Machine Learning (2021) 110:1527–1547
https://doi.org/10.1007/s10994-021-06014-6

MODES: model-based optimization on distributed embedded systems

Junjie Shi[1] · **Jiang Bian**[2,3] · **Jakob Richter**[4] · **Kuan-Hsun Chen**[1] · **Jörg Rahnenführer**[4] · **Haoyi Xiong**[2] · **Jian-Jia Chen**[1]

Received: 20 September 2020 / Revised: 25 May 2021 / Accepted: 27 May 2021 /
Published online: 4 June 2021

Abstract

The predictive performance of a machine learning model highly depends on the corresponding hyper-parameter setting. Hence, hyper-parameter tuning is often indispensable. Normally such tuning requires the dedicated machine learning model to be trained and evaluated on centralized data to obtain a performance estimate. However, in a distributed machine learning scenario, it is not always possible to collect all the data from all nodes due to privacy concerns or storage limitations. Moreover, if data has to be transferred through low bandwidth connections it reduces the time available for tuning. Model-Based Optimization (MBO) is one state-of-the-art method for tuning hyper-parameters but the application on distributed machine learning models or federated learning lacks research. This work proposes a framework *MODES* that allows to deploy MBO on resource-constrained distributed embedded systems. Each node trains an individual model based on its local data. The goal is to optimize the combined prediction accuracy. The presented framework offers two optimization modes: (1) *MODES*-B considers the whole ensemble as a single black box and optimizes the hyper-parameters of each individual model jointly, and (2) *MODES*-I considers all models as clones of the same black box which allows it to efficiently parallelize the optimization in a distributed setting. We evaluate *MODES* by conducting experiments on the optimization for the hyper-parameters of a random forest and a multi-layer perceptron. The experimental results demonstrate that, with an improvement in terms of mean accuracy (*MODES*-B), run-time efficiency (*MODES*-I), and statistical stability for both modes, *MODES* outperforms the baseline, i.e., carry out tuning with MBO on each node individually with its local sub-data set.

This work was partly supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, A1 and A3 (http://sfb876.tu-dortmund.de/).

Editors: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautmann.

✉ Junjie Shi
junjie.shi@tu-dortmund.de

Extended author information available on the last page of the article

1 Introduction

Nowadays, statistical and machine learning algorithms are used more frequently and intensively to solve problems in a wide range of applications, e.g., smart home, medical diagnosis, and environment analysis. These algorithms are often highly parameterizable and their performances are sensitive to hyper-parameter settings. For example, the well-known Multi-Layer Perceptron (MLP) (Gardner and Dorling 1998) suffers from a large variance of prediction accuracy with different hyper-parameter settings for the same task, where the hyper-parameters include the number of layers, the number of neurons in each layer, the type of the activation functions, the learning strategies, etc. All of these settings should be well configured before a machine learning model is applied to a real application.

Hyper-parameter tuning is essential to achieve good predictive performance, while it quickly becomes expensive as the data size and/or search space grows. In the past decades, many hyper-parameter tuning algorithms have been developed and analyzed. As the state-of-the-art, Model-Based Optimization (MBO) iterates between fitting models and uses them to make choices about which configurations to investigate. One concrete strategy is *Bayesian optimization* (Jones et al. 1998), which solves the expensive optimization problem by fitting a Gaussian process (GP) regression to approximate the predictive performance in dependence of the hyper-parameters.

Normally, such hyper-parameter tuning requires the dedicated machine learning model to be trained and evaluated on centralized data to obtain a performance estimate. However, the original design of a centralized hyper-parameter tuning process is no longer suitable and efficient, if centralized data is not available, e.g., a distributed setting is considered.

Distributed embedded (as well as edge computing) systems are widely utilized to run various machine learning algorithms due to their high flexibility (mobility), scalability, and low energy consumption in real-world applications (Bian et al. 2018; Gu et al. 2012; Levinson et al. 2011). For example, modern air quality monitoring systems consist of multiple nodes located around the target area, in order to increase robustness and eliminate possible bias. Each node can be regarded as an individual system. It has a sensor module used for monitoring the environment and collecting data, and a processing module, which is able to load light-weighted machine learning tasks based on locally collected data, and supports efficient training and fast inference. These distributed embedded systems are more powerful and intelligent than traditional sensors that are only used for collecting data.

In such a distributed setting, if data is transferred through low bandwidth connections, merging all sub-data sets to one central node consumes a large amount of communication resources and leads to large overheads, and, hence reduces the available time for tuning. In some scenarios, it is impossible to collect and store the raw data due to privacy concerns or limited storage of the central node. In addition, distributed nodes have overlapping sensing areas and the redundant data (repeatedly uploaded) causes further burdens to the central node. Moreover, the execution time of machine learning algorithms is usually sensitive to the adopted hardware platforms. In an extensive study of unsupervised methods, the impact of particular implementations, frameworks, programming languages and libraries on the run-time performance has been shown in Kriegel et al. (2017). Particularly for run-time considerations, it has been stated that caching behavior determines the performance of implemented algorithms even more than algorithmic differences (Nijssen and Kok 2006). For example, the run-time of a random forest in Buschjager et al. (2018) is optimized for different platforms using different settings due to the different hardware designs, e.g., cache size. Therefore, if the objective of the tuning is to speed up the algorithm, the optimal setting on the central node may not be optimal for the dedicated distributed embedded systems due to different hardware architectures.

Alternatively, each node can conduct hyper-parameter tuning independently based on its local data. However, for each node the storage and detecting area are limited. Hence each node can only keep one part of the whole data set collected in this area. If each node tunes the hyper-parameters independently using its local sub-data set, the performance of the machine learning algorithm will vary due to the small size of the training data. The main challenge of the hyper-parameter tuning on a distributed embedded systems lies on how to utilize these decentralized sub-data sets to generate a universal hyper-parameter setting, which can be applied to all the nodes in this system. Towards this, a new method is desired to achieve the following three objectives:, i.e., (1) increase the (mean) accuracy of prediction; (2) improve the statistical stability; and (3) improve the run-time efficiency.

In this work, we propose *MODES*, a Model-Based Optimization method to tune hyper-parameters for machine learning algorithms on Distributed Embedded Systems **locally** and **efficiently**. Each node is treated as a small black box. It trains an individual model based on its local data. The whole distributed embedded system is considered as a big black box, and the goal is to optimize the performance of this black box, w.r.t. the mean accuracy of prediction, statistical stability, and/or run-time efficiency. Our contributions are as follows:

- We design a framework *MODES* to apply MBO on resource-constrained distributed embedded systems, which not only speeds up the tuning process to obtain the optimal hyper-parameters efficiently, but also improves the generalization ability of the obtained hyper-parameter setting. The proposed *MODES* tremendously mitigates the data communication cost by only transferring hyper parameter settings and performance values, i.e., accuracy of classifications.
- We further categorize *MODES* into two optimization modes: (1) the Black-box mode (*MODES*-B) considers the whole ensemble as a single black box and optimizes the hyper-parameters of each individual model jointly by considering the weights for different nodes and (2) the Individual mode (*MODES*-I) considers all models as clones of the same black box which allows it to efficiently parallelize the optimization in a distributed setting. Moreover, as an extensible and flexible framework, *MODES* is capable of fitting a wide range of applications with minor adaptation and *MODES* switch.
- We conduct extensive evaluations to compare our proposed two modes of *MODES* with a baseline method, in which each single node tunes its own hyper-parameter setting by applying MBO using its local data independently. The results show that: (1) *MODES*-B outperforms all the other methods in most of the evaluated cases. (2) *MODES*-I highly improves the run-time efficiency, where the improvement depends upon the number of nodes in the distributed system, at a cost of slightly degraded performance in some cases. The implementation of *MODES* and corresponding experiments are released in Shi et al. (2021).

