14th International Conference on Applications of Statistics and Probability in Civil Engineering, ICASP14
Dublin, Ireland, July 9-13, 2023

Uncertainty-Aware Surrogates for Early Stage Design Prototyping

Archie Luxton
Data Scientist, Arup, London, United Kingdom

Dr. Gihan Weerasinghe
Senior Computational Scientist, Arup, Manchester, United Kingdom

Dr. Ramaseshan Kannan
Associate, Arup, Manchester, United Kingdom

ABSTRACT: Surrogate modelling is a powerful tool that can help to solve inverse problems and serve as a means for early stage rapid prototyping. It enables the creation of models that can be embedded in interactive visualisation environments that allow clients, architects, and other stakeholders to understand the performance of specific design configurations without necessarily setting up complex models. With the advancement of machine learning-based regression and the ease of access to tools, surrogate modelling has become increasingly accessible to practitioners. However, a major barrier to the widespread adoption of ML-based surrogates is the lack of firm accuracy guarantees. In this work, we develop uncertainty-aware surrogates using Bayesian inference. Our surrogates provide not only predictions, but also an associated uncertainty expressed through credibility intervals. By comparing the uncertainty with a threshold informed by the problem domain, the prediction can either be accepted or rejected, allowing practitioners to take advantage of the rapid computational speed without losing accuracy guarantees on unseen data. A key challenge is communicating this uncertainty to practitioners, as most engineers are not familiar with probabilistic machine learning. We demonstrate how the uncertainty-aware surrogate can be incorporated into a Rhino-based prototyping environment for developing geometries of steel-framed buildings, showcasing how the uncertainty can be effectively communicated to the user.

1. INTRODUCTION

Rapid prototyping in early project stages using digital tools is common across many disciplines within the built environment industry (see, for example, Gerber et al. (2012) or Holzer et al. (2007)). Designers and engineers often use visual programming environments such as Grasshopper/Rhino or Arup inForm, to rapidly generate geometries and configurations parametrically. The latter states in its documentation [1]:

> A parametric design framework that untangles the complex combination of project requirements and key performance indicators to deliver a data-driven design as a starting point to your project.

Aided by interactive and rich graphical interfaces, practitioners can obtain quantitative assessment of each option. Selected configurations are often iteratively refined in collaboration with the client or architect. The quantitative assessment used to select, reject, or refine design iterations can be a combination of discipline-specific objectives such as noise, lighting, wind loads, or structural assessment. A thorough and extensive exploration of the design

[1] `https://lively-water-096984503.1.azurestaticapps.net/introduction/`

space can create solutions that are more sustainable or meet the end design goals in more optimal ways such as by saving costs, constructibility or space utilisation.

Evaluating these objectives often relies on running computer simulations that solve an underlying physics-based problem and hence rely on mathematical modelling. Each time the user changes a design configuration in the interface, a new analysis is retriggered. The analysis must be resolved in near real-time to facilitate a smooth and coherent design discussion. On the other hand, if the simulations are computationally expensive, only a limited slice of the design space can be explored. Surrogate modelling can provide an effective solution to this challenge by resolving the analysis for each design iteration.

In this paper we present 'uncertainty-aware surrogates', which are surrogates with a twist in that they can produce both a prediction and the accompanying confidence in it. We demonstrate how these can be trained and embedded in design prototyping tools to radically accelerate the optioneering process without losing confidence in accuracy of its results.

1.1. Uncertainty-Aware Surrogates

The rise of machine learning (ML) has sparked significant interest in its use for surrogate modelling, as the authors have observed in their experience creating and distributing commercial digital tools for engineering simulation and design. ML-based surrogates, particularly neural networks for regression trained on simulation-generated datasets, are attractive due to their ease of use and accessible barriers to entry. This is due to their innate ability to generalize predictions without requiring a deep understanding of ML from the practitioner.

