A deep surrogate approach to efficient Bayesian inversion in PDE and integral equation models

Teo Deveney*, Eike Mueller†, Tony Shardlow‡
Department of Mathematical Sciences, University of Bath, Bath, BA2 7AY, UK

Abstract

We propose a novel deep learning approach to efficiently perform Bayesian inference in partial differential equation (PDE) and integral equation models over potentially high-dimensional parameter spaces. The contributions of this paper are two-fold: the first is the introduction of a neural network algorithm for approximating the solutions of Fredholm and Volterra integral equations of the first and second kind. The second is the description of a deep surrogate model which allows for efficient sampling from a Bayesian posterior distribution in which the likelihood depends on the solutions of PDEs or integral equations. For the latter, our method relies on the approximation of parametric solutions by neural networks. This deep learning approach allows for parametric solutions to be approximated accurately in significantly higher dimensions than is possible using classical techniques. These solutions are very cheap to evaluate, making Bayesian inference over large parameter spaces tractable for these models using Markov chain Monte Carlo. We demonstrate this method using two real-world examples; these include Bayesian inference in the PDE and integral equation case for an example from electrochemistry, and Bayesian inference of a function-valued heat-transfer parameter with applications in aviation.

*email: T.J.Deveney@bath.ac.uk
†email: E.Mueller@bath.ac.uk
‡email: T.Shardlow@bath.ac.uk
1 Introduction

Deep learning methodologies have seen significant development in the past few decades. The advent of efficient optimisation algorithms has led to state-of-the-art results in many high-dimensional tasks, particularly in settings where there is an abundance of data. Recently there has been much focus on the use of deep learning to address mathematical challenges involving differential equations. In the short time of just a few years, this has led to much progress, both in algorithmic ingenuity and theoretical understanding of neural network methodology applied to mathematical problems. For example we now have proof that deep neural networks can overcome the curse of dimensionality in the approximation of the solutions of certain classes of PDE [5, 11, 16], and efficient algorithms capable of achieving these approximations have been identified [12, 23, 8, 29]. These methods have already found applications in a range of areas such as fluid dynamics [24] and financial mathematics [34, 4].

Exploration of how these methods can further benefit researchers is well underway. Some developments include methods tackling optimal control problems by exploiting relationships with the Hamilton-Jacobi-Bellman equation [21, 15], and PDE based regularisation in regression tasks for which the governing laws are partially known [20]. In forward uncertainty quantification surrogate models have been proposed to solve PDEs with random coefficients [19], and probabilistic generative surrogate models to capture response uncertainty [35].

This work leverages recent algorithmic developments and extends them into new areas with two contributions:

- The development of a deep learning approach to solving integral equations,
- The design of a fast and accurate deep learning based surrogate methodology for Bayesian inversion in PDEs and integral equations.

PDEs are widely used to describe many physical processes with applications in areas such as fluids dynamics, atmospheric science, and quantum mechanics. Methods for solving these have received a lot of attention from the deep learning community. Integral equations are less well investigated, despite also having widespread applications in areas such as radiative transfer, viscoelasticity, and electrochemistry. We
will briefly review the physics-informed neural network [25] and the deep Galerkin method [29] of solving PDEs, before addressing integral equation problems by describing a deep learning algorithm to approximate the solutions of Fredholm and Volterra integral equations of the first and second kind [22].

Using these neural network based solvers for PDEs and integral equations we derive an efficient and accurate approach to Bayesian inverse problems [31]. This is based on a deep neural network surrogate model which is trained to approximately solve the PDE or integral equation over parameter space $\Theta \subset \mathbb{R}^p$ as well as input space $\Omega \subset \mathbb{R}^d$. The neural network approximation is then used as a surrogate model, which can then be coupled with a Markov chain Monte Carlo scheme in order to efficiently sample from an approximation of the posterior distribution of the parameters. By using neural networks the approximation of solutions spanning $\Omega \times \Theta \subset \mathbb{R}^{d+p}$ are tractable when $d + p$ is large [29, 12]. This is in contrast to traditional discretisation schemes, which would require $\mathcal{O}(N^{d+p})$ parameters, making them too expensive to execute for surrogate modelling of high dimensional PDEs or problems where there are many parameters.

For our forward model, we will first consider the situation where we have a given PDE

$$\mathcal{N}(u(x|\theta), x|\theta_N) = h(x|\theta_h) \quad x \in \Omega, \; \theta \in \Theta \quad (1)$$

with initial and boundary conditions defined by

$$u(x|\theta) = b(x|\theta_b) \quad x \in \partial \Omega, \; \theta \in \Theta. \quad (2)$$

Here $\theta = (\theta_N, \theta_h, \theta_b) \in \Theta$ is a parameter vector. We assume that $\mathcal{N}$ is a known (possibly non-linear) differential operator parameterised by $\theta_N$, and $h, b$ are known functions parameterised by $\theta_h$ and $\theta_b$. The parametric forward problem is to approximate the solution $u(x|\theta) : \Omega \times \Theta \to \mathbb{R}$ which satisfies this. If $\theta$ is fixed we write the solution as $u_\theta(x) : \Omega \to \mathbb{R}$. In the inverse problem our goal is to infer $\theta$ given some data samples $(\hat{x}_i, \hat{z}_i)_{i=1:M}$. This data consists of input data $\hat{x}_i \in \mathbb{R}^d$ and measured responses $\hat{z}_i \in \mathbb{R}$ corresponding to those inputs. This is a PDE inverse problem for scalar parameters.

