Efficient Uncertainty Quantification for Under-constraint Prediction following Learning using MCMC *

Gargi Roy and Dalia Chakrabarty (✉)

Brunel University London, Kingston Lane, London, Uxbridge UB8 3PH, UK
{Gargi.Roy,Dalia.Chakrabarty}@brunel.ac.uk

Abstract. We present an illustration of a method to ensure reliable uncertainty percolation, within supervised learning performed using Gaussian Processes (GP), and Markov Chain Monte Carlo based inference. We show the effect of variously propagating the uncertainty, on predictions undertaken on the output variable, at test inputs, subsequent to the learning of the functional relationship between the input and the output, where this functional relation is modelled as a realisation from a GP. The efficiency of imposing a physically motivated constraints on the output - via priors imposed on the GP covariance kernel hyperparameters - is compromised under certain strategies adopted to propagate uncertainty. Tools such as DNNs, that are relatively more blind to uncertainty learning/propagation, are found to be diversely inaccurate in their output prediction.

Keywords: Covariance kernel hyperparameter · MCMC · Learning under constraint · Gaussian Process · Uncertainty Propagation.

1 Introduction

Learning the functional relationship between the response and the predictor variables is a crucial task in any supervised mechanistic learning set up, where automated and reliable learning of this function is sought, given a training data set. Probabilistic learning provides the framework for undertaking inference on models that explain the data under consideration, while also quantifying uncertainties objectively. Reliable learning of the uncertainties in the learning of the sought function is crucial, since multiple models can be consistent with the data [3], and the data itself can include uncertainties due to measurement errors. There could be further complications owing to hidden variables [3] that cannot be addressed just by gathering more data under the same data collection set up. The prediction of new outputs using the learnt functional relationship, is also uncertain, where such uncertainties can be interpreted to reflect on limitations of the model; misrepresentative training set; noise in the test inputs; and possibility of the input-output relationship to be non-bijective. Hence the uncertainty

* Supported by EPSRC DTP Studentship.

quantification has enormous significance, especially in sensitive domains [1] and when the available training data is small and/or expensive to procure. Examples of sensitive domains include medicine [1]; traffic management systems [7]; information security [13], etc.

However, despite its importance, uncertainty quantification is still an emerging area and requires careful addressing, in order to achieve high quality intelligence from the data [4]. A poorly chosen representation of uncertainty can affect prediction greatly. In this work we illustrate what we imply by a poorly chosen way of addressing uncertainties and show the effect of the same on the quality of predictions.

In this paper, we undertake learning and prediction within Gaussian Process (GP) regression [11]. GP-based regression was introduced in [9]; however, it gained popularity as a non-parametric modelling approach following [8]. [10] states that GP predictive performance is comparable to several other state-of-the-art modelling approaches including neural networks. Predictive distributions are computed for each test point in GP regression, and we consider the mean of these predictive as the predicted output value, while the closed-form prediction of the respective variance provides the uncertainty on the prediction. Treating a function - of unknown form, and therefore treated as random in the Bayesian setting - as a sample from a GP implies that the joint probability of a finite number of realisations of this function - if scalar-valued - is multivariate Normal with a mean vector and variance-covariance matrix. This covariance matrix, if parameterised using a covariance kernel may be of a chosen parametric form, with hyper-parameter(s) assigned to this chosen form. Values of such hyperparameters are unknown and need to be learned from the data, rather than imposed by hand, to avoid erroneous predictions. In this work, we have used Squared Exponential Kernel (SQE) with length-scale hyperparameter ℓ for the GP. We learn ℓ from data using Markov Chain Monte Carlo (MCMC) inference techniques. Generally a GP does not pose any constraint on its predictive computation; however, when the application demands the outputs to abide by certain physically-motivated constraints, then said constraints need to be taken on board to render predictions correct. For example, if in the application, the output is a probability, then we need the GP output prediction to lie between 0 to 1. Here we present the learning of ℓ using MCMC, under the constraint that the output predictions lie in the range of 0 to 1.

We will also present the comparative results for regression undertaken with Deep Neural Network (DNN) systems (which do not account for uncertainties), with varying architecture. Such a comparative exercise will indicate the sensitivity of the results to the chosen architectural details.

2 Data and Model

In 1986, the Space Shuttle Challenger exploded very soon after launch, killing all seven astronauts on board. The explosion was the result of an O-ring failure. An O-ring is a rubber ring that seals parts of the Shuttle. The O-rings are made of

a material that stiffen at the ambient coldness [2], thereby causing these O-rings to fail to provide the required sealing, resulting in the explosion of the Shuttle. The accident was believed to have been caused by the unusually cold weather (with ambient temperatures of about 31°F) at the time of the launch.

We use the real test flight data on O-ring failure status Y. Here the input is a scalar, namely, temperature $X \in \mathbb{R}$ at which test flights of the Space Shuttle were undertaken, The failure status variable, i.e. the output variable, is binary, and is coded to attain 1 if the O-ring fails; else $Y = 0$.

