Extrapolative Bayesian Optimization with Gaussian Process and Neural Network Ensemble Surrogate Models

Yee-Fun Lim,*, Chee Koon Ng, U.S. Vaitesswar, and Kedar Hippalgaonkar*

Bayesian optimization (BO) has emerged as the algorithm of choice for guiding the selection of experimental parameters in automated active learning driven high throughput experiments in materials science and chemistry. Previous studies suggest that optimization performance of the typical surrogate model in the BO algorithm, Gaussian processes (GPs), may be limited due to its inability to handle complex datasets. Herein, various surrogate models for BO, including GPs and neural network ensembles (NNEs), are investigated. Two materials datasets of different complexity with different properties are used, to compare the performance of GP and NNE—the first is the compressive strength of concrete (8 inputs and 1 target), and the second is a simulated high-dimensional dataset of thermoelectric properties of inorganic materials (22 inputs and 1 target). While NNEs can converge faster toward optimum values, GPs with optimized kernels are able to ultimately achieve the best evaluated values after 100 iterations, even for the most complex dataset. This surprising result is contrary to expectations. It is believed that these findings shed new light on the understanding of surrogate models for BO, and can help accelerate the inverse design of new materials with better structural and functional performance.

1. Introduction

Automated high-throughput experimentation guided by artificial intelligence (AI) has emerged as a valuable tool for accelerating scientific research and materials discovery.[1,2] At the core of these efforts is a data-driven approach, the first generation of which mainly utilized forward predictive machine learning algorithms operating on open-source databases[3–5] or accumulated data from the literature,[6–9] where algorithms trained on such datasets are then able to predict the desired material property or output given a set of values for the input features.[10–13] Recently, optimization algorithms have been gaining attention and importance in this field. In contrast to forward prediction, these algorithms work in reverse to converge on the inputs that give the optimum desired output, and have been demonstrated to drive various automated experimental setups.[14–24] In these efforts, Bayesian optimization (BO) has been the algorithm of choice,[14–21] although “Stable Noisy Optimization by Branch and Fit” (SNOBFIT) has been used in some instances,[22,23] while genetic algorithm is the basis of another optimization work.[24]

BO is well suited for finding the extremum of black box functions that are expensive to evaluate and derivative-free (lacking information on the first and second derivatives).[25,26] Typically, Gaussian process (GP) regression is used as the surrogate model, which tries to model the objective function via Bayesian inference.[25] Furthermore, BO is able to achieve a balance between exploitation (converging toward optimum point) and exploration (choosing inputs in parameter space region with largest uncertainty).[25] Given its strengths, BO should be ideal for the optimization of scientific experiments, at least in principle. However, in practice, the results have been mixed. While some researchers concluded that BO is well suited for handling the high-dimensional parameter space that is typical of experimental input variables,[15,17] others opined that BO with GP is inadequate at modeling the complex space that has been explored in their study.[22] In particular, we note that in one of the published studies which reported BO-driven automated experiments, the BO required almost 400 iterations to arrive at the final optimum results.[14] In another study, the final optimized value that was obtained by the BO was only around 20–30% higher than that achieved by the initial runs.[19] These results suggest that BO with GP may not be very efficient at converging toward the optimum value.

Some of the shortcomings of the BO may be attributed to the limitations of GP regression.[25] It has been noted that the computational requirements of GPs scale cubically with the number of observations, therefore they become impractical with large datasets.[27,28] Furthermore, GPs do not scale well to high dimensions, and become inefficient when the number of input features become very large.[28] These shortcomings will transfer to the BO when GP is used as the surrogate model,
and the BO may thus not converge effectively to the optimum value. As an alternative, random forest regression has been used as the surrogate model for BO, but may not be a desirable approach due to the inability of random forest to extrapolate beyond existing data.\cite{26}