In the integral equation case we will consider non-homogeneous equations of the
first kind

\[ 0 = v(x|\theta_v) + \int_a^{b(x)} k(x,y|\theta_k)u(y|\theta)dy, \quad x \in [a,b^*], \; \theta \in \Theta \]  

(3)

and the second kind

\[ u(x|\theta) = v(x|\theta_v) + \int_a^{b(x)} k(x,y|\theta_k)u(y|\theta)dy, \quad x \in [a,b^*], \; \theta \in \Theta. \]  

(4)

Here \( \theta = (\theta_v, \theta_k) \in \Theta \) is a parameter vector, and we assume \( v(x|\theta_v) \) is a given function parameterised by \( \theta_v \), and \( k(x,y|\theta_k) \) is a kernel function parameterised by \( \theta_k \). We will describe a deep learning algorithm that can be applied to approximate the fixed parameter solution \( u_\theta(x) \), and the parametric solution \( u(x|\theta) \) to these equations. Our approach will cover both Fredholm equations \((b(x) = b^*)\), and Volterra equations \((b(x) = x)\). Similarly to the PDE setting, the corresponding inverse problem is to identify the unknown parameters \( \theta = (\theta_v, \theta_k) \in \Theta \), given data samples of input-output pairs \((\hat{x}_i, \hat{z}_i)_{i=1:M}\).

In both the PDE and integral equation cases, we model the data using the solution of the equation with an additional error term

\[ z_i = u(x_i|\theta) + \epsilon_i. \]  

(5)

This is similar to the common scenario in parametric statistics

\[ z_i = f(x_i|\theta) + \epsilon_i. \]  

(6)

In statistical applications \( f(x_i|\theta) \) is typically a carefully constructed parametric function, designed using specialist knowledge to take input variables \( x_i \) and produce outputs \( z_i \) which model the mean behaviour of the system. In our case this function uses laws imposed by a PDE or integral equation to govern its behaviour. The last term is a random variable which accounts for divergences of the data from the mean. A standard choice for this, which we will adopt, is to assume unbiased i.i.d Gaussian deviations of the form \( \epsilon_i = N(0,\sigma^2) \). The objective is then to infer the unknown parameters \( \{\theta,\sigma^2\} \) such that the model fits the data.

This paper is organised as follows: In Section 2 of this paper we review some current deep learning schemes used to solve PDEs and estimate parameters, and then describe a new method based upon these which can be used to solve integral
equations. In Section 3 we outline the deep surrogate method; this is based upon using neural networks to accurately approximate the parametric solution to PDEs and integral equations. These surrogates are analytic, and allow for efficient approximate evaluations of the solution for different model parameters without need of further numerical solves; this makes a host of techniques from parametric statistics accessible. We focus on describing how we can perform Bayesian inference, by applying an MCMC scheme which uses surrogate evaluations to significantly reduce the computational cost per iteration. In Section 4 we give some examples with real applications, demonstrating accelerated Bayesian inference using the deep surrogate method. These examples cover PDE problems and integral equation problems, as well as the inference of scalar and function-valued parameters.

2 Solving PDEs and integral equations

2.1 Summary for PDEs

We will first consider the PDE model defined by (1, 2). Recent works in deep learning for PDEs [25, 29] have uncovered a simple method designed to solve such equations where \( \Omega \) is a closed domain. This leverages the fact that for a fixed \( \theta \in \Theta \) the following is non-negative, and zero only when \( u_\theta(x) \) is a solution

\[
\|N(u_\theta(x), x|\theta_N) - h(x|\theta_h)\|_{L^2(\Omega)}^2 + \|u_\theta(x) - b(x|\theta_b)\|_{L^2(\partial\Omega)}^2. \tag{7}
\]

Taking inspiration from (7), the method uses a neural network to approximate the solution of the PDE. Denoting the neural network approximation as \( \hat{u}_\theta(x) \), the algorithm works by using some variant of the gradient descent scheme to update the neural network parameters such that they minimise the loss function

\[
\frac{\nu_1}{N} \sum_{n=1}^{N} (N(\hat{u}_\theta(x^n), x^n|\theta_N) - h(x^n|\theta_h))^2 + \frac{\nu_2}{J} \sum_{j=1}^{J} (\hat{u}_\theta(y^j) - b(y^j|\theta_b))^2. \tag{8}
\]

Here \( x^n \in \Omega, y^j \in \partial\Omega \) are collocation points over which the evaluations for the gradient descent take place. The coefficients \( \nu_1, \nu_2 \) are weighting parameters which can be adjusted to reflect the relative importance of the terms. The descent algorithm depends on gradients, which are computed via back-propagation using automatic differentiation. This is applied to compute both the descent direction of the network parameters, and to apply the differential operator \( N \) to the neural network at the collocation points. Such computations can be routinely implemented using an
automatic differentiation package in software such as TensorFlow [1].

If the collocation points are fixed throughout training we arrive at the physics-informed neural network method in [25]. This method is based on a fixed mesh which requires a sufficiently fine resolution in order to yield accurate results, since the neural network must interpolate the solution between training points. In higher-dimensional spaces, this becomes prohibitively expensive since the number of collocation points required to saturate a unit domain grows exponentially with the dimension of the problem.

In contrast, if the collocation points are randomly sampled before each gradient descent iteration the minimisation of (7) results in the deep Galerkin method described in [29]. This mesh free approach has proven efficient at accurately approximating high-dimensional solutions since it allows for substantially fewer collocation points to be used at each iteration, instead relying on mini-batch stochastic gradient descent by sequentially changing the collocation points in order to adequately cover the domain. An outline of this algorithm is:

1. Initialise a neural network approximation of the solution, \( \hat{u}_\theta(x) : \mathbb{R}^d \rightarrow \mathbb{R} \)
2. Randomly sample \( (x^n)_{n=1:N} \in \Omega, (y_j)_{j=1:J} \in \partial \Omega \)
3. Using these sampled points, update neural network parameters by taking 1 gradient descent step to reduce loss function (8)
4. Repeat 2-3 until loss function (8) converges

An early approach to inverse problems using deep learning was proposed in [26]. Here the author suggests assimilating data by augmenting the loss function (8) with an additional term conceived to reduce the distance of the learned solution to the data. In this case the augmented loss function is

\[
\frac{\nu_1}{N} \sum_{n=1}^{N} (\mathcal{N}(\hat{u}_\theta(x^n), x^n | \theta_A) - h(x^n | \theta_h))^2 + \frac{\nu_2}{J} \sum_{j=1}^{J} (\hat{u}_\theta(y^j) - b(y^j | \theta_b))^2 + \frac{\nu_3}{M} \sum_{i=1}^{M} (\hat{u}_\theta(\hat{x}_i) - \hat{z}_i)^2. \tag{9}
\]
This loss function is then minimised with respect to both the parameters of the neural network, and the model parameters $\theta$ simultaneously. The addition of the final term encourages the optimiser to achieve model parameters which reduce the distance of the learned solution from the data.