Table 1 depicts this test flight data, [12]. We refer to this data as D. Given that the output is binary, its regression against the predictor variable (temperature X) will be modelled with the logistic regression model [12, 15]. Thus, given that we have only one predictor variable, the linear predictor in this regression model has two regression coefficients, namely, the slope $\beta \in \mathbb{R}$ and the intercept parameter $\alpha \in \mathbb{R}$. Under the assumption of *iid* data points, likelihood is the product of the probability of each value of Y, at the corresponding temperature. This probability is the Bernoulli probability mass function, with parameter that is a function of temperature. We use Gaussian priors on each parameter, and ultimately write down the joint posterior of the slope and intercept parameters given the test flight data. Inference on model parameters is undertaken via posterior sampling with MCMC. We compute marginal posterior of each parameter given data D, using which we learn the 95% Highest Probability Density credible region (HPDs) [5] on each parameter.

Our learning of the parameters of this logistic regression model will allows us to compute the probability for O-ring failure at given temperatures. Such computation is relevant to our supervised learning endeavour, in which this $\Pr(Y = 1||X = x)$ is the output and X is the input variable. The set $D_{train} := \{(x_i, \Pr(Y = 1|x_i))\}_{i=1}^{M}$ of M pairs of value of this design input $X = x_i$ and the corresponding output will constitute the training data using which we will learn the functional relationship between this output and input, by modelling the said relationship as a random function that is realisation from a GP. Thus, our supervised learning endeavour will reduce to the learning of the parameters of the covariance structure of this GP. As we will parameterise the covariance structure using a parametric kernel, our supervised learning will basically reduce to our learning of the hyperparameters of this covariance kernel. There is in fact only one such hyperparameter that is relevant - this is the length scale ℓ. So we will learn ℓ using D_{train}, where D_{train} is arrived at from the learning of α and β, as motivated above.

In our work, we know the parametric form of how output values - $\{\Pr(Y = 1|X = x_i)\}_{i=1}^{M}$ - are distributed across the range of the input temperature values; this is the logistic function in X, with parameters α and β. This knowledge will benefit the inspection of the learnt input-output functional relation that is modelled with a GP, (and learnt using generated training data D_{train}). The point is that the GP-based learning of this function permits prediction of the output at test temperatures, and as we know the theoretical distribution of the output against temperatures, we can check for the veracity of our predictions. Also, the

uncertainty on the output value computed at a design temperature is available from the uncertainty in our learning of the slope and intercept parameters, (β and α), of the logistic regression model. This offers us a tool to check how noise in the training set affects the prediction performance.

As stated in the previous section, we could compute the output, i.e. probability of O-ring failure at a design temperature, by using a summary (eg. mean, median, etc.) of the marginal of the slope and intercept parameters - as learnt using the MCMC. We could alternatively compute the output using any value of the slope and intercept parameters from within the 95% HPDs on each such parameter, that are learnt by MCMC. We will show that with different choices from within the 95% HPD range, parameterisation of the GP covariance structure varies, thereby affecting the final prediction. This method of computing the probability of O-ring failure at a design temperature then appears unsatisfactory.

One way to avoid this is by using a pipe-lined algorithm, in which we learn the slope and intercept parameters at each iteration of MCMC, given the test flight data D, and then proceed in that same iteration, to predict the probability $\Pr(Y = 1|vertX = x_i)$ of O-ring failure at each design temperature x_i, at the values of α and β that are current as of this iteration. This is repeated across iterations and uncertainties in each probability value accrued across post-burnin iterations of the chain. This pipe-line enables best propagation of uncertainties in our learning of parameters of the original (logistic) model, towards generation of D_{train}.

In the next section we will present the results obtained using the former approach towards training set generation, namely, by computing output using summaries of the parameters. We display results of predictions made following the GP-based learning of the relation between probability of O-ring failure and temperature, as well as with the DNNs. Then, in Section 4 we will present the pipe-lined architecture for prediction.

Table 1: Test flight data D reproduced from page 15 of [12]. O-ring failure status is presented with temperature (in °F) in the second row.

Failure	1	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0
Temperature	53	57	58	63	66	67	67	67	68	69	70	70	70	70	72	73	75	75	76	76	78	79	81

2.1 Learning Parameters of the Logistic Regression Model

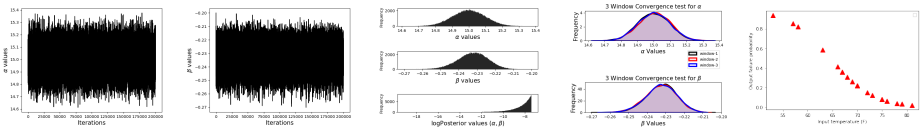
For binary response variable $Y \in \{0, 1\}$ that represents O-ring failure status, and the temperature $X \in \mathbb{R}$, we want to model the probability of failure given a temperature, assuming that Y follows a Bernoulli distribution. The probability of failure given the temperature $X = x$ is modeled within logistic regression using the slope and intercept parameters $\beta \in \mathbb{R}$ and $\alpha \in \mathbb{R}$ in general. $P(Y = 1|X = x) = \dfrac{e^{(\alpha+\beta x)}}{1 + e^{(\alpha+\beta x)}}$. Then likelihood is depicted as follows, (with n being the number of data points which is 23 here).

