Data-Centric Engineering (2020), 1: e4
doi:10.1017/dce.2020.4

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Deep kernel learning approach to engine emissions modeling

Changmin Yu[1], Marko Seslija[1], George Brownbridge[2], Sebastian Mosbach[1,2,3], Markus Kraft[1,2,3,4], Mohammad Parsi[5], Mark Davis[5], Vivian Page[5] and Amit Bhave[2]

[1]Department of Chemical Engineering and Biotechnology, University of Cambridge, Cambridge, United Kingdom
[2]CMCL Innovations, Sheraton House, Castle Park, Cambridge, United Kingdom
[3]Cambridge Center for Advanced Research and Education in Singapore (CARES), Singapore, Singapore
[4]School of Chemical and Biomedical Engineering, Nanyang Technological University, Singapore, Singapore
[5]Perkins Engines Co. Ltd., Frank Perkins Way, Peterborough, United Kingdom
*Corresponding author. E-mail: mk306@cam.ac.uk

(Received 31 January 2020; revised 01 May 2020; accepted 06 May 2020)

Keywords: Deep kernel learning; emissions; surrogate models; Gaussian processes; internal combustion engines

Abstract

We apply deep kernel learning (DKL), which can be viewed as a combination of a Gaussian process (GP) and a deep neural network (DNN), to compression ignition engine emissions and compare its performance to a selection of other surrogate models on the same dataset. Surrogate models are a class of computationally cheaper alternatives to physics-based models. High-dimensional model representation (HDMR) is also briefly discussed and acts as a benchmark model for comparison. We apply the considered methods to a dataset, which was obtained from a compression ignition engine and includes as outputs soot and NO_x emissions as functions of 14 engine operating condition variables. We combine a quasi-random global search with a conventional grid-optimization method in order to identify suitable values for several DKL hyperparameters, which include network architecture, kernel, and learning parameters. The performance of DKL, HDMR, plain GPs, and plain DNNs is compared in terms of the root mean squared error (RMSE) of the predictions as well as computational expense of training and evaluation. It is shown that DKL performs best in terms of RMSE in the predictions whilst maintaining the computational cost at a reasonable level, and DKL predictions are in good agreement with the experimental emissions data.

Impact Statement

Surrogate models, also known as emulators, meta-models, or response surfaces, are models, which are used in place of other models or data. In practice, there are two main reasons for using surrogate models. The first one is computational expense, in applications, which require large numbers of model evaluations, such as optimization problems, involving detailed, physics-based models. The second reason is to obtain functions, which describe behavior parametrically, in case of discrete data points. As requirements vary widely across applications, there is no one-size-fits-all surrogate. In addition, the challenge of constructing a suitable surrogate becomes dramatically harder as the number of input variables increases—a phenomenon sometimes referred to as the curse of dimension. In this work, we show that a state-of-the-art machine-learning technique performs well as a surrogate of a high-dimensional, highly nonlinear, industrial dataset of engine emissions, and present a method, which allows fitting the surrogate in a way convenient to users.

1. Introduction

Complex physical systems such as internal combustion engines are generally studied via physical or computational experiments, or their combinations. In many situations, physical experiments may not be feasible or can be highly expensive to perform, and computer experiments are then preferred in such cases. These involve experimenting on a rigorous first-principles or physics-based model instead of the real system (Garud et al., 2017a). However, expressive computer experiments often involve high computational cost. Therefore, in any application that requires large numbers of data points or model evaluations such as optimization or parameter estimation, it is inevitable to replace high-fidelity models and real physical experiments by computationally cheap surrogate models.

A surrogate model is an empirical analytical or numerical expression for the quantification of relationships between relevant input features and output labels/values of a physical system (Garud et al., 2018a). Surrogate models can be viewed as a substitute model, which mimics the behavior of a detailed model or data as closely as possible while keeping the computational cost at a minimum. The construction of a surrogate model requires one to choose a mathematical or numerical form for modeling. Surrogate modeling has been studied extensively in applications in numerous areas across science and technology, too numerous to review here, and for a variety of purposes, the most popular ones being parameter estimation (Frenklach et al., 1992; Kastner et al., 2013) and sensitivity analysis (Azadi et al., 2014). Many statistical and machine learning methods have been proposed as surrogate models in the literature, such as high-dimensional model representation (HDMR) (Rabitz and Aliş, 1999; Sikorski et al., 2016), support vector regression (Drucker et al., 1997), and radial basis function (RBF) fitting (Park and Sandberg, 1991), to name a few. The variety of available surrogate modeling techniques is reflective of the fact that there is no such thing as a "universal surrogate." Any given technique may work well in some applications, but relatively poorly in others, depending on the unique characteristics in each case, such as dimensionality, oscillatory or discontinuous behavior, and many others. Learning based evolutionary assistive paradigm for surrogate selection (LEAPS2) is a framework that was proposed to recommend the best surrogate(s) with minimal computational effort given the input/output data of a complex physico-numerical system (Garud et al., 2018b). A frequently encountered issue, particularly in engine applications, is that the input spaces of the datasets are high-dimensional. Another common problem, and again this applies particularly to engine applications, is that the size of the datasets is usually quite limited due to the high cost induced by the data measurement, which further compounds the problem of high dimensionality, although techniques exist to alleviate this (Garud et al., 2017a, Garud et al., 2017b).

The literature is replete with work related to the application of various machine-learning methods to internal combustion engine modeling, optimization, and calibration. Ghanbari et al. (2015) used support vector machines (SVMs) to predict the performance and exhaust emissions of a diesel engine and showed that SVM modeling is capable of predicting the engine performance and emissions. In the study by Najafi et al. (2016), an SVM and an adaptive neuro-fuzzy inference system (ANFIS) are applied to predicting performance parameters and exhaust emissions such as CO_2 and NO_x of a spark ignition (SI) engine and are compared in terms of their performance, and they showed ANFIS is significantly better than SVM. Silitonga et al. (2018) applied a method known as kernel-based extreme learning machine (K-ELM) to evaluate the performance and exhaust emissions of compression ignition (CI) engines with biodiesel–bioethanol–diesel blends at full-throttle conditions. Lughofer et al. (2011) investigated the modeling of NO_x emissions of a diesel engine using a fuzzy model directly from measurement data, which is then shown to be a good alternative to physics-based models. Yilmaz et al. (2016) compared the response surface methodology (RSM), a commonly used surrogate model, with least-squared support vector machine (LSSVM) based on their performance in predicting the performance and exhaust emissions of a diesel engine fueled with hazelnut oil, and showed that LSSVM narrowly outperforms RSM. Ghobadian et al. (2009) studied the application of a multilayer perceptron (MLP) to predicting exhaust emissions of a diesel engine using waste cooking biodiesel fuel and showed that MLP performs quite well in emissions prediction. Further studies in modeling and predicting the performance and exhaust emissions of diesel engines under different conditions can be found in the literature, such as the work of Niu et al. (2017) on the comparison of artificial neural network (ANN) and SVM on emissions prediction of a marine diesel engine, and the study by

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

Wong et al. (2015) on using relevance vector machine in modeling and prediction of diesel engine performance. Various studies on the application of ANNs to emissions modeling in diesel engines under different conditions can be found for example in Najafi et al. (2009), Sayin et al. (2007), and Yusaf et al. (2010).

In the field of machine learning applications in diesel engine modeling, one of the most widely used methods is known as ELM, which has inspired many extensions and applications in the diesel engine community since its introduction. ELMs are feedforward neural networks, with a single hidden layer in most cases (Huang et al., 2006). ELMs are an alternative to conventional neural networks in the sense that each hidden unit in an ELM is a computational element, which can be same as classical nodes in an MLP, as well as basis functions or a subnetwork with hidden units (Huang and Chen, 2007). Vaughan and Bohac (2015) proposed an online adaptive ELM named weighted ring-ELM, which provides real-time adaptive, fully causal predictions of near-chaotic homogeneous charge compression ignition engine combustion timing. Janakiraman et al. (2016) proposed a stochastic gradient based ELM (SG-ELM), a stable online learning algorithm, designed for systems whose estimated parameters are required to remain bounded during learning. Wong et al. (2018) studied the ELM-based modeling and optimization approach for point-by-point engine calibration. Silitonga et al. (2018) studied the application of K-ELM (Huang et al., 2012) for prediction of the engine performance of biodiesel–bioethanol–diesel blends.

