Journal of Global Optimization (2021) 81:203–231
https://doi.org/10.1007/s10898-020-00912-0

Surrogate optimization of deep neural networks for groundwater predictions

Juliane Müller[1] · Jangho Park[1] · Reetik Sahu[1] · Charuleka Varadharajan[2] · Bhavna Arora[2] · Boris Faybishenko[2] · Deborah Agarwal[1]

Received: 27 August 2019 / Accepted: 25 April 2020 / Published online: 26 May 2020

Abstract
Sustainable management of groundwater resources under changing climatic conditions require an application of reliable and accurate predictions of groundwater levels. Mechanistic multi-scale, multi-physics simulation models are often too hard to use for this purpose, especially for groundwater managers who do not have access to the complex compute resources and data. Therefore, we analyzed the applicability and performance of four modern deep learning computational models for predictions of groundwater levels. We compare three methods for optimizing the models' hyperparameters, including two surrogate model-based algorithms and a random sampling method. The models were tested using predictions of the groundwater level in Butte County, California, USA, taking into account the temporal variability of streamflow, precipitation, and ambient temperature. Our numerical study shows that the optimization of the hyperparameters can lead to reasonably accurate performance of all models (root mean squared errors of groundwater predictions of 2 meters or less), but the "simplest" network, namely a multilayer perceptron (MLP) performs overall better for learning and predicting groundwater data than the more advanced long short-term memory or convolutional neural networks in terms of prediction accuracy and time-to-solution, making the MLP a suitable candidate for groundwater prediction.

Keywords Hyperparameter optimization · Machine learning · Derivative-free optimization · Groundwater prediction · Surrogate models

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10898-020-00912-0) contains supplementary material, which is available to authorized users.

✉ Juliane Müller
JulianeMueller@lbl.gov

[1] Computational Research Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA

[2] Earth and Environmental Sciences Area, Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA

1 Introduction

The massive amount of data obtained from field observations and computer simulations has resulted in heightening an interest in the application of machine learning (ML) models and artificial intelligence to interpret observational data, detect patterns, and derive operational decisions related to management of water resources. Although scientists have been exploiting ML tools for image recognition [53], development of autonomous vehicles [47], inferring physical and chemical phenomena [77], and many more, their application for hydrological predictions is still limited.

Selecting an appropriate ML model is a challenging problem and the model architecture needs to be designed with an optimal set of hyperparameters such as the number of hidden layers and nodes per layer in deep learning models. Once these hyperparameters have been set, a very large-scale global optimization problem has to be solved in which we fit the learning model to the data by optimizing the network's weights. Stochastic optimization methods such as stochastic gradient descent [15,76] or the ADAM optimizer [44] have been successfully used for this task. In practice, the hyperparameters are often tuned "by hand" [69,91], i.e., hyperparameters are chosen, perhaps, at random or based on previous experience, and then the ML model's weights are optimized. One of the drawbacks of this approach is that training the model can be computationally expensive and, depending on the dataset, may require from a few minutes to several hours of compute time, and thus, the final hyperparameters obtained with this approach are usually not optimal.

To date, only a few systematic approaches have been developed to automate HPO in learning models. This includes the widely-used random and grid search approaches [10]. However, the grid search does not scale well with the number of hyperparameters in the model. Ilievski et al. [39] use a deterministic surrogate model algorithm based on radial basis functions to tune the hyperparameters of deep neural networks. The authors show the performance of the method on image data only, and it is unclear if the results can be generalized to time series data. Snoek et al. [80] developed a Gaussian process (GP) based algorithm for HPO, and the authors investigate how the choice of the GP kernel and the GP's own hyperparameters can influence the performance of the ML model. Bergstra [11] used Bayesian methods for HPO. Bayesian optimization has also been used by [45] to train support vector machines and convolutional neural networks (CNNs) for large data sets. The authors train their learning model on a subset of the data in order to decrease the amount of required training time. The developers of Auto-Keras [40] also used Bayesian optimization with a new type of kernel for the GP model, and a new acquisition function that enables the optimization of network morphism. The disadvantages of Bayesian optimization and GPs, in general, are the underlying assumptions that are made regarding the distributions of the errors and the covariance structure, which is usually not available a priori. Moreover, GPs do not scale very well with the number of observations because optimizing the GP's own hyperparameters becomes computationally expensive. Another approach to HPO is the use of genetic algorithms. For example, the Multinode Evolutionary Neural Networks for Deep Learning [90] tool uses a genetic algorithm and support vector machines for tuning the hyperparameters of CNNs. Evolutionary algorithms are known to require hundreds to thousands of function evaluations to find a solution, which is not always feasible to do. The authors of HyperNOMAD [48] extended the NOMAD algorithm [51] to automatically optimize the hyperparameters and the learning process of deep neural networks using the mesh adaptive direct search algorithm. The authors show the performance of HyperNOMAD on image data. Finally, surrogate models have been used for the automated HPO of deep

learning models by [9]. Their strategy is based on exploiting high performance computers and asynchronous computations to evaluate a large number of model architectures.

In this article, we focus on the problem of ML model selection and hyperparameter optimization (HPO) for a pressing application problem of predicting daily groundwater levels in California (CA), United States, for groundwater wells that are affected by streamflow and climatic changes. Due to the non-stationary nature of the groundwater levels, traditional time series *forecasting* like ARIMA and *regression* methods are not suitable. We analyze the performance of a long short-term memory recurrent neural network (LSTM), a multilayer perceptron (MLP), a simple recurrent neural network (RNN), and a convolutional neural net (CNN) because they represent the most widely used deep learning models and their suitability for hydrological timeseries data has not been studied thoroughly. We compare the models based on their prediction accuracy and the time it takes to find the best hyperparameters for each model.

In order to find the best hyperparameters, we pose the HPO problem as a bilevel black-box optimization problem. The upper level problem objective function, the model performance, is not given in closed form as it requires solving a computationally expensive optimization lower level problem, the model fitting problem. The goal is to find the best hyperparameters (upper-level variables) with as few calls to the lower level problem as possible. We solve this problem by using two types of adaptive, derivative-free surrogate model algorithms, namely one based on radial basis functions (RBFs) and one based on GPs. We compare the surrogate model approaches for HPO to the widely used method of randomly sampling the hyperparameter space. We demonstrate the application of the surrogate model approaches for a groundwater prediction application. Here, we consider two cases: one, in which we use a single groundwater observation well, and the second, in which we use multiple groundwater observation wells for model training and predictions.

The remainder of this paper is organized as follows. In Sect. 2, we introduce the mathematical problem description of HPO. We describe the details of the surrogate model based optimization algorithms in Sect. 3. In Sect. 4, we briefly review the learning models we use in our numerical experiments. In Sect. 5, we provide the details of the groundwater prediction problem, and we describe the setup and the results of our numerical experiments. Finally, Sect. 6 concludes the paper and outlines future research directions. Supplemental materials contain additional data and results of numerical experiments.

2 Mathematical problem description of hyperparameter optimization

The problem of finding the best parameters of a learning model can be stated as a bilevel optimization problem:

$$\min_{\boldsymbol{\theta}, \mathbf{w}^*} \quad \ell(\boldsymbol{\theta}, \mathbf{w}^*; \mathcal{D}_{\text{val}}) \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{\theta} \in \Omega \tag{2}$$

$$\mathbf{w}^* \in \arg\min_{\mathbf{w} \in \mathcal{W}} L(\mathbf{w}; \boldsymbol{\theta}, \mathcal{D}_{\text{train}}). \tag{3}$$

Here, we divide the dataset $\mathcal{D}$ into training data $\mathcal{D}_{\text{train}}$ and validation data $\mathcal{D}_{\text{val}}$. At the upper level, we optimize the d hyperparameters (model architecture), $\boldsymbol{\theta} = [\theta^{(1)}, \ldots, \theta^{(d)}]^T$ using the validation data $\mathcal{D}_{\text{val}}$ [Eqs. (1) and (2)]. At the lower level, we fit the model to the training data $\mathcal{D}_{\text{train}}$ by determining the optimal weights $\mathbf{w}^*$ given the current network architecture $\boldsymbol{\theta}$ [Eq. (3)]. Note that there may be more than one vector $\mathbf{w}^*$ that solve problem (3) for a given

network architecture, however the python package we use to solve this lower level problem in our implementation returns only a single solution.

The set Ω is modeled as a discrete finite set

$$\Omega = \prod_{i=1}^{d} \mathcal{I}_i \tag{4}$$

where $\mathcal{I}_i$ denotes a discrete set of values that can be assumed by parameter $\theta^{(i)}$. The domain $\mathcal{W}$ over which the weights are optimized is continuous. We use the notation $L(\mathbf{w}; \boldsymbol{\theta}, \mathcal{D}_{\text{train}})$ in (3) to indicate that we optimize over the parameters $\mathbf{w}$ *given* a set of parameters $\boldsymbol{\theta}$ and the training data set $\mathcal{D}_{\text{train}}$.

Our assumption that all hyperparameters can only assume a limited number of values is motivated by practical considerations. Some parameters, such as the number of network layers can only be integers. Other parameters, such as the dropout rate can range between 0 and 1. However, in practice, only a finite number of options is considered for all hyperparameters due to limited resources and an assumption that very small changes in these hyperparameters are not likely to affect the performance of the learning model significantly.

The loss functions ℓ and L in Eqs. (1) and (3) may be the same, for example, a root mean squared error between the data (actual value) and the fitted model (estimated values), but they may also be different depending on the goal of the analysis. For example, the upper level function could also include a regularization term to encourage model simplicity or low training and prediction times. In order to obtain the objective function value $\ell(\boldsymbol{\theta}, \mathbf{w}^*; \mathcal{D}_{\text{val}})$ for a certain architecture $\boldsymbol{\theta}$ at the upper level, we have to solve the fitting problem (3) at the lower level. The lower level problem is usually a difficult large-scale global optimization problem and stochastic optimizers are commonly used to solve it. Due to the stochasticity of the optimization method, solving problem (3) twice for the same hyperparameters but with different random number seeds will in general lead to two different solutions. Therefore, the upper level problem can be considered as a stochastic optimization problem. In order to obtain a good estimate of ℓ for a given $\boldsymbol{\theta}$, the lower level problem should be solved several times, and thus instead of minimizing ℓ at the upper level, we minimize its expected value $\mathbb{E}_\zeta[\ell(\boldsymbol{\theta}, \mathbf{w}^*(\zeta); \mathcal{D}_{\text{val}})]$ where ζ is a random variable that encodes the randomness due to the lower level optimizer. Depending on the application, it might also be appropriate to minimize the worst case loss. Thus, the optimal solution $\mathbf{w}^*$ as well as the best hyperparameter choice $\boldsymbol{\theta}^*$ depend on ζ.

Solving the lower level optimization problem is in general time-consuming and, depending on the network architecture (defined by $\boldsymbol{\theta}$), may require many minutes to hours (a large number of layers, nodes, and epochs lead to long compute times at the lower level). This compute time requirement restricts how many times the lower level problem can be solved for each hyperparameter set in order to obtain statistically significant point estimates of the expectation value.

Due to the problem structure described above, there is no algebraic description of the upper level ℓ available (black-box, see, e.g., [7] for a discussion of black-box optimization) and neither is gradient information. The loss function is generally multimodal with several local and global optima. Therefore, HPO requires an efficient, gradient-free, optimization method that can search both locally and globally in order to be able to find good solutions, yet be able to escape from local optima.