$$\mathcal{L}(\alpha, \beta|D) \propto \prod_{i=1}^{n} \left(\frac{e^{(\alpha+\beta x)}}{1 + e^{(\alpha+\beta x)}} \right)^{y_i} \left(\frac{1}{1 + e^{(\alpha+\beta x)}} \right)^{1-y_i} \tag{1}$$

We use Normal priors for both α and β:

$$\pi_0(\alpha) = \frac{1}{\sigma_\alpha^P \sqrt{2\pi}} e^{-(\alpha-\mu_\alpha)^2/2\sigma_\alpha^{P2}}; \quad \pi_0(\beta) = \frac{1}{\sigma_\beta^P \sqrt{2\pi}} e^{-(\beta-\mu_\beta)^2/2\sigma_\beta^{P2}}.$$

Thus, the unscaled joint posterior $\pi(\alpha, \beta|D, \pi_0(\alpha), \pi_0(\beta))$ of the model parameters, given the test flight data D is given as the product of the aforementioned likelihood and priors. Fig. 1 displays results of learning α and β given data D. The 95% HPDs on the parameters are: [14.8, 15.19], (with a mean of 14.995) for α and [-0.249, -0.216], (with a mean of -0.2325) for β.



(a) (b) (c) (d) (e) D_{mean}

Fig. 1: Results of learning α, β using O-ring data D. From left, (a,b) trace (variation with iteration index) of α, β respectively; (c) histogram of marginal posterior of α (upper), β (lower) and log of their joint posterior; (d) histograms of α (upper) and β (lower) values sampled during the MCMC chain, over three distinct (colored in blue, red and black), equally-wide (covering 20000 consecutive iterations), non-overlapping iteration windows for convergence test which are overplotted; (e) $\Pr(Y = 1|X = x_i)$ computed at 16 different temperatures using the mean of the α and β learnt in this chain, using data D. This MCMC chain was run using a Metropolis-within-Gibs algorithm. Total number of iteration in this chain is 200000; the variances of the truncated Normal proposal densities for α and β are 0.074^2, and 0.008^2 respectively with Normal priors on α and β.

3 Generating Training Data Sets at Parameter Summaries

In this section we first discuss the generation of training data D_{train} comprising pairs of values of design input, and the probability for Y to be 1 at that design temperature. As motivated above, we will undertake this generation in two distinct ways - for the D_{train} generated under a given approach, we refer to it by its updated name.

In the first approach, at the end of the MCMC chain that is used to learn the parameters α and β of the logistic regression model, we compute the mean of the sampled α and β, to generate the $\Pr(Y = 1|X = x)$, at the design temperature value x; such $(x, \Pr(Y = 1|x))$ pairs will constitute a training set D_{mean} (depicted in Fig. 1). D_{mean} is one of the training data sets that we will use to learn the functional relationship between this output (i.e. $\Pr(Y = 1|x)$ and the input X. (As stated above, such supervised learning will reduce to our learning of the length scale hyperparameter ℓ of the covariance structure of the GP that we will model our sought function with).

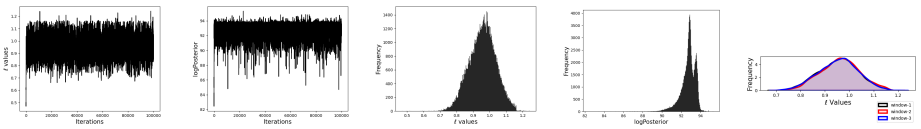
Another training data D_{right} will be built by using the learnt values of α and β at the right edge of the 95% HPD learnt on α and β using data D. In principle, any values from within the 95% HPDs on α and β - inferred upon using data D - can be used in this context.

We will perform with-uncertainty learning of the kernel hyperparameter ℓ, to undertake predictions on the output variable - which is $\Pr(Y = 1|X = x)$ - at test inputs. When making this learning of the relationship between the output $\Pr(Y = 1||x)$ and input X, there is no information available to our learning mechanism to have the output predictions - at test temperatures - to remain confined to the interval [0,1]. In other words, the supervised learning algorithm does not distinguish on the basis of the nature of the output variable. We will impose such a constraint on the learning to ensure meaningful predictions, using MCMC-based inference.

3.1 Learning ℓ Under the Constraint Given D_{mean}

We attempt learning the function $f(\cdot)$, where $W = f(X)$, with the output $W \equiv \Pr(Y = 1|X = x)$, s.t. the unknown $f : \mathbb{R} \longrightarrow [0, 1]$. We do this by treating $f(\cdot)$ as an unknown, which in the Bayesian paradigm translates to $f(\cdot)$ being modelled as a random variable, or rather a random function in this case. Then by definition, $f(\cdot)$ is ascribed a probability distribution. A probability distribution on the space of functions is a stochastic process. So we model $f(\cdot)$ as a realisation from a process. For maxima generalisability and ease of computation, we choose this process to be a GP, s.t. minimal constraints are imposed on the sample function $f(\cdot)$. So $f \sim GP(\mu(x), cov(x, x^{/}))$, where $\mu(\cdot)$ and $cov(\cdot, \cdot)$ are mean and covariance functions of this GP. Then by definition, the joint probability of a finite number of realisations of $f(\cdot)$ - such as the M realisations of $f(\cdot)$ at each of the design points in the training data D_{train} - is Multivariate Normal, with the M-dimensional mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}^{(M\times M)}$. We parameterise matrix $\boldsymbol{\Sigma}$ by saying that the ij-Th element of this matrix that represents the covariance between W_i and W_j, is modelled as a declining function $K(\cdot, \cdot)$ of the distance between the inputs s_i and x_j at which these outputs are realised. Here $i, j = 1, \ldots, M$.This distance is chosen to be Euclidean, and $K(\cdot, \cdot)$ a Squared Exponential (or SQE) kernel function, s.t. $\boldsymbol{\Sigma} = [cov(W_i, W_j)] = [K(x_i, x_j)]$, with $K(x_i, x_j) := \exp(-(x_i - x_j)^2/\ell^2)$, $\ell > 0$. We choose to model the covariance structure of this GP using an SQE kernel with a fixed length scale hyperparameter $\ell > 0$. We compute the empirical mean $\bar{w}$ of the output values $f(x_1) = w_1, \ldots, f(x_M) = w_M$ that live in the training data D_{train}, as each component of $\boldsymbol{\mu}$.