In addition to the above-mentioned studies, there exist a large number of works on the application of machine learning to diesel engine calibration and control. For instance, Tietze (2015) studied the application of Gaussian process (GP) regression for calibrating engine parameters. Jeong et al. (2008) applied a hybrid evolutionary algorithm consisting of a genetic algorithm and particle swarm optimization to optimize diesel engine design with respect to decreasing exhaust emissions. Berger and Rauscher (2012) and Berger et al. (2011) discussed various learning methods such as linear regression, feedforward neural networks, and GP regression for modeling and optimization for stationary engine calibration. Malikopoulos et al. (2007) proposed a reinforcement-learning-based decentralized control method, which allows an internal combustion engine to learn its optimal calibration in real time while running a vehicle.

In this study, we focus on data-driven engine emissions modeling using deep kernel learning (DKL) (Wilson et al., 2016)—a state-of-the-art machine learning technique, which can be viewed as a standard deep neural network (DNN) with a GP as its last layer instead of a fully connected layer. In this way, we can not only use a deep feedforward network to extract a high-level representation of the data, but also take advantage of the nonparametric flexibility induced by the GP regression. We implement DKL for a diesel engine emission dataset, taking 14 input variables including speed, load, injection timing, and others to make predictions of NO_x and soot emissions. We then use a systematic two-stage procedure to determine several of the hyperparameters in DKL, and compare the resulting surrogate to a plain deep feedforward network, a plain GP, as well as HDMR.

The paper is structured as follows. In Section 2, we discuss in some detail HDMR, deep feedforward networks, GPs and DKL, respectively. In Section 3, we apply the three methods to the target engine emission data, and compare their performance. Conclusions are drawn in Section 4.

2. Methods

2.1. High-dimensional model representation

Here, we briefly recall a well-established surrogate modeling method, high-dimensional model representation (HDMR), which we use as a reference method for comparison.

HDMR is a finite expansion for a given multivariable function (Rabitz and Aliş, 1999). Under its representation, the output function y can be approximated using the following expression:

$$y \approx f(x) = f_0 + \sum_{i=1}^{N_x} f_i(x_i) + \sum_{i=1}^{N_x} \sum_{j=1}^{N_x} f_{ij}(x_i, x_j), \tag{1}$$

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

where N_x is the dimension of the input space and f_0 represents the mean value of $f(x)$. The above approximation is sufficient in many situations in practice since terms containing functions of more than two input parameters can often be ignored due to their negligible contributions compared to the lower-order terms (Li et al., 2002). The terms in Equation (1) can be evaluated by approximating the functions $f_i(x_i)$ and $f_{ij}(x_i,x_j)$ with some orthonormal basis functions, $\phi_k(x_i)$, that can be easily computed. Popular choices for the basis functions include ordinary polynomials (Li et al., 2002) and Lagrange polynomials (Baran and Bieniasz, 2015). Apart from applications in chemical kinetics, HDMR has been applied in process engineering (Sikorski et al., 2016) and also in engine emissions modeling (Lai et al., 2018).

2.2. Deep neural networks

Deep learning, or deep ANNs, are composed of multiple processing layers to learn a representation of data with multiple levels of abstraction (LeCun et al., 2015). Deep learning has gained exploding popularity in the machine learning community over the past two decades, and it has been applied in numerous fields including computer vision, natural language processing, recommendation systems, and so forth, due to its ability to find both low- and high-level features of the data as well as its scalability to high-dimensional input spaces. However, the construction of deep learning requires specification of many hyperparameters including number of epochs, regularization strength, dropout rate, and so forth, and there is no universal paradigm for determining the optimal settings for these hyperparameters. Hence the performance of a DNN can heavily depend on engineering techniques such as architecture specification and parameter tuning.

In our study, we are only concerned with fully connected neural networks. The goal of a neural network is to learn the underlying representation of the given datasets. A deep feedforward network follows the standard forward propagation and backpropagation to tune the learnable parameters of the neural network, then use the trained network for further predictions on similar problems. In Figures 1 and 2, we show simple graphical representations for forward and backpropagation in a fully connected network, respectively.

The architecture of a neural network can be described as a directed graph whose vertices are called neurons, or nodes, and directed edges, each of which has an associated weight. The set of nodes with no incoming edges are called input nodes, whereas the nodes with no outgoing edges are called output nodes.

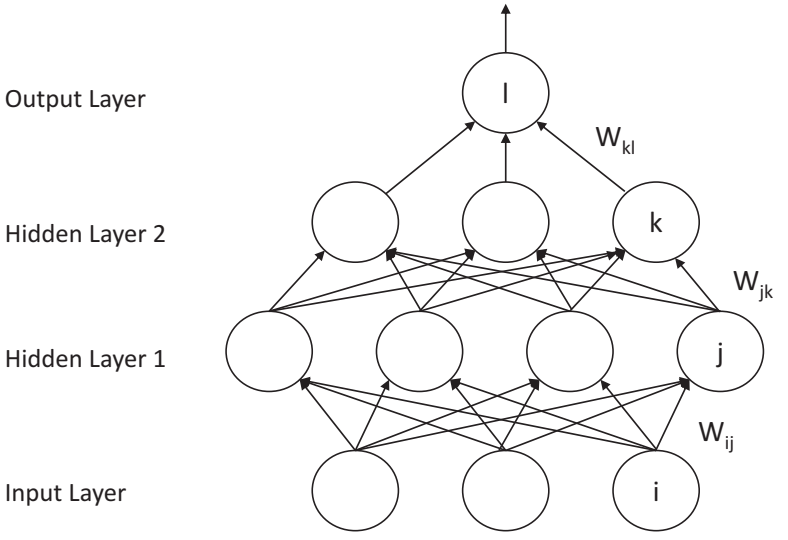


Figure 1. *Forward propagation in a three-layer feedforward neural network. For each unit in the layers other than the input layer, the output of the unit equals the inner product between all the outputs from the previous layer and the weights followed by a nonlinearity (e.g., the ReLU function).*

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

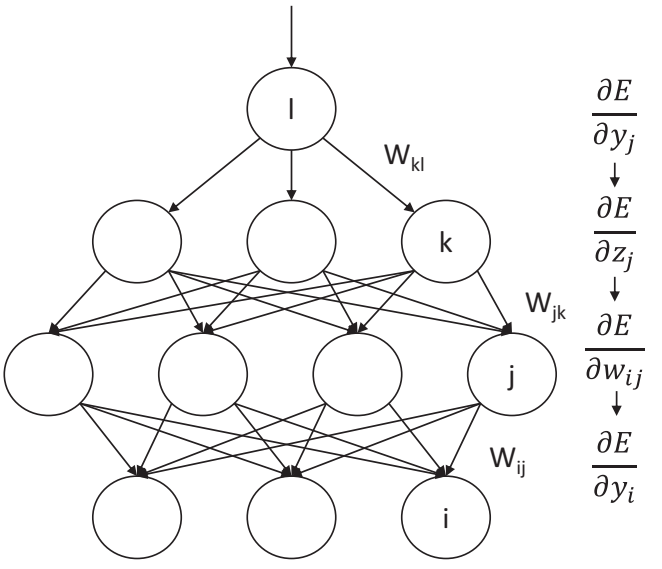


Figure 2. *Backpropagation in a three-layer feedforward neural network. Computing the derivatives of the cost function with respect to the weight parameters using chain rules, then the parameters are updated using gradient descent with the computed derivatives.*

We define the first layer to be the input layer and the last layer to be the output layer, and all other layers in between are called the hidden layers. The input data are propagated in a feedforward fashion as follows:

Figure 1 shows how input signals are propagated forward in a three-layer fully connected neural network, where for each unit in any layer other than the input layer, the input to the unit equals the inner product between the output signals from all the units from the previous layer and the associated weights:

$$z_j = \sum_{i \in \text{previous layer}} w_{ij} y_i. \tag{2}$$

The unit then outputs a scalar by applying a nonlinearity, g, to this inner product: $y_j = g(z_j)$. The signals are propagated forward in this way layer by layer until the output layer is reached. Commonly used nonlinear activation functions include the rectified linear unit (ReLU) $f(z) = \max(0,z)$, the sigmoid function $\sigma(z) = [1 + \exp(-z)]^{-1}$ and the hyperbolic tangent function $\tanh(z)$. In this case, we have omitted bias terms for simplicity.