This method has been shown to work well with large amounts of simulated data as there is no model error in this case. However there are significant downsides to this approach in a real world setting. Firstly in applications the PDE typically will not be a perfect representation of the data generating process. Thus by minimising (9) we train the network to strike the optimal balance between approximating the solution to the PDE, and interpolating the data. By setting $\nu_3$ to be small we can be confident of a good approximation to the PDE upon convergence. However in practice the data can have little influence on the descent algorithm in this case, leading to convergence to an accurate approximation of the solution, and model parameters which allow for fast convergence but don’t necessarily reflect the data. Alternatively if $\nu_3$ is large this function will typically not be a good approximation of the solution, as the first two terms will not be reduced sufficiently. In this case the first two terms act more as a physics-informed regulariser in a regression task as in [20]; this technique could help in making more robust predictions where some physical laws are known, however parameters estimated this way will be unreliable due to the deviation of the learned solution from the PDE.

2.2 Extension to integral equations

As an extension to the current literature we show how to leverage ideas from the last section in order to solve integral equations. Motivated by (3,4), we begin by defining the function

$$w(x, y|\theta_k) = \int_a^y k(x, \gamma|\theta_k)u_\theta(\gamma)d\gamma.$$  \hfill (10)

Using this (4) can be re-written

$$u_\theta(x) = v(x|\theta_v) + w(x, b(x)|\theta_k).$$ \hfill (11)

We will solve (11) by approximating the solution $u_\theta(x)$ by a neural network and minimising the residual in a similar fashion to the PDE case. Here there is an added complication which arises as $w(x, b(x)|\theta_k)$ is an integral involving the unknown solution $u_\theta(x)$. We overcome this by introducing a second integrator network denoted
\( \hat{w}_\theta(x, y) \), which is trained to evaluate this integral in the sense that it approximates the solution \( w(x, y|\theta_k) \) of the initial value problem

\[
\frac{\partial w_\theta}{\partial y}(x, y|\theta_k) = k(x, y|\theta_k)u_\theta(y),
\]

\( w_\theta(x, a|\theta_k) = 0. \) \( \tag{12} \)

By using an integrator network to approximate the integral term in this way, the differential equation techniques described in 2.1 are applicable. We use the two neural networks \( \hat{u}_\theta(x) \), \( \hat{w}_\theta(x, y|\theta_k) \) to approximate functions which satisfy (11) and (12). This is achieved by jointly optimising these networks to minimise a loss function of the form

\[
\frac{1}{N} \sum_{n=1}^{N} \left[ \nu_1 \left( \frac{\partial \hat{w}_\theta(x^n, y^n)}{\partial y} - k(x^n, y^n|\theta_k)\hat{u}_\theta(y^n) \right)^2 + \nu_2 \hat{w}_\theta(x^n, a|\theta_k)^2
+ \nu_3 (\hat{u}_\theta(x^n) - v(x^n|\theta_v) - \hat{w}_\theta(x^n, b(x^n)|\theta_k))^2 \right] \quad \tag{13}
\]

An algorithm for solving integral equations of the second kind on the domain \( x \in \Omega \subset \mathbb{R} \) is then:

1. Initialise a neural network approximation of the solution, \( \hat{u}_\theta(x) : \mathbb{R} \rightarrow \mathbb{R} \)
2. Initialise an integrator network, \( \hat{w}_\theta(x, y|\theta_k) : \mathbb{R}^2 \rightarrow \mathbb{R} \)
3. Randomly sample \( (x^n)_{n=1:N} \in [a, b^*], (y^n)_{n=1:N} \in [a, b(x_n)] \)
4. Using these sample points, jointly update both neural networks by taking 1 gradient descent step to reduce loss function (13)
5. Repeat 3-4 until convergence

The algorithm for the solving integral equations of the first kind in (3) is similar, with (11) becoming

\[
0 = v(x|\theta_v) + w(x, b(x)|\theta_k), \quad \tag{14}
\]

and the corresponding loss function is

\[
\frac{1}{N} \sum_{n=1}^{N} \left[ \nu_1 \left( \frac{\partial \hat{w}_\theta(x^n, y^n)}{\partial y} - k(x^n, y^n|\theta_k)\hat{u}_\theta(y^n) \right)^2 + \nu_2 \hat{w}_\theta(x^n, a|\theta_k)^2
+ \nu_3 (v(x^n|\theta_v) + \hat{w}_\theta(x^n, b(x^n)|\theta_k))^2 \right]. \quad \tag{15}
\]
An application of this algorithm is shown in an electrochemistry example in Section 3.