3 Surrogate models for efficient hyperparameter optimization (HPO)

Surrogate model-based derivative-free optimization algorithms have been developed to efficiently solve computationally expensive black-box problems. A surrogate model m approximates the costly objective function [13]: $\ell(\boldsymbol{\theta}) = m(\boldsymbol{\theta}) + e(\boldsymbol{\theta})$, where $e(\cdot)$ denotes the difference between the true objective and the surrogate approximation. Here we omit the dependence of ℓ on $\mathbf{w}^*(\zeta)$ for ease of notation and because we are only interested in approximating the relationship between $\boldsymbol{\theta}$ and the corresponding value of ℓ, in which the optimization over the $\mathbf{w}$'s is implied.

Most surrogate model optimization algorithms have been developed for problems with continuous parameters, which are constrained only by lower and upper bounds, see, e.g., [32,37,41,75]. However, there are some algorithms for problems with other characteristics such as computationally expensive-to-compute constraints [5,64,70,74], integer constraints on all parameters [66], mixed-integer parameters [2,24,36,60,65], multiple conflicting objectives [23,61], hidden constraints [30,54,63], and multiple levels of objective function fidelity [8,26,62]. For an overview of derivative-free and black-box optimization topics, we refer the reader to [6].

Many difficult optimization problems have been tackled successfully using surrogate model methods, e.g., in the structure optimization of nanomaterials [43], in cloud simulations [50], in climate modeling [67]; in watershed water quality management [64], and in aerodynamic design [86].

In this article, we will use a surrogate model approach for optimizing the integer-constrained hyperparameters of deep learning models. We implement and compare two different approaches: (a) a radial basis function (RBF) as surrogate and a weighted score to iteratively select sample points (similar to SO-I [66]); (b) a Gaussian process (GP) model and an expected improvement [41] acquisition function, which we optimized over an integer lattice, using a genetic algorithm to iteratively generate a new sample point (a new set of hyperparameters $\boldsymbol{\theta}$).

Global surrogate model algorithms generally follow the same structure: (1) Generate an initial experimental design and evaluate the expensive objective function at the selected points; (2) Use all input-output data pairs to compute the parameters of the surrogate model; (3) Optimize a computationally cheap auxiliary function on the surrogate model to determine the next point in the parameter space where the expensive objective will be evaluated. We iterate between steps (2) and (3) until a stopping criterion has been met. Natural stopping criteria include a maximum number of allowed function evaluations, a maximum CPU time, or a maximum number of failed consecutive improvement trials. In step (2), different types of surrogate models can be used, such as GP models [56], RBFs [72,73], or polynomial regression models [68]. Different methods have been developed for iteratively selecting new sample points. For example, [41] maximized an expected improvement criterion, [75] used a stochastic sampling approach, and [32] minimized a "bumpiness" measure. In this article, we compare the performance of the expected improvement criterion and the stochastic sampling when all parameters are assumed to be integer values. The pseudocode of the hyperparameter optimization method is shown in Algorithm 1, and the individual steps are described in more detail in the following subsections.

Algorithm 1 Algorithm for computationally expensive HPO of learning models

1: **Prepare hyperparameters for optimization:** Map the finite set of possible values using sequences of consecutive integers, denote this domain by Ω.
2: **Initial design:** Create an initial experimental design with n_0 points $\Theta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n_0}\}$ by randomly selecting integer-valued vectors from Ω. For each $\boldsymbol{\theta}_j \in \Theta$, compute the upper level objective function value ($\ell(\boldsymbol{\theta}_j)$) by solving the lower level problem (3).
3: Set $n \leftarrow n_0$.
4: **Adaptive sampling:**
5: Use all input-output pairs $\{(\boldsymbol{\theta}_j, \ell(\boldsymbol{\theta}_j))\}_{j=1}^{n}$ to compute the parameters of the surrogate model.
6: Optimize an acquisition function to select the next evaluation point $\boldsymbol{\theta}^{\text{new}}$.
7: Solve the lower level problem (3) to obtain the objective function value at $\boldsymbol{\theta}^{\text{new}}$.
8: Set $n \leftarrow n + 1$ and go to Step 5.
9: **Stop when the termination condition is satisfied.**

3.1 Step 1: preparing the hyperparameters for optimization

When preparing the hyperparameters in Step 1, we have to take into account that they may live on very different ranges and therefore we map them to sequences of consecutive integers. For example, if the first parameter can assume the values $\{0, 0.1, \ldots, 0.5\}$ and the second parameter can assume the values $\{50, 100, 150\}$, then the search domain is $\Omega = \{0, 1, \ldots, 5\} \times \{1, 2, 3\}$. This decreases the difference in parameter ranges.[1] To evaluate the loss function, we map the parameters back to their original values, e.g., by multiplying with 0.1 and 50, respectively, for the two examples above.

3.2 Step 2: initial experimental design

In Step 2, we generate n_0 randomly selected initial points from Ω at which we evaluate the upper level objective function. These points satisfy the integer constraints by definition. For each point, we solve the lower level problem $N = 5$ times and compute the upper level objective function as the average of N evaluations. These N evaluations can be done in parallel to reduce the required evaluation time.

3.3 Steps 4–9: the adaptive sampling loop

During the adaptive sampling loop, we compute the parameters of the surrogate model using *all* previously evaluated points (which is different from trust region type methods that may use a subset of the evaluated points). Generally, any surrogate model can be used in the algorithm. We consider RBFs and GPs in this article. In the following, we describe the details of these surrogate models and the associated adaptive sampling methods.

Radial basis functions and stochastic sampling Our first approach ("RBF approach") to the HPO of learning models is based on RBF models with a cubic kernel. These models have previously been shown to perform well for various optimization problems including problems with integer constraints, see e.g., [43,64,88]. RBFs have the general form

[1] Note that we usually use a similar approach in continuous optimization where we scale the parameters to the unit hypercube, which improves the surrogate models and eliminates difficulties when sampling by perturbation.

$$m_{\mathrm{RBF}}(\boldsymbol{\theta}) = \sum_{j=1}^{n} \lambda_j \varphi(\|\boldsymbol{\theta} - \boldsymbol{\theta}_j\|_2) + p(\boldsymbol{\theta}), \tag{5}$$

where $\boldsymbol{\theta}_j$, $j = 1, \ldots, n$, are already evaluated points, $\varphi(r) = r^3$ is the RBF function with a cubic kernel (other kernels are possible, see e.g., [32]), $\|\cdot\|_2$ denotes the Euclidean norm, and $p(\boldsymbol{\theta}) = \beta_0 + \boldsymbol{\beta}^T\boldsymbol{\theta}$ is a linear polynomial tail. The parameters $\lambda_1, \ldots, \lambda_n, \beta_0$, and $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]^T$ are determined by solving a linear system of equations:

$$\begin{bmatrix} \boldsymbol{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \tilde{\boldsymbol{\beta}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\ell} \\ \mathbf{0} \end{bmatrix}, \tag{6}$$

where the elements of the matrix $\boldsymbol{\Phi}$ are $\Phi_{k,l} = \varphi(\|\boldsymbol{\theta}_k - \boldsymbol{\theta}_l\|_2)$, $k, l = 1, \ldots, n$, $\mathbf{0}$ is a matrix with all entries 0 of appropriate dimension, and

$$\mathbf{P} = \begin{bmatrix} \boldsymbol{\theta}_1^T & 1 \\ \vdots & \vdots \\ \boldsymbol{\theta}_n^T & 1 \end{bmatrix} \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} \quad \tilde{\boldsymbol{\beta}} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \\ \beta_0 \end{bmatrix}, \quad \boldsymbol{\ell} = \begin{bmatrix} \mathbb{E}_\zeta[\ell(\boldsymbol{\theta}_1, \mathbf{w}_1^*(\zeta); \mathcal{D}_{\mathrm{val}})] \\ \mathbb{E}_\zeta[\ell(\boldsymbol{\theta}_2, \mathbf{w}_2^*(\zeta); \mathcal{D}_{\mathrm{val}})] \\ \vdots \\ \mathbb{E}_\zeta[\ell(\boldsymbol{\theta}_n, \mathbf{w}_n^*(\zeta); \mathcal{D}_{\mathrm{val}})] \end{bmatrix}. \tag{7}$$

The matrix in (6) is invertible if and only if $\mathrm{rank}(\mathbf{P}) = d + 1$ [72], where d denotes the problem dimension. Here, $\mathbf{w}_j^*$ is an optimal weight vector (lower level solution obtained by solving problem (3)) corresponding to the jth evaluated parameter vector $\boldsymbol{\theta}_j$.

In order to identify the next sample point, we find the point $\boldsymbol{\theta}^{\mathrm{best}} \in \arg\min\{\mathbb{E}_\zeta[\ell(\boldsymbol{\theta}_j, \mathbf{w}_j^*(\zeta); \mathcal{D}_{\mathrm{val}})], j = 1, \ldots, n\}$ that has the lowest objective function value (if there are multiple, we select one at random). Then, we generate a large number $M = 500$ of candidate points by perturbing each value of the best point found so far by either adding or subtracting a value randomly chosen from $\{1, 2\}$. If the perturbed point falls outside of Ω, we reflect it over the corresponding boundary to the inside of Ω. The points created by perturbation enable a local search around $\boldsymbol{\theta}^{\mathrm{best}}$. In addition, we generate a second set of candidate points by randomly selecting another M points from Ω. These points represent our global search. Thus, we have $2M$ candidate points in total.

In order to select the next evaluation point $\boldsymbol{\theta}^{\mathrm{new}}$ from the candidate points, we follow the ideas of [75] in which two scores are computed for each point, namely the distance to the already evaluated points (distance score) and the objective function value as predicted by the RBF surrogate (RBF score). We compute a weighted sum of both scores and the candidate point that achieves the best weighted sum will become the next evaluation point (for details, see the online supplement Section A). By placing a large weight on the distance score, we preferentially sample at points that are far away from already sampled points (global search). If the weight for the RBF score is large, we preferentially sample points with low predicted objective function values. This represents a local search, because points with low predicted objective function values are usually in the vicinity of the best point found so far. Note that candidates that have already been evaluated previously are discarded. Thus, if we did not have a stopping criterion and sampling continued, the globally optimal solution would eventually be found, following a simple counting argument.

Gaussian process models and expected improvement Our second approach (GP approach) to the HPO of learning models is based on GP models and expected improvement sampling [41] over an integer lattice. According to a literature review, GP (or kriging) models have been widely used as surrogate models to approximate expensive-to-compute objective functions. By definition, GP models not only provide a prediction of the function values at unsampled

points, but also an estimate of the associated uncertainty. In kriging, we treat the function we want to approximate like the realization of a stochastic process, and write the surrogate as

$$m_{\mathrm{GP}}(\boldsymbol{\theta}) = \mu + Z(\boldsymbol{\theta}). \tag{8}$$

Here, μ represents the mean of the stochastic process and $Z(\boldsymbol{\theta})$ is distributed as $\mathcal{N}(0, \sigma^2)$. Thus, the term $Z(\boldsymbol{\theta})$ represents a deviation from the underlying global model (the mean). We assume a correlation between the errors, which is based on the distance to the other points. At an unsampled point $\boldsymbol{\theta}^{\mathrm{new}}$, the kriging prediction is treated like a realization of a random variable $M_{\mathrm{GP}}(\boldsymbol{\theta}^{\mathrm{new}})$ that is normally distributed with mean μ and variance σ^2. The correlation between two random variables $Z(\boldsymbol{\theta}_k)$ and $Z(\boldsymbol{\theta}_l)$ is defined as

$$Corr\left(Z(\boldsymbol{\theta}_k), Z(\boldsymbol{\theta}_l)\right) = \exp\left(-\sum_{i=1}^{d} \boldsymbol{\gamma}_i |\theta_k^{(i)} - \theta_l^{(i)}|^{q_i}\right), \tag{9}$$

where γ_i determines how fast the correlation decreases in the ith dimension, and q_i reflects the smoothness of the function in the ith dimension (we use $q_i = 2$ for all i). We denote by $\mathbf{R}$ the $n \times n$ matrix whose (k, l)th element is given by (9). The kriging parameters μ, σ^2, and γ_i are estimated by maximum likelihood. Then, the prediction at a new point $\boldsymbol{\theta}^{\mathrm{new}}$ is defined as

$$m_{\mathrm{GP}}(\boldsymbol{\theta}^{\mathrm{new}}) = \hat{\mu} + \mathbf{r}^T \mathbf{R}^{-1}(\boldsymbol{\ell} - \mathbf{1}\hat{\mu}), \tag{10}$$

where $\mathbf{1}$ is a vector of ones of appropriate dimension and $\boldsymbol{\ell}$ as in (7) and

$$\mathbf{r} = \begin{bmatrix} Corr\left(Z(\boldsymbol{\theta}^{\mathrm{new}}), Z(\boldsymbol{\theta}_1)\right) \\ \vdots \\ Corr\left(Z(\boldsymbol{\theta}^{\mathrm{new}}), Z(\boldsymbol{\theta}_n)\right) \end{bmatrix}. \tag{11}$$

