\right)(x')d\rho(x') \\ &= \int r(x,x') \left(\int k(x',t)f(t)d\rho(t)\right) d\rho(x') \\ &= \int\int r(x,x')k(x',t)f(t)d\rho(x')d\rho(t) \\ &= \int (r*k)(x,t)f(t)d\rho(t) \end{aligned}$$

where $(r*k)(x,t) \coloneqq \int r(x,x')k(x',t)d\rho(x'), \forall x,t \in \mathcal{X}$.

This means $C_{\mathbb{Q}}C_{\mathbb{P}}$ is also an integral operator with (non-symmetric) kernel $r*k$. We can again approximate ρ with the data samples:

$$\widehat{(r*k)}(x,t) = \frac{1}{N}\sum_{n=1}^{N} r(x,x_n)k(x_n,t)$$

Thus, the spectrum of $C_{\mathbb{Q}}C_{\mathbb{P}}$ (set of its eigenvalues), we can calculate the spectrum by choosing a subsample of the data X_S of size $N_S < N$, we can approximate:

$$\begin{aligned} \lambda\left(C_{\mathbb{Q}}C_{\mathbb{P}}\right) &\approx \lambda\left(\frac{1}{N_S}\widehat{(r*k)}(X_S,X_S)\right) \\ &= \lambda\left(\frac{1}{N_S}\frac{1}{N}r(X_S,X)k(X,X_S)\right) \end{aligned}$$

We can then use this to approximate:

$$\begin{aligned} tr\left[\left(C_{\mathbb{P}}^{\frac{1}{2}}C_{\mathbb{Q}}C_{\mathbb{P}}^{\frac{1}{2}}\right)^{\frac{1}{2}}\right] &= \sum_{n=1}^{\infty}\sqrt{\lambda_n\left(C_{\mathbb{Q}}C_{\mathbb{P}}\right)} \\ &\approx \sum_{s=1}^{N_S}\sqrt{\lambda_s\left(\frac{1}{N_S}\frac{1}{N}r(X_S,X)k(X,X_S)\right)} \\ &= \frac{1}{\sqrt{NN_S}}\sum_{s=1}^{N_S}\sqrt{\lambda_s\left(r(X_S,X)k(X,X_S)\right)} \end{aligned}$$

Combining, we arrive at the approximation:

$$\begin{aligned} \hat{W}^2 \coloneqq &\frac{1}{N}\sum_{n=1}^{N}(m_{\mathbb{P}}(x_n) - m_{\mathbb{Q}}(x_n))^2 + \frac{1}{N}\sum_{n=1}^{N}k(x_n,x_n) \\ &+ \frac{1}{N}\sum_{n=1}^{N}r(x_n,x_n) - \frac{2}{\sqrt{NN_S}}\sum_{s=1}^{N_S}\sqrt{\lambda_s(r(X_S,X)k(X,X_S))} \end{aligned}$$

2.2 Prediction

We are concerned with:

$$p(y^*|x^*, \{x_n, y_n\}_{n=1}^N)$$

the posterior distribution for a new point y^* when conditioning our Gaussian measure on the training data $\{x_n, y_n\}_{n=1}^N$.

2.2.1 Gaussian Process Prediction

For a Gaussian process, the posterior distribution is given by:

$$p(y^*|x^*, \{x_n, y_n\}_{n=1}^N) = \mathcal{N}(\mu^*, \Sigma^* + \sigma^2)$$

where $\mathbf{X} = (x_n)_{n=1}^N$ and $\mathbf{y} = (y_n)_{n=1}^N$ such that:

$$\mu^* = k(x^*, \mathbf{X}) \left(k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}\right) \mathbf{y}$$

and

$$\Sigma^* = k(x^*, x^*) - k(x^*, \mathbf{X}) \left(k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}\right)^{-1} k(\mathbf{X}, x^*)$$

2.3 Tempering the Negative Log-Likelihood for Prediction

Consider the negative log-likelihood for the prediction for a point (x^*, y^*):

$$\begin{aligned} NLL &\coloneqq -\log p(y^*|x^*, \{x_n, y_n\}_{n=1}^N) \\ &= -\log \mathcal{N}\left(m_P(x^*), k(x^*, x^*) + \sigma^2\right) \\ &= \frac{1}{2}\left(\log\left(2\pi(k(x^*, x^*) + \sigma^2\right) + \frac{(y^* - m_P(x^*))^2}{k(x^*, x^*) + \sigma^2}\right) \end{aligned}$$