2 Background and related works

In this section, we first introduce several hyper-parameter tuning algorithms. Afterwards, the *Model-Parallelism* and *Federated Learning* are discussed briefly, which motivate our work. Then, we discuss the most relevant works to our study.

2.1 Hyper-parameter tuning algorithms

The most direct and easy to implement tuning algorithm is grid search (LeCun et al. 2012) which discretizes the hyper-parameter search space and exhaustively evaluates all possible combinations in a Cartesian grid to find the setting with the best performance. Another variation is random search (Bergstra and Bengio 2012), which randomly samples hyper-parameter settings from the search space. The drawback of both tuning methods is that they do not make use of information obtained from previous tries, which implies a waste of computational resources. In contrast, Sequential Model-Based Optimization (SMBO) (Jones et al. 1998) takes advantage of the previous search trajectory. Several benchmarks (Hutter et al. 2013; Bischl et al. 2017; Berk et al. 2018) demonstrate the superiority of MBO over grid and random search as well as evolutionary approaches. In the classical approach, Gaussian process regression, also called Kriging, is used as its regression model (Snoek et al. 2012). For certain scenarios and hierarchical search spaces, tree-based surrogates, such as the Tree-structured Parzen Estimator (TPE) (Bergstra et al. 2013) or random forests (Hutter et al. 2011), have proved beneficial. Also, Bayesian Neural Networks (BNN) (Graves 2011) can serve as a surrogate. There, a probability distribution is learnt for each weight of the network to produce a variance around the prediction. However the training process is very time-consuming. Several extensions are proposed to speed up the BNN, e.g., sample multiple sub-networks from a network trained with Dropout (Srivastava et al. 2014; Gal and Ghahramani 2016).

In order to extend MBO with parallel evaluations, various techniques have been developed to propose and evaluate multiple points in each iteration. Ginsbourger et al. (2010) proposed several approaches based on imputing the results of currently running experiments. Hutter et al. (2012) proposed the qUCB, which uses the Gaussian process upper confidence bound (GP-UCB). By optimizing the GP-UCB with different weights for the uncertainty, we obtain a set of proposals, i.e., q denotes the number of obtained proposals. Recently, Coy et al. (2020) proposed the parallelized Bayesian optimization by keeping the number of evaluations low (sample efficient) and executing parallel evaluations to reduce wall-clock time, which outperforms the state-of-the-art parallel CMA-ES (Hansen and Ostermeier 2001) even on higher dimensions, e.g., 20-dimensional Sharp Ridge function. To account for heterogeneous run-times of different proposals, asynchronous parallel strategies (Janusevskis et al. 2012) as well as scheduling methods (Richter et al. 2016; Kotthaus et al. 2019) have been developed.

2.2 Model-parallelism and federated learning

Due to the increasing demands of distributed data collection, storage, and processing as well as the privacy-preserved concerns in many applications, federated learning (Konečnỳ et al. 2016; Li et al. 2019) has become one of the popular computing paradigms, where a machine learning model is trained across multiple decentralized edge devices or servers with their local data. In most federated computing platforms, “no raw data sharing” is an important requirement, where a machine learning algorithm should be trained using all data stored in all the distributed machines but without any cross-machine raw data sharing. Specifically, the aforementioned hyper-parameter tuning algorithms can be accelerated by federated learning and typically be divided into two types: *Data-Parallelism* (Baek 2011) and *Model-Parallelism* (Xing et al. 2015). On each embedded system (node), the *Data-Parallelism* algorithm first trains the model by using the local data. Afterwards, a global model is obtained via model-averaging (Claeskens et al. 2008). The aggregated model is considered as the trained model based on the overall data (from multiple nodes). Due to the construction of Data-Parallelism, parallel computing method can be easily applied. The *Model-Parallelism* requires multiple nodes to learn a shared prediction model collaboratively. Such an algorithm has to update parameters synchronously or asynchronously across all nodes causing additional overheads. In many applications, parameters updating can be a tough nut.

Both aforementioned approaches keep all the training data local on corresponding nodes. Compared with the Data-Parallelism (as the chosen baseline algorithms that are named starting with *S*-), the Model-Parallelism (which *MODES* adopts) usually can achieve better performance, as it globally optimizes the performance of the model (Xing et al. 2015). As one of the most popular branch of Neural Architecture Search (NAS) using Model-Parallelism, federated NAS (Garg et al. 2020; He et al. 2020; Zhu and Jin 2020) have been proposed to search for global and personalized models automatically for non-IID data. To further preserve privacy, differentially-private FNAS (Singh et al. 2020), which adds random noise to the gradients of architecture variables, has been designed for a higher level of privacy protection. These algorithms mainly focus on federated learning solutions for NAS with computationally expensive method(s) (e.g., reinforcement learning-based surrogate method) and powerful GPUs (e.g., RTX 2080Ti in He et al. (2020)).

2.3 Discussions

In this work, we study the problem of tuning the hyper-parameters for learning algorithms on resource constrained distributed embedded systems, where we consider the prediction accuracy, statistical stability and run-time efficiency as the objectives of hyper-parameter tuning. We formulate the problem as a black box optimization problem in a distributed nature, while each black box function is subject to a specific data source. Specifically, we propose to leverage Model-Based Optimization (MBO) methods to solve the problem. To the best of our knowledge, few studies in the field have taken resource constraints into consideration (Kotthaus et al. June 2017, 2019), or optimize the execution of MBO on a single multi-core embedded system with completed data set (Kotthaus 2018). However, none of them has been carried out w.r.t. Model-Parallelism in connection with MBO on distributed embedded systems, which have limited computational resources and different sub-data sets on each node.

3 Model based optimization

Model-Based Optimization (MBO) solves the optimization problem:

$$\boldsymbol{x}^* = \arg\underset{\boldsymbol{x}\in\mathcal{X}}{max} f(\boldsymbol{x})$$

for a given function $f(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$ with $\mathcal{X} \subset \mathbb{R}^p$. We assume that the true expensive black box function can be approximated through a surrogate. This surrogate is a regression method that is comparably inexpensive to be evaluated. In this work we use a Gaussian process regression, which is a typical choice for MBO. To start the optimization, an initial design $\mathcal{D}$ of k points, laid out in a Latin hyper-cube design across $\mathcal{X}$, is evaluated on the expensive function and yields the outcomes $\boldsymbol{y}$. In the following, the sequential model-based optimization iteratively repeats the following steps until a predefined budget is exhausted:

1. A Gaussian process is fitted to all past evaluations $\mathcal{D}$ and their outcomes $\boldsymbol{y}$, serving as a surrogate to estimate f globally.
2. An acquisition function is derived from the current surrogate.
3. The acquisition function ($acq(\boldsymbol{x})$) is optimized to determine the most promising point $\hat{\boldsymbol{x}}$:
$$\hat{\boldsymbol{x}} = \arg \underset{\boldsymbol{x} \in \mathcal{X}}{max}\, acq(\boldsymbol{x}).$$
4. $y = f(\hat{\boldsymbol{x}})$ is evaluated, $\hat{\boldsymbol{x}}$ and y are added to $\mathcal{D}$ and $\boldsymbol{y}$.

The acquisition function has to balance exploration (evaluate points where the surrogate's prediction is uncertain) and exploitation (evaluate points that are predicted to be optimal by the surrogate). The final optimal result $\hat{\boldsymbol{x}}^*$ is the input that leads to the maximal observed objective value, e.g., prediction accuracy.