The disadvantage, however, is that we cannot reliably use the predictions from ML in safety critical scenarios. In the general case we cannot measure the accuracy of a prediction on an unseen data point from an ML model that is deployed in production[2]. Given this background, we can see a clear need for a surrogate that can communicate its confidence in a given prediction. In particular, an ideal surrogate should be able to communicate its limitations to the engineer who may not have knowledge about the information on which it has been trained.

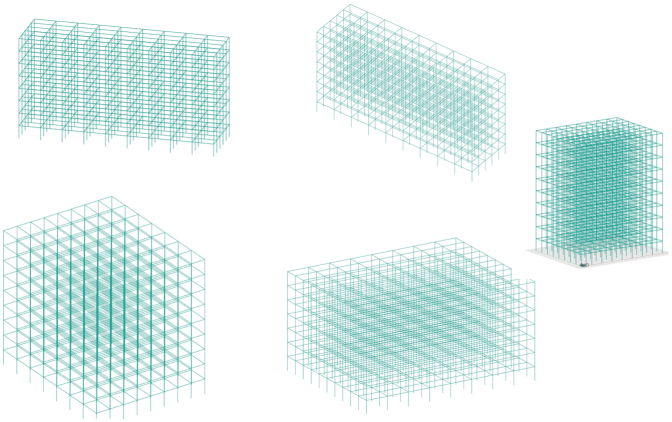


Figure 1: Frame configurations modelled in finite element software Oasys GSA

In this paper, we present the use of Bayesian machine learning to train uncertainty-aware surrogates for structural engineering prototyping. Our approach predicts a distribution of outcomes, rather than a single point prediction, to communicate the confidence in the prediction. Additionally, we present a novel method for deploying the uncertainty-aware surrogate to non-expert practitioners within the industry, ensuring its practicality and usability for those who may not possess extensive technical knowledge.

2. A SURROGATE FOR PREDICTING NATURAL FREQUENCIES

To demonstrate the training and deployment of an uncertainty-aware surrogate, we consider the problem of predicting the frequencies of fundamental vibration modes of a steel framed structure. We emulate a design prototyping scenario where a range of different frame geometries are examined, varying the bay dimensions and the number of bays as shown in Figure 1 and Table 2. Predicting the fundamental frequencies of the frames requires a modal analysis of the structure, which we accom-

[2]Noting that the probability assigned to classifications from a softmax output layer shouldn't be confused with an uncertainty, as a model can be uncertain in its predictions even with a high softmax output (Gal and Ghahramani (2016)).

plish with Oasys GSA[3], the commercial finite element structural software developed by Arup. To determine the vibration modes, Oasys GSA uses a sparse eigensolver to calculate the eigenvaulues and eigenvector (Oasys Software (2023)), with complexity scaling with $\mathcal{O}(n^2)$, where n is the number of nodes in the structure. The geometry of these framed structures can be wholly defined by the length 6 vector $\mathbf{X} = (X_0, X_1, ..., X_5)^T$, where each parameter details either *a):* the number of bays in each direction (x, y, z), or *b):* the length of members parallel to each direction (x, y, z):

Table 1: Input parameters to the surrogate

No. bays in direction:			Member length in direction:		
x	y	z	x	y	z
X_0	X_1	X_2	X_3	X_4	X_5

The geometry for such frames can be rapidly generated using Rhino, and subsequently (with suitable assumptions for boundary conditions and material/section properties) analysed in Oasys GSA using an application programming interface (API) to predict their frequencies. Each generated frame serves as a point in the training dataset.

3. Neural Networks and Uncertainty Quantification

3.1. Overview

Neural Networks are a popular mechanism to construct surrogate models as they can mimic complex, non-linear behaviour and can scale well to large amounts of high-dimensional data. The surrogates described here utilise simple feed-forward neural networks with a variety of modifications to improve their performance and allow them to produce credibility intervals with every prediction.