At an output node, after taking the linear combination of its predecessor values, instead of applying an activation function, an output rule can be used to aggregate the information across all the output nodes. In a regression problem, as opposed to a classification problem, we simply keep the linear combination as the output prediction.

The weights and bias parameters in neural networks are usually learnt via backpropagation using the cached quantities from the forward propagation stage. Backpropagation, as represented in Figure 2, is executed using gradient-descent–based optimization methods. Given a cost function, $E(\hat{y},y)$, which quantifies the difference between the current predictions of the output ($\hat{y}$) and the actual outputs (y), the derivatives of E with respect to each weight in each layer of the network can be derived using the chain rule,

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}, \quad \frac{\partial E}{\partial w_{ij}} = y_i \frac{\partial E}{\partial z_j}, \tag{3}$$

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

where the initial $\partial E/\partial y_j$ is derived given the activation function and the cost function. Then we compute $\partial E/\partial z_j$, $\partial E/\partial w_{ij}$ and $\partial E/\partial y_i$ iteratively for each node i in layer l and node j in layer $l-1$ until we reach the input layer. Then we use methods such as stochastic gradient descent to update the weights in the network:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}}, \tag{4}$$

where α is some scalar known as the learning rate.

Iterative application of forward propagation and backpropagation will gradually decrease the cost function value in general, and if the cost function is convex in the input space, it will eventually converge to the global minimum or somewhere close to the global minimum.

The power of neural networks lies in the composition of the nonlinear activation functions. From the results on universal approximation bounds for superpositions of the sigmoid function by Barron (1993), it can be implied that a neural network with sigmoid activation can arbitrarily closely approximate a nonparametric regression mode.

Here we only discuss fully connected neural networks, where the only learnable parameters are the weight parameters w_{ij} between each node i in layer $l-1$ and each node j in the previous layer l. For a general review of various architectures of deep learning such as convolutional neural networks and recurrent neural networks, the reader is referred to LeCun et al. (2015) for example.

2.3. Gaussian processes

A GP is a stochastic process, which is a collection of random variables, with the property that any finite subcollection of the random variables have a joint multivariate Gaussian distribution (Williams and Rasmussen, 2006). We denote the fact that a stochastic process $f(\cdot)$ is a GP with mean function $m(\cdot)$ and covariance function $k(\cdot,\cdot)$ as

$$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot)). \tag{5}$$

The definition implies that for any $x^{(1)}, x^{(2)}, \ldots, x^{(n)} \in \mathcal{X}$, where $\mathcal{X}$ denotes the set of possible inputs, we have

$$\left[f\left(x^{(1)}\right), \ldots, f\left(x^{(n)}\right)\right]^{\mathrm{T}} \sim \mathcal{N}\left(\left[m\left(x^{(1)}\right), \ldots, m\left(x^{(n)}\right)\right]^{\mathrm{T}}, K\right), \tag{6}$$

where the covariance matrix K has entries $K_{ij} = k\left(x^{(i)}, x^{(j)}\right)$.

GPs can be interpreted as a natural extension of multivariate Gaussian distributions to have infinite index sets, and this extension allows us to think of a GP as a distribution over random functions. A GP is fully determined by its mean and covariance functions. The mean function can be any real-valued function, whereas the covariance function has to satisfy that the resulting covariance matrix K for any set of inputs $x^{(1)}, \ldots, x^{(n)} \in \mathcal{X}$ has to be a valid covariance matrix for a multivariate Gaussian distribution, which implies that K has to be positive semidefinite and this criterion corresponds to Mercer's condition for kernels (Minh et al., 2006). Hence the covariance function is sometimes also known as the kernel function. One of the most popular choices of kernel function is the RBF (or squared error) kernel

$$k_{\mathrm{RBF}}(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2l^2}\right), \tag{7}$$

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

where l is a kernel parameter which quantifies the level of local smoothness of the distribution drawn from the GP.

GPs have gained increasing popularity in the machine learning community since Neal (1996) showed that Bayesian neural networks with infinitely many nodes converge to GPs with a certain kernel function. Therefore, GPs can be viewed as a probabilistic and interpretable alternative to neural networks.

In our study, the focus of the application of GPs lies within regression tasks. Suppose we are given a dataset $\mathcal{D} = \left\{x^{(i)}, y^{(i)}\right\}_{i=1,\ldots,n}$, that one may refer to as the training set, of independent samples from some unknown distribution, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ for $i = 1, \ldots, n$. A GP regression model is then

$$y^{(i)} = f\left(x^{(i)}\right) + \epsilon^{(i)}, \text{for } i = 1, \ldots, n, \tag{8}$$

with a GP prior over the functions f, that is $f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot))$, for some mean function $m(\cdot)$ and valid kernel function $k(\cdot,\cdot)$, and the $\epsilon^{(i)}$ are independent additive Gaussian noises that follow $\mathcal{N}(0, \sigma^2)$ distributions.

Suppose we are given another dataset $\mathcal{D}_* = \left\{x_*^{(i)}, y_*^{(i)}\right\}_{i=1,\ldots,n_*}$, that one may refer to as the blind-test set, drawn from the same distribution as $\mathcal{D}$. By the definition of a GP, we have

$$\begin{bmatrix} F \\ F_* \end{bmatrix} \Bigg| X, X_* \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right), \tag{9}$$

where $X = \left\{x^{(i)}\right\}_{i=1,\ldots,n}$, F represents $\left[f\left(x^{(1)}\right), \ldots, f\left(x^{(n)}\right)\right]^\top$, $K(X,X)$ represents an $n \times n$ kernel matrix whose (i,j) entry is $K\left(x^{(i)}, x^{(j)}\right)$, and analogous definitions for quantities with asterisk subscripts, for example, $X_* = \left\{x_*^{(i)}\right\}_{i=1,\ldots,n_*}$. Then, given the additive Gaussian noises, the joint distribution of $Y = \left[y^{(1)}, \ldots, y^{(n)}\right]^\top$ and Y_* becomes

$$\begin{bmatrix} Y \\ Y_* \end{bmatrix} \Bigg| X, X_* \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) + \sigma^2 I \end{bmatrix}\right). \tag{10}$$

Using the rules for conditioning Gaussian distributions, the posterior distribution of the blind-test data given the training data is then given by

$$\begin{aligned} Y_* | X_*, X, Y \; &\sim \mathcal{N}(\mu_*, \Sigma_*), \\ \mu_* \quad &= m(X_*) + K(X_*,X)\left[K(X,X) + \sigma^2 I\right]^{-1} Y, \\ \Sigma_* \quad &= K(X_*,X_*) - K(X_*,X)\left[K(X,X) + \sigma^2 I\right]^{-1} K(X,X_*). \end{aligned} \tag{11}$$

Equation (11) is the posterior GP regression model for predictions. The kernel parameters θ of the kernel function in the GP regression model can be learnt by maximizing the (log) posterior marginal likelihood

$$\log p(Y|\theta, X) \propto -Y^\top \left(K_\theta(X,X) + \sigma^2 I\right)^{-1} \mathrm{Y} - \log\left|K_\theta(X,X) + \sigma^2 I\right| \tag{12}$$

with respect to the kernel parameters, where we have emphasized the dependence of the kernel matrix K on its parameters θ through a subscript.

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

2.4.. Deep kernel learning

We now briefly discuss the main method we apply as a surrogate model, DKL (Wilson et al., 2016). DKL can be intuitively interpreted as a combination of a DNN and a GP. A graphical representation of a DKL model is shown in Figure 3, where we can see that the structure consists of a DNN followed by a GP. As mentioned in the previous section, a GP is the limit of a Bayesian neural network with an infinite number of nodes, hence the GP at the end of the DKL architecture can be interpreted as another hidden layer in the DNN, but with an infinite number of nodes, and this greatly increases the expressiveness compared to a stand-alone DNN. When the data enters the DKL model, it is first propagated in a forward fashion through the neural network. The high-dimensional input data is thus transformed by the neural network into a lower-dimensional feature vector, which is then used as the input arguments for GP regression. The expectation of the resulting posterior distribution is then taken as the value predicted by DKL as a function of the input data. As GPs naturally do not perform well in high-dimensional input spaces, the DNN acts as a feature extractor and dimensionality reduction method for more robust GP regression. Being a combination of deep learning and kernel learning, DKL encapsulates the expressive power for extracting high-level features and capturing nonstationary structures within the data given its deep architectures and the nonparametric flexibility in kernel learning induced by its probabilistic GP framework.