A popular acquisition function is the expected improvement (EI). Using a Gaussian process as a surrogate yields a Gaussian posterior with mean $\hat{\mu}(\boldsymbol{x})$ and standard deviation $\hat{s}(\boldsymbol{x})$ at each point $\boldsymbol{x}$. Accordingly the expected improvement can be derived as follows:

$$\begin{aligned} \mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}(max(\hat{\mu}(\boldsymbol{x}) - y_{\max}), 0) \\ &= (\hat{\mu}(\boldsymbol{x}) - y_{\max})\Phi(\frac{\hat{\mu}(\boldsymbol{x}) - y_{\max}}{\hat{s}(\boldsymbol{x})}) + \hat{s}(\boldsymbol{x})\phi(\frac{\hat{\mu}(\boldsymbol{x}) - y_{\max}}{\hat{s}(\boldsymbol{x})}) \end{aligned} \tag{1}$$

where Φ is the distribution, and ϕ is the density function of the standard Gaussian distribution and $y_{\max}$ is the best observed value in $\boldsymbol{y}$ so far. This classical formulation of MBO only yields one proposal $\hat{\boldsymbol{x}}$ in each iteration. However, in our work, it is necessary to obtain multiple proposals in each iteration in order to make use of parallel computing infrastructures. Batch expected improvement (qEI) (Ginsbourger et al. 2010; Rezende et al. 2014) is proposed as an acquisition function for multiple proposals. It transforms the p-dimensional optimization problem for finding one promising point into a $p \cdot q$-dimensional optimization problem for finding q promising points. As the qEI lacks of an exact analytical representation for $q > 2$ it is usually solved approximately by Monte Carlo (MC) sampling methods. In Balandat et al. (2020) the qEI for $X = (\boldsymbol{x}_1 \dots \boldsymbol{x}_q)'$ is calculated as follows: We sample $\tilde{\boldsymbol{y}} \in \mathbb{R}^q$ from the joint posterior of X, which is given by the Gaussian process surrogate. We calculate the individual improvements $I = \max(\tilde{\boldsymbol{y}} - y_{\max}, 0)$. Then, we obtain $\max(I)$ for the current sample. Finally, we repeat those steps multiple (e.g. 1000) times and average over the obtained maximal improvements to obtain an MC approximation of the qEI for a given X. To obtain the set of q points that maximize the qEI, BoTorch uses gradient-based optimization.

The extensions of the qEI to noisy problems, namely the qNEI (Balandat et al. 2020), can be derived by replacing the fixed value $y_{\max}$ with $\max(\tilde{\boldsymbol{y}}_{\text{obs}})$ from the sample $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{y}}_{\text{obs}})'$ of the joint posterior of $(\boldsymbol{x}_1 \dots \boldsymbol{x}_q\, \boldsymbol{x}_{\text{obs},1} \dots \boldsymbol{x}_{\text{obs},t})'$, with $D = (\boldsymbol{x}_{\text{obs},1} \dots \boldsymbol{x}_{\text{obs},t})'$.

Similarly, when q = 1, the NEI can be calculated by introducing MC-sampling into the calculation of the EI. Therefore, we replace $y_{\max}$ with the average of multiple samples of $\max(\tilde{\boldsymbol{y}}_{\text{obs}})$.

In this work, single proposal MBO, i.e., EI and NEI are applied for *MODES*-B while parallelization through multiple proposals using the qEI and qNEI criterion is applied for *MODES*-I.

4 Distributed model-based optimization

In this section, the model of the distributed embedded system is introduced at first. Afterwards, to meet the requirements mentioned in Sect. 1, two categories of proposed *MODES* with different structures are explained in detail.

4.1 System model

In a distributed embedded system, also denoted as a *cluster*, several embedded systems cooperate towards a common objective. In this work, we assume a homogeneous cluster[1], in which all the nodes have identical characteristics. For this cluster, we assume:

- It consists of n nodes, denoted as ES_1, ES_2, $\cdots ES_n$. Each node is one embedded system.
- Each node has limited storage and can only store a certain amount of data.
- Data collected by different nodes are (at least partially) different and can be treated as sub-sets of a completed data set.
- Connections among nodes are of low bandwidth and only the tiny data can be transferred, i.e., hyper-parameter settings and performance results (accuracy of classifications).

In our setting, a host-client model is applied on all the available nodes. Although all nodes run a dedicated machine learning algorithm, only one node runs the MBO algorithm. The node where the MBO is deployed, is called *host*, which runs MBO and the dedicated machine learning algorithm at the edge at the same time. The remaining nodes, called *clients*, only run the dedicated machine learning algorithm. Due to our setting of limited computational power of embedded systems, only light-weighted machine learning algorithms are applied, which results in a relatively small search space for hyper-parameters. The number of hyper-parameters of the machine learning algorithm is denoted by p.

4.2 Black-box mode *MODES*-B

In *MODES*-B, the whole distributed system is treated as a single black box. The hyper-parameter setting as well as the weight of each individual node is optimized jointly in order to improve the performance as a way of ensemble learning. The whole system only generates one prediction at a time. Such a method can be utilized in a wide range of applications, e.g., air quality prediction in one area utilizing all the embedded sensors in that area (Bian et al. 2018), and object recognition by using images taken from different angles (Gu et al. 2012).

The structure of *MODES*-B is shown in Fig. 1, and the corresponding workflow is presented in Algorithm 1. MBO runs initial setup at first to construct the surrogate denoted as $\mathcal{S}$. At the beginning of each iteration, MBO only generates one set of hyper-parameters with the highest expected improvement w.r.t. the current surrogate, which consists of $(n \times p + n)$ hyper-parameters, denoted as $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, w_1, \ldots, w_n\}$. In each setting, first

[1] The proposed method can also be applied on heterogeneous clusters, with the effort to synchronize the execution of different nodes, e.g., assign the heavy workloads to nodes with more resources and better computational performance, which is out of scope for this work.

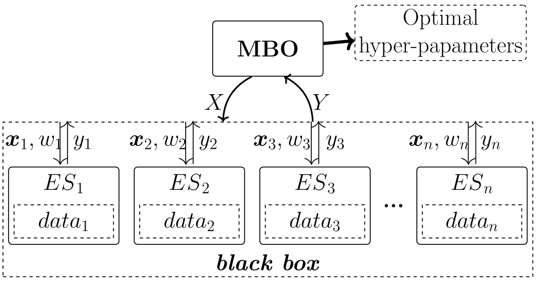
MBO
Optimal hyper-papameters
X
Y
$\boldsymbol{x}_1, w_1$ y_1
$\boldsymbol{x}_2, w_2$ y_2
$\boldsymbol{x}_3, w_3$ y_3
$\boldsymbol{x}_n, w_n$ y_n
ES_1
$data_1$
ES_2
$data_2$
ES_3
$data_3$
...
ES_n
$data_n$
black box

Fig. 1 *MODES*-B: The distributed embedded system is treated as a single black box

set $\boldsymbol{x}_1$ contains p elements that represent the hyper-parameters of the dedicated machine learning model for the first node, second set $\boldsymbol{x}_2$ represents the hyper-parameters for the second node and so on. Moreover, n weights indicating the importance of each node and its local data are represented through X as well.

The dedicated machine learning model ML is trained on each node by using the given hyper-parameter setting (i.e., $\boldsymbol{x}_j$ where j is the node id) and the local sub-data set. Each node generates one local performance result (accuracy of classification) of the trained machine learning model by using an evaluation test set. The final result Y is averaged according to the weights of results from all the nodes, i.e., $Y = \sum_{j=1}^{n} w_j \times y_j$, where y_j is the local performance result of node j, and $\sum_{j=1}^{n} w_j = 1$. In practice, each weight is a real number with the range [0.1, 1]. Afterwards, a normalization is operated to obtain the real weight for accuracy calculation, i.e., $w_j = \frac{w_j}{\sum_{i=1}^{n} w_i}$. Afterwards, the final result is utilized to update the surrogate of MBO. The process is repeated until the maximum number of iterations is reached or the time budget is exhausted.

```
Algorithm 1 Workflow of MODES-B
Input: number of nodes $n$, dedicated machine learning model $ML$, number of hyper-
    parameters $p$, time budget $T$, and maximum tuning iterations $Itr$;
Output: Optimal hyper-parameter setting: $HP\text{-}B$;
Initialize: MBO surrogate $\mathcal{S}$, iteration $i \leftarrow 0$, time $t \leftarrow 0$;
while $i \leq Itr$ and $t \leq \mathrm{T}$ do
    $X \leftarrow$ MBO $(\mathcal{S}, n, p)$;
    for $j$ from 1 to $n$ do
        $y_j = ML(\boldsymbol{x}_j, ES_j, data_j)$
    end for
    $Y \leftarrow \sum_{j=1}^{n} w_j \times y_j$;
    Update surrogate according to $(X, Y)$;
    $i$ ++;
    Accumulate consumed time $t$;
end while
MBO generates the optimized $HP\text{-}B$ according to current surrogate;
return $HP\text{-}B$;
```

In this mode, the number of dimensions of the search space is $n \times p + n$. Therefore, the large number of nodes (n) in the dedicated cluster and/or the large number of hyper-parameters (p) of the dedicated machine learning model can result in a search space with a large number of dimensions. The computation power that MBO needs to update the surrogate and to propose new settings is proportional to the size of the search space. However, due to the limited computational capability, embedded systems may not be able to find the optimal hyper-parameter setting from such a huge search space within a certain time budget.