So as the joint of $f(x_1), \ldots, f(x_M)$ is Multivariate Normal, with mean $\boldsymbol{\mu}$, and covariance matrix $\boldsymbol{\Sigma} = [K(x_i, x_j)]$, it implies that the joint of $w_1, \ldots, w_M$ is this Multivariate Normal. But the joint of these M values of the output W that live in training data, is the probability of the data on W. In fact, it is a conditional probability - conditional on the parameters of the mean and covariance, i.e. on ℓ. But the probability of the data conditional on model parameters is the likelihood.



(a) (b) (c) (d) (e)

Fig. 2: Results of learning ℓ using training data D_{mean}. Here the 2-stepped prior is imposed on ℓ. From left, (a,b) traces of learned values of ℓ and logarithm of posterior of ℓ respectively; (c,d) histogram representations of the marginal of ℓ and posterior values respectively; (e) 3-window convergence test, (see caption of Fig. 1) to confirm convergence of the MCMC chain.

Thus, the likelihood is Multivariate Normal with mean $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ that is kernel parameterised as above, with hyperparameter ℓ,i.e. the likelihood is $\mathcal{L}(\ell|D_{train}) = (1/\sqrt{(2\pi)^M|\boldsymbol{\Sigma}|})\exp\left(-(\boldsymbol{w}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{w}-\boldsymbol{\mu})/2\right)$, where $\boldsymbol{w} = (w_1, \ldots, w_M)^T$, and $\boldsymbol{\Sigma}$ is kernel parametrised using the SQE kernel with hyperparameter ℓ. We impose Gaussian priors on ℓ. Then the unscaled posterior on ℓ given the training data is this likelihood times the Gaussian prior. We perform learning with Metropolis Hastings in which ℓ is proposed from a truncated Normal density, and training data that is generated as the data D_{mean}, using the mean of the sampled α and β that are learnt using D. We update ℓ in each iteration of the chain, and subsequently, predict the mean and variance of the output W at test values of the input. However, these predictions do not necessarily offer output values to remain in [0,1].

To remedy this, we impose the constraint that we learn only those ℓ that predict outputs at any test input, to lie in [0,1]. We accomplish this by inputting a prior on ℓ that offers a density of $\rho_{hi} > 0$ if the predictions on an arbitrarily chosen set of test temperatures are within [0,1], and the prior is set at the value ρ_{lo}.

In the t-th iteration of the MCMC chain, after proposing ℓ as $\ell^{\star,t} \sim$ Truncated Normal$(0, \ell^{(t-1)}, 0.03^2)$, a 2-stepped prior is invoked, as discussed below. Here $\ell^{(t-1)}$ is the current value of ℓ in the $t-1$-th iteration of the MCMC chain.

$$f_{stepPrior}(\ell_{prop}) = \begin{cases} \rho_{hi}\pi_{0,Normal}, & \text{if } W \text{ predicted using } \ell = \ell_{prop} \text{ is } \in [0,1] \\ \rho_{lo}\pi_{0,Normal}, & \text{otherwise,} \end{cases} \tag{2}$$

where $\pi_{0,Normal} = \mathcal{N}(0.9, 0.08^2)$ prior placed on ℓ. We choose $\rho_{hi} = 1$, and $\rho_{lo} \leq e^{-10^3}$. The MCMC chain that we run comprises 100000 iterations; and the 95% HPD learnt on the ℓ using D_{mean} is [0.79509924, 1.10960904], with mean 0.9523 approximately. Fig. 2 shows results obtained by running this chain. Fig. 5a shows the GP prediction for this set up using $\ell \approx 0.9523$.

3.2 DNN Results

We have also implemented Deep Neural Network systems with varying architectures to make prediction of output $W \equiv \Pr(Y = 1|X = x_{test})$ at test temperatures. Here the DNN architecture varies in number of hidden layers (one to four), with 64 neurons in each layer. We make these predictions using Tensorflow, following the code that is available in the official documentation/tutorial cite of Tensorflow [14]. To allow for compatible comparison, here we perform predictions at the same test data, as used when output predictions were made following the learning of ℓ using MCMC, as in subsection 3.1. We perform these DNN-based predictions using Adam optimiser that has a learning rate=0.001, loss function='mean_absolute_error', epoch=500.