Similarly to the PDE based problem we could attempt to solve the inverse problem by augmenting this loss function with an additional term to encourage the learned solution to match the data. The minimisation would then be executed with respect to the parameters of the two neural networks and the model parameters $\theta$ simultaneously. For equations of the second kind the resulting loss function has the form

$$
\sum_{n=1}^{N} \left[ \nu_1 \left( \frac{\partial \hat{u}_\theta}{\partial y}(x^n, y^n) - k(x^n, y^n | \theta_k) \hat{u}_\theta(y^n) \right)^2 + \nu_2 \hat{w}_\theta(x^n, a | \theta_k) \right] + \nu_3 (\hat{u}_\theta(x^n) - v(x^n | \theta_v) - \hat{w}_\theta(x^n, b))^2 + \nu_4 \sum_{i=1}^{M} (\hat{u}_\theta(\hat{x}_i) - \hat{z}_i)^2.
$$

(16)

This method of inferring parameters suffers from the same issues as described at the end of Section 2.1 for PDE based problem.

3 Deep surrogate approach for Bayesian inference

The benefits of neural network based techniques are most evident when the function which is being approximated is high-dimensional. This is due to the ability of these networks to accurately approximate high-dimensional functions using few parameters, relative to more traditional function approximators. This principle has theoretical grounding for multiple classes of PDE, due to proof that the number of parameters required to approximate a solution grows at most polynomially with the inverse of required error and the dimension of the problem [5, 11, 16]. In contrast the number of parameters required by a linear function approximator to achieve the same will typically grow exponentially. Because of this, deep learning techniques have shown great success in the solutions of PDEs with dimensions of up to 200 [12, 23, 8, 29], which would have been intractable to solve with traditional techniques. Here we show how we can benefit from these high-dimensional capabilities in Bayesian inference by using neural networks to approximate parametric solutions $u(x | \theta) : \Omega \times \Theta \rightarrow \mathbb{R}$; that is solutions which can be evaluated for any input values $x \in \Omega$ and model parameters $\theta \in \Theta$. 

9
3.1 Bayesian Inference

Bayesian Inference is a means of inferring the distribution of a set of parameters $\tilde{\theta}$, given observed data $\tilde{z}$ using Bayes’ rule. It calculates the posterior probability of an event, as a consequence of a prior distribution, a likelihood function, and some observed data. The Bayesian approach allows us to make inferences about model parameters through analysis of the posterior distribution via Bayes’ rule

$$p(\tilde{\theta} | \tilde{z}) \propto p(\tilde{z} | \tilde{\theta}) \times p(\tilde{\theta}).$$

The prior distribution, with density $p(\tilde{\theta})$, is chosen to describe ones existing knowledge about the parameters prior to data being observed. The likelihood, $p(\tilde{z} | \tilde{\theta})$ is the conditional density of the data $\tilde{z}$ given parameters $\tilde{\theta}$ according to some model. The product of the prior and likelihood is proportional to the posterior density, $p(\tilde{\theta} | \tilde{z})$. This density represents an updated belief about the unknown parameters $\tilde{\theta}$, having observed the data $\tilde{z}$.

In general the exact functional form of the posterior cannot be directly computed and thus must be approximated. A popular method in statistics and data science is to apply a Markov chain Monte Carlo (MCMC) scheme to sample from the posterior distribution over the parameters $\tilde{\theta}$ [18, 30]. Each iteration of an MCMC scheme requires the evaluation of the prior density and likelihood function. The prior density is known and usually easily evaluated. The likelihood function depends on the model and the data. In our setting the model is (5), the data is $\tilde{z} = (\hat{x}, \hat{z})$, and the parameters we wish to infer are $\tilde{\theta} = \{\theta, \sigma^2\}$. The likelihood function in this case takes the form

$$p(\tilde{z} | \tilde{\theta}) = \frac{1}{(2\pi\sigma^2)^{K/2}} \exp \left( -\frac{1}{2\sigma^2} \sum_{i=1}^{M} (\hat{z}_i - u_\theta(x_i))^2 \right).$$

In the Bayesian inverse problem literature MCMC methods are sometimes used, however the computational cost of executing an MCMC iteration includes the evaluation of $u_\theta(x_i)$ within (18) for each instance of $\theta$; this requires an expensive numerical forward solve if traditional grid based numerical solvers are employed. Many MCMC iterations are needed to obtain a sufficient posterior sample, therefore this repeated solving leads to a time consuming procedure to execute accurately on all but the simplest models. Multiple approaches have been devised to improve efficiency of MCMC methods for inverse problems, such as multilevel methods [7],

10
the use of inexpensive coarse scale models as pre-conditioners for the fine scale model [10, 9], surrogate models based on interpolation [6], and surrogates based on parametric approximations to the solutions [28]. These have all been successful in accelerating MCMC for specific applications, however each is restricted either by the dimensionality of the problems that they can be applied to, or the accuracy of the surrogates that they construct. Our method aims to address this by augmenting the latter approach with a neural network surrogate which can efficiently approximate accurate solutions to high-dimensional problems.

3.2 Deep parametric solutions as surrogate models

A surrogate model is a function \( \hat{u}(x|\theta) : \Omega \times \Theta \rightarrow \mathbb{R} \) which takes model parameters \( \theta \in \Theta \), and inputs \( x \in \Omega \) as arguments, and outputs an approximation to the solution of the model for those values. Surrogates can typically be efficiently evaluated for any instance of inputs and model parameters, leading to them becoming a popular tool in uncertainty quantification and Bayesian inverse problems. Researchers who have taken advantage of this approach have been successful in accelerating Bayesian inference in differential equation models by pairing the surrogate model with an MCMC scheme; allowing them to efficiently sample from an approximation to the posterior distribution [28, 13, 6]. The benefit of using a surrogate is that the likelihood evaluation step of Monte Carlo algorithms — a bottleneck which typically requires a PDE solver to be run each time — becomes a cheap evaluation of a known analytic function. Additionally, the decoupling of the computation of the solution from MCMC scheme means that once the surrogate is constructed, it can be saved and used to do inference with different data sets without the need to re-solve the PDE.