Against this limitation, we enforce all the nodes to share the same setting of hyper-parameters but with different weights, i.e., $\forall i,j \leq n, i \neq j : \boldsymbol{x}_i = \boldsymbol{x}_j$ and $\exists i,j \leq n, i \neq j : w_i \neq w_j$. As a result, the search space is significantly reduced to $(p + n)$ dimensions. In each MBO iteration, all the nodes receive the same set of hyper-parameters, and train the dedicated machine learning model using their local data sets independently. Afterwards, the evaluation test set is utilized to evaluate the performance of these trained machine learning models on different nodes, and the weighted mean is returned to the *host* node, which is used to update the MBO surrogate. In the end, one set of optimized hyper-parameters along with the weights of nodes are obtained.

Please note, the proposed *MODES*-B with different hyper-parameters for each node, i.e., $\exists i,j \leq n, i \neq j : \boldsymbol{x}_i \neq \boldsymbol{x}_j$, can also be applied on powerful distributed systems. However, the performance evaluation is out of scope for this work.

4.3 Individual mode *MODES*-I

In *MODES*-I, each node is treated as an instance of the same black box. The whole cluster acts like a multi-processor system and each node is a single processor. This enables us to apply MBO in a parallelized manner. In this scenario, the performance of multiple proposed hyper-parameter settings can be evaluated at the same time, i.e., each node trains a dedicated machine learning model using one set of the proposed hyper-parameter settings and its local data set. In this mode, improving the timing efficiency is the most important objective, e.g., in some real world timing-sensitive applications like autonomous driving systems (Levinson et al. 2011).

The structure of *MODES*-I is shown in Fig. 2, the workflow is presented in Algorithm 2. In each iteration, MBO proposes n different hyper-parameter settings based on the knowledge obtained from the current surrogate, using the qEI or qNEI acquisition function as explained in Sect. 3. Each node uses one hyper-parameter setting to independently train the dedicated machine learning model using their local data. Afterwards, these trained models are evaluated by using a local evaluation test set. The individual performance measures, i.e., the classification accuracies, are sent back to the *host* node. In our setting, synchronized updating of the surrogate is applied, where the MBO updates the surrogate, once all nodes finished their evaluation. Therefore, the execution time of each iteration equals to the longest execution time of all these nodes. The iterations are repeated until the time budget is exhausted or the maximum number of iterations is reached. The optimization result is one hyper-parameter setting that can be utilized for all the nodes. The whole system can generate the prediction by a simple average with equal weights from different nodes. Alternatively, a single node can do the prediction itself with a lack of robustness.

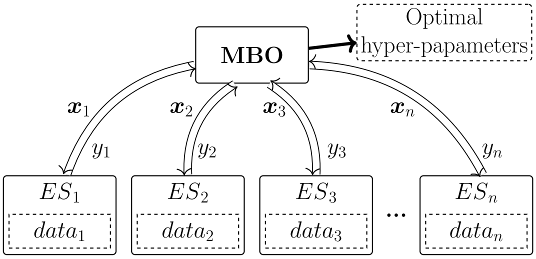

Fig. 2 *MODES*-I: Each embedded system acts as an individual black box

Algorithm 2 Workflow of *MODES*-I

```
Input: number of nodes $n$, dedicated machine learning model $ML$, number of hyper-
    parameters $p$, time budget $T$, and maximum tuning iterations $Itr$;
Output: Optimal hyper-parameter setting: $HP\text{-}I$;
Initialize: MBO surrogate $\mathcal{S}$, iteration $i \leftarrow 0$, time $t \leftarrow 0$;
while $i \leq Itr$ and $t \leq$ T do
    $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\} \leftarrow$ MBO $(\mathcal{S}, n, p)$;
    for $j$ from 1 to $n$ do
        $y_j \leftarrow$ ML$(\{\boldsymbol{x}_j, ES_j, data_j\})$;
    end for
    Update surrogate according to $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$;
    $i \leftarrow i$ + n;
    Accumulate consumed time $t$;
end while
MBO generates the optimized $HP\text{-}I$ according to current surrogate;
return $HP\text{-}I$;
```

MODES-I significantly improves the run-time efficiency of the hyper-parameter tuning process, by fully utilizing the computational power of all the nodes inside the distributed system, i.e., it evaluates *n* proposed settings in parallel by considering all the information from the local data in different nodes. Although the performance of the tuned hyper-parameters may not be improved significantly, due to the fact that different data in different nodes creates noisy results, it is still practical in running time sensitive applications on distributed embedded systems, since *MODES*-I has shorter response time in general. In some applications, the learned model becomes useless if it is not delivered within a specific time window, e.g., real-time traffic flow prediction and human activity recognition. For such applications, the tuning speed is as important as the accuracy. In case *MODES*-B is too slow to react, *MODES*-I would be a better choice, provided that the sub-data sets are consistent.

4.4 Comparison between *MODES*-B and *MODES*-I

The aforementioned *MODES*-B and *MODES*-I focus on different requirements with different assumptions. *MODES*-B tries to improve the performance of the whole system by considering the difference among different nodes. While *MODES*-I tries to improve the run-time efficiency of the tuning process by assuming the nodes and its local sub-data sets are with high similarity.

In *MODES*-B, the whole distributed embedded system is treated as an ensemble. Each hyper-parameter setting involves not only the hyper-parameters for the dedicated machine learning model, but also the weights for different models. In each iteration of the optimization process, only one single proposal is trained and evaluated in the entire system. In the end, the obtained optimized hyper-parameter setting is applied for the whole ensemble, and only one classification result is generated by the system. Theoretically, since the tuned weights represent the importance of different nodes and corresponding sub-data sets, *MODES*-B can outperform other hyper-parameter tuning algorithms if sub-data sets held by different nodes are imbalanced or some sub-data sets have great noise.

In *MODES*-I, multiple nodes in a distributed embedded system are treated as multiple clones of a single node. In addition, the local sub-data sets are considered as subsets of a consistent data set. This treatment relies on an assumption that the optimal hyper-parameters of the dedicated machine learning model for different nodes are highly similar. Therefore, multiple proposals are trained and evaluated on all the available nodes at the same time, in order to accelerate the optimization of the corresponding surrogate. Ideally, the tuning process can be sped up by n times, where n is the number of nodes in the dedicated distributed embedded system. However, some nodes with short execution time have to wait until the node with the longest execution for synchronized updating the surrogate. Hence the improvement of efficiency is actually less than n times. Although asynchronous parallel strategies (Janusevskis et al. 2012) as well as scheduling methods (Richter et al. 2016; Kotthaus et al. 2019) are developed for heterogeneous run-time of different proposals, the comparison of different surrogate updating strategies is considered out of scope. When there are many nodes, the resulting surrogate may not be able to generate a sufficient number of valuable proposals for evaluating the machine learning algorithms in parallel in the next iteration. That is, some of the proposed hyper-parameter settings to be evaluated have to be generated randomly without any contributions to the corresponding surrogate. Moreover, since each node can make the prediction independently, node(s) can be easily added or removed without affecting the functionality of the distributed system. Hence, *MODES*-I is more scalable, compared to *MODES*-B.