There is however a limitation: as mentioned previously, feed-forward neural networks are incapable of quantifying uncertainty in a prediction. There are many methods for endowing neural networks with uncertainty quantification, each having their own advantages and disadvantages. Notable methods include, among others, MC Dropout (Gal and Ghahramani (2016)), SGLD (Welling and Teh (2011)), Variational Inference (Jaakkola and Jordan (1999), Graves (2011)), Bayes by Backprop (Blundell et al. (2015)), and Probabilistic Backpropagation (Hernández-Lobato and Adams (2015)).

The simplest of these methods are MC Dropout and SGLD, both of which require minimal modifications to the training and inference routines, but still provide results with high accuracy and suitable credibility intervals. This inherent simplicity speeds up the development and deployment of the surrogates, and also more easily enables existing neural-network based surrogates to be retrofitted with uncertainty quantification abilities. These methods do not require the construction of priors, nor do they need to be fine-tuned to obtain reasonable reported credibility intervals.

The key property that MC Dropout and SGLD add to traditional feed-forward neural networks is stochasticity during both training and evaluation. When performing inference, i.e. obtaining a prediction $\mathbf{y}$ for a given input $\mathbf{x}$, multiple forward passes are performed, with stochasticity introduced by perturbing the model parameters θ_t in each pass. In SGLD, the weights are perturbed by injecting Gaussian noise, and in MC Dropout the pertubations arise due to a Bernoulli mask applied to the neurons. When performing a prediction, T forward passes are carried out through the networks (each parameterised by θ_t), with each forward pass acting as an additional sample from the predictive posterior distribution $p(\mathbf{y}|\mathbf{x})$[4]. The mean predicted value can then be calculated as the mean of all collected samples:

$$p(\mathbf{y}|\mathbf{x}) \approx \frac{1}{T}\sum_{t=1}^{T} p(\mathbf{y}|\mathbf{x}, \theta_t) =: q(\mathbf{y}|\mathbf{x}). \qquad (1)$$

Using each of the individual samples $p(\mathbf{y}|\mathbf{x}, \theta_t)$ that were obtained from the approximate predictive posterior, credibility intervals can be calculated using the highest density interval (HDI). Here, we use 95% HDI, describing the shortest interval that contains 95% of the total probability mass. See Section

[3] `oasys-software.com/gsa`

[4] When limited to a finite number of samples, this is termed the approximate predictive posterior $q(\mathbf{y}|\mathbf{x})$

4.4 for a brief summary of the investigation into the effect of number of samples T drawn versus the accuracy and credibility interval width.

3.2. MC Dropout

MC Dropout (Gal and Ghahramani (2016)) is a Monte-Carlo (MC) method which requires very little modification of existing training pipelines and network architecture, making it very suitable for scalable surrogates. It applies a Bernoulli distribution mask to every hidden layer neuron in the network, which gives each neuron a $p_{dropout}$ probability of having its output set to zero. $p_{dropout}$ is a constant value across all neurons in the hidden layers.

Dropout as a stochastic regularisation technique (SRT) was originally introduced by Hinton et al. (2012), rapidly proving its ability to reduce overfitting and improve the generalisation capabilities of neural networks. As it is a very popular regularisation method, it is already implemented in many modern architectures, potentially simplifying its use as an uncertainty quantification method.

The most important difference between dropout as a SRT versus MC Dropout as an uncertainty quantification method is the retaining of the Bernoulli distribution mask when carrying out inference on the network. Typically, the dropout mask is "turned off" after training has completed, but MC Dropout performs multiple forward passes on a network with dropout still enabled to effectively generate samples from the predictive posterior as explained in Equation (1).

MC Dropout operates similarly to an ensemble method (Gal (2016)), whereby a large number of networks are trained independently, then the result of forward passes through all networks is averaged to obtain a final result. Using ensembles of deterministic models is a bone-fide method for constructing BNNs, but it has a high memory footprint and the uncertainty it produces is inaccurate for OOD predictions, as shown by Osband et al. (2016).