Tempering is the shrinking of the variance of the posterior predicted distribution by a factor of $\alpha_T \in [0, 1]$. It can be viewed as a process of uncertainty calibration for the model's predictions. This can be done experimentally by using a validation set of (x^*, y^*) points to find the optimal α_T for a Gaussian measure conditions on the training data.

The tempered negative log-likelihood for the prediction for a point (x^*, y^*):

$$\begin{aligned} NLL_{\alpha^T} &\coloneqq -\log p_{\alpha^T}(y^*|x^*, \{x_n, y_n\}_{n=1}^N) \\ &= -\log \mathcal{N}\left(m_P(x^*), \alpha_T \cdot (k(x^*, x^*) + \sigma^2)\right) \\ &= \frac{1}{2}\left(\log\left(2\pi\left[\alpha_T \cdot (k(x^*, x^*) + \sigma^2)\right]\right) + \frac{(y^* - m_P(x^*))^2}{\alpha_T \cdot (k(x^*, x^*) + \sigma^2)}\right) \end{aligned}$$

We can find the optimal α_T by minimising the negative log-likelihood on the validation set, while holding all other previously learned hyper-parameters fixed.

3. Gaussian Processes for Classification

Notes taken from chapter 4 of Matthews (2017).

For Gaussian process regression (GPR), a class of models is defined:

$$f \sim \mathcal{GP}(0, K(\theta))$$

where $f : X \to \mathbb{R}$, mapping to the set of real numbers $\mathbb{R}$ and K is the covariance function $K : X \times X \to \mathbb{R}$ parameterised by θ.

For binary Gaussian process classification (GPC), a mapping is defined:

$$g : \mathbb{R} \to [0, 1]$$

transforming a value on the real line to the unit interval to represent a probability. A Bernoulli random variable $\mathcal{B}$ can be defined such that:

$$f_c \sim \mathcal{B}(g(f))$$

where $f_c : X \to \{0, 1\}$, the desired binary classifier.

For multiclass classification of J different classes, models are defined:

$$f^{(j)} \sim \mathcal{GP}(0, K(\theta^{(j)}))$$

where $j = 1, \ldots, J$, defining J i.i.d. Gaussian processes. Concatenating $\mathbf{f} = [f_1 \cdots f_J]^T$, the classification operation can be defined:

$$\mathbf{f}_c \sim Cat(\mathcal{S}(\mathbf{f}))$$

where $\mathbf{f}_c : X \to \{0, \ldots, J\}$, the desired multiclass classifier. $\mathcal{S} : \mathbb{R}^J \to \Delta(J)$, a mapping from a J dimensional real vector to a J dimensional probability simplex. Cat is the categorical distribution (generalisation of Bernoulli distribution for Categorical Data).

There are different possible choices for $\mathcal{S}$. The multiclass generalisation of the logit likelihood:

$$\mathcal{S}_{softmax}(\mathbf{f})_i = \frac{\exp(f^{(i)})}{\sum_{j=1}^{J} \exp(f^{(j)})}$$

The robust max function:

$$\mathcal{S}^{(\epsilon)}_{robust}(\mathbf{f})_i = \begin{cases} 1 - \epsilon, & \text{if } i = \arg\max(\mathbf{f}) \\ \epsilon, & \text{otherwise} \end{cases}$$

taking class label of the maximum value with probability of $1-\epsilon$ and probability ϵ of picking one of the other classes uniformly at random, where ϵ is chosen. This formulation provides robustness to outliers, as it only considers the ranking of the GPR models for each class.