Table 1 compares all the aforementioned schemes, where Single is that each node tunes on its local data and Central tunes on centralized data from all nodes. The performance related features, i.e., accuracy, efficiency, and statistical stability will be demonstrated in the following section. Please note that *Central* is not evaluated as it is considered out of scope.

5 Evaluation

To validate the performance of *MODES*, we consider a distributed embedded system with four nodes. An emulation platform is established by using a cache-coherent SMP, consisting of two AMD 3990X processors and 256 GB main memory. The *host* is a desktop with one Intel i7-8700K processor, two Nvidia GTX1080 GPUs, and 32 GB main memory, which only runs the MBO. Our implementation is based on Balandat et al. (2020), which is a Bayesian optimization implementation in PyTorch. We adopt 4 popular real-world data sets with reasonable size, i.e., at most 60, 000 instances, to evaluate the proposed *MODES* framework:

1. The MNIST (LeCun et al. 1998) data set: it contains 60, 000 handwritten digits (from 0 to 9) images with 28 × 28 grey-scale resolution. The MNIST data set is widely used for evaluating the performance of machine learning algorithms. Here, we fit our learning task as an image classification problem on the MNIST data set.

Table 1 The comparison of *MODES*-B, *MODES*-I, Single, and Central with the notations {+: improved; o: no change; -: decreased}

	MODES-B	*MODES*-I	Single	Central
Privacy	+	+	+	−
Data parallelism	o	+	o	o
Model parallelism	+	o	o	o
Accuracy	+	o	o	+
Efficiency	o	+	o	−
Statistical stability	+	o	−	+

2. The Fashion-MNIST (Xiao et al. 2017) data set: it consists of Zalando's article images, where the statistics are exactly the same as the original MNIST data set, i.e., with the same number of instances, the same image size, and the same distribution of different classes. The Fashion-MNIST is more representative for modern computer vision tasks. It usually serves as a replacement for the original MNIST data set when benchmarking machine learning algorithms, since the original MNIST classification task is easy (e.g., MLP can easily achieve the accuracy of 95%) and overused in the machine learning domain.
3. The Covertype (Blackard and Dean 1999) data set: it is a non-vision data set, coming from the US Forest Service inventory information. This data set is originally used to predict forest cover type from cartographic variables, and it is sensitive for the model settings (parameter tuning) of some popular machine learning algorithms (e.g., MLP, SVM and RF). The original data set contains 581, 012 instances and 7 classes. However, the number of instances for different classes are extremely unbalanced, i.e., 100 times difference. Hence, we downsized the data set according to the size of the smallest class, i.e., each class now contains 2747 instances, and in total 19, 229 instances.
4. The HAR (Anguita et al. 2013) data set: it consists of 10, 299 instances, which are built from the recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. Therefore, the HAR data set naturally fits the distributed embedded systems scenario and it satisfies the assumptions of *MODES* well. As a sensing data set, six human activities are included, i.e., walking, climbing the stairs, walking down the stairs, sitting, standing, and laying.

Based on the selected data sets and the computational power of the platform, two machine learning algorithms that represent the state of the art are selected as the optimization objects: (1) Multi-Layer Perceptron (MLP) (Gardner and Dorling 1998) and (2) Random Forest (RF) (Liaw and Wiener 2002). The performance of these two benchmark machine learning algorithms have been well-reported on the aforementioned data sets, where they can be used as the references for the performance of our *MODES*. Moreover, the performances of MLP and RF are both sensitive to the hyper-parameters, which makes MBO tuning necessary.

5.1 Experimental setup

To efficiently evaluate the performance of fine-tuned machine learning algorithms, for the most accuracy-sensitive hyper-parameters among all adjustable hyper-parameters in MLP and Random Forest, we select values based on experience. There are 5 hyper-parameters for MLP and 7 hyper-parameters for RF need to be tuned, details can be found in Tables 2 and 3 respectively.

To simulate the possible patterns of distributed data storage, data sets are pre-processed. Firstly, each data set is randomly split into a training set, an evaluation test set and an unseen final test set with a ratio of 10:1:1. The evaluation test set is only used for hyper-parameter tuning, i.e., verify the performance of proposed hyper-parameter setting and the result is used to update the MBO surrogate. The unseen final test set is used to evaluate the final performance of hyper-parameters optimized by different methods and their data storage situations accordingly. Please note, although different evaluation and test sets can be applied on different nodes in real applications, we apply the same evaluation and test sets for all the nodes in our evaluation to eliminate the potential disturbance from the evaluation and test data sets. Finally, in order to simulate the situation of data storage on real distributed embedded systems, the sub-data set for each node is generated from the overall training set by applying the following strategies:

- **Uniform Split (D1)**: Equally divide the training set into four parts.
- **Duplicated Split (D2)**: Each of the four training sets from D1 is extended with 30% data randomly selected from the remaining three parts. Therefore, each sub-data set has overlapping data with the other sub-data sets.
- **Unbalanced Split (D3)**: Divide the training set unequally with shares of $20\%, 20\%, 30\%$, and 30%.

5.2 Selection of baselines

In order to compare the performance of our proposed methods, 9 algorithms are evaluated. These algorithms are named according to the following rules: (1) B/I/S in the first part: *MODES*-B, *MODES*-I, or Single is applied. (2) EI/NEI in the second part: expected improvement or noisy expected improvement is applied to generate a proposal for next iteration. qEI/qNEI is applied for *MODES*-I correspondingly, when q proposals are generated for parallel execution.

Each MBO tuning procedure has the same budget of maximal 100 iterations and 12 h run-time. For *MODES*-I, only 25 iterations and 3 h run-time are assigned, since it can evaluate four different hyper-parameter settings at the same time in each iteration, and in total 100 proposals are evaluated at the end. Afterwards, the optimized hyper-parameters are applied to train the dedicated machine learning algorithms. To be fair, the training data sets are the same during hyper-parameter tuning. At last, the identical testing data is adopted which is unseen for all methods. Since MBO itself has randomized decisions (including the selection of the initialized points and the proposals based on the surrogates), it is necessary to analyze the variance to verify the correctness of our evaluation results. Therefore, we repeated each experiment setting for 10 times, to show the statistical stability of proposed methods.

5.3 Experimental results

We evaluated all combinations and report the accuracy of the classification results for two machine learning algorithms and three data splitting strategies separately for the different data sets. Since the MLP and RF architectures are modularized and standardized (i.e., Scikit-learn (Pedregosa et al. 2011)), the randomness from the algorithm itself in reloading

Table 2 The 5 hyper-parameters that are tuned for MLP.

Parameter	Type	Range
Number of layers	ℕ	[1, 15]
Units per layer	ℕ	[10, 150]
Activation	Dictionary	{identity, logistic, tanh, relu}
L2 penalty	ℝ	[−5, −2] (log-10 scale)
Learning rate for Adam	ℝ	[−4, −1] (log-10 scale)

Table 3 The 7 hyper-parameters that are tuned for Random Forest.

Parameter	Type	Range
Number of trees	ℕ	[5, 150]
Maximal number of features at every split	Dictionary	{auto, sqrt, log2}
Maximal number of levels in trees	ℕ	[2, 40], or (`None` for auto mode)
Minimal number of samples to split a node	ℕ	[2, 20]
Minimal number of samples at leaf node	ℕ	[1, 20]
Function to measure the quality of a split	Dictionary	{gini, entropy}
Usage of bootstrap samples	Boolean	{True, False}

(training with the same hyper-parameters and the training set) can be ignored by averaging. This implies that even a tiny accuracy improvement is only incurred by a better hyper-parameter setting. The results are shown in Figs. 3, 4, 5, and 6.