Gal and Ghahramani show that the predictive log-likelihood and RMSE of predictions made using MC Dropout are considerably improved over state-of-the-art methods at that time (Variational Inference and Probabilistic Backpropagation).

3.3. Stochastic Gradient Langevin Dynamics (SGLD)

SGLD (Welling and Teh (2011)) combines the Stochastic Gradient Descent (SGD) optimisation method with Langevin Dynamics, an approach used to model the dynamics of molecular systems. By adding the right amount of noise to a SGD algorithm and by annealing the learning rate, Welling and Teh showed that the iterates converge to samples from the true posterior distribution. The Langevin dynamics portion of the method injects normally distributed noise (scaled according to the learning rate) into the weights of the network, resulting in the parameters converging to the full posterior instead of the maximum a posteriori mode.

In our implementation, the model's weights are saved at regular intervals during training, after an initial burn-in period. The injection of Langevin noise adds stochasticity to the network, so a forward pass of each save state of the model can form a sample from the approximate predictive posterior distribution.

In practice (for example, Palacci (2018)) SGLD is usually implemented by adding a noise term into a custom SGD optimiser, for example Adam (Kingma and Ba (2014)), however here we construct a method that can add the noise after any optimiser after the weights have been calculated. Equation 2 shows the addition of this Langevin noise to a typical SGD algorithm. If $\partial J(\theta_{t-1})/\partial \boldsymbol{\theta}$ is the gradient of the loss function at the previous iteration w.r.t. the model weights (calculated using backpropagation), then a simple SGLD update step becomes

$$\Delta\theta_t = \varepsilon_t \frac{1}{n} \sum_{i=1}^{n} \frac{\partial J(\theta_{t-1})}{\partial \boldsymbol{\theta}} + c\eta_t, \tag{2a}$$

$$\eta_t \sim N(0, \varepsilon_t) \tag{2b}$$

Where c is a noise multiplier term that serves as an additional hyperparameter. Note that here we use Root Mean Squared Error (RMSE) as our loss function as is common in evaluating regression models.

Welling and Teh (2011) specify that the learning rate ε should be annealed according to the Robbins-Monro conditions (Robbins and Monro (1951)) to reduce autocorrelation between samples, but many authors modify this annealing schedule or remove it altogether (e.g. Ahn (2015)). After some investigation into the effects of the annealing schedule, here we excluded it from the network to simplify hyperparameter tuning.

4. Methodology

4.1. Generating and Splitting Data

Oasys GSA was used to programmatically generate a corpus of $1,978$ framed structures with a range of geometries. Each of these structures can be wholly defined by $\mathbf{X} \in \mathscr{R}^6$ as previously detailed in Table 1. For each structure, the first 6 modal frequencies were calculated by GSA, ultimately generating a csv file with $1,978$ rows of 12 variables (6 inputs, 6 outputs).

The main corpus of $1,978$ structures is made up of multiple subsets of data, where each subset's geometry is defined by $X_n \sim \text{Uniform}(min_n, max_n)$. Greater amounts of data are generated for geometries that are defined to be 'in-domain' in order to maximise the size of the final training set, and smaller amounts of data are generated for the input space that is defined to be 'out of domain' for evaluation.

The full corpus is then split according to defined limits to guarantee the range of geometries contained in the training, validation and test sets. Maximising the amount of in-domain data emulates the process of acquiring data from a live system or historical data behaving normally, where testing against OOD geometries emulates the process of making a prediction on some new (or rarely seen) system configuration. See Table 2 for full details of the datasets used. All three sets are standardised before passing through the neural network.