A benefit of the robust max function is that the variational expectation is analytically tractable with respect to the normal CDF ($q(\mathbf{f}) = \mathcal{N}(\mu, C), \mathbf{f} \in \mathbb{R}^J$) and one dimensional quadrature ($\mathcal{S}^{(\epsilon)}_{robust}(\mathbf{f})_i \in \mathbb{R}$):

$$\int_{\mathbb{R}^J} q(\mathbf{f}) \log(\mathcal{S}^{(\epsilon)}_{robust}(\mathbf{f})_y) d\mathbf{f} = \log(1 - \epsilon) S + \log\left(\frac{\epsilon}{J - 1}\right)(1 - S)$$

where S is the probability that the function value corresponding to observed class y is larger than the other function values at that point:

$$S = \mathbb{E}_{\mathbf{f}^{(y)} \sim \mathcal{N}(\mathbf{f}^{(y)} | \mu^{(y)}, C^{(y)})} \left[\prod_{i \neq y} \phi \left(\frac{\mathbf{f}^{(y)} - \mu^{(i)}}{\sqrt{C^{(i)}}} \right) \right]$$

where ϕ is the standard normal CDF. This one dimensional integral can be evaluated using Gauss-Hermite quadrature.

4. GWI for Multiclass Classification

Notes taken from A.6 of Wild et al. (2022).

4.1 Objective Function

The likelihood:

$$p(y|f_1, \ldots, f_J) = \prod_{n=1}^{N} p(y_n|f_1, \ldots, f_J)$$

where $p(y_n|f_1, \ldots, f_J) \coloneqq \mathcal{S}^{(\epsilon)}_{robust}(f_1(x_n), \ldots, f_J x_N)$ and $y_n \in \{1, \ldots, J\}$. $\mathcal{S}^{(\epsilon)}_{robust}$ is the robust max function as described in Matthews (2017). [JK: Don't define the same object twice with different names. Easy fix if you can't decide on which notation you want to use and perhaps want to change later: introduce a macro for the max function.] [JW: Changed to $\mathcal{S}_{robust}$ for now] Wild et al. (2022) used $\epsilon = 1\%$.

The model consists of J independent Gaussian Random Elements such that:

$$f_j \sim P_j = \mathcal{N}(m_{\mathbb{P},j}, C_{\mathbb{P},j})$$

with the corresponding variational measures:

$$Q_j = \mathcal{N}(m_{\mathbb{Q},j}, C_{\mathbb{Q},j})$$

The objective to minimise:

$$\mathcal{L} = -\mathbb{E}_{\mathbb{Q}} \left[\log p(y_n|F_1, \ldots, F_J) \right] + \sum_{j=1}^{J} W_2^2(P_j, Q_j)$$

4.1.1 Expected Log-likelihood

The variational (posterior) approximation of the probability of $\{(F_1(x), \ldots, F_J(x)) \in A\}$ will be denoted:

$$\mathbb{Q}\left((F_1(x), \ldots, F_J(x)) \in A\right)$$

where $A \subset \mathbb{R}^J$. We get the expected log-likelihood: [JK: Would be good to explain where this approximation comes from] [JW: I've copied over the reasoning from Veit's paper but not entirely sure on how each step follows yet.]

$$\begin{aligned}\mathbb{E}_{\mathbb{Q}}\left[\log p(y|F_1,\ldots,F_J)\right] &= \sum_{n=1}^{N}\mathbb{E}_{\mathbb{Q}}\left[\log p(y_n|F_1,\ldots,F_J)\right] \\ &= \sum_{n=1}^{N}\log(1-\epsilon)\mathbb{Q}\left(\underset{j=1,\cdots,J}{\arg\max}\{F_j(x_n)\}=y_n\right) \\ &\quad + \log\left(\frac{\epsilon}{J-1}\right)\mathbb{Q}\left(\underset{j=1,\cdots,J}{\arg\max}\{F_j(x_n)\}\neq y_n\right) \\ &\approx \sum_{n=1}^{N}\log(1-\epsilon)S(x_n,y_n)+\log\left(\frac{\epsilon}{J-1}\right)(1-S(x_n,y_n))\end{aligned}$$

$$\mathbb{E}_{\mathbb{Q}}\left[\log p(y_n|F_1,\ldots,F_J)\right] \approx \sum_{n=1}^{N}\log(1-\epsilon)S(x_n,y_n)+\log\left(\frac{\epsilon}{J-1}\right)(1-S(x_n,y_n))$$

where:

$$S(x,j) \coloneqq \frac{1}{\sqrt{\pi}}\sum_{i=1}^{I} w_i \prod_{l\neq j}\phi\left(\frac{\sqrt{2r_j(x,x)}\xi_i + m_{Q,j}(x) - m_{Q,l}(x)}{\sqrt{r_l(x,x)}}\right)$$

for any $x \in \mathcal{X}$, $j = 1,\ldots,J$ where $(w_i,\xi_i)_{i=1}^{I}$ are the weights and roots of the Hermite polynomial of order $I \in \mathbb{N}.$, calculated with `scipy.special.roots_hermite`. ϕ is the standard normal cumulative distribution function.

4.2 Prediction

For an unseen point $x^* \in \mathcal{X}$, the probability that it belongs to class $j \in \{1,\ldots,J\}$:

$$\mathbb{Q}(Y^*=j) = (1-\epsilon)S(x^*,j) + \frac{\epsilon}{J-1}(1-S(x^*,j))$$

where the predicted label class is the maximiser of this probability: [JK: Style thing: define a macro for *Cat* so that i looks like Cat in math environments if you're going to use it more often. Otherwise, just write Cat in-text. Also unclear what Cat means here; undefined.] [JW: Added a Cat operator, thanks for the tip.]

$$\mathrm{Cat}(\mathbb{Q}(Y^*)) = \underset{j\in\{1,\ldots J\}}{\arg\max}\,\mathbb{Q}(Y^*=j)$$

where Cat is the categorical operator, choosing the class from $1,\cdots J$ with the highest probability given by $\mathbb{Q}$.

5. Uncertainty Quantification Review

Notes taken from a review paper by Abdar et al. (2021).

There are two main types of uncertainty: aleatoric and epistemic. Epistemic uncertainty is the model uncertainty (i.e. choosing to fit the data with a quadratic function when the

data is sinusoidal) and can be formulated as a probability distribution over the model parameters. Aleatoric uncertainty is the irreducible uncertainty of the data (data uncertainty) and considered an inherent property of the data distribution. Aleatoric uncertainty can be further divided into homoscedastic and heteroscedastic uncertainties.

5.1 Monte Carlo Dropout

Estimate epistemic uncertainty by applying MC dropout with Bernoulli distribution at the output of the neurons of a NN. Different options for dropout-based methods include Bernoulli/Gaussian dropout of either the nodes of a NN or the weights of a NN.

5.2 Markov Chain Monte Carlo

This uses MCMC to estimate intractable posterior distributions. There are issues with the required iterations for sufficient burn-in of the sampler being unknown, an issue with MCMC that extends beyond uncertainty quantification.

5.3 Variational Inference

An approximation method learning the posterior distribution over BNN weights.

5.4 Ensemble Techniques

NNs generally have competitive accuracy but poor predictive uncertainty quantification, usually generating overconfident predictions. Calibration and domain shift are two evaluation measures used to evaluate the quality of predictive uncertainty. Calibration measures the discrepancy between long-run frequencies and subjective forecasts. Domain shift quantifies the generalisation of predictive uncertainty to a domain shift in the data (i.e. trained on cats and dogs but then asked to make a prediction on a bird). Quantifies if the model is aware of what it does/doesn't know.

An ensemble of models enhances predictive performance, but it's not immediately obvious why it would generate good uncertainty estimation. Bayesian model averaging (BMA) holds belief that the true model lies within the hypothesis class of the prior $\mathcal{H}$. Ensembles combine models to discover more powerful models, so they can be expected to be better when true model does not lie in $\mathcal{H}$.

An evaluation approach for measuring uncertainty estimators in vision problems can be found in Gustafsson et al. (2020).