These results show that the B-EI outperforms all the other methods in most of the evaluated cases w.r.t. the mean prediction accuracy and/or statistical stability. In general, EI-based methods have better performance than NEI-based methods w.r.t. the mean prediction accuracy and statistical stability, which represents that the MLP and RF themselves are noiseless. Although *MODES*-I (I-q(N)EI) shows less competitiveness in classification accuracy, it significantly improves the run-time efficiency, the detailed improvement will be presented in Sect. 6. In addition, when extra overlapped data is added to training data set (i.e., from D1 to D2), the performance of I-qEI improves significantly in most evaluated cases, due to the increased data size and increased similarity of different data sets.

For both MNIST and Fashion-MNIST datasets (Figs. 3 and 4), B-EI shows its advantages if the data size is unbalanced in different nodes, i.e., D3 data sets. However, B-EI performs worse than I-qEI and S-EI for D2 data sets, because: (1) high similarity of data sets in different nodes reduces the influence from tuning the weights of nodes, i.e., simple average in I-qEI already performs well. (2) the increased size of training data allows each single node to train a machine learning model individually with good prediction accuracy.

For Covertype data set, RF outperforms MLP in all the three data splitting strategies. Hence, only the results for RF are analyzed. In both D1 and D2, B-EI performs slightly worse than S-EI, since each node can train a machine learning model with good prediction accuracy based on the relatively easy data set. However, when the size of data sets in different nodes are unbalanced, i.e., D3 data sets, B-EI outperforms all the other methods, by taking the weights of different nodes into consideration.

Since the HAR data set has fewer dimensions than MNIST (562:784), considering the much smaller sample size (1:6), HAR is more difficult to train especially by MLP. In comparison with RF as an ensemble of decision trees, MLPs are attributed to have a higher sensitivity to inputs, which tend to result in a higher risk of deviations in the case of relatively high dimensional and low-sample sized inputs. Specifically, when data size in each node is relatively small (D1), B-EI can better reconstruct the true distribution of HAR data set through the weighted optimization scheme so as to achieve a higher accuracy on test data set, i.e., increase the weights of nodes (learning with bias) that the distribution of data is closer to the true one. However, when the size of each data set increases (D2), the risk of over-fitting decrease (Hastie et al. 2009), i.e., the noise compensated in MBO dominates the optimization to prevent over-fitting. Thus, NEI-based methods outperform EI-based methods on D2 data sets. Note that the nature of data unbalance in D3 leads to a trivial trade-off between the weight and the noise, which results in similar performance of the NEI- and EI-based methods. The similar phenomenon is also discussed in Chan and Hall (2009). In most of the evaluated cases, one of our proposed *MODES* still outperforms the Singles w.r.t. mean prediction accuracy, and show better statistical stability. In contrast, RF shows more robust behavior for HAR data set than MLP does, where the results are promising and similar to what we have observed on the previous data set. In summary, for a great variety of data sets and/or applications without data aggregation, the method *MODES*, with two different modes, outperforms the traditional approach S-EI in terms of either accuracy (*MODES*-B) or run-time efficiency (*MODES*-I) without much accuracy degradation.

6 Scalability and applicability

In order to investigate the scalability of the *MODES*, we evaluated it on the Infinite-MNIST (Loosli et al. 2007) data set with 16 (emulated) nodes. The Infinite-MNIST (also known as MNIST8M) data set produces an infinite supply of digit images derived from the well-known MNIST data set using pseudo-random deformations and translations.

To mitigate the effect of inadequate training samples in each node, e.g., a machine learning model may not be well trained if only small size of training data is available, following the size of MNIST data set used in Sect. 5 (i.e., 60,000 training samples for 4 nodes), we enlarge the size of data set linearly with the same termination condition. That is, only 240,000 training samples in total for both data sets were chosen individually in our experiments. Meanwhile, similar sub-data sets generation strategies are applied: (1) equally divided the training set into 16 parts, denoted as **D1**; (2) each sub-data set from D1 is extended with 5000 samples randomly selected from the remaining samples, denoted as **D2**; (3) divide the training samples unequally, i.e., 8 sets with 5% share and 8 sets with 7.5% share, denoted as **D3**.

The results of the Infinite-MNIST data set are shown in Fig. 7. In general, *MODES*-B outperforms other methods in all the evaluated cases for MLP and most cases for RF. The performance of *MODES*-I for MLP shows large variance, since the key assumption of *MODES*-I, i.e., sub data sets in different nodes have high similarity that the optimal hyper-parameters are similar, does not hold any more for 16 nodes. The difference of optimizing models for different nodes brings significant noise to the *centralized* MBO surrogate model

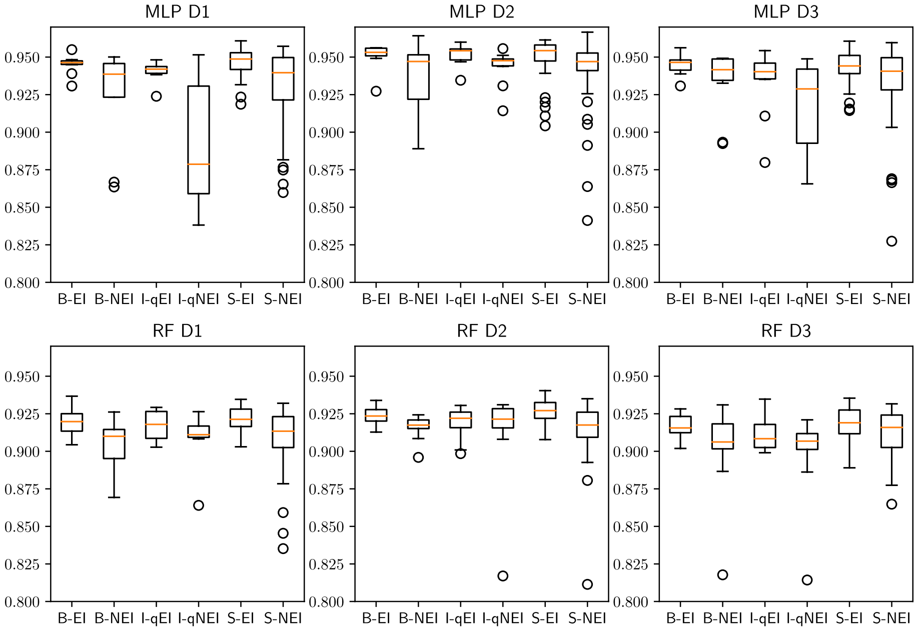

Fig. 3 The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **MNIST** data set

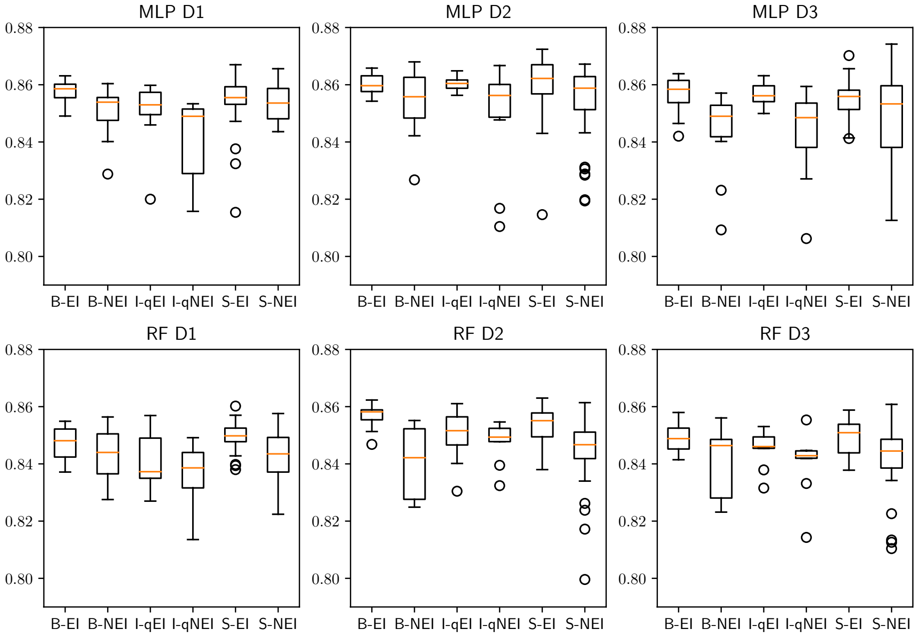

Fig. 4 The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Fashion-MNIST** data set