Surrogate models which accurately approximate parametric solutions are typically expensive to compute. The most common approach is to use a generalised polynomial chaos representation [33, 17], however this suffers from the curse of dimensionality, and has such been restricted to applications with few model parameters [28, 13]. Data driven surrogates have been applied in cases where the number of parameters is larger (at around 6-10). These usually apply traditional solvers to generate training data for several instances of \( \theta \). The surrogate is then constructed by solving a regression problem using the generated data. This allows higher-dimensional param-
eter spaces to be explored, but relies on interpolants which typically won’t satisfy
the required equations, and may be inaccurate in complicated regions of the domain
[35]. Here we propose a new surrogate model for Bayesian inference, which uses a
neural network to approximate the parametric solution. This allows the accurate
approximation of surrogates which satisfy the equations in higher dimensions with
a modest computational budget.

The neural network surrogate \( \hat{u}(x|\theta) \) is obtained by extending the methods in Sec-
section 2. In this case we treat the model parameters analogously to the input variables,
meaning each iteration we sample over the combined space \( \Omega \times \Theta \subset \mathbb{R}^{d+p} \) and use
gradient descent to minimise the loss function corresponding to the problem. For
the PDE based problem this loss function is

\[
\begin{align*}
\nu_1 & \frac{1}{N} \sum_{n=1}^{N} (N(\hat{u}(x^n|\theta^n), x^n|\theta^n_h) - h(x^n|\theta^n_h))^2 + \\
\nu_2 & \frac{1}{J} \sum_{j=1}^{J} (\hat{u}(y^j|\phi^j) - b(y^j|\phi^j))^2.
\end{align*}
\]

(19)

An algorithm to construct the parametric solution is then:

1. Initialise a neural network approximation of the parametric solution, \( \hat{u}(x|\theta) : \mathbb{R}^{d+p} \rightarrow \mathbb{R} \)

2. Randomly sample \( (x^n)_{n=1:N} \in \Omega, (y^j)_{j=1:J} \in \partial \Omega, \)
\( (\theta^n, \theta^n_h, \theta^n_b)_{n=1:N} \in \Theta, (\phi^j, \phi^j_h, \phi^j_b)_{j=1:J} \in \Theta. \)

3. Using these sampled points, update neural network parameters by taking 1
   gradient descent step to reduce loss function (19).

4. Repeat 2-3 until loss function converges

When solving integral equations, the same adaptation is made to compute paramet-
ric solutions. In this case we augment the algorithm in Section 2.2 by sampling over
the parameter space in addition to the input space, and using these samples to train
both \( \hat{u}(x|\theta) \) and \( \hat{w}(x,y|\theta) \). The resulting neural network approximates a parametric
solution. Such parametric solutions can then be used as surrogate models within an
MCMC scheme for Bayesian inference.

In the next section we give some examples to demonstrate that neural networks
allow us to compute computationally efficient and accurate parametric solutions,
which can then be used in conjunction with an MCMC sampler to significantly
accelerate Bayesian inference.
4 Examples

4.1 Voltammetry Example

Voltammetry is an experimental technique used in electrochemistry to infer properties of a chemical. A standard experiment is to apply a potential to an electrode in an electrochemical cell. This causes electron transfer, which is measured as an electrical current. The current measurements can then be used to deduce information about the chemical system. Here we focus on the rate of reaction taking place at the electrode surface, which is dependent on the concentration of a chemical species. The inverse problem we will consider is to infer the formal potential $E^0$ of the chemical. This quantity is related to the amount of energy which is required to stimulate a reaction. This problem can be posed as a PDE problem or as an integral equation [3].

4.1.1 PDE Approach

To construct a PDE model for this problem we will assume a 1-dimensional model where chemicals are transported only through diffusion

$$\frac{\partial C_s}{\partial t} = D_s \frac{\partial^2 C_s}{\partial x^2}. \quad (20)$$

Here $x \in \Omega = \mathbb{R}^+$ is the distance from the electrode surface and $C_s$ represents the concentration of the chemical species $s$. $D_s$ is the scalar diffusion coefficient of species $s$.

The experiment we will consider has 2 species, $A$ and $B$. It begins with only chemical $A$ present over the whole domain. We assume that the oxidation reaction $A - e^- \rightarrow B$ (the loss of an electron from $A$) takes place at the electrode surface at a rate dependent on the intensity of the applied potential. For simplicity we assume that $A$ and $B$ have the same diffusion coefficient then after non-dimensionalising appropriately we have a system of equations

$$\frac{\partial a}{\partial t} = \frac{\partial^2 a}{\partial x^2}, \quad x, t \in \mathbb{R}^+, \quad (21)$$

$$\frac{\partial b}{\partial t} = \frac{\partial^2 b}{\partial x^2}, \quad x, t \in \mathbb{R}^+. \quad (22)$$
Here $a$ and $b$ represent the concentrations of chemicals $A$ and $B$. These are coupled, with initial conditions

$$a(x,0) = 1, \quad x \in \mathbb{R}^+, \quad (23)$$
$$b(x,0) = 0, \quad x \in \mathbb{R}^+, \quad (24)$$

and far field boundary conditions for $t \in \mathbb{R}^+$

$$a(x,t) \to 1, \quad x \to \infty, \quad (25)$$
$$b(x,t) \to 0, \quad x \to \infty. \quad (26)$$

We assume conservation of matter, so that

$$a(x,t) + b(x,t) = 1, \quad x,t \in \mathbb{R}^+. \quad (27)$$

This is imposed through the boundary condition

$$\frac{\partial a}{\partial x} + \frac{\partial b}{\partial x} = 0, \quad x = 0, t \in \mathbb{R}^+, \quad (28)$$

therefore it is sufficient to solve for just $a(x,t)$. The last boundary condition at $x = 0$ depends on how the potential is applied. In linear sweep voltammetry a linearly increasing current of the form

$$E(t) = E_{start} + t \quad (29)$$

is applied. It can be shown that this leads to a boundary condition of

$$a(0,t) = \frac{1}{1 + e^{E(t) - E_0}}, \quad t > 0. \quad (30)$$

The current is measured as the rate of reaction at the electrode surface

$$I(t) = \frac{\partial a}{\partial x} \bigg|_{x=0}. \quad (31)$$

The inverse problem is then to infer the formal potential $E^0$, given current measurements from experiments.