We can also view DKL as a GP with a stand-alone deep kernel. Starting from a base kernel $k\left(x^{(i)},x^{(j)}|\theta\right)$ with kernel parameters, the deep kernel can be constructed as

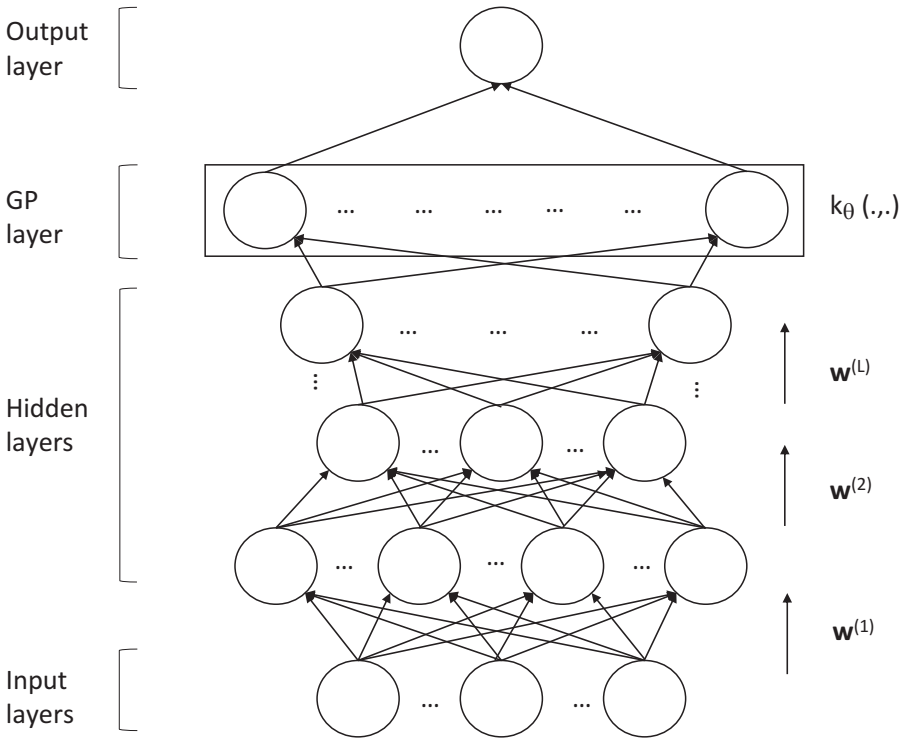


Figure 3. *Deep kernel learning: input data is propagated in a forward fashion through the hidden layers of the neural network parameterized by the weight parameters. Then, the low-dimensional high-level feature vector as the output of the neural network is fed into a GP with a base kernel function $k_\theta(\cdot,\cdot)$ for regression. The posterior mean of the Gaussian regression model is taken as the prediction given the input data (Adapted from Wilson et al., 2016).*

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

$$k\left(x^{(i)},x^{(j)}|\theta\right)\rightarrow k\left(g\left(x^{(i)},w\right),g\left(x^{(j)},w\right)|\theta,w\right), \tag{13}$$

where $g(x;w)$ is a nonlinear mapping induced by the neural network with weight parameters w. A popular choice for the base kernel $k\left(x^{(i)},x^{(j)}|\theta\right)$ is again the RBF kernel (Equation 7). Inspired by Wilson et al. (2016), we also look at the spectral mixture (SM) base kernel

$$k_{\mathrm{SM}}(x,x'|\theta)=\sum_{q=1}^{Q}a_q\frac{|\Sigma_q|^{\frac{1}{2}}}{(2\pi)^{\frac{D}{2}}}\exp\left(-\frac{1}{2}\|\Sigma_q^{\frac{1}{2}}(x-x')\|^2\right)\cos\langle x-x',2\pi\mu_q\rangle \tag{14}$$

by Wilson and Adams (2013), where the learnable kernel parameters $\theta=\left\{a_q,\Sigma_q,\mu_q\right\}$ consist of a weight, an inverse length scale, and a frequency vector for each of the Q spectral components, and where $\langle\cdot,\cdot\rangle$ denotes the standard inner product. The spectral mixture kernel is meant to be able to represent quasi-periodic stationary structures within the data.

We denote by $\gamma=\{w,\theta\}$ the parameters of the DKL model, consisting of the neural network weight parameters w and the GP kernel parameters. These parameters are learnt jointly via maximizing the log-posterior marginal likelihood of the GP (Equation 12) with respect to.

3. Applying DKL to engine emissions

3.1. Dataset

The dataset used in this work was obtained from a diesel-fueled compression ignition engine whose main geometric features are provided in Table 1. The data consists of soot and NO_x emissions taken engine-out during steady-state operation at 1,861 distinct operating points. Each of these points is characterized by 14 operating condition variables, which include engine speed and torque, intake manifold temperature, injection pressure, mass fraction of recirculated exhaust gas, start, end, and fuel mass of injection, and combustion chamber wall temperatures. The set of points is spread roughly evenly over the entire engine operating window in terms of speed and load. NO_x emissions are measured in units of parts per million by volume (ppmv) with a nominal error bar of 3% and soot represent the carbon fraction of emitted particulate matter with a stated measurement uncertainty of 5%.

Since the numerical values of the soot response vary over several orders of magnitude, it is necessary to consider their logarithms instead of their raw values. The NO_x response values can be used as is. We split the dataset randomly into disjoint training and blind-test sets, which comprise 90% and 10% of the total, respectively.

3.2. Implementation

In all numerical experiments, our DKL implementation consists of a five-layer fully connected network and a GP with RBF kernel. The neural network employs the rectified linear unit (ReLU) function LeCun

Table 1. Specification of the turbocharged four-stroke diesel-fueled compression ignition engine used in this work.

Quantity	Value
Bore	98 mm
Stroke	120 mm
Connecting rod length	180 mm
Compression ratio	17:1

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

et al. (2015) as the activation function for each hidden layer, and all the weights are initialized with the He normal initialization (He et al., 2015). We use the standard root mean squared error (RMSE) loss function, and the Adam optimizer for optimization (Kingma and Ba, 2014).

We implement DKL as a surrogate model into model development suite (MoDS) (CMCL Innovations, 2018)—an integrated software written in C++ with multiple tools for conducting various generic tasks to develop black-box models. Such tasks include surrogate model creation (Sikorski et al., 2016), parameter estimation (Kastner et al., 2013), error propagation (Mosbach et al., 2014), and experimental design (Mosbach et al., 2012). Our MoDS-implementation of DKL uses PyTorch (Paszke et al., 2017) and GPyTorch (Gardner et al., 2018).

All simulations were performed on a desktop PC with 12 3.2 GHz CPU-cores and 16 GB RAM. Even though MoDS allows parallel execution, all simulations were conducted in serial in order to simplify quantification of computational effort. For the same reason, no GPU-acceleration of Torch-based code was considered. We also did not explore the use of kernel interpolation techniques (Wilson and Nickisch, 2015; Quiñonero-Candela and Rasmussen, 2005) to speed up GP learning.

3.3. *Network architecture, kernel, and learning parameters*

The DKL framework involves learnable parameters such as network weights and kernel parameters, as well as hyperparameters such as the learning rate, number of iterations, and number of nodes in each layer of the neural network. Before training can be carried out, suitable values of the hyperparameters need to be chosen. We approach this in two ways: First, we make this choice manually, based on previous experience and cross-validation over a small hyperparameter search-space, and second, we employ a systematic, optimization-based procedure. In both cases, we determine the following seven hyperparameters: the number of nodes in each of the four hidden layers, the prior white-noise level of the GP, the number of epochs, that is training iterations, and the learning rate.