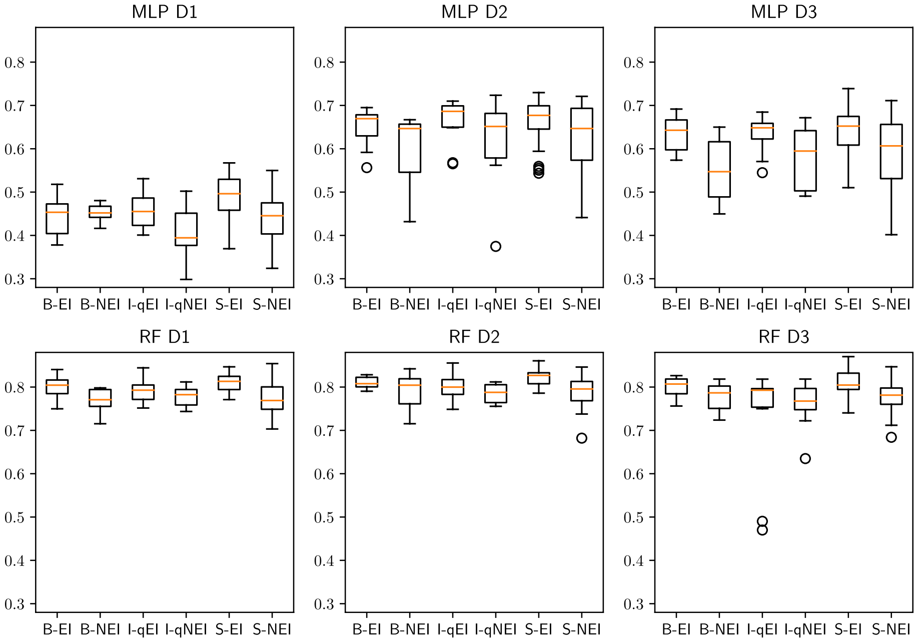

Fig. 5 The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Covertype** data set

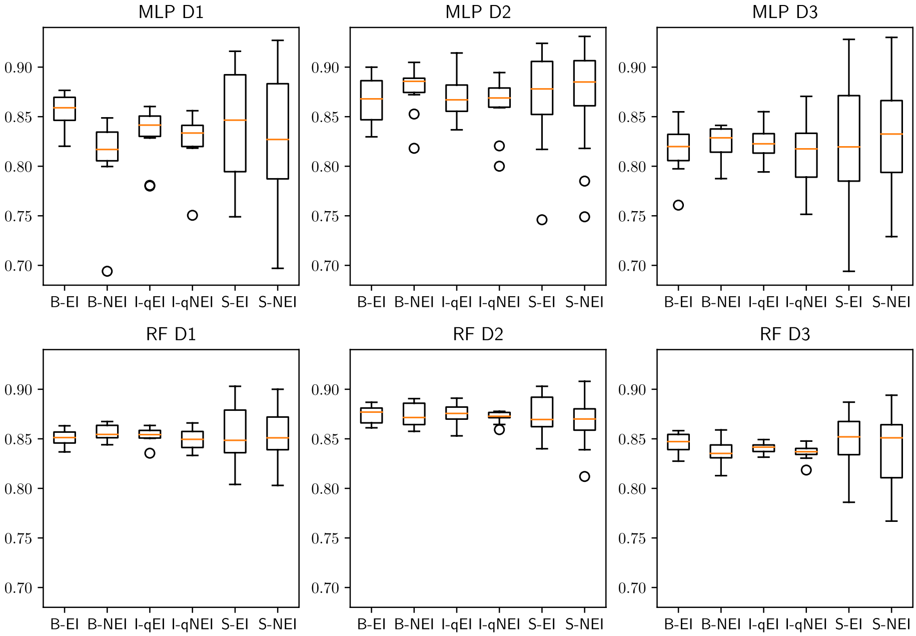

Fig. 6 The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **HAR** data set

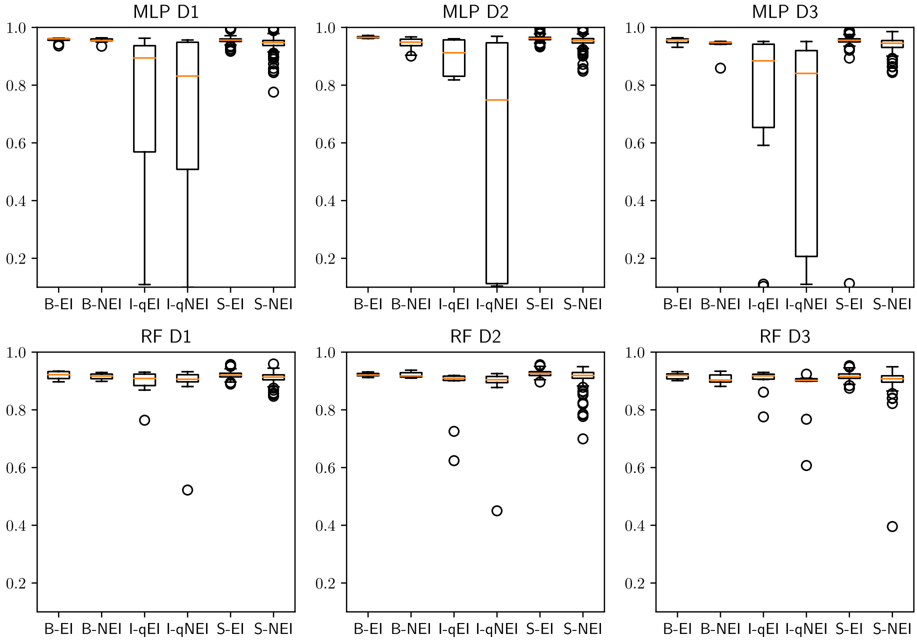

Fig. 7 The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Infinite-MNIST** data set

of *MODES*-I, which can make the tuned hyper-parameters infeasible for the final test set. It also explains why there is an outlier for S-EI on D3 data set. When the similarity of data in different nodes increases, i.e., in D2 data sets, the variance of I-EI reduces significantly. Therefore, *MODES*-B can still work well when the number of nodes increases. However, *MODES*-I can only work well when the applied sub-data sets on each node have certain similarities.

To validate the applicability of *MODES* on distributed embedded systems, we consider a distributed embedded system with four ODROID-N2 boards (https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram/). Each of them integrates a quad-core ARM Cortex-A73 CPU, a dual-core Cortex-A53 CPU, and 32GB storage. The ODROID-N2's DDR4 RAM is running at 1320Mhz with 1.2 Volt low power consumption. All four boards execute the machine learning algorithms (MLP or RF) for a specific task. These nodes are connected with each other, which makes the data transmission possible. The real evaluation shows similar accuracy results as previous figures. The *MODES*-I on a real cluster shows 2.2−3.7 times faster than other methods respectively.