### 4.1.2 Integral Equation Approach

This problem can also be posed as an integral equation by taking a Laplace transform. Letting $\tilde{a}(x,s)$ denote the Laplace transform of $a(x,t)$ with respect to $t$, it can be shown that

$$\tilde{a}(x,s) = c_1(s)e^{-\sqrt{s}x} + \frac{1}{s}. \quad (32)$$
To calculate the current we take the Laplace transform of (31), giving

$$\tilde{I}(s) = \frac{\partial \tilde{a}}{\partial x} \bigg|_{x=0},$$

(33)

where $\tilde{I}(s)$ denotes the Laplace transform of $I(t)$. Differentiating (32) with respect to $x$, and equating it to (33) with $x = 0$ we deduce

$$c_1(s) = -\frac{\tilde{I}(s)}{\sqrt{s}},$$

(34)

so that we have

$$\tilde{a}(0, s) = -\frac{\tilde{I}(s)}{\sqrt{s}} + \frac{1}{s}.$$  

(35)

To recover $a(0, t)$ from (35) we invert the Laplace transform using the convolution theorem, and the identities: $\mathcal{L}^{-1}\left(\frac{1}{\sqrt{s}}\right) = \frac{1}{\sqrt{\pi t}}$, $\mathcal{L}^{-1}\left(\frac{1}{s^2}\right) = 1$. This gives

$$a(0, t) = 1 - \frac{1}{\sqrt{\pi}} \int_0^t I(\tau) \cdot \frac{1}{\sqrt{t-\tau}} d\tau.$$  

(36)

Together with the boundary condition (30) this gives rise to the Volterra integral equation of the first kind

$$\sqrt{\pi} \frac{1}{1 + e^{-(E(t) - E_0)}} = \int_0^t \frac{I(\tau)}{\sqrt{t-\tau}} d\tau.$$  

(37)

The solution $I(t)$ of this integral equation is the same as the gradient at the boundary under the PDE construction defined in (31). The objective here is the same as before, that is we wish to infer the formal potential $E_0$ given measurements of the current from experimentation.

4.1.3 Results

The neural network based surrogate model described in Section 3 are used to approximate parametric solutions to both the PDE and integral equation problems. We use $E_{\text{start}} = -10$, and truncate the domain to $(x, t) \in [0, 200] \times [0, 20]$. The neural networks were constructed and trained using TensorFlow [1]. For the PDE problem, a fully connected neural network with 2 hidden layers of 45 neurons is used to parameterise the spatio-temporal domain of the solution. This neural network defines a parametric solution by taking $E_0$ as an input, and outputting coefficients, means, and variances of 1000 Gaussian radial basis functions. The linear combination of these basis functions approximates the solution $a(x, t | E_0)$ to the PDE in
the spatio-temporal domain for the given $E^0$. This architecture was adopted since radial basis solutions attained much better accuracy for this equation than the more common fully connected or residual architectures applied in [25, 29]. The approximation to the current is recovered from the solution by computing the gradient at the boundary using automatic differentiation. For the integral equation the approximation of the solution and the integrator network are both fully connected neural networks, each with 4 hidden layers of 45 neurons. The training time for the networks in both cases was less than one hour.

As reference solutions we numerically solve the integral equation by approximating the integral in (37) by a midpoint quadrature rule and solving the resulting system of linear equations. To achieve a high accuracy we apply this with a very small discretisation width of $\Delta t \approx 5.7 \times 10^{-4}$. These reference solutions are used as reliable approximations of the true solutions. For $E^0 = \{-4,0,4\}$, Figure 1 compares these
reference solutions to the parametric solutions achieved by the surrogates. In each case the neural network solutions are close to the references.

Simulated data generated by adding Gaussian noise to the reference solutions at 100 random time points is shown in the first row of Figures 2 and 3. The second rows show the posterior distributions for $E^0$ obtained by combining this data with the surrogate models and using a Metropolis–Hastings sampler [18]. The prior distribution was uniform $E^0 \sim U([-6,6])$ for all examples. The posterior distributions of $\sigma^2$ were also estimated for each example using truncated normal priors. These gave estimates consistent with the true noise levels, however we omit their plots.

Figure 2: Bottom: Approximate posterior distributions and 95% credible intervals computed using deep PDE surrogate. The true parameter values used to generate the data are marked by vertical lines. Top: Data used and solutions fitted using the mean of the posterior. The true standard deviations used to generate the Gaussian noise are given at the top of each column.
We observe that the mean of the MCMC sample is close to the true parameter used to generate the data, and the width of the credible intervals reflect the noise levels of the data as we would expect. 500,000 iterations of the Metropolis–Hastings scheme were used to sample from the posterior, and kernel density estimation [27] was used to visualise the distributions. In each case the proposal distributions were tuned to achieve acceptance rates close to 40%. The time to generate this sample size was no longer than 10 minutes in both the PDE and integral equation case, implying that we were able to probe the forward map over 50,000 times per minute. This was achieved by executing 10 MCMC samplers in parallel by using TensorFlow to distribute the computation to a mobile RTX 2080 graphics processing unit.

Figure 3: Bottom: Approximate posterior distributions and 95% credible intervals computed using integral equation surrogate. The true parameter values used to generate the data are marked by vertical lines. Top: Data used and solutions fitted using the mean of the posterior. The true standard deviations used to generate the Gaussian noise are given at the top of each column.
4.2 Inferring the non-constant Biot number in rotating discs

Heat conduction in rotating-disc systems is a conjugate problem; the convective heat transfer from fluid particles and the diffusive heat transfer in the solid are coupled. This phenomenon can be modelled using the Fin equation [14]. The relative effects of diffusion and convection are related to a spatially varying parameter known as the Biot number. It is of interest in aviation applications to estimate the Biot number in order to control for material expansion in aircrafts.