To overcome these shortcomings, it has been proposed that using neural networks as the surrogate model would allow the BO to scale much more efficiently for the optimization of complex datasets.\cite{27} To retain the statistical properties provided by GPs that allow for inference of uncertainty information, it will be necessary to make use of an ensemble of neural networks with a range of predictions rather than a single network with a fixed prediction. The uncertainty is then inferred from the variance in the predictions given by the neural networks. An example of such implementation would be a Bayesian neural network (BNN) with weights that are not fixed but rather drawn from a distribution.\cite{27,28} A single BNN would therefore represent an ensemble of neural networks. As exact Bayesian inference for neural networks is computationally intractable, all of the published BNN approaches are in fact clever approximations.\cite{16,27–30} In one notable approach, Gal and coworkers showed that neural networks with dropout were an effective approach as a Bayesian approximation to model uncertainty.\cite{30} In yet another implementation, Abolhasani et al. utilized randomly generated neural network architectures to produce an ensemble of neural networks as the surrogate model.\cite{22,31}

While the aforementioned approaches have laid the groundwork on neural network ensembles, a systematic comparison of the different surrogate models for BO is still lacking. Furthermore, there are only a very limited number of experimental efforts using BO that make use of neural network surrogate models\cite{16,22,31} and more detailed studies are needed. In this work, we conducted a comprehensive study on the performance of various surrogate models for BO applied to the optimization of several open source material science datasets, with particular emphasis placed on the comparison between GP and neural network surrogate functions. We used some datasets from the UCI Machine Learning Repository,\cite{32} as well as a computationally calculated electronic transport database for inorganic materials\cite{33} from which we extracted a high-dimensional dataset with 22 inputs. Machine learning algorithms are first trained on these datasets and then used as predictive oracles to simulate new experimental runs within the parameter space of these datasets (refer to Figure 1 for a summary of the workflow). The performance of different BO surrogate models will be determined by their speed of convergence toward the optimum and also the best evaluated value that is obtained for each dataset. Here, we would also like to emphasize that in the choice of algorithms for prediction and optimization, the ability to extrapolate accurately into unexplored regions of the parameter space is critical. Good extrapolation would be especially important if the goal is to find experimental inputs that can outperform the previous best. In this work, we have thus established a workflow to evaluate the extrapolative performance of the various algorithms and surrogate models, and then relate this to their optimization performance. We found that the neural network ensemble not only

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{Summary of the simulated experimental workflow that is enabled by a machine learning predictive oracle, with different surrogate models for the BO.}
\end{figure}
extrapolates well, but also enables the fastest convergence toward the optimum values. However, GP eventually catches up and consistently achieved the best evaluated values after 100 iterations for each dataset.

2. Results

2.1. Optimization on Synthetic Functions

The optimization runs were implemented using the open source Python library, scikit-optimize (skopt). The traditional BO with GP as the surrogate model was implemented in skopt using the built-in method gp_minimize. To implement the neural network ensemble surrogate functions, we coded our own custom regressors that are compatible with the scikit-learn (sklearn) machine learning Python library so as to be able to use them with skopt. Two different NN ensembles (NNEs) have been implemented. The first is a straightforward dense neural network (DNN) implemented with sklearn that is trained multiple times on the same data to obtain ensemble statistics (henceforth referred to as the neural ensemble); although the training data is the same, random initialization will result in different predictions each time. The second is a NN with dropout (implemented with keras) which, as described earlier, is an ensemble in itself (henceforth referred to as the neural dropout). These regressors are able to return both the mean and the standard deviation of the ensemble predictions. More details on the implementation can be found in the Experimental Section.

Before running the optimization on materials science datasets, we first tested our surrogate models on synthetic functions.
We chose the Branin function and the Rosenbrock function, which are two test functions that are commonly used for testing of optimization algorithms.[16,38] These 2D functions are plotted and visualized in Figure 2; the Branin function has global minima of 0.398 at three separate points, while the Rosenbrock function has a long, parabolic shape with a global minima of 0 at a single point. Figure 2 shows that our NNE surrogates are competent when applied to the optimization of the synthetic functions, outperforming dummy, random forest, and gradient boosting surrogate models. We found that GP still shows the best performance, although our neural ensemble is capable of comparable performance for the optimization of the Rosenbrock function in terms of convergence speed and minimum value achieved.