The corresponding mean squared error is

$$s^2(\boldsymbol{\theta}^{\mathrm{new}}) = \hat{\sigma}^2 \left(1 - \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r} + \frac{(1 - \mathbf{1}^T \mathbf{R}^{-1} \mathbf{r})^2}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}}\right) \tag{12}$$

with

$$\hat{\mu} = \frac{\mathbf{1}^T \mathbf{R}^{-1} \boldsymbol{\ell}}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}} \tag{13}$$

and

$$\hat{\sigma}^2 = \frac{(\boldsymbol{\ell} - \mathbf{1}\hat{\mu})^T \mathbf{R}^{-1} (\boldsymbol{\ell} - \mathbf{1}\hat{\mu})}{n}. \tag{14}$$

From the properties of the GP model, we can now define an expected improvement function which we use to iteratively select a new sample point $\boldsymbol{\theta}^{\mathrm{new}}$. We denote by $\mathcal{L}$ a random variable that represents our uncertainty about the function value. Its mean and variance at a point $\boldsymbol{\theta}$ are defined by the kriging predictor, namely $m_{\mathrm{GP}}(\boldsymbol{\theta})$ and $s^2(\boldsymbol{\theta})$. We denote by $\ell^{\mathrm{best}} = \ell(\boldsymbol{\theta}^{\mathrm{best}})$ the best function value we have found so far during the upper level optimization. If the value $\mathcal{L}$ is less than ℓ^{best}, we have an improvement $I = \ell^{\mathrm{best}} - \mathcal{L}$. The expected value of this improvement is computed as

$$\mathbb{E}(I) = s(\boldsymbol{\theta})\left(v\Phi(v) + \phi(v)\right), \tag{15}$$

where

$$v = \frac{\ell^{\text{best}} - m_{\text{GP}}(\boldsymbol{\theta})}{s(\boldsymbol{\theta})} \tag{16}$$

and Φ and ϕ are the normal cumulative distribution and density functions, respectively, and $s(\boldsymbol{\theta}) = \sqrt{s^2(\boldsymbol{\theta})}$ is the squared root of the mean squared error.

In order to select the next evaluation point $\boldsymbol{\theta}^{\text{new}}$, we have to maximize the expected improvement. Since the maximum argument will not necessarily fall onto a point that satisfies the integer constraints, we have to maximize the function over an integer lattice. Therefore, we use a genetic algorithm (see [58] for introducing genetic algorithms), which returns an integer-feasible point $\boldsymbol{\theta}^{\text{new}} \in \Omega$ at which we do the next expensive function evaluation.

3.4 Algorithm parameters

Our algorithm has several parameters that can be adjusted and may impact the results of our simulations. In our numerical experiments, we use $n_0 = d + 1$ points as the number of initial experimental design points. Generally, we would like to use a large number, but the more initial points we use, the fewer points can be acquired iteratively (as the bottleneck is usually the number of allowed evaluations). For each hyperparameter vector, we solve the lower level problem $N = 5$ times to obtain an estimate of the expected value of the outer objective function. The value of N should be adjusted based on the available compute resources.

In each iteration, we generate $2M$ candidate points, with $M = 500$. If the number of hyperparameters and the number of values each hyperparameter can assume are small, we can alternatively generate candidate points by completely enumerating all possible solutions for which we would then compute the weighted scores or the expected improvement values, and choose the best candidate, but in our HPO setting, this is not applicable due to the huge number of possible solutions (see also Table 1 in Sect. 5.3).

The weights for the selection criteria in the RBF approach cycle through the pattern $\{0, 0.25, 0.5, 0.75, 1\}$. In the first iteration, the weight is 0; in the second iteration, the weight is 0.25, and so on, and after the weight 1 was used, it will be set back to 0 in the next iteration. This cycling enables the sampling to transition from searching locally to globally.

Another important parameter is the set of randomly chosen perturbations $\{1, 2\}$ that are added or subtracted to the current best solution. The goal of the perturbation is to enable a local search and given that our parameter ranges are relatively small, using the values $\{1, 2\}$ is appropriate and a maximum perturbation of 2 is reasonable to prevent creating too many candidate points that are outside the upper and lower bounds. If larger parameter ranges were present, larger perturbation values could be chosen.

In general, the amount of available compute time and the number of hyperparameters to be optimized dictate the algorithm parameter values for N, M, and the maximum number of allowed upper level evaluations. We use a maximum number of 50 upper level function evaluations as stopping criterion.

4 Review of learning models

In this section, we provide a brief review of the different deep learning models that we are using in our numerical study and their associated hyperparameters. These hyperparam-

eters determine the model architecture, and thus how well the model performs in terms of minimizing the loss.

The number of layers, the number of nodes in each layer, the type of activation functions, and the type of optimizer are examples of hyperparameters. Each node in a network layer may have a different activation function. The Rectified Linear Unit (ReLU) activation function, $f(\cdot) = \max(0, \cdot)$, is commonly used. The learning model corresponding to a given architecture ($\boldsymbol{\theta}$) is trained to find the weight vector ($\mathbf{w}^*$) that minimizes the loss L. The optimizers used for this task share the idea of backpropagation [78] which is an iterative method based on gradient descent and the chain rule. Therefore, the number of iterations (=the number of epochs) and the batch size (=the number of samples in the gradient update) are additional hyperparameters that must be determined. Among many existing optimizers, ADAM [44], Adamax, and RMSprop [34] are widely used. Each of these optimizers has their own hyperparameters such as the learning rate and the decay rate. Finally, a dropout rate ($\alpha \in (0, 1)$) can be set to help prevent overfitting at each layer except for the output layer. The dropout rate means that randomly selected $\alpha \times$(number of nodes in the layer) nodes are ignored during training.

4.1 Multilayer perceptron model (MLP)

The MLP [12] is a feed-forward neural network that consists of an input layer, at least one hidden layer, and an output layer. It can model nonlinear patterns by introducing multiple layers and nonlinear activation functions that transform input signals. It has been primarily used for classification and function estimation problems [87]. MLPs have been used for, e.g., forecasting drought [3], estimating evapotranspiration [84], and predicting water flow [4]. Unlike other statistical models, the MLP does not make any assumptions regarding the prior probability density distribution of the training data [28].

4.2 Convolutional neural networks (CNN)

The CNN [52] is also a feed-forward neural network that applies a sliding window filter on the input dataset, and has three main characteristics, namely local connectivity, shared weight, and spatial or temporal sub-sampling. The local connectivity means that the weighted linear combination is restricted to within the filter (*not fully connected*). All local fields share the same weights. The weighted values are transformed with the activation function $f(\cdot)$, and the function returns a feature map. The sub-sampling is an optional step to extract other features in the feature map by reducing its size. An example method of sub-sampling is to pool the data to replace the input by a summary statistic such as the maximum (max-pooling) or mean (avg-pooling) value. The filters can also be used to extract features. The filter can be applied to both the input and the feature map to make deeper neural networks. Finally, the extracted features are fully connected to the output layer. Note that additional hidden layers can be constructed between the final feature layer and the output layer. Additional CNN specific hyperparameters that must be optimized include the number of filters, the filter size, whether or not to apply sub-sampling, and the method and the size of the sub-sampling.

In practice, CNN's have been used for a variety of application areas ranging from image recognition (e.g., ResNet [33]) to time series classification [21], and time series forecasting [14].

4.3 Simple recurrent neural networks (RNN)

Recurrent Neural Networks (RNNs) can represent dynamic temporal behavior by using a directed graph structure between nodes to store sequence representations. The decision an RNN sees at time $t-1$ affects the decision at time t. Unlike feed-forward neural networks, which take the entire sequence input as is, the RNN first takes in the output generated from the previous sequence steps (memory) before taking the input step. In the simplest RNN, the memory is carried forward through the hidden state, which is a function of the product of the input at time t and a weight matrix that is added to a product of the hidden state at time $t-1$ and another hidden weight matrix. The output is a function of the product of the last hidden state and a weight matrix. The activation function in the RNN is a hyperbolic tangent function that allows an increase or decrease in the value of the state. The weight matrices are adjusted by error backpropagation. A drawback of RNNs is that often, despite choosing the optimal set of hyperparameters, errors flowing "backward in time" in a fully connected RNN may either vanish or blow up, which leads to oscillating weights or long training times. RNNs have been used in applications like speech recognition [57], text generation [83], and time series forecasting in hydrology [17,20], where system memory and context are critical to prediction.

4.4 Long short-term memory recurrent neural networks (LSTM)

Long short-term memory networks [35] are an improvement of RNNs and are able to learn long-term dependencies. LSTMs have a structure that allows them to represent temporal behavior and remember previous behavior through a special arrangement of modules. Besides a hidden state, LSTMs also have a cell state that runs through the entire chain and is responsible for storing the long term dependencies. The information flow between the cell state, the hidden state, and the input state is controlled through different types of gates. A simple LSTM unit contains an input gate, an output gate, and a forget gate. The forget gate [29] enables the network to overcome the potentially unbounded growth of cell states when a continuous time series is used. The forget gate is a sigmoid activation function that decides how much information from the previous cell state should be passed to the next. The output from the forget gate is a function of the input, the previous hidden states with their corresponding weight matrices, and a bias vector. It enables resetting the memory blocks once their contents are out of date.

The input gate decides how to update the cell state. The hyperbolic tangent activation function generates new candidate solutions using the weighted input and hidden state and a bias vector. The new cell state is generated from the weighted sum of the previous cell state and the input state. The output gate generates the new hidden state from the new cell state and the input carrying the most recent information.

LSTMs have been used in many different applications, e.g, for language modeling [82], speech recognition [31], traffic speed prediction [55], and time series modeling [38].

5 Numerical experiments for timeseries data

In the following, we describe our groundwater application problem, the data used in the numerical experiments, and the setup and results of the experiments. Additional results and data are provided in the online supplement.

We designed numerical experiments to answer two main questions: (1) Which learning model is best suited for making accurate predictions of groundwater levels? (2) Which HPO method leads to the best results fastest? In order to answer these questions, we use our HPO methods to train and validate the learning models on historical observations and we compare the performance of the four learning models to each other. We use the optimized model architectures to predict the future groundwater levels and we compare these predictions to "future" data, i.e., groundwater data that was not used for optimizing the hyperparameters or training the models.

5.1 Motivation and description of the groundwater prediction application

In California (CA), United States, groundwater has always been an important resource for agriculture, drinking, and other beneficial purposes. An unprecedented multi-year drought endured in the recent past (2012–2016) led to regulations on long-term water conservation and management of groundwater (see the 2014 Sustainable Groundwater Management Act (SGMA) [16] in CA).