The optimization approach consists of two stages: a global quasi-random search followed by a local optimization with a gradient-free grid-based method, both of which are conducted using MoDS. For the first stage, a Sobol sequence (Joe and Kuo, 2008), a low-discrepancy sampling method, is employed. We generate 1,000 Sobol points within the space spanned by all of the hyperparameters given in Table 2, which also provides the range and scaling type for each parameter. We then fit the DKL under each of these 1,000 hyperparameter settings to the training data. The quality of each fit is assessed by calculating the objective function

$$\Phi(\theta) = \left[\frac{1}{N_\mathrm{b}-1}\sum_{i\in\mathcal{J}_\mathrm{b}}\left(f\left(x^{(i)}\right)-y^{(i)}\right)^2\right]^2 + \left[\frac{1}{N_\mathrm{t}-1}\sum_{i\in\mathcal{J}_\mathrm{t}}\left(f\left(x^{(i)}\right)-y^{(i)}\right)^2\right]^2, \tag{15}$$

Table 2. Deep kernel learning hyperparameters considered for optimization.

Parameters	Lower bound	Upper bound	Scaling	Rounded
N nodes layer 1	1	1,000	Log	Yes
N nodes layer 2	1	1,000	Log	Yes
N nodes layer 3	1	1,000	Log	Yes
N nodes layer 4	1	15	Linear	Yes
White noise scale	0.0	0.5	Linear	No
Number of epochs	10	1,000	Log	Yes
Learning rate	0.001	0.1	Log	No

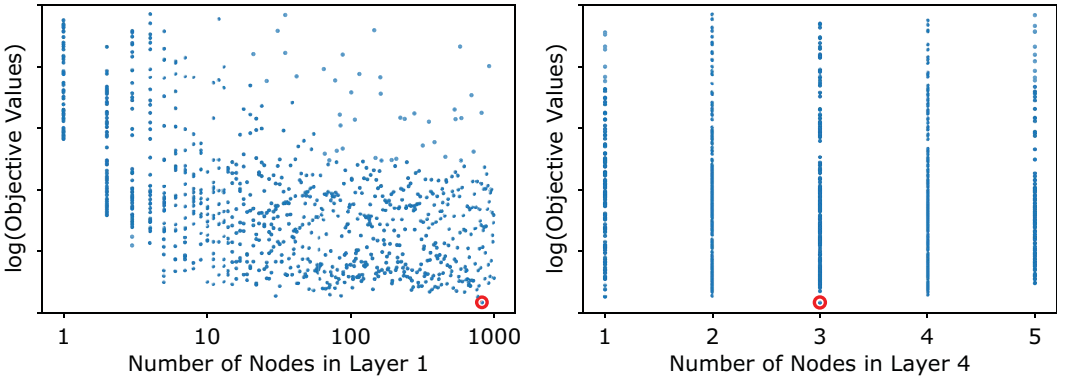
https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

where f denotes the surrogate model, that is, the trained DKL, and $x^{(i)}$ and $y^{(i)}$ the experimental operating conditions and responses (soot or NO_x), respectively, of the ith data point. $\mathcal{I}_b$ denotes the set of indices belonging to the blind-test data points, and N_b denotes their number, whereas $\mathcal{I}_t$ and N_t refer to the analogous quantities for the training data points. The normalization of the two parts of the objective function, that is, the training and blind-test parts, by the number of points they contain, implies that the two parts are equally weighted with respect to each another, irrespective of how many points they contain. We tested other forms of the objective function but found empirically that this form yields the best results. Furthermore, we note that including the blind-test points into the objective function is not a restriction. In any application, whatever set of points is given, it can arbitrarily be split into training and blind-test subsets. Again, we made this choice because we found empirically that it produces the best results.

Scatter plots of the Sobol points, showing their objective values against two of the architecture parameters, are given in Figure 4. In Figure 4a, we observe that best performance is achieved if the number of nodes in the first layer well exceeds the number of inputs. From Figure 4b, we conclude that the number of features extracted by the neural network part of DKL, that is, the number of nodes in the fourth hidden layer or in other words the number of quantities fed as input to the GP, for which the objective function attains its minimum is three. The objective value deteriorates appreciably for four and five features.

The best Sobol point as measured by Equation (15) (highlighted in Figure 4) is then optimized further with respect to the white noise of GP, number of epochs and learning rate using the algorithm by Hooke and Jeeves (1961)—a gradient-free grid-based optimization algorithm, which is also part of MoDS. Since both the Sobol and Hooke–Jeeves method are designed for continuous variables, we treat the discrete parameters, that is, the number of nodes in each layer and the number of epochs, internally as continuous and simply round their values to the nearest integers when passing them on to DKL. This is also indicated in Table 2. Other optimization methods more suitable for discrete problems, such as genetic algorithms, are expected to perform at least as well, however, it is beyond the scope of the present work to explore this.

We note that the Hooke-Jeeves optimization step achieves only a relatively small improvement upon the best Sobol point, with the algorithm terminating after 68 and 78 iterations for NO_x and soot, respectively. However, one should bear in mind that DKL training itself being based on stochastic optimization, and thus noise inherently being present in the quality of the fit, presents a challenge to any local optimization method. We furthermore find that there is little to no benefit in including the architecture parameters of the network, that is, the number of nodes in the hidden layers, into the Hooke-Jeeves optimization, so we excluded them from this stage.



(a) Objective against number of nodes in layer 1. DKL performs relatively poorly for small numbers of nodes in hidden layer 1.

(b) Objective against number of nodes in layer 4. Best performance is found for 3 extracted features, *i.e.* 3 nodes in layer 4.

Figure 4. *Combined training and blind-test objective function value (Equation 15) for 1,000 Sobol points in the space of hyperparameters of Table 2. The point with the lowest objective value overall is circled.*

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

Table 3. Best values found for the hyperparameters in deep kernel learning through optimization.

Output	*N* nodes Layer 1	*N* nodes Layer 2	*N* nodes Layer 3	*N* nodes Layer 4	White noise Scale	Number of Epochs	Learning Rate
NO_x	822	46	356	3	0.39	210	0.019
Soot	690	5	6	3	0.48	466	0.026

The average CPU-time for (serial) evaluation of each Sobol or Hooke-Jeeves point was about 1 min for NO_x and 2 min for soot. The reason for the larger evaluation time for soot is that the optimization favored a number of epochs on average about twice as high for soot as for NO_x.

Table 3 shows the best values for the hyperparameters we find using the procedure described above. We note that due to the random nature of both the global search and the DKL training process itself, our procedure does not guarantee a global minimum.

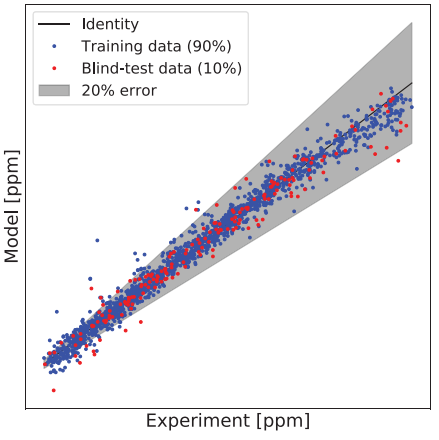
The best values found manually for the hyperparameters in DKL are 1000, 500, 50, and 3 for the numbers of nodes in the hidden layers, a white-noise level of 0.1, 200 training epochs, and a learning rate of 0.01.