2.2. Concrete Compressive Strength Dataset

We chose a dataset reporting concrete compressive strength[39] as our first test case of applying our surrogate models on a materials science dataset. This dataset has eight input parameters representing various composition and process variables and comprises 1030 entries. Thus, it is a reasonably large dataset with sufficient complexity to be used as a good test case of a real-world material science dataset. A snapshot and data distribution of the dataset are shown in Figure S1 and S2, Supporting Information.

To test the performance of our surrogate models on this dataset, it will be necessary to make new queries and perform new evaluations of the concrete compressive strength, which in practice will be done via new experiments using the same input parameters. Here, we will simulate these experiments using machine learning algorithms that have been trained on the dataset, which will serve as oracles that will predict the outcome of the new experiments. The use of machine learning enabled simulated experiments has been reported previously in various publications, to speed up the evaluation of various optimization parameters which would otherwise be too slow to implement in actual experimental iterations.[15–17]

Before choosing the machine learning algorithm to serve as the predictive oracle, it would first be necessary to determine their predictive performance on the dataset. We have chosen to evaluate commonly used regressors, including linear regression, polynomial regression (degree 2 and degree 3), random forest, and gradient boosting, as well as the GP regressor and the NNEs that we will utilize for our surrogate models. Each regressor was trained on the training dataset, and then evaluated on both the training and test sets. The performance of each regressor is shown in Figure 3, which shows the scatter plots of the predicted versus actual values together with the metric root mean squared error (RMSE) of actual versus predicted values of compressive strength. As can be seen from the metrics comparison summary in Figure 4, our custom regressors neural ensemble...
and neural dropout perform favorably when compared with the others.

It is worthwhile to discuss the effect of the hyperparameters for the GP regressor on the algorithm performance. Here, we have chosen to use the Matern kernel \((\nu = 5/2)\), which allows control of the smoothness of the kernel and is a popular choice for machine learning.\(^{[40]}\) In our skopt implementation, it is possible to control the bounds for the length scale of the kernel variation. Figure S3, Supporting Information, shows the effects of different kinds of bounds. For a fixed length scale, the algorithm shows severe overfitting, as it is able to predict very well on the training set but fails on the test set. The performance on the test set improves as the bounds are allowed to relax gradually and the algorithm is allowed to converge on the optimum length scale, which indicates that finding the correct length scale is important for predicting unseen data. In view of this, we have opted to relax the bounds for our predictive oracle and also for the GP surrogate model that will be used in the optimization algorithm.

There are several algorithms that seem to perform almost equally well as evaluated by test set metrics (Figure 4), including polynomial (degree 3), gradient boosting, random forest, GP, neural ensemble, and neural dropout regression. Based purely on such goodness-of-fit metrics, it would appear that any of these algorithms can be a suitable choice for the predictive oracle that would be used for the optimization step. However, it should be noted that these algorithms have been evaluated on the test set, which while different from the training set is actually still quite similar in terms of input experimental bounds as well as the output compressive strength values. From an optimization perspective, it is desirable to find experimental inputs that can go beyond the bounds of the original dataset. Therefore, the algorithms need to be evaluated based on their extrapolative ability, which means the test set needs to be significantly different from the training set. In view of evaluating which algorithm can predict inputs for higher compressive strength, we use a new strategy where we select the training set to correspond to data with the lower half of the compressive strength values, and test set to be the data with the higher half of compressive strength.

The extrapolative performance (RMSE score for test set predictions) of each algorithm can be gleaned from Figure 5. As expected, the trained algorithms can predict reasonably well on the training set, but now the test set performance is a lot worse due to the large differences between the two datasets. The polynomial regressors predict values for the test set that are completely different from the original, resulting in high RMSE scores relative to the other algorithms. The tree-based regressors also performed badly with fairly high RMSE scores as well, as the regressors seem to have an inherent limit in their

---

**Figure 4.** Comparison of various metrics (obtained from the test set) for the different algorithms trained on the concrete compressive strength dataset, including RMSE and R2 score.
predictions, and therefore they are poor extrapolators. This behavior may be attributed to the way the Decision Tree algorithm works, which is by generating nodes at each decision point and binning the dataset into several end nodes, resulting in nodes which cannot go beyond the bounds of the dataset. In this regard, gradient boosting can do slightly better than random forest because the boosting stages of the algorithm can increase the original predicted value and therefore allow predictive values that are slightly above the previous maximum.