However, this situation is not unique to CA, and drought severity is expected to increase over many regions with changes in climate [19] increasing the agricultural dependencies on groundwater [85]. There is therefore an increased urgency for time series forecasting of groundwater depths.

Groundwater depths are typically estimated using mechanistic multi-scale, multi-physics simulation models (see, e.g., [49,81,89]). However, these models require extensive characterization of hydrostratigraphic properties and accurate quantification of boundary conditions, including recharge sources, climate variability as well as changes in pumping [79], which are not always known apriori. Moreover, these models' computational complexity does not make them desirable when groundwater predictions for multiple future climate scenarios are needed to make actionable decisions for sustainable groundwater management. Thus, it is appealing to use purely data-driven machine learning methods that are computationally inexpensive and that can be continually updated as new observations are obtained. Moreover, in some cases data-informed learning models have shown better performance than process-informed simulations, possibly due to deficiencies in model structure from incomplete process representations [42].

In the groundwater modeling context, artificial neural networks (ANNs) have previously been used for both modeling and prediction purposes (see e.g., [22]). More recently, RNNs have been used to predict groundwater levels (e.g., [20,91]). The application of LSTMs is still limited in the field of hydrologic sciences. For example, LSTMs have recently been used to predict changes of river discharge from precipitation across the Continental United States using publicly available datasets [46]. [91] use an LSTM to predict groundwater levels in the Hetao Irrigation District in China.

In this article, we focus in particular on the Butte County watershed in CA. This area belongs to CA's Central Valley, which is one of the largest agricultural lands in the world and produces over 250 crops a year [25]. Butte County is considered a high-priority watershed by SGMA and it is therefore vital to develop tools for groundwater predictions that will enable actionable and timely decision support for the planning and management of groundwater resources in this region in the future.

5.2 Observation data in Butte County, CA, USA

At the Butte County watershed, we have high-frequency (daily) observations of groundwater levels available for the duration of approximately eight years (2010–2018). We train our learning models on historic daily observations of temperature, precipitation, streamflow, groundwater levels, as well as features such as the week and month of the year that the measurement was taken.

Since there were gaps in the observations for temperature, precipitation, and groundwater levels (e.g., which may happen due to faulty sensors or bad data quality), we imputed the missing values using the R package imputeTS "Time Series Missing Value Imputation" [59]. This package provides state-of-the-art imputation algorithms along with plotting functions for univariate time series missing data statistics. We use the na.seadec function with the Kalman Smoothing and State Space Models algorithm. This algorithm removes the seasonal component from the time series, performs imputation on the deseasonalized time series, and then adds the seasonal component again. To create an uninterrupted time series (due to some faulty data entries at the end of the hydrological year, i.e., end of September), we used the na.locf algorithm ("Imputation by Last Observation Carried Forward") of the imputeTS package.

In the following study, we consider two scenarios, namely a"single well case" and "multi-well case". In the single well case, we train the model on the groundwater levels measured only at a single well and we predict only for this one well. In the multi-well case, we use the groundwater level data from three different wells to train the learning models, and we make predictions for all three wells. Note that we do *not* train a separate model for each well, but rather we have one model for all three wells. A summary of the data sources we use in our study as well as the time series for all variables are shown in the online supplement in Sect. B. The time series for the groundwater levels at all wells reflect the significant decline in the amount of available groundwater between 2010 and 2016.

5.3 Setup of the numerical experiments

We conducted our numerical experiments with python ($version$ 3.7) on Ubuntu 16.04 with Intel® Xeon(R) CPU E3-1245 v6 @ 3.70 GHz × 8, and 31.2 GiB memory. We use the Keras package [18] with the TensorFlow [1] backend for our deep learning architectures. For the genetic algorithm in the GP approach we use the DEAP python package [27]. In the genetic algorithm we use *100 generations* with *100 individuals* each and we use a *crossover probability of 0.75*. For the GP itself, we use the implementation in Sklearn [71].

Preparing the data We divide our observation data into two sets—one for training the models ($\mathcal{D}_{\text{train}}(\boldsymbol{\theta})$) and one for validating the models ($\mathcal{D}_{\text{val}}(\boldsymbol{\theta})$), i.e., comparing the model's groundwater predictions to the true observations. In our setting, there is a dependency of the number of daily observations used for training and validating the models on a hyperparameter that we are optimizing. This dependency is described in more detail in the following. The model output is the groundwater level for the next day and the inputs are the values of all the variables at the current day and all values at some previous days. More specifically, we define the number of previous days whose data are used as input as a *lag number*, G. The lag number G means that the variable values of the G previous days are used. This lag thus determines the size of $\mathcal{D}_{\text{train}}(\boldsymbol{\theta})$ and $\mathcal{D}_{\text{val}}(\boldsymbol{\theta})$. To illustrate it with an example, suppose that there are a total 5 days of observations and for each day we have measurements of 6 variables. We denote $\delta(t)$, $t = 1, \ldots, 5$, as the day of the measurement. If the lag $G = 1$, each

sample consists of two days of observations (the current and the previous day) and therefore has 2×6 =12 variable values. The total number of samples we obtain is thus four: $(\{\delta(1), \delta(2)\}, \{\delta(2), \delta(3)\}, \{\delta(3), \delta(4)\}, \{\delta(4), \delta(5)\})$. If the lag $G = 2$, we use the measurements for each variable at the two previous days in addition to the data for the current day, and thus each sample consists of three days and has 3×6=18 variable values. The total number of samples we have is then three: $(\{\delta(1), \delta(2), \delta(3)\}, \{\delta(2), \delta(3), \delta(4)\}, \{\delta(3), \delta(4), \delta(5)\})$. Thus, as is, the lag would determine the total number of samples for our model input. However, we divide the dataset such that the number of training samples is the same regardless of the lag. In other words, the number of samples in $\mathcal{D}_{\text{train}}(\boldsymbol{\theta})$ is always the same and the dimension of each sample (= the number of days $\times$ the number of variables in each day) varies depending on the lag. The samples that were not used for defining $\mathcal{D}_{\text{train}}(\boldsymbol{\theta})$ constitute the validation set $\mathcal{D}_{\text{val}}(\boldsymbol{\theta})$. Therefore, the lag determines how much historical information will be used in each sample. By setting the lag as a hyperparameter, the optimization aims to not only find the best network architecture, but also the optimal amount of historical information in each sample that leads to minimizing the outer objective function.

In order to make future groundwater predictions, we use the given values (here from the dataset $\mathcal{D}_{\text{val}}(\boldsymbol{\theta})$) of temperature, precipitation, and streamflow discharge and predict the corresponding groundwater levels with the trained models. Because the input groundwater data in $\mathcal{D}_{\text{val}}(\boldsymbol{\theta})$ are assumed to be unknown, we use the predicted groundwater level value from the model. Therefore, the input groundwater data are dynamically obtained as we make predictions for each additional day.

Before using the variable values for training and validating our learning models, we normalize all variables' values to [0,1]. The reason is that different observed variables affect the groundwater level in specific ways and their values are generally on different ranges. Scaling them to [0,1] enables us to better capture the variables' interactions within the learning models. For the groundwater data, this is, however, not straight-froward. The recent drought in CA has caused the groundwater levels to drop to unprecedented low levels. Thus, for scaling purposes, it is difficult to assign a maximum and a minimum groundwater level as future groundwater levels may go below or above recorded values. We therefore assigned a minimum groundwater level of 0, which here refers to the mean sea level and it is a fair assumption that the water level can never drop below 0 even though this value has never been observed (unless it is located near the sea coast). For the upper bound, we use the maximum observed groundwater level. The reason is that over the past, we have observed a steadily decreasing trend in the amount of groundwater. Although wet years may cause the groundwater levels to go above the maximum recorded values, it is a good choice as it reflects the reality observed so far. Another possible upper limit could be the ground surface elevation (GSE), as groundwater cannot go above the ground. However, the GSE is also a dynamic parameter, as several places in CA suffer from land subsidence due to groundwater withdrawal. For the other variables (temperature, precipitation, streamflow discharge) the scaling is trivial as we have full information of the minimum and maximum values of the whole time series (past observations as well as future predictions). For the streamflow discharge, we applied a log-transformation before scaling the data to [0,1] because of the huge variations that occur (compare the time series in the online supplement). Note that the scaling does not mean that the predicted groundwater levels will be below the chosen maximum, and the future groundwater level predictions can go beyond their historical maximum levels depending on the input values of the other variables.

Replications of inner and outer optimizations and hyperparameters The training of the models in the inner optimization and the search for the optimal hyperparameter in the outer optimization both have stochastic elements. For performing the inner large-scale opti-

mization for a given hyperparameter set, we use stochastic optimizers. Therefore, given a specific hyperparameter set $\hat{\boldsymbol{\theta}}$, we solve the inner problem five times and we define the inner optimization's loss function as the average over these five trials (as an approximation of $\mathbb{E}_{\zeta}[\ell(\hat{\boldsymbol{\theta}}, \mathbf{w}^*(\zeta); \mathcal{D}_{\text{val}}(\hat{\boldsymbol{\theta}}))]$). The outer optimization also has stochastic elements (in the initial experimental design and in the adaptive sampling methods). Therefore, in order to obtain an average outer optimization performance, we repeat the HPO for each model with each algorithm five times. We allow each HPO method to try 50 different hyperparameter sets.

The hyperparameters we optimize as well as the possible (unmapped) values for each hyperparameter are summarized in Table 1. We keep the learning and the decay rates at their default values of 0.001 and 0.0, respectively. From a practical perspective, allowing a lag of 365 days is reasonable as this would allow the model to use the information from a whole year worth of data, and thus recognize seasonality and correlations between the time of the year and the groundwater levels. However, for RNN and also for LSTM, a maximum lag of 365 days was too large, causing problems with backpropagation in training and thus problems in the optimization. We were able to run the LSTM and RNN five times successfully by restricting the maximum lag to 60 days for the single well case.

Although the structure of the networks and their detailed mechanics are different (see Sect. 4), the number of input/output nodes are formed in the same way. The number of input nodes are the same as the number of variables in a sample. For the earlier example with five days and lag $G = 1$, all models would have 12 input nodes. The number of output nodes are the same as the number of wells whose groundwater data we are learning, i.e., one for the single well case and three for the multi-well case.

For LSTM, we use the ReLu activation function in all the deeper layers except the outer layer. For training the LSTM for a given hyperparameter set, we use the RMSprop optimizer. In the MLP, we also use the ReLu activation function for all nodes except those in the output layer. The nodes of the output layer have the linear activation function because we are dealing with real-valued outputs. We use ADAM for the lower level optimization. During the construction of the CNN, we add a fully connected hidden layer with 50 nodes between the last convolution layer and the output layer. For simplicity, we do not apply the sub-sampling option here. The activation function and the lower level optimizer for the CNN and the RNN are the same as for the MLP. We leave the dropout rate of the recurrent step fixed at 0. Even with our simplifying assumptions that all layers in the model have the same hyperparameter values for the dropout rate, the number of nodes, the number of filters, the filter size, and the large step sizes (e.g., steps of 50 for the number of epochs), we can see from Table 1 (bottom) that the number of possible hyperparameter combinations ranges from hundred thousands to millions. Trying all of these combinations is computationally infeasible, especially in our setting where optimal hyperparameters have to be determined fast to enable timely groundwater management decisions.

5.4 Numerical results

In the following subsections, we compare the results of our numerical experiments obtained with LSTM, MLP, CNN, and RNN for the single well case. For the multi-well case, we compare only LSTM, MLP, and CNN. We did not include the RNN model for the multi-well case as it suffers from the problems with backpropagation mentioned earlier. It therefore often fails to fit a model to a given architecture. Also, as we will see for the single well results, the RNN performed worst among all learning models. Therefore, we do not include the RNN in the multi-well analysis.