Table 2: Corpus generated from GSA and training, validation and testing splits

Set	Ref	No. Points	Range $X\,[0-2]$	Range $X\,[3:5]$
Main corpus	A	1,000	$(3,10)$	$(5,10)$
	B	30	$(1,30)$	$(0.5,25)$
	C	50	$(1,15)$	$(0.5,15)$
	D	400	$(1,15)$	$(0.5,15$
	E	300	$(1,10)$	$(0.5,10)$
	F	200	$(1,5)$	$(0.5,10)$
Original Training	Train	929	$(3,10)$	$(5,10)$
	Val	104	$(3,10)$	$(5,10)$
	Test	863	$(0,3)$, $(10,25)$	$(0,5)$, $(10,12)$
	Unused	82	-	-
Retraining	Train	1,233	$(2,15)$	$(2,15)$
	Val	138	$(2,15)$	$(2,15)$
	Test	607	$(0,2)$, $(15,30)$	$(0,2)$, $(15,25)$

4.2. Hyperparameters and Training Regime

The surrogates described in this paper were trained using the main corpus of data, split according to the ranges defined in Table 1. The final hyperparameters used here (after tuning) are presented in Table 3.

The neural networks are constructed using PyTorch (Paszke et al. (2019)), and are are trained in two environments: on a notebook laptop[5] using CPU only, and in a free Google Colab notebook[6] using CUDA GPU acceleration. The learned model weights are saved to a file, allowing a user to load the models and perform inference very quickly. The Adam optimiser is used in both surrogate architectures.

4.3. Performing Inference

Once the models have been trained, inference can be performed[7]. A Flask app is started, which opens an API endpoint for the Grasshopper interface to communicate with. The neural networks are instantiated and the model weights are loaded from file when the Grasshopper interface is initialised, with every call to the API performing the required num-

[5]Windows 10 64-bit, Intel core i7-1165G7 Quad core 2.8GHz, 8Gb RAM, integrated Intel Iris Xe graphics

[6]`https://colab.research.google.com/`

[7]Note that here, inference refers to the machine learning definition (i.e. performing a prediction), rather than the Bayesian sense (integration over model parameters).

Table 3: Hyperparameters after tuning

Model	Parameter	Value
MC Dropout	Epochs	1,000
	$p_{dropout}$	0.3
	Hidden layers	2
	Neurons per hidden layer	100
	No. samples (T)	150
	Batch Size	50
	LR (ε)	5×10^{-4}
SGLD	Epochs	1,000
	Burnin epochs	250
	Weight decay (λ)	0
	Weight annealing	None
	Noise multiplier (c)	1
	Hidden layers	1
	Neurons / hidden layer	100
	No. samples (T)	150
	Batch size	50
	LR (ε)	1×10^{-3}

ber of forward passes through the network and returning the mean prediction and credibility interval. The models can also be instantiated and called in a pure Python environment.

4.4. Number of Samples Drawn vs Prediction Accuracy and Uncertainty

Using MC Dropout and SGLD, the mean prediction and credibility intervals are dependent on the number of samples drawn from the approximate predictive posterior distribution, so it is important to define a suitable number of samples to draw. We investigated how the number of samples drawn effected the predictions for in-domain and OOD geometries, with the number of samples ranging from $1-500$. A range of between $100-200$ was found to be sufficient to achieve a stable mean prediction and credibility interval width; any more samples than this and unnecessary computational cost is incurred, any fewer and the final result fluctuates excessively every time a round of inference is performed. Here we choose 150 samples per prediction to strike a balance between computational expense and reliability of results. Generally, we see that the predictions from SGLD – in terms of mean prediction and credibility intervals – are more stable with respect to the number of samples when compared to MC Dropout.

5. RESULTS

5.1. Deployment of Surrogate

A Grasshopper interface has been developed and deployed, allowing practitioners to parametrically design frame geometries (using sliders) that are instantly visualised in Rhino. Grasshopper serves as a rapid prototyping environment, where an engineer can very quickly perform preliminary design of structures.

For every configuration drawn, Grasshopper queries a RESTful API which carries out forward passes of the trained surrogate (leveraging MC Dropout or SGLD), returning a frequency prediction and an uncertainty. The relative uncertainty $(100 \times CI/\mu)$ is used to colour the structure; green indicates a high certainty, and red indicates a low certainty. The script also queries a larger deterministic reference neural network that has been trained on the full corpus to obtain a reference result closer to the ground-truth solution.