For the choice of the best extrapolative algorithm among those that have been studied here, it is a toss-up between GP and neural ensemble. Both achieved relatively low RMSE scores for the test set; furthermore, they correctly indicated that the test set predictions should have much higher uncertainties than the training set. We have decided to go with the trained neural ensemble as the algorithm for our predictive oracle, due to its better RMSE score and a smaller number of outliers in its predictions. The concrete compressive strength is thus optimized using this predictive oracle function, and the best concrete strength evaluations at each iteration (averaged over 10 runs where each run evaluates the predictions over 100 iterations) are shown in Figure 6. Both the neural ensemble and neural dropout models seem to converge to a high value after a small number of iterations, but saturate after a while. In contrast, the GP starts slowly but eventually is able to beat out the other models (after ≈30 iterations) and ultimately able to find the highest compressive strength among all the models.

Figure 5. Regressor evaluations repeated with the training set chosen to be the data with the lower half of the concrete compressive strength, while the test set is the upper half. For regressors that can provide ensemble uncertainty information, the 95% confidence interval is indicated as well. The extrapolative performance is indicated by the prediction metrics (RMSE scores) on the test set.

Figure 6. Best concrete strength evaluations averaged over ten runs, using the trained neural ensemble algorithm as the predictive oracle and compared across different surrogate models. It can be seen that the GP model converges slowly at the start, but is ultimately able to find the highest compressive strength among all the models.
The predictions of the surrogate model at each iteration are a complex function of all the eight input variables and would be too complicated to plot on a graph. However, in an attempt to understand the behavior of the different models, we have decided to visualize the predicted values and uncertainties at each iteration by varying just one variable (the cement component). While this is not by any means an accurate representation of the full complex model, it does provide some insightful information about each model. Figure 7 shows the plots of the model predicted values and uncertainties after going through eight optimization iterations (with four initial random sampling points). It can be seen that the predictions and uncertainties of the gradient boosting and random forest models are "blockish" and do not follow too closely the ground truth, which may be expected as they are tree-based models. The neural dropout is able to better follow the ground truth, although the prediction uncertainty seems relatively constant within the range being plotted (when the range is expanded, the uncertainty increases in regions far away from the observed points; see Figure S4, Supporting Information). Better uncertainty estimates can be obtained from the GP and neural ensemble models, showing greater uncertainties in unsampled regions. The GP model is able to model the uncertainties smoothly, although the prediction and uncertainty manifolds can be quite unstable especially for the model with the larger length scale bounds (see Figure S5 and S6, Supporting Information). This could explain its low convergence speed at the beginning, although its smoother model of the uncertainty information probably enables it to converge toward the most optimum values in the later iterations.

2.3. Thermoelectrics Power Factor Dataset

The GP model performed well on the concrete compressive strength dataset, but this dataset may not have enough complexity to truly stress test the model. We have seen that for the Branin and Rosenbrock functions that there is not a huge difference in performance across the different models. Furthermore, we have also looked at another simple dataset, a combined cycle power plant dataset that relates four input variables to its electrical energy output (Figures S7 and S8, Supporting Information).[41,42] Applying a similar optimization workflow to this dataset, we found that all surrogate models show fairly similar performance (Figure S9, Supporting Information). It appears that for a simple dataset such as this one, most of the surrogate models do equally well, and it is difficult to differentiate between them. Therefore, a greater level of dataset complexity is needed to truly test the limits of these models.
In search of a dataset with a larger number of input parameters, we turned to an open-source electronic transport database that was published by Ricci et al.[43] which was obtained by mining data from the Materials Project[3] followed by subsequent computation of the desired parameters. The dataset contains a large number of electronic transport properties, but we limit our dataset output to thermoelectric power factor, defined as the square of the Seebeck coefficient multiplied by the electrical conductivity, and select only the materials with a cubic crystal structure. Despite these restrictions, we still obtained a high-dimensional dataset with 22 input parameters and 94 250 entries. A snapshot of the dataset is shown in Figure S10, Supporting Information. Note here that a constant relaxation time approximation was used to obtain this dataset (scattering of charges not considered) and therefore the original units were in (W m\(^{-1}\)K\(^{-1}\)). The Seebeck coefficient in the dataset was reported in \(\mu\)VK\(^{-1}\) with conductivity as (1/\(\Omega\)ms). Considering typical constant relaxation times of 10\(^{-14}\) s, we have converted the power factor to convenient units of mW/mK\(^2\) (allowing for comparison to state-of-the-art values).