Table 1 Hyperparameters in the optimization and their possible values

	LSTM	MLP	RNN	CNN
#Epochs	{50, 100, . . . , 450, 500}	{50, 100, . . . , 450, 500}	{50, 100, . . . , 450, 500}	{50, 100, . . . , 450, 500}
Dropout rate	{0.0, 0.1, . . . , 0.5}	{0.0, 0.1, . . . , 0.5}	{0.0, 0.1, . . . , 0.5}	{0.0, 0.1, . . . , 0.5}
Batch size	{50, 100, . . . , 450, 500}	{50, 55, . . . , 195, 200}	{50, 55, . . . , 195, 200}	{50, 55, . . . , 195, 200}
#Layers	{1, 2, 3, 4}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5, 6}
Lag	{3, 6, 9, . . . , 60}	{30, 35, . . . , 360, 365}	{3, 6, 9, . . . , 60}	{30, 35, . . . , 360, 365}
#Nodes per hidden layer	{3, 6, . . . , 27, 30}	{5, 10, . . . , 45, 50}	{5, 10, . . . , 45, 50}	NA
#Filters	NA	NA	NA	{5, 10, . . . , 45, 50}
Filter size	NA	NA	NA	{2, 3, 4, 5}
Possible combinations	480,000	7,588,800	2,232,000	30,355,200

NA means that a model does not have that particular hyperparameter
LSTM long short-term memory network, *MLP* multilayer perceptron, *RNN* recurrent neural network, *CNN* convolutional neural network
#Indicates "number of"

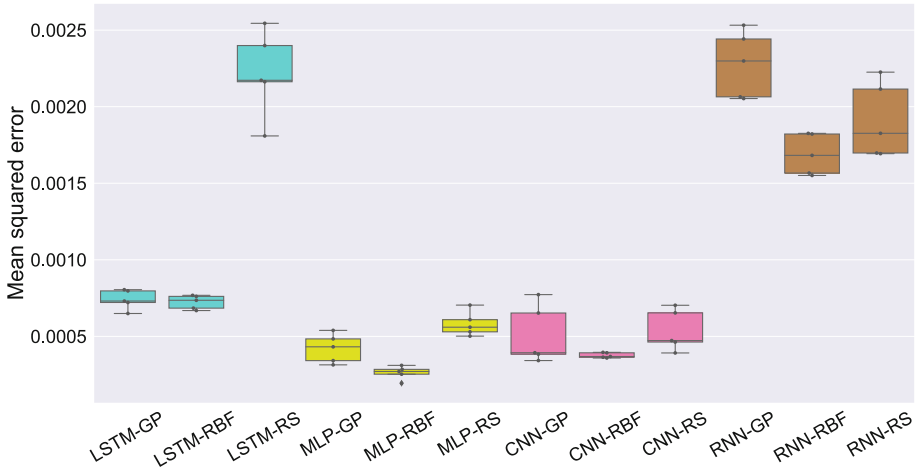


Fig. 1 Boxplots of mean squared errors for the single well case over five HPO trials. Low values are better. *LSTM* long short-term memory network, *MLP* multilayer perceptron, *CNN* convolutional neural network, *RNN* recurrent neural network, *GP* Gaussian process, *RBF* radial basis function, *RS* random sampling

We compare the performance of our two optimization approaches (the RBF approach and the GP approach) with a random sampling (RS) approach for HPO. We illustrate our results in the form of progress plots, time series plots, wall clock time plots, and plots of the distributions of the optimal hyperparameters found in each of the five trials. In addition, we illustrate the models' predictive performance for data in the future that was not used during optimizing the hyperparameters or during training the models. This "out-of-sample prediction" allows us to gain insights into the models' extrapolation performance. We report additional results including tables with the optimal hyperparameter values for each model and each optimization strategy as well as the corresponding losses (and the resulting groundwater prediction errors) in the online supplement in Sect. C.

5.5 Results for predicting groundwater at a single well in Butte County, CA

Comparison of mean squared errors In Fig. 1 we show the distribution of the optimal outer objective function values over five trials for all models and all optimization methods after the optimization has finished. The final mean squared errors (MSEs) are fairly low for all learning models and all optimization approaches. In the online supplement in Table 2, we show the numerical MSE values for all trials as well as the corresponding deviation of the predicted groundwater level from the truth. Almost all trials of all models reach a deviation of about 2 meters or less, which is considered highly accurate. Figure 1 shows that there is a noticeable difference between the performance of the types of learning models, with MLP and CNN generally leading to the smallest (best) final MSEs. We also observe a difference in the variation of the results over the five trials with RBF leading to the smallest variation for all HPO methods. The RNN performs worst regardless of the HPO method used. This may be related to the fact that all HPO methods have difficulties in finding optimal solutions or that the RNN is simply not able to capture the groundwater data any better. Figure 1 also indicates that the RBF approach generally leads to the lowest errors among all HPO methods.

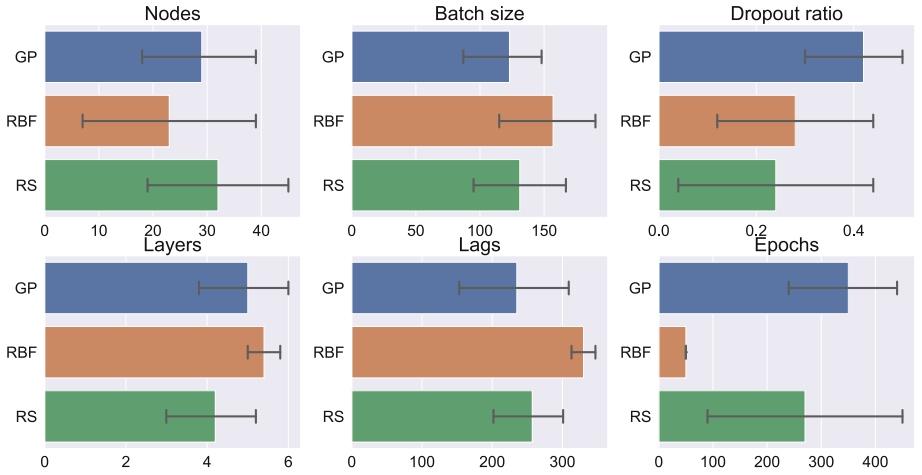


Fig. 2 Distribution of the optimized hyperparameter values for MLP for the single well case. The colored bars represent the mean and the black lines show the standard deviation over five solutions. *LSTM* long short-term memory network, *MLP* multilayer perceptron, *CNN* convolutional neural network, *RNN* recurrent neural network, *GP* Gaussian process, *RBF* radial basis function, *RS* random sampling

Analysis of optimized hyperparameters We compare the distributions of the final hyperparameter values over all five trials obtained with the different HPO methods. Figure 1 showed that the MLP optimized with the RBF approach leads overall to the best results. Figure 2 shows the corresponding hyperparameters. Here, the colored bars represent the mean and the black lines show the standard deviation over the five solutions for the MLP. We see that generally all HPO methods lead to a more complex MLP network, the RBF method leading to approximately 5 layers with approximately 25 nodes each. For the RBF method, the lag is large, using almost a full year of past data. The dropout rate and the optimal number of epochs for the RBF method are smaller than those obtained with the GP. Note that larger dropout rates discourage overfitting of the models to the data. We observe the largest differences between the HPO methods for the number of epochs.

The distributions of the optimal hyperparameters for LSTM, CNN, and RNN are shown in the online supplement in Sect. C.1.1 in Fig. 3. For the CNN, we observe a similar behavior as for the MLP: the optimal RBF solutions have a large number of layers, filters, and lags. For LSTM, all HPO methods prefer smaller, less complex networks, and for the RNN, the HPO methods return networks of different complexities with generally larger standard deviations on the optimal hyperparameter values.

Analysis of HPO convergence Besides the final solutions obtained with the different HPO methods, we are also interested in how fast the methods converge, i.e., how many calls to the inner optimization loop are necessary. For this purpose, we use progress plots that illustrate how quickly the average and the standard deviation of the outer objective function values over five trials decrease as the number of function evaluations increases. Ideally, the graphs should drop off fast, which means that improved network architectures were found within very few hyperparameter evaluations. The plot of the standard deviation reflects how robust each method is. Small standard deviation envelopes indicate that all five trials with the HPO methods converged to an approximately equally good solution. We show these convergence plots in Fig. 3.

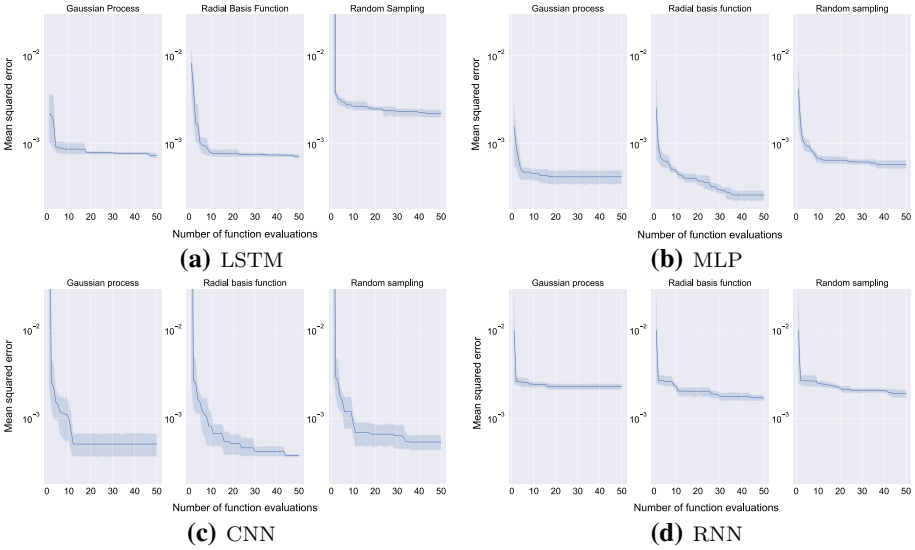


Fig. 3 Convergence plots showing the average and the standard deviation of the outer objective function values over five trials versus the number of function evaluations for all four learning models for the single well case

For the LSTM (Fig. 3a) we can see that the GP and RBF approaches lead to more robust solutions (narrow standard deviation bands) than the RS approach. The standard deviation associated with the random sampling remains the largest throughout. This can be expected as the random sampling does not have a systematic search approach that allows it to intensify the search locally around good solutions.

For MLP, on the other hand, the RBF performed best and the convergence plot (Fig. 3b) reflects that the RBF approach also converged faster than the GP and the RS. For MLP, GP has wider standard deviation bands than RBF throughout indicating less robustness. The standard deviations obtained with RS are smaller for MLP than for LSTM. Since the RS is lacking any systematic search for improvements, this may be an indicator that there are several solutions for MLP with similar performance.

For CNN (Fig. 3c), we can see significant differences in the convergence behavior between the three HPO methods. The RBF approach has smaller standard deviations than GP and RS. We notice that the search with the GP gets stuck after about 12 function evaluations and it gets stuck in different solutions as is reflected by the width of the standard deviation envelopes.

Finally, for RNN (Fig. 3d), we observe the overall worst performance. All HPO models make only slow progress towards improved solutions and the five trials with the RBF method show the average best performance.

Analysis of computing times As training larger, more complex networks requires more compute time, we also plot the cumulative function evaluation time for each network type and each HPO method in order to compare the wall clock time to solution. Here, we only compare the amount of time spent on function evaluations and we assume that the HPO methods' own computational overheads are negligible. We show the average and the standard deviations of the computing times over all five trials in Fig. 4. Comparing the figures, we see that optimizing the hyperparameters of the MLP (Fig. 4b) is fastest for all HPO methods (within 5 hours) even though the optimal MLP architectures were fairly complex.