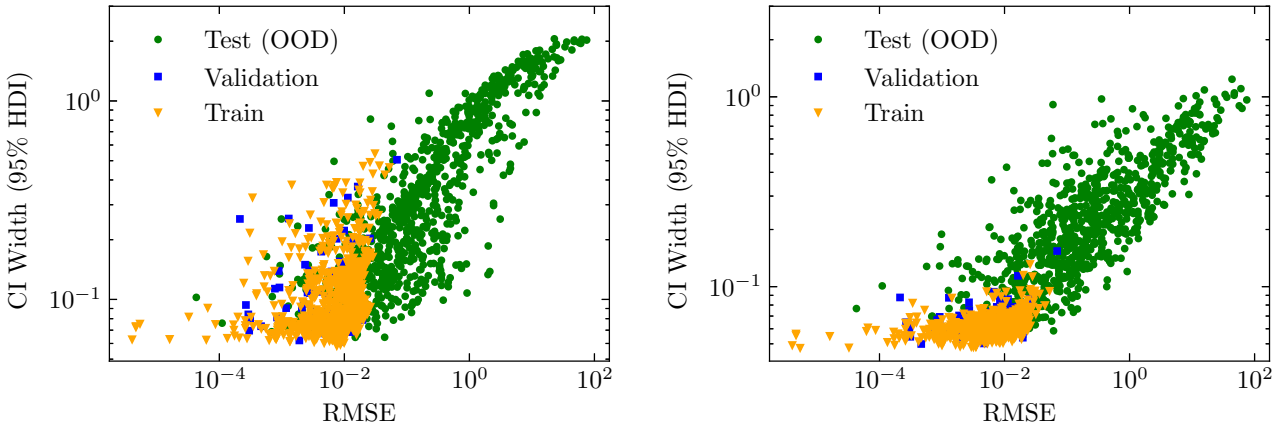
This interface is ideal for rapid prototyping of designs, allowing engineers to make informed decisions about a wide range of designs.

5.2. Timings and Scaling

5.2.1. Timings: Training

The MC Dropout-based surrogate finished training in approximately 200 seconds using CPU training on a notebook laptop, and approximately 45 seconds using CUDA GPU acceleration on Google Colab. The SGLD-based model finished training in 310 seconds using CPU and 156 seconds on GPU.

The difference in timings is primarily due to the additional computational and memory burden of SGLD introduced in two of its routines: *a)* the saving, storage and loading of 150 model states necessary for producing the samples from the predictive posterior; *b)* the loop that adds scaled Gaussian noise to every trainable model parameter at every epoch.



(a) MC Dropout *(b) SGLD*

Figure 2: Accuracy and credibility intervals of MC Dropout and SGLD on the training, validation, and OOD datasets. The credible interval is calculated using 95% HDI.

This additional computational complexity and memory consumption of SGLD scales linearly with the number of model parameters, i.e. $\mathcal{O}(n)$.

5.2.2. *Timings: Making Predictions*

The predictions made by the surrogates are fast enough to give the perception of being delivered in 'real time'. The MC Dropout-based model and the SGLD-based model all take on average between 35.8ms and 38.6ms to return a mean prediction and a credibility interval from the API, which is between 100 to $35,000\times$ faster than the GSA simulations (dependent on geometry size). The wall time is reduced further when running within a Python environment without API calls. The timings reported here are a result of 7 runs of 200 loops for each method.

5.3. *Analysis of Uncertainty Quantification and Accuracy*

Figure 2 shows the accuracy and credibility intervals of surrogates using MC Dropout and SGLD for the training, test and validation datasets.

Both surrogates demonstrate higher levels of uncertainty when making predictions on OOD inputs. Importantly, we observe that the credibility interval width increases as the RMS error increases. Generally the CI width is larger in MC Dropout for all datasets, but in particular it shows favourable performance in the test set, where the CI width increases at a faster rate with increasing RMSE compared to SGLD.