3.4. Comparison

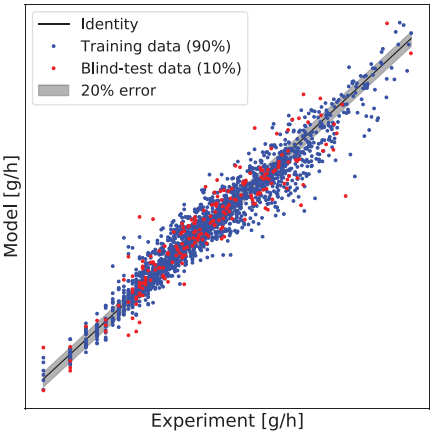
Figure 5 shows a comparison of model versus experiment using HDMR and DKL for both NO_x and soot responses. For DKL, two sets of results are shown: One (Figure 5c,d) obtained using the best values for the hyperparameters found manually, and another one (Figure 5e,f) using the optimized values. The NO_x values are scaled linearly, whereas the soot values are scaled logarithmically. The shaded areas in the plots represent an error margin of 20% with respect to the experimental values. We note that DKL generally produces more accurate regression fits for the data than HDMR for both NO_x and soot, and also that DKL with the optimized hyperparameters is more accurate than with the manual ones. In the NO_x regression with DKL, almost all of the predicted training and blind-test values lie within 20% of the experimental value and a large majority of the predictions align closely with the experimental values. For HDMR, although only few points lie outside the 20% margin, the predictions tend to be distributed further from the experimental values. For the soot regression, similar behavior is observed, but in this case, significantly more predictions lie outside the 20% error bar with respect to the experimental values. Soot being more challenging than NO_x in terms of regression is entirely expected, due to soot emissions depending much more nonlinearly on the engine operating condition and the measurements being intrinsically much more noisy. In addition, we note in Figure 5 that the spread of training points is quite similar to that of the blind-test points for HDMR, whereas the latter is much wider for DKL, with the effect being stronger for soot than for NO_x. These observations are quantified in Table 4 in terms of the percentage of points, which lie within 20% of the experimental values, as well as RSME. The blind-test RMSE is indeed seen to be larger than the training error for both NO_x and soot regression, and the difference is larger in relative terms for soot. This is indicative of over-fitting—a well-known issue with neural networks (Srivastava et al., 2014). Any surrogate with a large number of internal degrees of freedom is prone to over-fitting, especially if this number exceeds the number of data points. Given the number of layers and nodes typically used in a neural network, and hence the associated number of weights, it is clear that in our case the number of degrees of freedom in DKL is much larger than the number of available data points. Another factor contributing to over-fitting is the sparsity of the dataset in the input space, where there are less than 2000 available training points in 14 dimensions. As an aside, an additional consequence of this sparsity is that the variances predicted by Equation (11) are too small to be useful, which is why they are not shown in any of the plots.

Figure 6 shows density plots of relative values $\hat{y}/y$ for both outputs for DKL and HDMR, where $\hat{y}$ and y represent the predicted and experimental values, respectively. It can be observed that for both outputs, the relative values for both HDMR and DKL are mainly distributed near unity, and the majority of the predicted values are within 20% of the experimental values for both methods for both outputs. However, it is clear from the plots that the overall distribution of errors of DKL is significantly more centralized at unity than

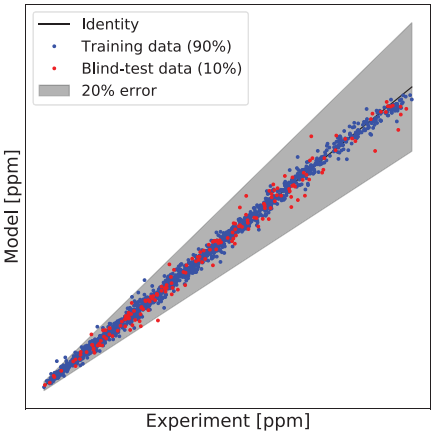
https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press



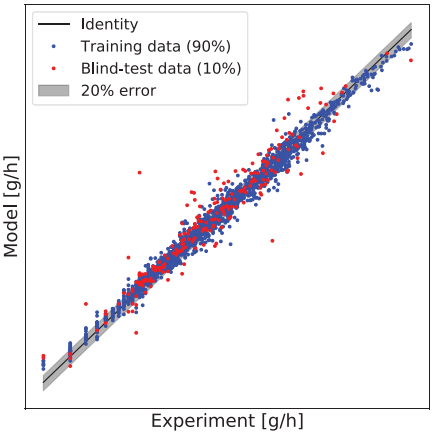
(a) HDMR regression of NO_X response.



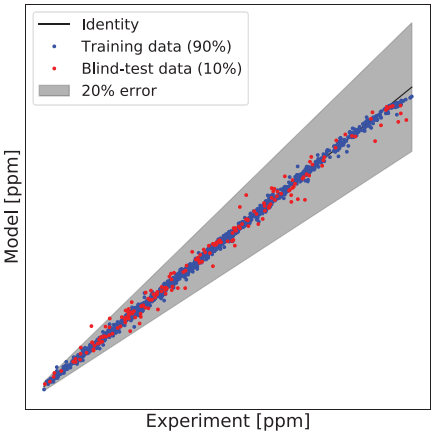
(b) HDMR regression of soot response.



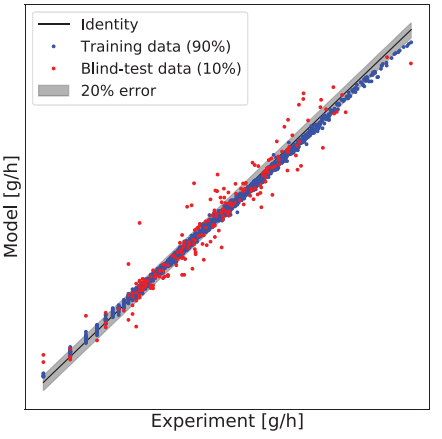
(c) DKL (manual) for NO_X response.



(d) DKL (manual) for soot response.



(e) DKL (optimised) for NO_X response.



(f) DKL (optimised) for soot response.

Figure 5. *Modeled NO_x and soot responses against experimental ones using high-dimensional model representation (HDMR) and two sets of deep kernel learning (DKL) architectures and hyperparameter values. Soot values are logarithmic. For confidentiality reasons, no values are shown on the axes.*

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

Table 4. Percentage of predictions within 20% of the experimental value and root mean square errors (RSME) of NO_x and soot regressions for high-dimensional model representation (HDMR) and deep kernel learning (DKL).

		Percentage w/in 20% of experience			RSME	
Output	Surrogate	Training	Blind test	Total	Training	Blind test
NO_x	HDMR	94.9%	90.3%	94.5%	44.78	69.17
	DKL (man.)	99.7%	99.4%	99.6%	22.07	38.93
	DKL (opt.)	99.9%	99.5%	99.9%	11.91	34.94
Soot	HDMR	47.6%	36.9%	46.5%	0.17	0.30
	DKL (man.)	74.0%	44.0%	71.0%	0.09	0.22
	DKL (opt.)	94.4%	56.1%	90.6%	0.05	0.17

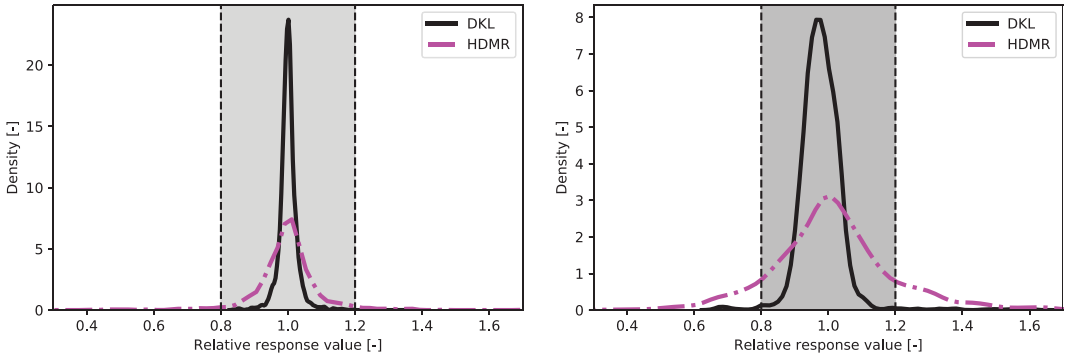


Figure 6. *Densities of prediction values relative to experiment using high-dimensional model representation (HDMR) and deep kernel learning (DKL) regression for* NO_x *and soot emissions, respectively. DKL generates better predictions in terms of the number of points within 20% of the experimental values, and the difference is greater for soot.*

HDMR for the regression of both outputs as the density is more centralized at unity for DKL than HDMR for both outputs. It is also clear that the number of predictions within 20% of the experimental value made by DKL is more than those made by HDMR, and this difference is greater for the soot prediction.

In Table 5, we show a comparison between DKL and HDMR on NO_x regression in terms of their training time and evaluation time. We see that HDMR is significantly computationally cheaper than the DKL method. The training time for DKL is 56 seconds whereas the HDMR fitting takes less than 2 s on the same device. Hence, the trade-off between accuracy and computational cost needs to be taken into

Table 5. CPU-time comparison between deep kernel learning (DKL) and high-dimensional model representation (HDMR).

Surrogate	Training time (s)	Evaluation time (s)
DKL	56	3
HDMR	2	1

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

Table 6. Root mean squared error (RMSE) performance of the considered surrogates on the diesel dataset.