All the four predictive plots from DNN are shown in Fig. 3. The comparative results of DNN and GP are presented in the Table 2 and it is to be noted that DNN results are sensitive to the choice of architecture, even for this simplistic data set. Such sensitivity is cause for worry when reliable and robust ML implementation is sought. For more complex real-world data sets with higher noise, such sensitivity is likely to be more pronounced. Also, DNN does not produce the (probability) predictions within the strict range of 0 and 1, while predictions following GP-based learning could be successfully constrained to lie in this interval.

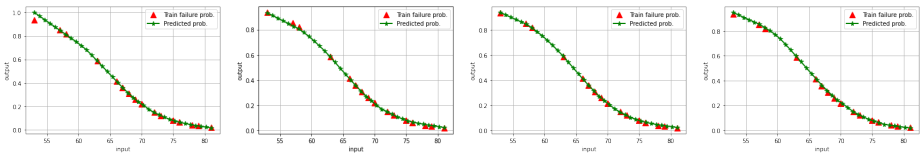


Fig. 3: DNN predictions with varying hidden layers, from left, DNN with 1, 2, 3 and 4 hidden layers with 64 neurons in each layer.

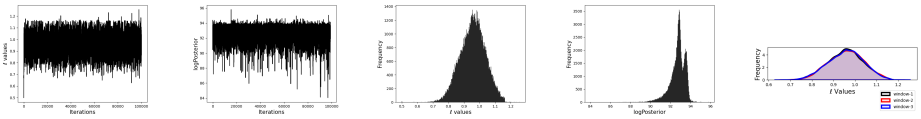


Fig. 4: Results as in Fig. 2, except obtained using D_{right}, from left: (a) to (e).

3.3 Learning ℓ Under Constraint, with MCMC, using D_{right}

Instead of using the training data D_{mean} one could potentially use any value of α and β from within 95% HPDs learnt on these parameters, in the chain run with test flight data D, to compute $\Pr(Y = 1|X = x_i)$ where x_i is the i-th design temperature; $i = 1, \ldots, M$. The training data that will then result, will not be D_{mean} since the output at any design temperature in D_{mean} is computed at the mean of the α and β samples generated in the MCMC chain run with test flight data D. As stated above, when we populate a training data by computing

$\Pr(Y = 1|X = x_i)$ using the value of α=15.19 and of β=-0.216 from the rightmost edge of their respective 95% HPD that is learnt in this MCMC chain, for $i = 1, \ldots, M$, we get the training set that we refer to as D_{right}. Results of GP-based learning of ℓ undertaken with training data D_{right} are presented in Fig. 4. From this chain, we learn the 95% HPD on ℓ to be [0.7999134, 1.11657139], with mean ℓ of 0.9582 approximately. Fig. 5b shows the GP prediction for this set up using $\ell = 0.9582$.

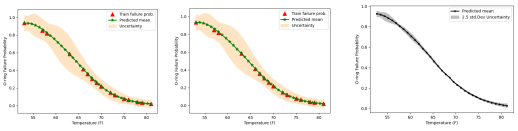
Prediction Once ℓ is updated in any iteration of the MCMC chain run with training data D_{train} - which could be D_{mean} or D_{right} - we predict the output W at a test input. These predicted mean outputs at each of the considered test temperatures are plotted against temperature, in green stars, in Fig. 5a, and Fig. 5b; W from the training data at a design temperature is depicted in these figures in red triangles, while the uncertainty predicted at a test input, on the output, is depicted as the salmon-pink shaded region. This depicted "uncertainty" is 2.5 times the standard deviation that is predicted - in a closed form way, along with the mean - at any test temperature. This is an advantage of performing learning with GPs; the output mean and variance predictions are closed-form. One concern that we have about this learning is that the choice of the training set that was employed in undertaking this learning appears ambiguous. In fact, percolation of the uncertainties in the learning of α and β is not correct in the formulation of these training data. In particular, when we examine the values of the predicted outputs and the uncertainties predicted on the outputs at any test temperature, we can firstly see that the result varies depending on which training data we have considered in the learning, and secondly, the result shows that some probabilities within the uncertainty bands, are predicted as being in excess of 1. This is in spite our 2-stepped prior that we

Table 2: Prediction of output W at test inputs - with DNNs of varying architectures, and following GP-based learning of hyperparameter (ℓ) of the covariance kernel that parameterises the GP covariance structure. HL: number of Hidden Layers in the DNN, N: number of Neurons.