Considering its superior performance on OOD predictions and more efficient memory and computational requirements, MC Dropout may prove to be the preferred method in most cases. However it is worth noting that the credibility interval width from SGLD can be easily adjusted using the noise multiplication factor c introduced in Equation 2, which may be favourable in scenarios where the surrogate should be more conservative in its estimation of confidence. Increasing c in SGLD generates higher uncertainties on OOD data at the cost of reduced accuracy. Adjusting $p_{dropout}$ in MC Dropout produces a similar result, but with greatly reduced prediction accuracy and increased training times.

6. Conclusions

It is critical for surrogate models to be able to convey prediction uncertainty for them to be useful in commercial engineering workflows, particularly in safety critical applications such as structural engineering.

The surrogates described here can utilise MC Dropout or SGLD in feed-forward neural network architectures. The methods are shown to be fast - with speed-ups between $100\times$ and $30,000\times$ compared to full modal analysis using Oasys GSA. Since the surrogates have been developed using PyTorch, there is native CUDA GPU support, which

would provide substantial speed increases for larger surrogates using this framework. We have also demonstrated how the prediction confidence can be converted into a "traffic light" system that can be intergrated into an optioneering interface, thus giving the practitioner a tool to either accept prediction or reject it in favour of a full fidelity simulation. The speed of these surrogates allows the user to iterate over a vastly higher number of design iterations using the purpose-built Grasshopper interface.

Further work is ongoing to train these models in an online setting, as well as applying the techniques outlined here to more complex, non-linear modelling problems.

7. REFERENCES

Ahn, S. (2015). "Stochastic gradient mcmc: Algorithms and applications." Ph.D. thesis, Ph.D. thesis.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). "Weight uncertainty in neural network." *International conference on machine learning*, PMLR, 1613–1622.

Gal, Y. (2016). "Uncertainty in deep learning." Ph.D. thesis, University of Cambridge, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." *international conference on machine learning*, PMLR, 1050–1059.

Gerber, D. J., Lin, S.-H. E., Pan, B. P., and Solmaz, A. S. (2012). "Design optioneering: Multi-disciplinary design optimization through parameterization, domain integration and automation of a genetic algorithm." *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, SimAUD '12, San Diego, CA, USA, Society for Computer Simulation International.

Graves, A. (2011). "Practical variational inference for neural networks." *Advances in neural information processing systems*, 24.

Hernández-Lobato, J. M. and Adams, R. (2015). "Probabilistic backpropagation for scalable learning of bayesian neural networks." *International conference on machine learning*, PMLR, 1861–1869.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580*.

Holzer, D., Hough, R., and Burry, M. (2007). "Parametric design and structural optimisation for early design exploration." *International Journal of Architectural Computing*, 5(4), 625–643.

Jaakkola, T. S. and Jordan, M. I. (1999). "Variational probabilistic inference and the qmr-dt network." *Journal of artificial intelligence research*, 10, 291–322.

Kingma, D. P. and Ba, J. (2014). "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*.

Oasys Software (2023). *Oasys GSA Version 10.1 reference manual.* Arup, 13 Fitzroy Street London W1T 4BQ, <https://docs.oasys-software.com/structural/gsa/>. Available from `https://docs.oasys-software.com/structural/gsa/`.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). "Deep exploration via bootstrapped dqn." *Advances in neural information processing systems*, 29.

Palacci, H. (2018). "A look at sgd from a physicists's perspective - part 3, langevin dynamics and applications." *GitHub blog*. Accessed on 21/08/2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). "Pytorch: An imperative style, high-performance deep learning library." *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 8024–8035.

Robbins, H. and Monro, S. (1951). "A stochastic approximation method." *The annals of mathematical statistics*, 400–407.

Welling, M. and Teh, Y. W. (2011). "Bayesian learning via stochastic gradient langevin dynamics." *Proceedings of the 28th international conference on machine learning (ICML-11)*, Citeseer, 681–688.