7 Conclusion and future work

In this work, we proposed *MODES*, a novel framework for model-based optimization on distributed embedded systems. Instead of aggregating all the data at a centralized server, *MODES* leverages the local data to obtain the optimized hyper-parameter setting of dedicated machine learning algorithms without any raw data sharing. Specifically, *MODES*-B treats the whole system as a single black box and tunes the hyper-parameters jointly; *MODES*-I treats each node as a copy of the same black box and optimizes the hyper-parameters in parallel. The evaluation on real-world data sets demonstrates that *MODES* outperforms traditional localized MBO in general w.r.t. prediction accuracy and/or runtime efficiency. In future work, we plan to transfer our method to a more powerful platform, i.e., a server-based cluster, to evaluate the performance of *MODES* for more powerful machine learning models with mixed type of hyper-parameters, i.e., containing both continuous and discrete parameters on complex data sets.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Anguita, D., Ghio, A., et al. (2013). A public domain dataset for human activity recognition using smartphones. In Esann

Baek, O. K. (2011). Data-centric distributed computing. US Patent 8060464.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2020). BoTorch: A framework for efficient Monte–Carlo Bayesian optimization. In *Advances in neural information processing systems*

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research.*

Bergstra, J., Yamins, D.,&Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML* (vol. 28, pp. 115–123), Atlanta, Georgia, USA, 17–19 June PMLR.

Berk, J., Nguyen, V., Gupta, S., Rana, S., & Venkatesh, S. (2018). Exploration enhanced expected improvement for Bayesian optimization. In *Machine learning and knowledge discovery in databases—ECML/PKDD proceedings, volume 11052 of lecture notes in computer science* (pp. 621–637). Springer.

Bian, J., Xiong, H., Fu, Y., & Das, S. K. (2018). Cswa: Aggregation-free spatial-temporal community sensing. In *AAAI conference on artificial intelligence* (pp. 2087–2094).

Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., & Lang, M. (2017). mlrMBO: A modular framework for model-based optimization of expensive black-box functions. arXiv:1703.03373 [stat]

Blackard, J. A., & Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture, 24*(3), 131–151.

Buschjager, S., Chen, K.-H., Chen, J.-J., & Morik, K. (2018). Realization of random forest for real-time evaluation through tree framing. In *ICDM*, IEEE.

Chan, Y.-B., & Hall, P. (2009). Scale adjustments for classifiers in high-dimensional, low sample size settings. *Biometrika, 96*(2), 469–478.

Claeskens, G., Hjort, N. L., et al. (2008). *Model selection and model averaging*. Cambridge: Cambridge Books.

Coy, M. A. R., Rehbach, F., Eiben, A. E., & Bartz-Beielstein, T. (2020). Parallelized Bayesian optimization for problems with expensive evaluation functions. In Coello, C. A. C. (ed.), *GECCO* (pp. 231–232). ACM.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML* (pp. 1050–1059). PMLR.

Gardner, M. W., & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—A review of applications in the atmospheric sciences. *Atmospheric Environment, 32*(14–15), 2627–2636.

Garg, A., Saha, A. K., & Dutta, D. (2020). Direct federated neural architecture search. arXiv:2010.06223

Ginsbourger, D., Le Riche, R., & Carraro, L. (2010). Kriging is well-suited to parallelize optimization. In *Computational intelligence in expensive optimization problems* (pp. 131–162). Springer.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems* (pp. 2348–2356). Citeseer.

Gu, Y., Do, H., Ou, Y., & Sheng, W. (2012). Human gesture recognition through a kinect sensor. In *ROBIO* (pp. 1379–1384). IEEE.

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation, 9*(2), 159–195.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer.

He, C., Annavaram, M., & Avestimehr, S. (2020). Fednas: Federated deep learning via neural architecture search. arXiv:2004.08546

Hutter, F., Hoos, H., & Leyton-Brown, K. (2013). An evaluation of sequential model-based optimization for expensive blackbox functions. In *GECCO* (pp. 1209–1216).

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *LION*. Springer.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2012). Parallel algorithm configuration. Number 7219 in lecture notes in computer science. In Y. Hamadi & M. Schoenauer (Eds.), *Learning and intelligent optimization* (pp. 55–70). Springer.

Janusevskis, J., Le Riche, R., Ginsbourger, D., & Girdziusas, R. (2012). Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges. In *LION*. Springer.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization, 13*(4), 455–492.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492

Kotthaus, H. (2018). Methods for efficient resource utilization in statistical machine learning algorithms. Ph.D. thesis, Technical University of Dortmund, Germany

Kotthaus, H., Richter, J., Lang, A., Thomas, J., Bischl, B., Marwedel, P., et al. (2017). RAMBO: Resource-aware model-based optimization with scheduling for heterogeneous runtimes and a comparison with asynchronous model-based optimization. Lecture notes in computer science. In *Learning and intelligent optimization* (pp. 180–195). Cham: Springer.

Kotthaus, H., Schönberger, L., Lang, A., Chen, J., & Marwedel, P. (2019). Can flexible multi-core scheduling help to execute machine learning algorithms resource-efficiently? In *SCOPES* (pp. 59–62). ACM.

Kriegel, H.-P., Schubert, E., & Zimek, A. (2017). The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *KAIS, 52*(2), 341–378.

LeCun, Y., Cortes, C., & Burges, C. J. (1998). The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9–48). Springer.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., et al. (2011). Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)* (pp. 163–168). IEEE.

Li, L., Xiong, H., Wang, J., Xu, C.-Z., & Guo, Z. (2019). Smartpc: Hierarchical pace control in real-time federated learning system.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R News, 2*(3), 18–22.

Loosli, G., Canu, S., & Bottou, L. (2007). Training invariant support vector machines using selective sampling. In L. Bottou, O. Chapelle, D. DeCoste, & J. Weston (Eds.), *Large scale kernel machines* (pp. 301–320). Cambridge: MIT Press.

Nijssen, S., & Kok, J. (2006). Frequent subgraph miners: Runtimes don\'t say everything. In *Proceedings of the workshop on mining and learning with graphs* (pp. 173–180).

ODROID-N2. https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram/. Retrieved October 25,2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research, 12,* 2825–2830.

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *ICML* (pp. 1278–1286). PMLR.

Richter, J., Kotthaus, H., Bischl, B., Marwedel, P., Rahnenführer, J., & Lang, M. (2016). Faster model-based optimization through resource-aware scheduling strategies. In *Learning and intelligent optimization* (pp. 267–273). Springer.

Shi, J., Bian, J., & Richter, J. (2021). Model-based optimization on distributed embedded system. https://github.com/Strange369/MODES-public

Singh, I., Zhou, H., Yang, K., Ding, M., Lin, B., & Xie, P. (2020). Differentially-private federated neural architecture search. arXiv:2006.10559

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951–2959).

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

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747

Xing, E. P., Ho, Q., Dai, W., Kim, J. K., Wei, J., Lee, S., et al. (2015). A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data, 1*(2), 49–67.

Zhu, H., & Jin, Y. (2020). Real-time federated evolutionary neural architecture search. arXiv:2003.02793

Authors and Affiliations

Junjie Shi[1] · Jiang Bian[2,3] · Jakob Richter[4] · Kuan-Hsun Chen[1] · Jörg Rahnenführer[4] · Haoyi Xiong[2] · Jian-Jia Chen[1]

Jiang Bian
bjbj11111@knights.ucf.edu

Jakob Richter
richter@statistik.tu-dortmund.de

Kuan-Hsun Chen
kuan-hsun.chen@tu-dortmund.de

Jörg Rahnenführer
joerg.rahnenfuehrer@uni-dortmund.de

Haoyi Xiong
xhyccc@gmail.com

Jian-Jia Chen
jian-jia.chen@cs.uni-dortmund.de

[1] Department of Computer Science, TU Dortmund University, Dortmund, Germany

[2] Big Data Lab, Baidu Inc., Beijing, China

[3] Department of Electrical and Computer Engineering, University of Central Florida, Orlando, USA

[4] Department of Statistics, TU Dortmund University, Dortmund, Germany