Output	GP (RBF)	GP (SM)	DNN	DKL (RBF)	HDMR
NO_x	110.06	777.93	34.282	11.91	44.78
Soot	4.77	2.28	0.52	0.05	0.17

Abbreviations: DKL, deep kernel learning; DNN, deep neural network; GP, Gaussian process; HDMR, high-dimensional model representation.

consideration when selecting surrogates for particular applications. It is worth noting that evaluation of a trained DKL model is computationally much cheaper than its training process, which indicates that a pretrained DKL model may be feasible in some near real-time applications. We furthermore note that a large part of this evaluation time is one-off overhead, such that batch-evaluation of collections of points can be achieved with relatively minor additional computational expense.

Table 6 compares plain GP regression using RBF and SM kernels, a stand-alone DNN, and DKL with RBF kernel, as well as HDMR with respect to RMSE for both NO_x and soot emissions. We observe that the performance of DKL is significantly superior to the plain GPs with either the RBF or the spectral mixture kernel on both outputs. This is consistent with expectation, since, as discussed in Section 2.4, the 14-dimensional input space of our engine dataset would be expected to cause problems for plain GPs. DKL also outperforms a stand-alone DNN, indicating a genuine benefit in the combination of a GP and a DNN.

In Figure 7, we show the RMSE loss history of the NO_x regression for DKL and plain neural network during the training process. We observe that the prediction losses with respect to the experimental values of the DKL training throughout the training process is consistently lower than those of the plain DNN training. Even at the beginning of the training, it can be observed that the loss for the DKL prediction is lower than for the plain DNN prediction. This agrees with our expectation due to the existence of a GP at the end of DKL, which automatically computes the maximum a posteriori (MAP) estimate given the training dataset as the prediction. We can also see that the loss history curve for the DKL is much smoother than that for the DNN. Hence the RMSE loss of the DKL training process is a robust estimator for the performance of the trained model whereas for the DNN training, the RMSE estimates the performance of the model with larger uncertainty.

From Figures 5–7 and Table 6 we conclude that the DKL shows improvement over HDMR, plain GPs and plain neural networks in the regression tasks on the considered diesel engine emission dataset.

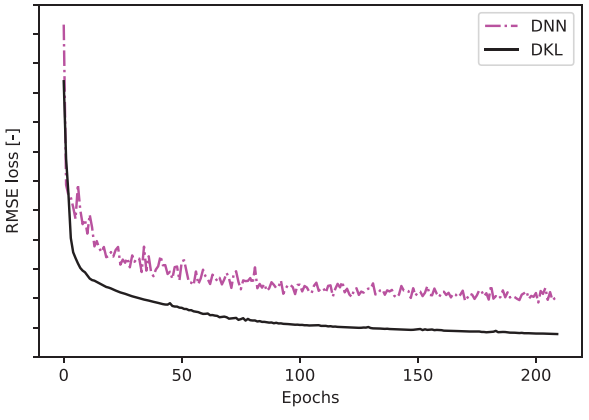


Figure 7. *Loss history of training deep kernel learning (DKL) as well as a plain deep neural network (DNN) for* NO_x *regression.*

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

4. Conclusions

In this paper, we studied DKL as a surrogate model for diesel engine emission data. Instead of using a physical-based model for modeling the complex system, we have taken a purely data-driven approach. DKL was applied to a commercial Diesel engine dataset for NO_x and soot emissions comprising 1,861 data points, with 14 operating condition variables as inputs. We employed a systematic two-stage procedure, consisting of a quasi-random global search and a local gradient-free optimization stage, to determine seven DKL hyperparameters, which include network architecture as well as kernel and learning parameters. It was found that the global search, conducted through sampling 1,000 Sobol points, was most effective in identifying a suitable set of hyperparameters. Local optimization was found largely ineffective for the network architecture hyperparameters, but led to minor improvement for the kernel and learning parameters. We compared DKL to standard deep feedforward neural networks, GPs, as well as HDMR, and the results indicate that, overall, DKL outperforms these methods in terms of regression accuracy as measured by RMSE on the considered 14-dimensional engine dataset for NO_x and soot modeling. Further research themes could involve designing DKL with nonstationary kernel functions to deal with the heteroskedasticity of the input arguments.

Acknowledgments. Changmin Yu is thankful to CMCL Innovations for the internship program offered.

Funding Statement. This work was partly funded by the National Research Foundation (NRF), Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) program, and by the European Union Horizon 2020 Research and Innovation Program under grant agreement 646121. Changmin Yu was funded by a Shell PhD studentship. Markus Kraft gratefully acknowledges the support of the Alexander von Humboldt foundation.

Competing Interests. The authors declare no competing interests exist.

Authorship Contributions. Conceptualization: M.S., S.M., M.K., and A.B.; Data curation: M.P., M.D., and V.P.; Formal analysis: C.Y., M.S., S.M., and G.B.; Funding acquisition: M.K., V.P., A.B., and S.M.; Investigation: C.Y., M.S., S.M., and G.B.; Methodology: M.S., S.M., and G.B.; Software: M.S., S.M., and G.B.; Supervision: S.M., M.K., V.P., and A.B.; Validation: S.M., G.B., M.P., and M.D.; Visualization: C.Y., M.S., and S.M.; Writing-original draft: C.Y., M.S., and S.M.; Writing-review & editing: S.M.; All authors approved the final submitted draft.

Data Availability Statement. The data used in this work are proprietary and confidential.

Ethical Standards. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

References

Azadi P, **Brownbridge GP**, **Mosbach S**, **Inderwildi OR and Kraft M** (2014) Production of biorenewable hydrogen and syngas via algae gasification: a sensitivity analysis. *Energy Procedia 61*, 2767–2770.

Baran M and Bieniasz LK (2015) Experiments with an adaptive multicut-HDMR map generation for slowly varying continuous multivariate functions. *Applied Mathematics and Computation 258*, 206–219.

Barron AR (1993) Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory 39*(3), 930–945.

Berger B and Rauscher F (2012) Robust Gaussian process modelling for engine calibration. *IFAC Proceedings Volumes 45*(2), 159–164.

Berger B, **Rauscher F and Lohmann B** (2011) Analysing Gaussian processes for stationary black-box combustion engine modelling. *IFAC Proceedings Volumes 44*(1), 10633–10640.

CMCL Innovations (2018) MoDS (Model Development Suite), Version 0.9.0. Available at https://cmclinnovations.com/products/mods/. Accessed 05 December 2018.

Drucker H, **Burges CJC**, **Kaufman L**, **Smola AJ and Vapnik V** (1997) Support vector regression machines. In Mozer MC, Jordan MI and Petsche T (eds), *Advances in Neural Information Processing Systems 9*. Cambridge, MA: MIT Press, pp. 155–161.

Frenklach M, **Wang H and Rabinowitz MJ** (1992) Optimization and analysis of large chemical kinetic mechanisms using the solution mapping method — Combustion of methane. *Progress in Energy and Combustion Science 18*, 47–73.

Gardner J, **Pleiss G**, **Weinberger KQ**, **Bindel D and Wilson AG** (2018) GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*, pp. 7587–7597. Red Hook, NY: Curran Associates Inc.

Garud SS, **Karimi IA**, **Brownbridge GP and Kraft M** (2018a) Evaluating smart sampling for constructing multidimensional surrogate models. *Computers & Chemical Engineering 108*, 276–288.

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

Garud SS, **Karimi IA and Kraft M** (2017a) Design of computer experiments: A review. *Computers & Chemical Engineering 106*, 71–95.

Garud SS, **Karimi IA and Kraft M** (2017b) Smart sampling algorithm for surrogate model development. *Computers & Chemical Engineering 96*, 103–114.

Garud SS, **Karimi IA and Kraft M** (2018b) LEAPS2: Learning based evolutionary assistive paradigm for surrogate selection. *Computers & Chemical Engineering 119*, 352–370.

Ghanbari M, **Najafi G**, **Ghobadian B**, **Mamat R**, **Noor M and Moosavian A** (2015) Support vector machine to predict diesel engine performance and emission parameters fueled with nano-particles additive to diesel fuel. In *IOP Conference Series: Materials Science and Engineering*, Vol. *100*. Bristol, UK: IOP Publishing, p. 012069.

Ghobadian B, **Rahimi H**, **Nikbakht A**, **Najafi G and Yusaf T** (2009) Diesel engine performance and exhaust emission analysis using waste cooking biodiesel fuel with an artificial neural network. *Renewable Energy 34*(4), 976–982.