This dataset is different from the previous two, not only in its size and complexity but also in the distribution of its data values. The power factor, for instance, spans over 12 orders of magnitude, while many of the inputs are also skewed in their distribution. It is thus more challenging to predict using common machine learning algorithms. To overcome these challenges, we apply a Yeo–Johnson transformation to make the input data closer to that of a normal distribution,[44] while a cube root transformation is applied to the power factor to reduce the range of its values. Both transformations are applied on top of the usual Standard Scaler transformation that is applied to the other two datasets. This approach allowed us to progressively reduce the predictive RMSE scores (see Figure S11, Supporting Information).

As before, we investigated the extrapolative properties of the various machine learning algorithms, with the results plotted in Figure 8. The GP regressor has been omitted, due to the large amounts of memory needed for computation of the whole dataset. Indeed, this large dataset showcases one of the limitations of the GP algorithm in handling datasets with a large number of entries, as discussed earlier. Similar to the results for the concrete dataset, the test RMSE scores are significantly worse compared with the training RMSE scores, which may be attributed to the large differences between the training (bottom half of dataset) and test (top half of dataset) sets. The point here is not to optimize the test set scores to become comparable to the training scores, but rather to pick the algorithm with the best scores. Once again, the neural ensemble regressor has the lowest

Figure 8. Plots of the extrapolative performance of the various machine learning algorithms on the thermolectric power factor dataset. Similar to the concrete dataset, the test set predictions here gave much larger RMSE when compared with the training set, due to large differences between the two. The algorithm with the lowest test set RMSE, neural ensemble regression, is deemed the best performing extrapolative algorithm and is chosen as the predictive oracle for the optimization runs.
RMSE score for the test set predictions; therefore, it is the best extrapolative algorithm and proves to be the best predictive oracle for the thermoelectrics dataset.

Although omitted as a predictive oracle, GP is still investigated as one of the surrogate models for the optimization workflow because the small number of iterations make the computation much more tractable. However, it should be noted that for such a complex dataset, engineering the bounds of the kernel length scale becomes even more important. Indeed, as shown in the 1D visualization in Figure 9, insufficient bounds will result in a predictive and uncertainty manifold that is vastly inadequate at extrapolating the correct values. When the bounds are extended by a sufficient amount, the manifold is able to follow the ground truth a lot more closely, so that the algorithm is able to converge better on the correct values. Furthermore, it is also important to tweak the value of the alpha parameter which is representative of the noise inherent in the data (see Experimental Section for more information, and Figure S12, Supporting Information, for an illustration of the effect of alpha on the prediction and uncertainty manifolds). The behavior of the other surrogate models is similar to that observed for the concrete dataset.

With the neural ensemble predictive oracle, the thermoelectrics dataset is optimized for the power factor. The best power factor evaluations at each iteration are shown in Figure 10.

Here, it can be seen that the GP model is still the best performing model, even for such a complex dataset. This result is somewhat
surprising because previous studies\textsuperscript{[27,28]} have suggested that GP should struggle with such a dataset. It is worth mentioning that the GP length-scale bounds and also alpha values have been optimized, and suboptimal performance can result from nonoptimized parameters (see Figure S13, Supporting Information). The neural ensemble model came in a close second; it showed fast convergence during early iterations, but was ultimately overtaken by the GP model. Although the performance is slightly inferior, there is no need to finely tune the hyperparameters of the neural ensemble; therefore, its performance could be more consistent and robust.