In a non-dimensional setting, we could consider a rotating disc with inner and outer radii \(a\) and \(1\), the Fin equation for this case is

\[
\frac{d^2 u}{dx^2} + \frac{1}{x} \frac{du}{dx} - Bi \ u = 0, \quad x \in [a, 1],
\]

\[
u(a) = u_a, \]

\[
u(1) = u_1. \tag{38}
\]

Here \(u = u(x)\) is a non-dimensional temperature, \(Bi = Bi(x)\) is the Biot number, \(x\) represents radial distance from the centre of rotation, and \(u_a, u_1 \in \mathbb{R}\) are fixed Dirichlet boundary conditions. Analytical solutions to this equation are intractable when the Biot number is non-constant, however (38) can still be solved numerically in this case. The inverse problem is to infer the functional form of \(Bi(x)\) given noisy measurements of the temperature profile \(u(x)\).

In principle, if the radial distribution of the temperature is known then \(Bi(x)\) can be determined numerically. But this is an example of an ill-posed inverse problem where very small uncertainties in the temperature measurements can create large uncertainties in the computed Biot number. This feature of the inverse problem has caused some early research to infer curves with large unrealistic values for \(Bi(x)\) [2]. Similar issues are raise if the naive deep learning scheme of minimising (9) is applied as Figure 5 shows. A maximum a posteriori approach to this problem indicated that a Bayesian approach can be robust to this sensitivity and allow for reliable inferences [32]. We apply the deep surrogate method to give this problem a complete Bayesian treatment.
4.2.1 Results

We approach the inference of $\text{Bi}(x)$ using the framework developed in this paper by using some parameterised function $\tilde{\text{Bi}}(x|\theta)$ to represent the class of possible Biot numbers. Then we use a neural network to approximate a parametric solution $\hat{u}(x|\theta)$ of the PDE, in terms of the parameters of $\tilde{\text{Bi}}(x|\theta)$. Specifically we use a linear combination of monomials up to degree 15 to represent the Biot number

$$\tilde{\text{Bi}}(x|\theta) = \sum_{n=0}^{15} \theta_n x^n, \quad x \in [a, 1], \theta \in \Theta.$$  \hspace{1cm} (39)

Given this representation we approximate the parametric solution $u(x|\theta)$ by minimising (19). For this we used a fully connected neural network with a 17-dimensional input layer with arguments $x \in \mathbb{R}$ and $\theta \in \mathbb{R}^{16}$, 4 hidden layers with 45 neurons per layer, and a one-dimensional output to approximate $u(x|\theta)$. Symmetric uniform priors were placed on the coefficients of the form $\theta_n \sim U(-d_n, d_n)$, where $d_0 = 20, d_1 = 20, d_{n+1} = d_n/2$. This decay in the prior support of the coefficients regularises the smoothness of the functions in the posterior distribution, whilst also constraining the parameter space over which the parametric solution is approximated to $\Theta = [-d_0, d_0] \times [-d_1, d_1] \times \ldots \times [-d_{15}, d_{15}]$. Using these priors it took less than one hour to train the neural network to approximate a parametric solution over this domain.

The neural network $\hat{u}(x|\theta)$ is then used as a surrogate model and coupled with a Metropolis–Hastings sampler to approximately sample from the posterior. The left of Figure 4 shows data generated by solving (38) with $\text{Bi}(x) = 18e^{x-0.3}$, and adding Gaussian noise with standard deviation $\sigma = 0.003$ at 30 equidistant points. Overlaid on this is the solution fitted by solving (38) using the $\tilde{\text{Bi}}(x|\theta)$ estimated by the MCMC scheme. The right side of Figure 4 shows the estimated $\tilde{\text{Bi}}(x|\theta)$ and its 95% credible interval compared to the true $\text{Bi}(x)$. We see that the true Biot number lies within the 95% credible region, and that the fitted solution matches the data well.
We compare our approach to the physics informed deep learning approach described in [26]. For this we represent the Biot number as in (39) and minimise the augmented loss function (9) with respect to the neural network parameters and model parameters $\theta$ simultaneously. Figure 5 shows the estimate achieved by applying this scheme using the same data. The neural network was trained until convergence, with the weighting coefficients in (9) set to $\nu_1 = \nu_2 = \nu_3 = 1$. In this case the inferred Biot number does not closely match its true value, and would clearly lie outside the 95% confidence interval computed by our Bayesian approach. Furthermore we have no uncertainty estimates over this parameter, giving us no indication of whether the estimate is accurate.
Figure 5: Right: $\tilde{Bi}(x|\theta)$ inferred by minimising (9) compared to true Biot number. Left: Fitted heat profile using the inferred Biot number.

5 Conclusion

We have proposed a new method of solving integral equations based on the use of two neural networks to approximate the solution of the integral equation, and the integral term of the equation. Both networks are trained simultaneously using mini-batch gradient descent to minimise a loss function constructed such that the minimising functions satisfy the equation. The method and extends existing deep learning methods for solving PDEs. Like these algorithms it benefits from being mesh-free, thus with an extension could be scaled to efficiently solve higher-dimensional integral equations without suffering from the curse of dimensionality.