He K, **Zhang X**, **Ren S and Sun J** (2015) Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, Los Alamitos, CA: IEEE Computer Society, Conference Publishing Services (CPS), pp. 1026–1034.

Hooke R and Jeeves TA (1961) "Direct search" solution of numerical and statistical problems. *Journal of the ACM (JACM) 8*(2), 212–229.

Huang G-B and Chen L (2007) Convex incremental extreme learning machine. *Neurocomputing 70*(16–18), 3056–3062.

Huang G-B, **Zhou H**, **Ding X and Zhang R** (2012) Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42*(2), 513–529.

Huang G-B, **Zhu Q-Y and Siew C-K** (2006) Extreme learning machine: Theory and applications. *Neurocomputing 70*(1–3), 489–501.

Janakiraman VM, **Nguyen X and Assanis D** (2016) Stochastic gradient based extreme learning machines for stable online learning of advanced combustion engines. *Neurocomputing 177*, 304–316.

Jeong S, **Obayashi S and Minemura Y** (2008) Application of hybrid evolutionary algorithms to low exhaust emission diesel engine design. *Engineering Optimization 40*(1), 1–16.

Joe S and Kuo FY (2008) Constructing Sobol sequences with better two-dimensional projections. *SIAM Journal on Scientific Computing 30*(5), 2635–2654.

Kastner CA, **Braumann A**, **Man PL**, **Mosbach S**, **Brownbridge GP**, **Akroyd J**, **Kraft M and Himawan C** (2013) Bayesian parameter estimation for a jet-milling model using Metropolis-Hastings and Wang-Landau sampling. *Chemical Engineering Science 89*, 244–257.

Kingma DP and Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Lai J, **Parry O**, **Mosbach S**, **Bhave A and Page V** (2018) Evaluating emissions in a modern compression ignition engine using multi-dimensional PDF-based stochastic simulations and statistical surrogate generation. SAE Technical Paper 2018-01-1739. https://doi.org/10.4271/2018-01-1739.

LeCun Y, **Bengio Y and Hinton G** (2015) Deep learning. *Nature 521*(7553), 436.

Li G, **Wang S-W and Rabitz H** (2002) Practical approaches to construct RS-HDMR component functions. *The Journal of Physical Chemistry A 106*(37), 8721–8733.

Lughofer E, **Macián V**, **Guardiola C and Klement EP** (2011) Identifying static and dynamic prediction models for NOx emissions with evolving fuzzy systems. *Applied Soft Computing 11*(2), 2487–2500.

Malikopoulos AA, **Papalambros PY and Assanis DN** (2007) A learning algorithm for optimal internal combustion engine calibration in real time. In *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, pp. 91–100.

Minh HQ, **Niyogi P and Yao Y** (2006) Mercer's theorem, feature maps, and smoothing. In *International Conference on Computational Learning Theory*. Springer, pp. 154–168.

Mosbach S, **Braumann A**, **Man PL**, **Kastner CA**, **Brownbridge GP and Kraft M** (2012) Iterative improvement of Bayesian parameter estimates for an engine model by means of experimental design. *Combustion and Flame 159*(3), 1303–1313.

Mosbach S, **Hong JH**, **Brownbridge GP**, **Kraft M**, **Gudiyella S and Brezinsky K** (2014) Bayesian error propagation for a kinetic model of n-propylbenzene oxidation in a shock tube. *International Journal of Chemical Kinetics 46*(7), 389–404.

Najafi G, **Ghobadian B**, **Moosavian A**, **Yusaf T**, **Mamat R**, **Kettner M and Azmi WH** (2016) SVM and ANFIS for prediction of performance and exhaust emissions of a SI engine with gasoline-ethanol blended fuels. *Applied Thermal Engineering 95*, 186–203.

Najafi G, **Ghobadian B**, **Tavakoli T**, **Buttsworth D**, **Yusaf T and Faizollahnejad M** (2009) Performance and exhaust emissions of a gasoline engine with ethanol blended gasoline fuels using artificial neural network. *Applied Energy 86*(5), 630–639.

Neal RM (1996) *Bayesian Learning for Neural Networks, Volume 118 of Lecture Notes in Statistics*. New York, NY: Springer Science+Business Media.

Niu X, **Yang C**, **Wang H and Wang Y** (2017) Investigation of ANN and SVM based on limited samples for performance and emissions prediction of a CRDI-assisted marine diesel engine. *Applied Thermal Engineering 111*, 1353–1364.

Park J and Sandberg IW (1991) Universal approximation using radial-basis-function networks. *Neural Computation 3*(2), 246–257.

Paszke A, **Gross S**, **Chintala S**, **Chanan G**, **Yang E**, **DeVito Z**, **Lin Z**, **Desmaison A**, **Antiga L and Lerer A** (2017) Automatic differentiation in PyTorch. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA.

https://doi.org/10.1017/dce.2020.4 Published online by Cambridge University Press

Quiñonero-Candela J and Rasmussen CE (2005) A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research 6*, 1939–1959.

Rabitz H and Aliş ÖF (1999) General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry 25*(2–3), 197–233.

Sayin C, **Ertunc HM**, **Hosoz M**, **Kilicaslan I and Canakci M** (2007) Performance and exhaust emissions of a gasoline engine using artificial neural network. *Applied Thermal Engineering 27*(1), 46–54.

Sikorski JJ, **Brownbridge G**, **Garud SS**, **Mosbach S**, **Karimi IA and Kraft M** (2016). Parameterisation of a biodiesel plant process flow sheet model. *Computers & Chemical Engineering 95*, 108–122.

Silitonga AS, **Masjuki HH**, **Ong HC**, **Sebayang AH**, **Dharma S**, **Kusumo F**, **Siswantoro J**, **Milano J**, **Daud K**, **Mahlia TMI**, **Cheng W-H and Sugiyanto B** (2018) Evaluation of the engine performance and exhaust emissions of biodiesel-bioethanol-diesel blends using kernel-based extreme learning machine. *Energy 159*, 1075–1087.

Srivastava N, **Hinton G**, **Krizhevsky A**, **Sutskever I and Salakhutdinov R** (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*(1), 1929–1958.

Tietze N (2015). *Model-Based Calibration of Engine Control Units Using Gaussian Process Regression.* PhD thesis, Technische Universität Darmstadt.

Vaughan A and Bohac SV (2015) Real-time, adaptive machine learning for non-stationary, near chaotic gasoline engine combustion time series. *Neural Networks 70*, 18–26.

Williams CK and Rasmussen CE (2006) *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press.

Wilson A and Adams R (2013). Gaussian process kernels for pattern discovery and extrapolation. *Proceedings of Machine Learning Research*, *28*(3), 1067–1075.

Wilson A and Nickisch H (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). *Proceedings of Machine Learning Research*, *37*, 1775–1784.

Wilson AG, **Hu Z**, **Salakhutdinov R and Xing EP** (2016) Deep kernel learning. *Proceedings of Machine Learning Research*, *51*, 370–378.

Wong KI, **Wong PK and Cheung CS** (2015) Modelling and prediction of diesel engine performance using relevance vector machine. *International Journal of Green Energy 12*(3), 265–271.

Wong PK, **Gao XH**, **Wong KI and Vong CM** (2018) Online extreme learning machine based modeling and optimization for point-by-point engine calibration. *Neurocomputing 277*, 187–197.

Yilmaz N, **Ileri E**, **Atmanlı A**, **Karaoglan AD**, **Okkan U and Kocak MS** (2016) Predicting the engine performance and exhaust emissions of a diesel engine fueled with hazelnut oil methyl ester: The performance comparison of response surface methodology and LSSVM. *Journal of Energy Resources Technology 138*(5), 052206.

Yusaf TF, **Buttsworth DR**, **Saleh KH and Yousif BF** (2010) CNG-diesel engine performance and exhaust emission analysis with the aid of artificial neural network. *Applied Energy 87*(5), 1661–1669.

Cite this article: Yu, C. Seslija, M. Brownbridge, G. Mosbach, S. Kraft, M. Parsi, M. Davis, M. Page, V. and Bhave, A. 2020. Deep kernel learning approach to engine emissions modeling. *Data-Centric Engineering,* 1: e4. doi:https://doi.org/10.1017/dce.2020.4