1 HL, 64N	2 HL, 64N each	3 HL, 64N each	4 HL, 64N each	GP mean
1.0310167	0.9359667	0.938025	0.9486213	0.9323123
0.9939897	0.91862655	0.91935253	0.9286917	0.934071
0.9569628	0.90128636	0.90068007	0.9093453	0.926627
0.9199359	0.8839463	0.8820075	0.8906931	0.911141
0.88290906	0.86660624	0.863335	0.87204087	0.888929
0.8458822	0.84823	0.8443388	0.8533887	0.861319
0.8088551	0.8216592	0.818619	0.82862586	0.82952
0.7718283	0.79057	0.78710705	0.80185705	0.794533
0.7348015	0.7537405	0.75483435	0.77277607	0.757104
0.6977744	0.7143501	0.7184024	0.73662823	0.717731
0.6607475	0.6749598	0.6793759	0.69309384	0.676709
0.62372065	0.6355695	0.63738376	0.6467888	0.634208
0.5862205	0.5937305	0.59471357	0.599609	0.590366
0.5407498	0.5457597	0.54709804	0.55195147	0.545373
0.4952766	0.49778882	0.4992079	0.5042205	0.49954
0.44980314	0.44990715	0.45125672	0.45653468	0.453327
0.40433016	0.4031423	0.40382358	0.40979505	0.407338
0.3588567	0.35748953	0.35848558	0.36351866	0.362288
0.31549928	0.31303966	0.31533292	0.32044703	0.318933
0.27524063	0.2705538	0.27200228	0.2804457	0.277996
0.23979539	0.23742433	0.24069557	0.2418064	0.240096
0.20099688	0.19988464	0.20456474	0.2145113	0.205685
0.16196822	0.16679865	0.17303263	0.18275131	0.175016
0.14389752	0.14620048	0.14776252	0.15101965	0.148136
0.12673585	0.12363774	0.12680508	0.12768927	0.124907
0.10957434	0.10223782	0.1055288	0.10772515	0.105044
0.09241267	0.08653921	0.09068373	0.08748056	0.088173
0.07525112	0.07084078	0.07638919	0.0738593	0.073881
0.06405788	0.05796587	0.06551494	0.06084543	0.06177
0.05543705	0.04907831	0.0565593	0.0475015	0.05149
0.04681628	0.04019085	0.04897907	0.04064081	0.042759
0.03819548	0.03227264	0.04272956	0.03435563	0.035364
0.02957465	0.02511932	0.03683984	0.02806261	0.029157
0.02095388	0.0179661	0.03095011	0.0229567	0.024029
0.0123331	0.01081278	0.02506035	0.01791653	0.0198959

imposed to ensure that all output values stay restricted to [0,1]. Fig. 5a depicts the result of predictions using D_{mean} while Fig. 5b shows the same with D_{right}.

4 Pipe-Lined Architecture for Efficient Uncertainty Percolation



(a) $\ell \approx 0.9523$. (b) $\ell \approx 0.9582$ (c)

Fig. 5: (a,b) show GP predictions with 2.5 standard deviation of uncertainties with different ℓs with non pipe-lined architecture and pipe-lined architecture (c).

As we have seen in the previous section, predictions following GP-based learning can result in unacceptable uncertainties, and ambiguous results stemming from generation of training sets based on ambiguity in summarising uncertainties in the earlier stages. In this section we will illustrate a pipeline architecture for efficient handling of uncertainties which can also be extended to higher dimensional data. Fig. 6 represents the two approaches of uncertainty modelling and percolation as flow charts. The pipeline architecture consists of three blocks within the MCMC chain that we run. These blocks are executed sequentially within each iteration of the MCMC chain, so that uncertainty gets propagated through the stages without incorporating further errors that can creep in via arbitrary summarisation of learning outcomes in previous stages.

Broadly, inside an iteration of the MCMC chain, in the first block, α, β are updated - as within an iteration of Metropolis Hastings - using the test flight data D. In the second block of this iteration, the training data D_{train} is then computed at the current α and β values, and subsequently the GP kernel hyperparameter ℓ is updated using this current training data D_{train}, under the 2-stepped prior to ensure that output predictions remains within 0 and 1 at all considered temperatures. Then in the third block of this iteration, with the current value of ℓ, closed form mean and variance predictions of output values are undertaken, at each test temperatures. Hence, after the full iteration is over, we get traces of the predictive means at each test temperature. We use the range of values of the output $W \equiv \Pr(Y = 1|X = x_{test})$ sampled across the iterations, at each test input x_{test}, to compute the 95% HPD credible region. We compute the mean and the standard deviation of this sample, as the central prediction at test temperature x_{test}, with uncertainties of 2.5 times this sample standard deviation on either side of the mean, as the uncertainty in the prediction. The predicted variance at each test temperature is also recorded, and this is compared to the uncertainty of predictions obtained using the variation across MCMC samples.

First block: In an iteration of the MCMC chain, in the first block α, β are learnt using data D, with Metropolis Hastings, with the same configuration that is used in Subsection 2.1. The log of the posterior defined in this subsection is used. Gaussian priors are used for both α, β. Once, the burnin phase is over,

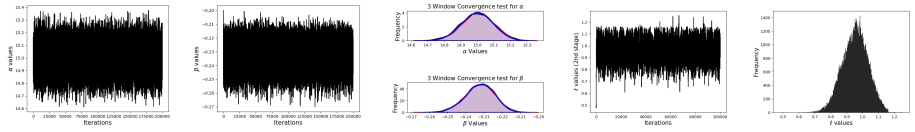
then next stages are switched on.

Second block: In this current iteration of the MCMC chain, the O-ring failure probability is computed at design temperatures $x_1, \ldots, x_M$ using the current values of α, β as in Section 2.1. This newly computed failure probability now becomes the output computed at a design input. When performed over the whole set of design inputs, the training data D_{train} is generated. this is employed in learning the GP kernel hyperparameter ℓ, as depicted in the earlier Subsection 3.1.