We have additionally described how to leverage neural network based algorithms for PDEs and integral equations to accelerate MCMC schemes for Bayesian inverse problems involving PDE and integral equation forward models. Our approach uses a neural network surrogate model which approximates the solution of the equation over the parameter space of the problem as well as the input space. The resulting deep surrogate model can be evaluated efficiently for any parameter values within the solution domain. It can also be easily parallelised to increase efficiency. When compared to vanilla MCMC for inverse problems our method avoids the necessity of repeated numerical solves, leading to significant speedups. Our method is supe-
rior to interpolation based surrogate models at accurately satisfying the equations, while still being capable of representing solutions with higher-dimensional parameter spaces. We presented numerical evidence displaying the accuracy of the integral equation algorithm, as well examples demonstrating the approximate posterior samples achieved are consistent with the true parameters.

Some topics of further research related to this work include an investigation of the performance of this method when extended to integro-differential equations or higher-dimensional integral equations. For function-valued parameter inference it would be interesting to explore different parametric forms of these functions. Options such as a Karhunen-Loève or random Fourier series representation could provide a means to solve problems where the prior distribution is a stationary Gaussian random field. Alternatively a grid based discretisation of the domain could be coupled with a convolutional encoder-decoder network as in [35] to allow a discrete approximation of any prior distribution.

Acknowledgements

Teo Deveney is supported by a scholarship from the EPSRC Centre for Doctoral Training in Statistical Applied Mathematics at Bath (SAMBa), under the project EP/L015684/1.

References

[1] M. Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org, 2015.

[2] A. Alexiou, N. J. Hills, C. A. Long, A. B. Turner, and J. A. Millward. Heat transfer in high-pressure compressor gas turbine internal air systems: A rotating disc-cone cavity with axial throughflow. Exp. Heat Trans., 13(4), 2000.

[3] A. J. Bard and L. R. Faulkner. Electrochemical Methods: Fundamentals and Applications. Wiley, 2nd edition, 2001.

[4] S. Becker, P. Cheridito, and A. Jentzen. Deep optimal stopping. J. Mach. Learn. Res., 20:74:1–74:25, 2018.
[5] J. Berner, P. Grohs, and A. Jentzen. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *ArXiv e-prints arXiv:1809.03062*, 09 2018.

[6] P. Constantine, Q. Wang, A. Doostan, and G. Iaccarino. A surrogate accelerated bayesian inverse analysis of the hyshot ii flight data. In 52nd *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 19th AIAA/ASME/AHS Adaptive Structures Conference 13t*, page 2037, 2011.

[7] T. Dodwell, C. Ketelsen, R. Scheichl, and A. L Teckentrup. A hierarchical multilevel Markov chain Monte Carlo algorithm with applications to uncertainty quantification in subsurface flow. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):1075–1108, 2015.

[8] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Dec 2017.

[9] Y. Efendiev, A. Datta-Gupta, V. Ginting, X. Ma, and B. Mallick. An efficient two-stage markov chain monte carlo method for dynamic data integration. *Water Resources Research - WATER RESOUR RES*, 411, 12 2005.

[10] Y. Efendiev, T. Hou, and W. Luo. Preconditioning markov chain monte carlo simulations using coarse-scale models. *SIAM J. Scientific Computing*, 28:776–803, 01 2006.

[11] P. Grohs, F. Hornung, A. Jentzen, and P. Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *ArXiv e-prints arXiv:1809.02362*, 09 2018.

[12] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115, 07 2017.
[13] S. Heidenreich, H Gross, M-A Henn, C Elster, and M Bär. A surrogate model enables a Bayesian approach to the inverse problem of scatterometry. *Journal of Physics: Conference Series*, 490:012007, mar 2014.

[14] F. P. Incropera and D. P. DeWitt. *Fundamentals of Heat and Mass Transfer*. Wiley, New York, 1996.

[15] D. Izzo, E. Öztürk, and M. Märtens. Interplanetary transfers via deep representations of the optimal policy and/or of the value function, 07 2019.

[16] A. Jentzen, D. Salimova, and T. Welti. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *ArXiv e-prints arXiv:1809.07321*, September 2018.

[17] Y. Marzouk and D. Xiu. A stochastic collocation approach to bayesian inference in inverse problems. *Communications in Computational Physics*, 6:826–847, 2009.

[18] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[19] M. A. Nabian and H. Meidani. A deep neural network surrogate for high-dimensional random partial differential equations, 2018.

[20] M. A. Nabian and H. Meidani. Physics-informed regularization of deep neural networks, 2018.

[21] T. Nakamura-Zimmerer, Q. Gong, and W. Kang. Adaptive deep learning for high-dimensional hamilton-jacobi-bellman equations, 2019.

[22] M. Rahman. *Integral Equations and Their Applications*. WIT, 2007.

[23] M. Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018.

[24] M. Raissi, H. Babaei, and P. Givi. Deep learning of turbulent scalar mixing, 11 2018.
[25] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *ArXiv pre-print arXiv1711.10561*, 2017.

[26] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations. *ArXiv pre-print arXiv1711.10566*, 2017.

[27] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[28] J. A. J. Rynn, S. L. Cotter, C. E. Powell, and L. Wright. Surrogate accelerated bayesian inversion for the determination of the thermal diffusivity of a material. *Metrologia*, 56(1):015018, Jan 2019.

[29] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018.

[30] A. F. M. Smith and G. O. Roberts. Bayesian Computation via the Gibbs Sampler and Related Markov Chain Monte Carlo Methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 3–23, 1993.

[31] A. M. Stuart. Inverse problems: A bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[32] H. Tang, T. Shardlow, and J. M. Owen. Use of Fin equation to calculate Nusselt numbers for rotating discs. *Journal of Turbomachinery*, 12(137), 2015.

[33] D. Xiu and G. Karniadakis. Modeling uncertainty in steady state diffusion problems via generalized polynomial chaos. *Computer Methods in Applied Mechanics and Engineering*, 191:4927–4948, 09 2002.

[34] L. Zhang. A general framework of optimal investment. *Journal of Mathematical Finance*, 09:535–560, 01 2019.

[35] Y. Zhu, N. Zabaras, P. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.