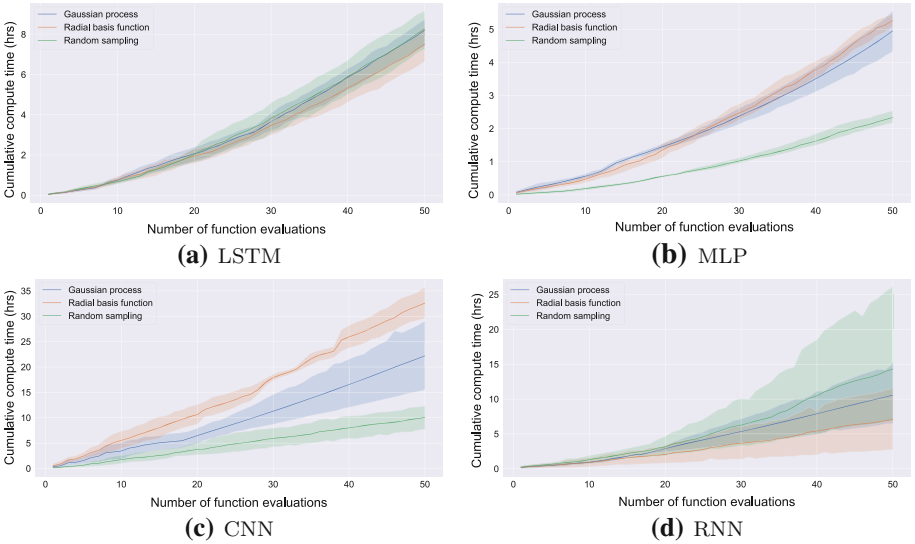
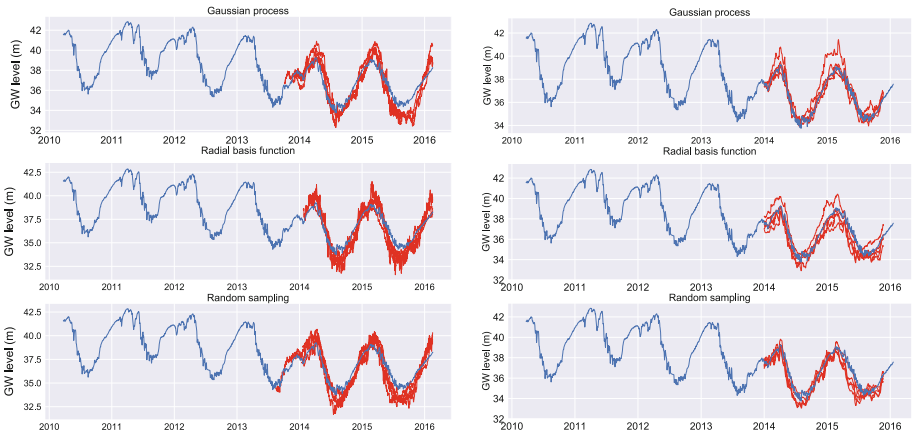


Fig. 4 Cumulative function evaluation time versus number of function evaluations for all four learning models for the single well case

For the LSTM (Fig. 4a), we can see that the computing times of all three HPO methods are approximately equal. Since the final optimized networks are smaller compared to MLP for all HPO methods and the number of epochs is approximately the same for RS and GP (compare Fig. 3a in Sect. C.1.1 in the online supplement), we conclude that on average all HPO methods tried networks of similar complexity. The computing times for the CNN (Fig. 4c) are significantly larger than for all other network types (RBF needs on average over 30 hours for the CNN compared to less than 8 hours for all other network types). For RNN (Fig. 4d), we observe large variations between the times for the individual optimization trials. RBF may take from 3 to 11 hours and RS takes between 7 hours and 25 hours. Thus, if the goal is to train learning models on a laptop or simple desktop computer to make predictions for future groundwater levels in a timely manner, one may prefer using the MLP or the LSTM in terms of expected computing time.

Analysis of performance on validation data We illustrate in Fig. 5 how well the optimized MLP and LSTM models capture the validation time series data (for CNN and RNN results, see the online supplement, Sect. C.1.2, Fig. 4). As described earlier, we train the models on a subset of the historical observations (approximately beginning 2010-end 2013) and we validate their performance on another subset (approximately beginning 2014-end 2015) to compute the outer objective function values. In Fig. 5a and b, we show the training data and for each optimization trial, we show the predictions of the optimized models for the validation data. The closer the predictions (red graphs) are to the true data (blue), the better. We can see that both MLP and LSTM are able to capture the seasonality in the data well. The predictions made with the MLP appear more erratic, i.e, we see many small jumps in daily groundwater predictions. This behavior is observed in the daily groundwater measurements and is due to, e.g., irrigation or other water withdrawal activities. The LSTM predictions are somewhat smoother, which indicates that the daily drawdown patterns are not learned as well.

Out-of-sample future groundwater predictions Finally, we are interested in how well the optimized learning models will predict groundwater data that was not used during the HPO.



(a) MLP **(b)** LSTM

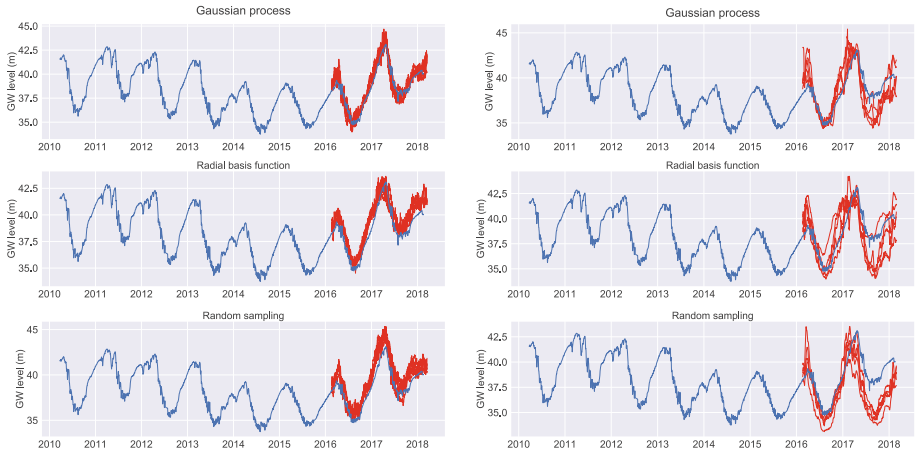
Fig. 5 Groundwater level predictions obtained from five hyperparameter optimizations of MLP and LSTM, respectively, for the single well case for each HPO method. *GW* groundwater

We retrain the models with the optimized hyperparameters on the whole dataset $\mathcal{D}$ (2010–2016) and we predict the groundwater levels from February 20, 2016 to February 28, 2018. We illustrate these "out-of-sample" predictions for each of the five trials for MLP and LSTM in Fig. 6. We see that both models are able to keep the trend in the data (decreasing groundwater levels in the summer, increasing levels in the winter), but the MLP predictions are much more accurate and less variable than the LSTM predictions. The LSTM is not able to capture the higher groundwater levels in the wet years (2017, 2018). The out-of-sample predictions for CNN and RNN are provided in the online supplement in Sect. C.1.3 in Fig. 5. The RNN makes significantly worse out-of-sample predictions than the MLP. The distributions of the MSEs for the out-of-sample predictions are provided in the form of boxplots in the online supplement in Fig. 6.

5.6 Results for predicting groundwater at multiple wells in Butte County, CA

Watersheds are large and generally have more than a single well from which groundwater is pumped or with the help of which the amount of available groundwater is monitored. Using multiple wells will allow us to better assess the average health of a watershed, and thus to avoid letting local fluctuations drive policies and decisions. Therefore, within Butte County, we selected three additional wells. Ideally, we do not want to train and optimize a separate learning model for each well, but rather we want one learning model that can be trained on and used for predicting the data at multiple wells simultaneously. We train and optimize the hyperparameters as before with the only difference that we now have three groundwater input data streams and three nodes in each output layer. We use these data to optimize the hyperparameters of MLP, LSTM, and CNN. For reasons of space considerations, we present detailed results only for the MLP. The LSTM and CNN results are in the online supplement in Sect. C.2.

Comparison of mean squared errors We show a comparison of the distributions of the final mean squared errors for all three learning models and all HPO methods obtained from five optimization trials in Fig. 7. The optimal mean squared errors are for almost all models and



(a) MLP **(b)** LSTM

Fig. 6 Out-of-sample groundwater predictions with the five optimized MLP and LSTM models, respectively, for the single well case. *GW* groundwater

HPO methods larger by an order of magnitude for the multi-well case than for the single well case (see also the numerical data provided in Table 3 in the online supplement). This is to be expected as more time series must be fit and one set of hyperparameters may not be optimal for the data of all three wells. We can see that the MLP and the CNN optimized with the RBF approach lead to the lowest errors. Also RS performs reasonably well for all three models in the multi-well case, while it showed significantly worse performance in the single well case. Thus, in terms of performance consistency, one should opt for an HPO method that systematically explores the search space rather than the RS method.

Analysis of optimized hyperparameters We show the distribution of the optimized hyperparameters only for the MLP (Fig. 8). The results for LSTM and CNN are in the online supplement in Sect. C.2.1 in Fig. 7. For the MLP, the RBF method chooses again a large complex network (4–5 layers and on average over 30 nodes per layer) and a large lag. Also the optimal solutions for the CNN (see online supplement) show that a complex network with many layers and filters is preferred and also the lag is large. On the other hand, the optimal LSTM network structure is significantly smaller with only 1–2 layers.

Analysis of HPO convergence As for the single well case, we illustrate how fast each HPO method converges. We show the progress plots on the example of MLP in Fig. 9a (results for LSTM and CNN are in the online supplement in Sect. C.2.2 in Fig. 8). We observe that all HPO methods converge fast and the RBF method achieves on average the best performance.

Analysis of computing times The cumulative computation time for the MLP over all function evaluations is shown in Fig. 9b. All HPO methods require approximately the same time and the HPO is completed within 4 hours. The CNN again requires the largest amount of time to solution (up to 30 hours) and the HPO for the LSTM completes within 10 hours (see Fig. 10 in Sect. C.2.4 in the online supplement). Thus, we conclude that the MLP optimized with the RBF should be the learning model and HPO method of choice as it attains high quality solutions within the lowest computation time.