3. Conclusion

We have performed a comprehensive evaluation of the extrapolative performance of the various machine learning algorithms into the unexplored regions of the parameter space. When experimentalists discover new materials, it is often desirable to search for material properties that are expected to be better than what has currently been reported. Thus, it is important for optimization frameworks to have the ability to go beyond what was the previous best for the dataset, rather than simply operating within the confines of the existing bounds and parameters. By separating the training and test sets to be the bottom and top half of the dataset, respectively, we have shown that the GP and neural ensemble regressors are the more consistent extrapolative algorithms, based on the performance on the test set. These results show that the design of the GP allows to uncover the interdependent relationships between the inputs via careful kernel construction, while the NNEs are able to accurately extract the complex nonlinear relationships between the inputs and output\textsuperscript{[27]}

Regarding the performance of the various surrogate models when used for the optimization of the different datasets, the random forest and gradient boosting models are fairly consistent, in that they typically outperform dummy or random optimization slightly but not by much. We postulate that this is due to the poor extrapolative performance of these algorithms. The GP model is typically the top performer for simpler datasets with small number of inputs, and therefore its popular choice in state-of-the-art materials science research. It is also worth mentioning that in order for GP to properly map the manifold between the inputs and output, it is important to tune the bounds for the kernel length scale so that the algorithm is able to converge on the correct values. It may also be necessary to fine-tune the alpha noise parameter as well. With fully tuned parameters, the GP model becomes the top performer for even the most complex high-dimensional dataset (thermoelectric dataset with 22 inputs), which could be due to the ability of GP to smoothly map out the uncertainty manifolds. The neural ensemble also performed well and came in a close second, which may be attributed to its better ability at mapping out the complex input–output relationships. The neural ensemble could be the model of choice if more robust and consistent performance is desired because there is less need for precise hyperparameter tuning. We believe that our findings have shed new light on the understanding of surrogate models for BO, and can help to improve the speed of convergence toward optimal values as well as the best evaluated values that are ultimately achieved. Properly tuned GP and neural ensemble models can help accelerate the inverse design of new materials with better structural and functional performance, when used in conjunction with BO for the optimization of materials science experiments.

4. Experimental Section

Dataset Processing: Data operations are performed using the scikit-learn (sklearn) machine learning python library\textsuperscript{[46]} All the datasets go through the usual normalization using StandardScaler before being fed into machine learning algorithms. For a given set of sample data $x$, the scaled standard score $z$ is given by

$$z = \frac{x - \mu}{\sigma}$$ (1)

where $\mu$ is the mean of the samples and $\sigma$ is the standard deviation. Thus, StandardScaler transforms the data to have mean 0 and variance 1. In addition, for the thermoelectric power factor dataset, an additional Yeo–Johnson PowerTransformer method is applied to make the data distributions closer to a normal distribution, which is important to achieve good prediction results. The PowerTransformer fits each of the 22 inputs to obtain the $\lambda$ values for the Yeo–Johnson transformation that minimizes skewness. Depending on $\lambda$, the Yeo–Johnson transformation for an input $x$ is given by\textsuperscript{[44]}

$$y_i(\lambda) = \begin{cases} \frac{\ln(x + 1)}{\lambda} & \text{if } \lambda \neq 0, x \geq 0 \\ \frac{\ln(x + 1)}{\lambda} & \text{if } \lambda = 0, x \geq 0 \\ \frac{\ln(x - 1)}{\lambda} & \text{if } \lambda \neq 2, x < 0 \\ \frac{\ln(x - 1)}{\lambda} & \text{if } \lambda = 2, x < 0 \end{cases}$$ (2)

The fitted values of $\lambda$ for each input are given in Table S1, Supporting Information, which range from −2.33 to 1.30. Following which, the data are split into training and test sets using train_test_split, the former is used to train the machine learning algorithms while the latter is used for validation of the trained algorithms. Both random selection (20% of the dataset) and deliberate selection (top half of the dataset) are used to create the test set, with the latter being for the specific purpose to test the extrapolative performance of the machine learning algorithms.