Third block: In the third block of this iteration, predictive mean and variance of the output at each test temperature is computed, using the current ℓ.

Fig. 7 represents the results of learning the parameters α and β, using the test flight data, within the MCMC chain - to update the training set $\{(x_i, w_i)\}_{i=1}^{M}$ - that simultaneously learns the hyperparameter (ℓ) of the covariance structure of the GP that models the functional relation between output W and input X. Here, the output W here is the probability of the O-ring failure and the input X is the ambient temperature to which such an O-ring is exposed. At each iteration of the MCMC chain, the uncertainty-included outputs are predicted at each test temperature. Thus, at the end of the MCMC chain, we predict values of mean and standard deviation of the output at each x_{test}; such predictions are shown in Fig. 5c. Traces of predicted values of the output W at few of the considered test temperatures are depicted in Fig. 8, along with uncertainties.

Algorithm 1 depicts the algorithm of the pipeline with the following inputs. $\alpha_0, \beta_0, \ell_0$ are seeds; $\sigma_\alpha, \sigma_\beta, \sigma_\ell$ are jump scales for α, β, ℓ respectively; μ_α, μ_β prior mean and $\sigma_\alpha^p, \sigma_\beta^p$ are prior standard deviation for α and β respectively, number of maximum iterations N; N_ℓ is starting iteration for learning ℓ, n number of test data points, ϵ noise in GP.



(a) Trace of α. (b) Trace of β. (c) C. test (d) Trace of ℓ. (e) ℓ histogram

Fig. 7: Traces and plots for learning α, β, ℓ in pipe-lined architecture; in c) C.:convergence test for α(top); β

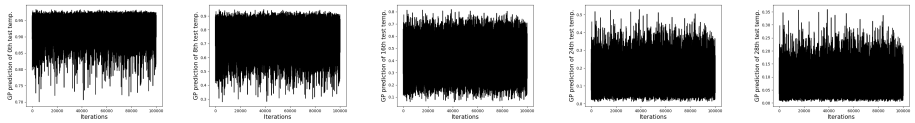
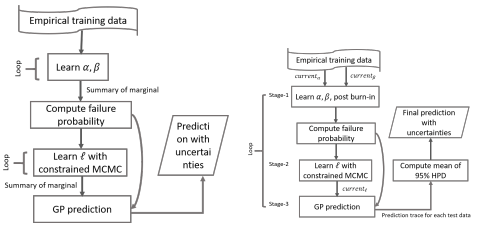


Fig. 8: Variation of predicted mean of the output W with iteration index, at few values of x_{test}, from left 0th, 8th, 16th, 24th and 28th test point.

5 Discussion and Conclusions



(a) Non pipe-lined. (b) Pipe-lined.

Fig. 6: Approaches of dealing uncertainty

In this work we present a robust method of uncertainty percolation in the context of a simple data situation; however the methodology is generic, and applicable to real-world complex datasets. This kind of integrated uncertainty percolation is useful for domains such as healthcare, where correct uncertainty acknowledgement is crucial. Results from our undertaken Bayesian approach have been compared to methods that consider uncertainty differently, and this comparison is the basic premise of the work. Importantly, we have also made comparison of our results with those obtained using against DNNs. In fact, the prediction performance in our work is benchmarked against DNN performance. Results obtained from DNNs are sensitive to the architectural parameters of the DNN. Indeed, the prediction made with DNNs could improve with further hyperparameter tuning; however, this very need for such tuning is our exact worry in regard to the employment of DNNs, when making prediction. We are not arguing against the possibility of enhancing prediction performance with DNNs; our quibble is that DNN prediction performance is sensitive to the choice of its architectural parameters, and when the correct answer is not known - as is always the case in real-world problems - we do not know which architectural details are optimal for the considered prediction task, given the data at hand. So the quality of the prediction made with DNNs is rendered questionable in general and tuning the DNN for a test case in a chosen data context, does not directly inform on how well such a tuned DNN will perform in a different data context. Such difficulties with DNN usage are corroborated by the works of [6] and references therein. As stated above, our small example can be generalised to broader real-world data situations. Indeed, depending on the architectural parameters, DNNs can produce diversely inaccurate predictions.

Via our simple illustration, we also show how the Bayesian setting allows for priors on the unknowns to facilitate the undertaken GP-based learning, such that the outputs - predicting which is the objective of the undertaken learning - can abide by relevant constraints.

References

1. Edmon Begoli, Tanmoy Bhattacharya, and Dimitri Kusnezov. The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23, 2019.
2. Siddhartha R Dalal, Edward B Fowlkes, and Bruce Hoadley. Risk analysis of the space shuttle: Pre-challenger prediction of failure. *Journal of the American Statistical Association*, 84(408):945–957, 1989.
3. Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112, 2009.