Measures of spread or "disagreement" of ensembles such as mutual information can be used to assess uncertainty in predictions due to knowledge uncertainty:

$$\mathcal{MI}\left[y, \theta | \mathbf{x}^*, \mathcal{D}\right] = H[\mathbb{E}_{p(\theta|\mathcal{D})}[P(y|\mathbf{x}^*, \theta)]] - \mathbb{E}_{p(\theta|\mathcal{D})}[H[y|\mathbf{x}^*, \theta]]$$

where:

- $\mathcal{MI}\left[y, \theta | \mathbf{x}^*, \mathcal{D}\right]$ is the knowledge uncertainty
- $H[\mathbb{E}_{p(\theta|\mathcal{D})}[P(y|\mathbf{x}^*, \theta)]]$ is the total uncertainty
- $\mathbb{E}_{p(\theta|\mathcal{D})}[H[y|\mathbf{x}^*, \theta]]$ is the expected data uncertainty (i.e. regions of severe class overlap)

Two situations: all models have similar uncertainty distribution are very uncertain for each label (data uncertainty) or the models in the ensemble have very different predictions (model uncertainty).

5.5 Other Uncertainty Quantification Methods

Neural Architecture Distribution Search (NADS) finds an appropriate distribution of different architectures that perform significantly well on a specified task.

explored the training dynamics of over-parameterised NNs under natural gradient descent. They showed that the discrepancy between NNs trained on non-linearised and linearised natural gradient descent is smaller than that of standard gradient descent. Also, that empirically there was no need to formulate a limit argument about the width of the neural network layers, as the discrepancy was small for over-parameterised NNs.

BNNs have been used as a solution for NN predictions but specifying priors is still an open problem. Independent normal prior in weight space leads to weak constraints on function posterior, allowing it to generalise in unanticipated ways on OOD data. Noise contrastive priors (NCPs) used to estimate consistent uncertainty by Hafner et al. (2020).

Mixup is a DNN training technique where extra samples are produced during training by convexly integrating random pairs of images and their labels. Thulasidasan et al. (2019) showed that this provided much better model calibration and was less likely to yield overconfident predictions using random noise and OoD data.

Adversarial training can eradicate the vulnerability in a single model by forcing it to learn more robust features, but this approach is rigid and suffers from substantial loss on clean data accuracy. Ensemble techniques can be induced to have diverse sub-models robust to a transfer adversarial example.

Gaussian processes do not scale well, but a common technique is to have a variational GP using inducing samples. Deep Gaussian processes represent a multilayer hierarchy of Gaussian processes.

Most weight perturbation-based algorithms suffer from high variance of gradient estimation due to sharing the same perturbations among all samples in a mini-batch. Flipout by Wen et al. (2018) is an approach that samples pseudo-independent weight perturbations for each input to decorrelate the gradients within a minibatch.

DNNs have been successful with complex high-dimensional image data but are not robust to adversarial examples as shown in Szegedy et al. (2013). Bradshaw et al. (2017) proposed a hybrid model of GP and DNNs (GPDNNs) to deal with the uncertainty caused by adversarial examples. Convolutional structures have also been introduced into GPs such as in Van der Wilk et al. (2017).

6. Neural Tangents

Notes taken from Novak et al. (2019).

The infinite-width limit of a large class of Bayesian neural networks become Gaussian Processes with specific, architecture-dependent, compositional kernel, forming a Neural Network Gaussian Process (NNGP) model. Kernels can be defined with recurrence relationships for a wide range of non-linearities (activation functions), convolutional layers, residual connections, and pooling. Neural Tangent Kernels (NTK) relates to the gradient descent

training of the infinite-width limit of a Bayesian Neural Network. Infinite-width kernels that cannot be constructed analytically can be approximated by Monte Carlo sampling.

Neural Tangents provide framework for automatic construction of infinite-width kernels that would otherwise need to be derived for each new architecture by hand.

References

Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schon. Evaluating scalable bayesian deep learning methods for robust computer vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 318–319, 2020.

Danijar Hafner, Dustin Tran, Timothy Lillicrap, Alex Irpan, and James Davidson. Noise contrastive priors for functional uncertainty. In *Uncertainty in Artificial Intelligence*, pages 905–914. PMLR, 2020.

Alexander Graeme de Garis Matthews. *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge, 2017.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. *arXiv preprint arXiv:1912.02803*, 2019.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional gaussian processes. *Advances in Neural Information Processing Systems*, 30, 2017.

Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.

Veit D Wild, Robert Hu, and Dino Sejdinovic. Generalized variational inference in function spaces: Gaussian measures meet bayesian deep learning. *arXiv preprint arXiv:2205.06342*, 2022.