Data visualizations make use of functions from the matplotlib library\textsuperscript{[45]} Various data visualizations are performed to compare the predicted values versus the actual values, uncertainty manifold of the predictions, and also the speed of convergence of the optimization runs.

Machine Learning: Machine learning regression algorithms are investigated for their predictive and extrapolative performance (as described previously), to select the best algorithm for use as the predictive oracle function for the dataset optimization. They are evaluated based on standard metrics, namely, RMSE and R2 score (coefficient of determination), calculated on the predicted $\hat{y}$ versus actual values $y$

$$\text{RMSE}(\hat{y}, y) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$ (3)

$$R^2(\hat{y}, y) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$ (4)

where $n$ is the number of samples, $y_i$ and $\hat{y}_i$ are the values for the $i$th sample, and $\bar{y}$ is the mean of the actual values $y$.

The sklearn algorithms LinearRegression, PolynomialFeatures (to implement polynomial regression with degree 2 and degree 3), GradientBoostingRegressor, and RandomForestRegressor are implemented using default parameters. The GaussianProcessRegressor is implemented using the scikit-optimize (skopt) library with an in-depth study on the kernel properties, while the NNEs are self-written customized regressors. We will describe these regressors in greater detail later.

GPs for machine learning have been previously documented in detail\textsuperscript{[40]} and also reviewed again in subsequent publications\textsuperscript{[25,26]}.
brief, a GP \( f(x) \) is a collection of variables with a joint Gaussian Distribution, and can be completely described by its mean \( m(x) \) and covariance \( k(x,x') \):\(^{[40]}\)

\[
m(x) = E[f(x)]
\]

\[
k(x, x') = E[(f(x) - m(x))(f(x') - m(x'))]
\]

(5)

where \( E[f(x)] \) is the expected value of the function \( f(x) \). When using GPs for machine learning regression, it will be necessary to select the covariance function or kernel \( k(x,x') \) to define the prior distribution for the function that maps the inputs to the output. The regression algorithm then calculates the posterior distribution using the observed values from the dataset. Clearly, the choice of the kernel function with its corresponding hyperparameters will have a great effect on the performance of the algorithm, and many options are available. We have chosen to use the Matern kernel, which is a popular class of kernels for machine learning that allows control of the function smoothness via the parameter \( \nu \), with typical values of \( 1/2, 3/2, \) and \( 5/2 \).\(^{[40]}\) The Matern kernel with \( \nu = 5/2 \) is given by\(^{[40]}\)

\[
k_{\nu=5/2}(x, x') = A \left( \frac{\sqrt{5d}}{\nu} + \frac{\sqrt{2}}{\nu} r \right) \exp \left( -\frac{\sqrt{5d}}{\nu} d \right)
\]

where \( A \) is a constant, \( r \) is the Euclidean distance between \( x_1 \) and \( x_2 \), and \( d \) is a length scale parameter. The \( \nu \) parameter has been chosen for deeper investigation for its effect on algorithm accuracy and optimization convergence performance. In the implementation, it is set as either fixed, variable within moderate bounds (\( 10^{-2} \) to \( 10^{2} \)) or allowed to vary in a large range (\( 10^{-5} \) to \( 10^{6} \)) for the concrete and power plant datasets, and \( 10^{-12} \) to \( 10^{12} \) for the thermoelectric dataset). Furthermore, we have also noted that in some cases it will be necessary to tweak the value of the alpha parameters of the regressor, which is equivalent to that of a White kernel\(^{[34]}\)

\[
k_{\nu=5/2}(x, x') = \text{MaternKernel} + \text{WhiteKernel}
\]

(8)

The NNE regressors are custom-written regressors. They are written using the sklearn regressor template, so as to be compatible with the skopt optimization algorithm that is used in the next step. These regressors are designed to implement the usual fit and predict API, with the added functionality that the predict function can return the standard deviation of the predictions if desired. To do so, an ensemble of NNS is required because a single NN is deterministic after being trained. For a single NN, the output of a hidden (intermediate) layer in the network is given by\(^{[45]}\)

\[
Z_m = \sigma\left( \alpha_{m0} + \sum_{p} \alpha_{mp}X_p \right), m = 1, \ldots, M
\]