Analysis of performance on validation data In Fig. 10, we show the predictions obtained with the optimized MLP models for all three wells for the validation data. All three HPO methods are able to capture the seasonality in the data and all three HPO methods appear to

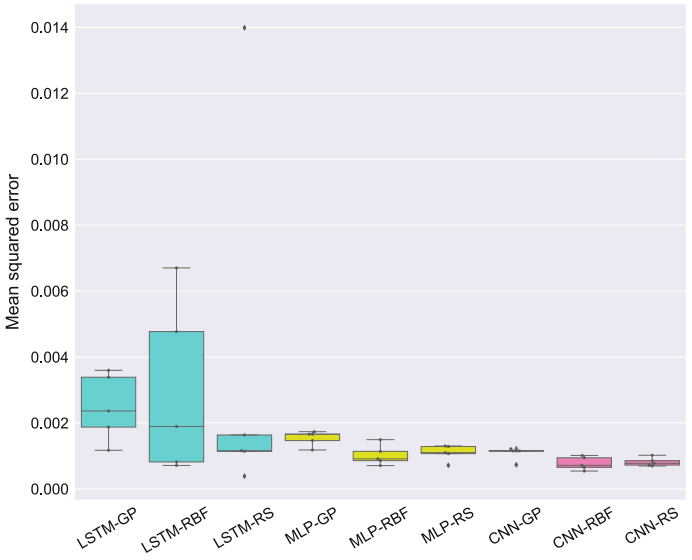


Fig. 7 Boxplots of mean squared errors for the multi-well case. *LSTM* long short-term memory network, *MLP* multilayer perceptron, *CNN* convolutional neural network, *RNN* recurrent neural network, *GP* Gaussian process, *RBF* radial basis function, *RS* random sampling

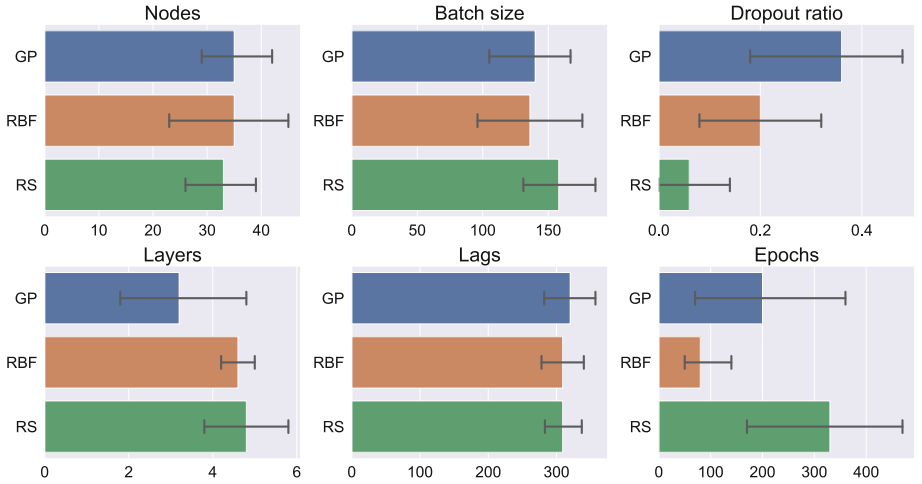
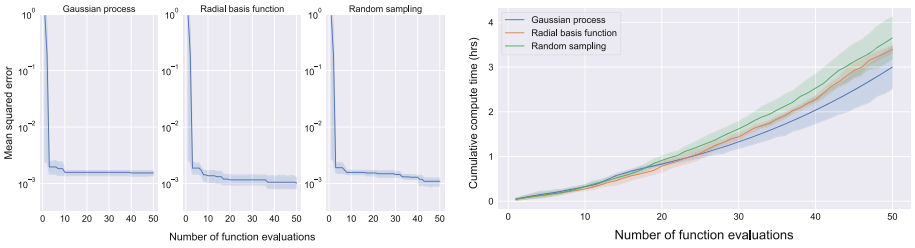


Fig. 8 Distribution of the optimized hyperparameter values for MLP for the multi-well case. The colored bars show the mean of the values over five trials and the black lines indicate the standard deviation. *GP* Gaussian process, *RBF* radial basis function, *RS* random sampling

underpredict the true groundwater levels for wells A and B. Well C is generally better fitted by all three HPO methods. This same observation holds true for the CNN predictions (Fig. 9b in Sect. C.2.3 in the online supplement). One possible explanation is that the errors made in the predictions for well C dominate the HPO. This may be due to the more erratic behavior of the groundwater levels for well C, which are manifested by more severe drawdowns and thus more patterns must be learnt (the most complex time series dominated the optimization).



(a) Convergence plot **(b)** Cumulative evaluation time

Fig. 9 Convergence plot showing the average and the standard deviation of the outer objective function value versus the number of function evaluations (left) and cumulative function evaluation time versus number of function evaluations (right) for MLP for the multiple well case

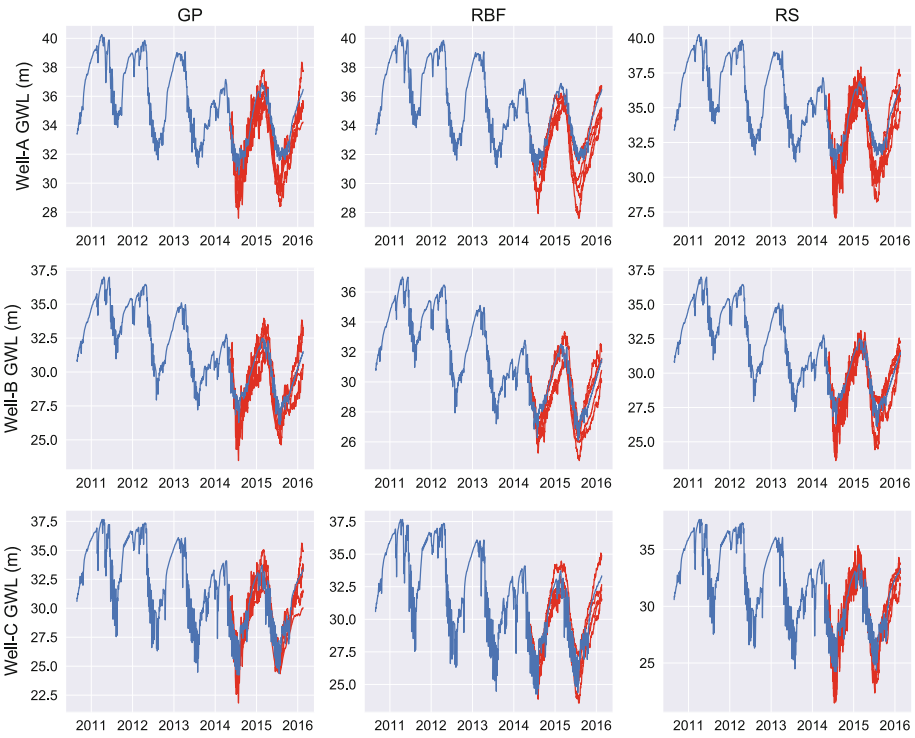


Fig. 10 Groundwater level predictions obtained from five hyperparameter optimizations of MLP for the multiple well case for each HPO method. *GWL* groundwater level

Using a weighted sum of the wells' errors as outer objective function could potentially help mitigating this problem. The LSTM predictions (Fig. 9a in Sect. C.2.3 in the online supplement) have generally a larger variability than CNN and MLP and we observe larger jumps in the daily groundwater predictions. Note that for the multi-well case, we do not have additional observations for all three wells for making "out-of-sample" predictions.

6 Summary and directions of future research

Our work presented in this article is motivated by the need for computationally lightweight approximation models that can run on a laptop or simple desktop computer and that can reasonably accurately predict groundwater levels with the overarching goal to enable timely and reliable water resources management decisions.

In this article, we studied the applicability of several purely data-informed learning models whose hyperparameters were optimized with three different methods for predicting the groundwater level. We used four deep learning models, namely convolutional neural networks (CNNs), recurrent neural networks (RNNs), multilayer perceptron (MLP), and long short-term memory networks (LSTMs). We formulated a bilevel optimization problem for finding the optimal hyperparameters of the learning models. At the upper level, we have a pure-integer stochastic problem, and at the lower level, we have a large-scale global optimization problem. We solved the hyperparameter optimization (HPO) problem with three different methods. We used two surrogate model based optimization methods (one based on radial basis functions (RBFs) and one based on Gaussian process (GP) models) and a random sampling method. In our numerical experiments, we trained the learning models on daily observation data of groundwater levels and meteorological variables at, respectively, one ("single well case") and three ("multi-well case") groundwater monitoring wells in Butte County, California, USA.

Our study shows that with the right HPO method, different types of deep learning models, including CNN, LSTM, and MLP (and to a limited extend also RNN), can be tuned to make accurate forecasts of daily groundwater levels (within two meters of the true values). The main difference between the performance of the methods is the total optimization time and the associated optimal model complexity. We showed that the surrogate model based HPO methods lead to better performing network architectures than randomly sampling the hyperparameter space. Our results show that *the "simplest" learning model, the MLP, optimized with the RBF method, generally performs best and its optimization time is lowest*. The RNN is the worst performing model and the most cumbersome to use as it often fails to fit a model for an architecture due to its problems during backpropagation. The CNN also performs well in terms of prediction accuracy, but it requires considerably more computation time than the other models. The results for the LSTM are less robust and have a large variation.

Directions of future research include applying the RBF-optimized MLP model for groundwater predictions at other watersheds in California and other parts of the United States to assess how well our method generalizes to watersheds with different characteristics (e.g., geology, groundwater use, weather patterns). We will study the sensitivity of the model performance to different input variables. This analysis will inform us about which data should be collected and thus where the necessary data acquisition infrastructure investments should be made. Similarly, we will study how sensitive the model predictions are to the amount of training data points (daily versus monthly observations) and when fewer years of data are used in training. This future research direction also includes applying our method for learning model optimization on other time series data such as predicting water quality.

Acknowledgements This work was supported by Laboratory Directed Research and Development (LDRD) funding from Berkeley Lab, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. et al.: Tensorflow: large-scale machine learning on heterogeneous distributed systems (2016). arXiv:1603.04467
2. Abramson, M.A., Audet, C., Chrissis, J., Walston, J.: Mesh adaptive direct search algorithms for mixed variable optimization. Optim. Lett. **3**, 35–47 (2009)
3. Ali, Z., Hussain, I., Faisal, M., Nazir, H.M., Hussain, T., Shad, M.Y., Shoukry, A.M., Gani, S.H.: Forecasting drought using multilayer perceptron artificial neural network model. Adv. Meteorol., 5681308, 9 pages (2017)
4. Araujo, P., Astray, G., Ferrerio-Lage, J.A., Mejuto, J.C., Rodriguez-Suarez, J.A., Soto, B.: Multilayer perceptron neural network for flow prediction. J. Environ. Monit. **13**(1), 35–41 (2011)
5. Audet, C., Dennis Jr., J.E.: Mesh adaptive direct search algorithms for constrained optimization. SIAM J. Optim. **17**, 188–217 (2006)
6. Audet, C., Hare, W.: Derivative-Free and Blackbox Optimization. Springer Series in Operations Research and Financial Engineering. Springer, Berlin (2017)
7. Audet, C., Kokkolaras, M.: Blackbox and derivative-free optimization: theory, algorithms and applications. Optim. Eng. **17**(1), 1–2 (2016)
8. Audet, C., Savard, G., Zghal, W.: A mesh adaptive direct search algorithm for multiobjective optimization. Eur. J. Oper. Res. **204**(3), 545–556 (2010)
9. Balaprakash, P., Salim, M., Uram, T.D., Vishwanath, V., Wild, S.M.: Deephyper: asynchronous hyperparameter search for deep neural networks. In: 2018 IEEE 25th International Conference on High Performance Computing (HiPC), pp. 42–51 (2018)
10. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(1), 281–305 (2012)
11. Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of the 30th International Conference on Machine Learning (2013)
12. Bishop, C.M., et al.: Neural networks for pattern recognition. Oxford University Press, Oxford (1995)
13. Booker, A.J., Dennis Jr., J.E., Frank, P.D., Serafini, D.B., Torczon, V., Trosset, M.W.: A rigorous framework for optimization of expensive functions by surrogates. Struct. Multidiscip. Optim. **17**, 1–13 (1999)
14. Borovykh, A., Bohte, S., Oosterlee, C.W.: Conditional Time Series Forecasting with Convolutional Neural Networks (2017). arXiv:1703.04691
15. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: 19th International Conference on Computational Statistics, pp. 177–186 (2010)
16. California Department of Water Resources. SGMA groundwater management. https://www.waterboards.ca.gov/water_issues/programs/gmp/docs/sgma/sgma_20190101.pdf. Accessed 18 May 2020
17. Chiang, Y.-M., Chang, L.-C., Chang, F.-J.: Comparison of static-feedforward and dynamic-feedback neural networks for rainfall-runoff modeling. J. Hydrol. **290**(3–4), 297–311 (2004)
18. Chollet, F.: keras. GitHub Repository (2015). https://github.com/fchollet/keras. Accessed 18 May 2020
19. Cook, B.I., Mankin, J.S., Anchukaitis, K.J.: Climate change and drought: from past to future. Curr. Clim. Change Rep. **4**(2), 164–179 (2018)
20. Coulibaly, P., Anctil, F., Aravena, R., Bobée, B.: Artificial neural network modeling of water table depth fluctuations. Water Resour. Res. **37**(4), 885–896 (2001)
21. Cui, Z., Chen, W., Chen, Y.: Multi-scale Convolutional Neural Networks for Time Series Classification (2016). arXiv:1603.06995
22. Daliakopoulos, I.N., Coulibaly, P., Tsanis, I.K.: Groundwater level forecasting using artificial neural networks. J. Hydrol. **309**(1–4), 229–240 (2005)
23. Datta, R., Regis, R.G.: A surrogate-assisted evolution strategy for constrained multi-objective optimization. Expert Syst. Appl. **57**, 270–284 (2016)
24. Davis, E., Ierapetritou, M.: Kriging based method for the solution of mixed-integer nonlinear programs containing black-box functions. J. Global Optim. **43**, 191–205 (2009)