Algorithm 1: Pipeline for Forward Prediction

Set $\alpha[0] \leftarrow \alpha_0, \beta[0] \leftarrow \beta_0, \ell[0] \leftarrow \ell_0$, $\mathbf{x} \leftarrow$ Non-standardised input vector from D
$\mathbf{x^{std}} \leftarrow Standardise(\mathbf{x})$, $\mathbf{x_{test}} \leftarrow$ generate n test data within range of $\mathbf{x^{std}}$
/* Block-1: MH for learning α, β */
for $i \leftarrow 1$ **to** N **increment by** 1 **do**
  $\alpha_{current} \leftarrow \alpha[i-1], \beta_{current} \leftarrow \beta[i-1]$, $\alpha_{proposed} \sim \mathcal{N}(\alpha[i-1], \sigma_\alpha)$, $u \sim U(0,1)$
  $a \leftarrow log(\pi^{\mathrm{I}}(\alpha_{proposed}, \beta[i-1]|D, \pi_0^\alpha, \pi_0^\beta)) - log(\pi^{\mathrm{I}}(\alpha[i-1], \beta[i-1]|D, \pi_0^\alpha, \pi_0^\beta)$
  if $a > log(u)$ **then** $\alpha[i] \leftarrow \alpha_{proposed}$;
  else $\alpha[i] \leftarrow \alpha[i-1]$;
  $\beta_{proposed} \sim \mathcal{N}(\beta[i-1], \sigma_\beta)$
  $b \leftarrow log(\pi^1(\alpha[i], \beta_{proposed}|D, \pi_0^\alpha, \pi_0^\beta)) - log(\pi^1(\alpha[i], \beta[i-1]|D, \pi_0^\alpha, \pi_0^\beta)$
  if $b > log(u)$ **then** $\beta[i] \leftarrow \beta_{proposed}$;
  else $\beta[i] \leftarrow \beta[i-1]$;
  /* Block-2: Vanilla MCMC for learning ℓ */
  $\mathbf{y_p} \leftarrow ComputeFailureProbability(\alpha_{current}, \beta_{current}, D)$ as in Subsection 2.1
  $\mathbf{y_p^{std}} \leftarrow Standardise(\mathbf{y_p})$
  if $i >= N_\ell$ **then**
    $j \leftarrow (i - N_\ell), \ell_{current} \leftarrow \ell[j-1], \pi^{\mathrm{II}}_{current} \leftarrow \pi^{\mathrm{II}}[j-1], a_\ell \leftarrow -\infty, b_\ell \leftarrow +\infty$
    $\ell_{prop} \sim \mathcal{N}_T(\ell_{current}, \sigma_\ell, a_\ell, b_\ell)$
    $p \leftarrow f_{stepPrior}(\mathbf{x^{std}}, \mathbf{y_p}, \mathbf{x_{test}}, \ell_{prop}, \epsilon)$
    if $p == 1$ **then**
      $\ell_Q \leftarrow (log(\mathcal{N}_T(\ell_{current}|\ell_{prop}, \sigma_\ell, a_\ell, b_\ell)) - log(\mathcal{N}_T(\ell_{prop}|\ell_{current}, \sigma_\ell, a_\ell, b_\ell)))$
      $\pi^{\mathrm{II}}_{prop} \leftarrow log(\pi^{\mathrm{II}}(\ell_{prop}|\mathbf{y_p^{std}}, \mathbf{x}, \pi_0^\ell))$
      if $(\pi^{\mathrm{II}}_{prop} - \pi^{\mathrm{II}}_{current} + \ell_Q) > log(u)$ **then** $\ell[j] \leftarrow \ell_{prop}, \pi^{\mathrm{II}}[j] \leftarrow \pi^{\mathrm{II}}_{prop}$;
      else $\ell[j] \leftarrow \ell_{current}, \pi^{\mathrm{II}}[j] \leftarrow \pi^{\mathrm{II}}_{current}$;
    else
      $\ell[j] \leftarrow \ell_{current}, \pi^{\mathrm{II}}[j] \leftarrow \pi^{\mathrm{II}}_{current}$
    $\{\mathbf{y_o}, \sigma_{\mathbf{gp}}\} \leftarrow GPprediction(\ell[j], \mathbf{x^{std}}, \mathbf{y_p}, \mathbf{x_{test}}, \epsilon)$; // Block-3: prediction using GP

4. Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.
5. John K. Kruschke. Doing bayesian data analysis (second edition). In *Academic Press*, 2015.
6. Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(3):1–40, 2022.
7. Tanwi Mallick, Prasanna Balaprakash, and Jane Macfarlane. Deep-ensemble-based uncertainty quantification in spatiotemporal graph neural networks for traffic forecasting. *arXiv:2204.01618*, 2022.
8. R. M. Neal. Regression and classification using gaussian process priors (with discussion). In J. M. Bernardo et. al, editor, *Bayesian Statistics 6*, pages 475–501. Oxford University Press, 1998.
9. Anthony O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(1):1–24, 1978.
10. Carl Edward Rasmussen. *Evaluation of Gaussian processes and other methods for non-linear regression.* PhD thesis, University of Toronto Toronto, Canada, 1997.
11. Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.
12. Christian P. Robert and George Casella. Monte carlo statistical methods. In *Springer Texts in Statistics*, 2004.
13. H Jeff Smith, Tamara Dinev, and Heng Xu. Information privacy research: an interdisciplinary review. *MIS quarterly*, pages 989–1015, 2011.
14. Tensorflow. Basic regression: Predict fuel efficiency, 2022. https://www.tensorflow.org/tutorials/keras/regression.
15. Derek S. Young. *Handbook of regression methods.* Chapman and Hall/CRC, 2018.