(9)

where \( Z_m \) is the output of the \( m \)th node of a hidden layer with \( M \) nodes, \( X_p \) are the inputs to this hidden layer, \( \alpha_{mp} \) are the weights connecting the previous layer with this layer, and \( \sigma \) is the activation function which we have chosen to be the Rectified Linear Unit (ReLU) to handle nonlinear relationships between the inputs and outputs

\[
\sigma(x) = \max(0, x)
\]

(10)

After training, the weights \( \alpha_{mp} \) of the NN become fixed; therefore, it will always produce the same output when given a certain input. However, if the NN is trained again on the same data, the weights will become slightly different due to a random reinitialization of the weights,\(^{[28]}\) and the new prediction will be different. Therefore, our first implementation of the NeuralEnsembleDropoutRegressor is simply to retrain the NN several times (the ensemble size is a hyperparameter that can be tuned), and then output the mean and standard deviation of the predictions. This NN is implemented using the MLPRegressor in sklearn. In our second implementation, we created the NeuralEnsembleDropoutRegressor, which is a NN imple-mented in the keras library using dense and dropout layers.\(^{[46]}\) The dropout layer will randomly set input values to zero (thus turning off certain nodes), which can be implemented for both the training and prediction phase (the latter by setting training=True).\(^{[43]}\) As such, the regressor only needs to be trained once, and the output predictions will be different every time and will essentially be an ensemble, due to the random nature of the dropout layer.

BO: The optimization is implemented using the skopt library.\(^{[34]}\) The built-in optimization functions dummy_minimize, forest_minimize, gbrt_minimize, and gp_minimize are used to run the optimization using random sampling (dummy), random forest, gradient boosting, and GP as their respective surrogate models. The gp_minimize implementation is able to accept a user defined GP regressor, which we have defined to be one with a Matern kernel that has a tunable length scale with specified bounds. Our custom neural ensemble and neural dropout surrogate models are implemented using the base_minimize function with the base_estimator specified to be a NeuralEnsembleRegressor or a NeuralEnsembleDropoutRegressor, respectively. The acquisition function “Expected Improvement” is used to select the next point for sampling in the new iteration of the optimization workflow, which is defined by\(^{[23]}\)

\[
EI_n(x) = E[f(x) - f_{n^*}]\]

(11)

where \( E[f(x)] \) is the expectation of posterior function \( f(x) \) obtained from fitting the chosen surrogate model on the observed function values at previous sample points \( x_1, x_2, \ldots, x_n \) and \( f_{n^*} \) is the best value of the \( n \) function evaluations performed. The next sampling point \( x_{n+1} \) that is chosen for the subsequent optimization iteration is then the value of \( x \) that gives the maximum amount of expected improvement

\[
x_{n+1} = \arg\max(EI_n(x))
\]

(12)

As the \( EI \) takes into account both mean and variance (uncertainty of predictions), it should provide a balance between exploitation and exploration.\(^{[25]}\)

For each dataset, the optimization is performed for 100 iterations, and is repeated 10 times for each surrogate model. The best value that has been evaluated at each iteration (averaged over the ten runs) is then plotted for each model. The optimization performance is evaluated based on the speed at which the model converges on the optimum value, and also on the best value that is ultimately obtained.

Supporting Information
Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements
The authors acknowledge helpful technical discussions with Swee Liang Wong and Jatin Kumar. This work was supported by the Agency of Science, Technology and Research (A*STAR), Singapore, via two programmatic research grants (grant nos. A18980043 and A19E90103). The Interactive Supporting Information of this article can be found at: https://www.authorea.com/doi/full/10.22541/au.162679475.54895263/v1.

Conflict of Interest
The authors declare no conflict of interest.

Data Availability Statement
The data that support the findings of this study are openly available in Authorea, at https://www.authorea.com/doi/full/10.22541/au.162679475.54895263/v1.
Keywords

automated experiments, Bayesian optimization, extrapolative algorithms, machine learning, neural network ensembles