25. Faunt, C.C.: Groundwater Availability of the Central Valley Aquifer, California. Professional paper 1766, 225 p., U.S. Geological Survey (2009). https://pubs.usgs.gov/pp/1766/PP_1766.pdf. Accessed 18 May 2020
26. Forrester, A.I.J., Sóbester, A., Keane, A.J.: Multi-fidelity optimization via surrogate modelling. Proc. R. Soc. **463**, 3251–3269 (2007)
27. Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Gagné, C., Parizeau, M.: DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. **13**, 2171–2175 (2012)
28. Gardner, M.W., Dorling, S.R.: Artificial neural networks (the multilayer perceptron): a review of applications in the atmospheric sciences. Atmos. Environ. **32**(14–15), 2627–2636 (1998)
29. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**, 2451–2471 (2000)
30. Gramacy, R., Le Digabel, S.: The mesh adaptive direct search algorithm with treed Gaussian process surrogates. Pac. J. Optim. **11**, 419–447 (2015)
31. Graves, A., Mohamed, A., Hinton, G.: Speech recognition with deep recurrent neural networks. In: Proceedings of the 2013 International Conference on Acoustics, Speech, and Signal Processing (2013)
32. Gutmann, H.-M.: A radial basis function method for global optimization. J. Global Optim. **19**, 201–227 (2001)
33. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
34. Hinton, G., Srivastava, N., Swersky, K.: Neural Networks for Machine Learning. lecture 6a, Overview of Mini-batch Gradient Descent. Lecture Notes (2012). https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed 18 May 2020
35. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**, 1735–1780 (1997)
36. Holmström, K.: An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer global optimization. J. Global Optim. **9**, 311–339 (2008a)
37. Holmström, K.: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. J. Global Optim. **41**, 447–464 (2008b)
38. Hsu, D.: Multi-period Time Series Modeling with Sparsity Via Bayesian Variational Inference (2018). arXiv:1707.00666v3
39. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.A.: Efficient hyperparameter optimization of deep learning algorithms using deterministic RBF surrogates. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (2017)
40. Jin, H., Song, Q., Hu, X.: Auto-Keras: An Efficient Neural Architecture Search System (2019). arXiv:1806.10282 [cs.LG]
41. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**, 455–492 (1998)
42. Karandish, F., Šimunek, J.: A comparison of numerical and machine-learning modeling of soil water content with limited input data. J. Hydrol. **543**, 892–909 (2016)
43. Karslıoğlu, O., Gehlmann, M., Müller, J., Nemšàk, S., Sethian, J., Kaduwela, A., Bluhm, H., Fadley, C.: An efficient algorithm for automatic structure optimization in x-ray standing-wave experiments. J. Electron Spectrosc. Relat. Phenom. **230**, 10–20 (2019)
44. Kingma, D.P., Ba, J.L.: ADAM: a method for stochastic optimization. In: ICLR 2015 (2015)
45. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast Bayesian optimization of machine learning hyperparameters on large datasets. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA, vol. 54 (2017)
46. Kratzert, F., Klotz, D., Brenner, C., Schulz, K., Herrnegger, M.: Rainfall-runoff modelling using long short-term memory (LSTM) networks. Hydrol. Earth Syst. Sci. **22**(11), 6005–6022 (2018)
47. Kuderer, M., Gulati, S., Burgard, W.: Learning driving styles for autonomous vehicles from demonstration. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 2641–2646 (2015)
48. Lakhmiri, D., Digabel, S. Le, Tribes, C.: HyperNOMAD: Hyperparameter Optimization of Deep Neural Networks Using Mesh Adaptive Direct Search (2019). arXiv:1907.01698 [cs.LG]
49. Langevin, C.D., Hughes, J.D., Banta, E.R., Niswonger, R.G., Panday, S., Provost, A.M.: Documentation for the MODFLOW 6 Groundwater Flow Model. Technical Report, US Geological Survey (2017)
50. Langhans, W., Müller, J., Collins, W.: Optimization of the Eddy-diffusivity/mass-flux shallow cumulus and boundary-layer parameterization using surrogate models. J. Adv. Model. Earth Syst. **11**, 402–416 (2019)
51. Le Digabel, S.: Algorithm 909: NOMAD–nonlinear optimization with the MADS algorithm. ACM Trans. Math. Softw. **37**, 1–15 (2011)
52. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)

53. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
54. Lee, H.K.H., Gramacy, R.B., Linkletter, C., Gray, G.A.: Optimization subject to hidden constraints via statistical emulation. Pac. J. Optim. **7**, 467–478 (2011)
55. Ma, X., Tao, Z., Wang, Y., Yu, H., Wang, Y.: Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. Transp. Res. C Emerg. Technol. **54**, 187–197 (2015)
56. Matheron, G.: Principles of geostatistics. Econ. Geol. **58**, 1246–1266 (1963)
57. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh Annual Conference of the International Speech Communication Association (2010)
58. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1996)
59. Moritz, S., Bartz-Beielstein, T.: imputeTS: Time Series Missing Value Imputation in R. R J. **9**, 207–218 (2017)
60. Müller, J.: MISO: mixed integer surrogate optimization framework. Optim. Eng. **17**(1), 177–203 (2015)
61. Müller, J.: SOCEMO: surrogate optimization of computationally expensive multiobjective problems. INFORMS J. Comput. **29**(4), 581–596 (2017)
62. Müller, J.: An algorithmic framework for the optimization of computationally expensive bi-fidelity black-box problems. INFOR Inf. Syst. Oper. Res. (2019). https://doi.org/10.1080/03155986.2019.1607810
63. Müller, J., Day, M.: Surrogate optimization of computationally expensive black-box problems with hidden constraints. INFORMS J. Comput. (2019). https://doi.org/10.1287/ijoc.2018.0864
64. Müller, J., Woodbury, J.: GOSAC: global optimization with surrogate approximation of constraints. J. Glob. Optim. (2017). https://doi.org/10.1007/s10898-017-0496-y
65. Müller, J., Shoemaker, C.A., Piché, R.: SO-MI: a surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. Comput. Oper. Res. **40**, 1383–1400 (2013a)
66. Müller, J., Shoemaker, C.A., Piché, R.: SO-I: a surrogate model algorithm for expensive nonlinear integer programming problems including global optimization applications. J. Glob. Optim. **59**, 865–889 (2013b)
67. Müller, J., Paudel, R., Shoemaker, C.A., Woodbury, J., Wang, Y., Mahowald, N.: CH4 parameter estimation in CLM4.5bgc using surrogate global optimization. Geosci. Model Dev. Discus. **8**, 141–207 (2015)
68. Myers, R.H., Montgomery, D.C., Anderson-Cook, C.M.: Response Surface Methodology: Process and Product Optimization Using Designed Experiments, 4th edn. John Wiley & Sons, Inc., Hoboken, NJ (2016)
69. Najah, A., El-Shafie, A., Karim, O.A., El-Shafie, A.H.: Application of artificial neural networks for water quality prediction. Neural Comput. Appl. **22**(1), 187–201 (2013)
70. Nuñez, L., Regis, R.G., Varela, K.: Accelerated random search for constrained global optimization assisted by radial basis function surrogates. J. Comput. Appl. Math. **340**, 276–295 (2018)
71. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
72. Powell, M.J.D.: Advances in Numerical Analysis, Vol. 2: Wavelets, Subdivision Algorithms and Radial Basis Functions. Oxford University Press, Oxford, pp. 105–210, Chapter The Theory of Radial Basis Function Approximation in 1990. Oxford University Press, London (1992)
73. Powell, M.J.D.: Recent Research at Cambridge on Radial Basis Functions New Developments in Approximation Theory, pp. 215–232. Birkhäuser, Basel (1999)
74. Regis, R.G.: Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. Comput. Oper. Res. **38**, 837–853 (2011)
75. Regis, R.G., Shoemaker, C.A.: A stochastic radial basis function method for the global optimization of expensive functions. INFORMS J. Comput. **19**, 497–509 (2007)
76. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**(3), 400–407 (1951)
77. Rudy, S., Alla, A., Brunton, S.L., Kutz, J.N.: Data-driven identification of parametric partial differential equations. SIAM J. Appl. Dyn. Syst. **18**(2), 643–660 (2019)
78. Rumelhart, D.E., Hinton, G.E., Williams, R.J., et al.: Learning representations by back-propagating errors. Cognit. Model. **5**(3), 1 (1988)
79. Sahoo, S., Russo, T.A., Elliott, J., Foster, I.: Machine learning algorithms for modeling groundwater level changes in agricultural regions of the US. Water Resour. Res. **53**(5), 3878–3895 (2017)
80. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems (2012)
81. Steefel, C.I., Appelo, C.A.J., Arora, B., Jacques, D., Kalbacher, T., Kolditz, O., Lagneau, V., Lichtner, P.C., Mayer, K.U., Meeussen, J.C.L., et al.: Reactive transport codes for subsurface environmental simulation. Comput. Geosci. **19**(3), 445–478 (2015)

82. Sundermeyer, M., Schluter, R., Ney, H.: LSTM neural networks for language modeling. In: Proceedings of the 12th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, pp. 601–608 (2012)
83. Sutskever, I., Martens, J., Hinton, G.E.: Generating text with recurrent neural networks. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 1017–1024 (2011)
84. Tabari, H., Talaee, P.H.: Multilayer perceptron for reference evapotranspiration estimation in a semiarid region. Neural Comput. Appl. **23**(2), 341–348 (2013)
85. Taylor, M.: Liquid Assets: Improving Management of the State's Groundwater Resources. Legislative Analyst's Office, Technical Report (2010)
86. Toal, D., Keane, A.: Efficient multi-point aerodynamic design optimization via co-kriging. J. Aircr. **48**(5), 1685–1695 (2011)
87. Trenn, S.: Multilayer perceptrons: approximation order and necessary number of hidden units. IEEE Trans. Neural Netw. **19**(5), 836–844 (2008)
88. Wild, S.M., Shoemaker, C.A.: Global convergence of radial basis function trust-region algorithms for derivative-free optimization. SIAM Rev. **55**, 349–371 (2013)
89. Xu, T., Spycher, N., Sonnenthal, E., Zhang, G., Zheng, L., Pruess, K.: TOUGHREACT version 2.0: a simulator for subsurface reactive transport under non-isothermal multiphase flow conditions. Comput. Geosci. **37**(6), 763–774 (2011)
90. Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.-H., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: MLHPC'15 Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, Volume Article No. 4 (2015)
91. Zhang, J., Zhu, Y., Zhang, X., Ye, M., Yang, J.: Developing a long short-term memory (LSTM) based model for predicting water table depth in agricultural areas. J. Hydrol. **561**, 918–929 (2018)