EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

# MFRLMO: Model-free reinforcement learning for multi-objective optimization of apache spark

Muhammed Maruf Öztürk[1,*]

[1]Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Suleyman Demirel University, West Campus, Isparta, 32040, Turkey

## Abstract

Hyperparameter optimization (HO) is a must to figure out to what extent can a specific configuration of hyperparameters contribute to the performance of a machine learning task. The hardware and MLlib library of Apache Spark have the potential to improve big data processing performance when a tuning operation is combined with the exploitation of hyperparameters. To the best of our knowledge, the most of existing studies employ a black-box approach that results in misleading results due to ignoring the interior dynamics of big data processing. They suffer from one or more drawbacks including high computational cost, large search space, and sensitivity to the dimension of multi-objective functions. To address the issues above, this work proposes a new model-free reinforcement learning for multi-objective optimization of Apache Spark, thereby leveraging reinforcement learning (RL) agents to uncover the internal dynamics of Apache Spark in HO. To bridge the gap between multi-objective optimization and interior constraints of Apache Spark, our method runs a lot of iterations to update each cell of the RL grid. The proposed model-free learning mechanism achieves a tradeoff between three objective functions comprising time, memory, and accuracy. To this end, optimal values of the hyperparameters are obtained via an ensemble technique that analyzes the individual results yielded by each objective function. The results of the experiments show that the number of cores has not a direct effect on *speedup*. Further, although grid size has an impact on the time passed between two adjoining iterations, it is negligible in the computational burden. Dispersion and risk values of model-free RL differ when the size of the data is small. On average, MFRLMO produced *speedup* that is 37% better than those of the competitors. Last, our approach is very competitive in terms of converging to a high accuracy when optimizing Convolutional Neural networks (CNN).

Received on 04 January 2024; accepted on 19 February 2024; published on 20 February 2024

**Keywords:** Spark, configuration tuning, multi-objective optimization, reinforcement learning



doi: 10.4108/eetsis.4764

## 1. Introduction

Apache Spark is a prevalent distributed data processing platform that provides various machine learning algorithms to interpret raw data. Generally, machine learning [1, 2], fog computing [3], event detection [4], and interactive analysis [5] are the foremost application areas of Spark. Optimization is key to understanding the underlying mechanism of parameters of Spark providing data processing and machine learning methods that enable us to plan computational sources. In order to obtain the best performance in Spark, it is necessary to seek the ultimate configuration of the parameter set that is determined before the execution and changes depending on the type of data set. That parameter set is called hyperparameter configuration.

Spark is not only optimized for making the use of time and hardware sources more efficient, but for obtaining reliable measurement methods that evaluate a machine learning model. Therefore, instead of relying on a single objective function, multi-objective function groups should be preferred in HO. To configure the parameters of Spark, there are various ways including choosing from a machine learning group [6], heuristic methods [6], cost estimation models [7], and utility-based designs [8] which have been commonly applied to

*Corresponding author. Email: muhammedozturk@sdu.edu.tr

this field. The major drawback in this field is the lack of dynamic models considering the internal states of HO rather than employing traditional black-box approaches in the tuning of Spark. As such, the HO performed regarding the interaction of sequential tiny tasks is more feasible than establishing optimization models [9] along with a single objective.

There are few works proposing HO with multi-objective designs [8, 10]. Although these works optimize Spark with more than one objective, they were not able to combine three or more objective functions. Further, generally, the objective functions are constructed in relation to the system's performance. In that case, there is a greater risk that the reliability of evaluation of the performance of sub-jobs of Spark degrades gradually. Therefore, traditional methods obtain the optimal hyperparameter configuration limited to accuracy and execution time.

In the last decade, random search [11], grid search [12], bayesian optimization [13], and evolutionary methods [14] have been commonly applied to HO problems. However, none of them always can yield the best solution for every type of HO problem due to their design constraints. Instead, there is a possible solution that a choice is made regarding problem-specific constraints and the size of experimental data. For instance, though grid search is an exhaustive algorithm, it takes a lot of time compared to the alternatives. On the other hand, although Bayesian optimization performs relatively fast in HO, it poses a significant threat that some of the optimal results may be ignored. Since Spark processes resilient distributed data (*RDD*) and divides the problem into small job tasks, there is a need to develop a sophisticated HO technique to reveal the internal dynamics of big data processing.

While RDD provides explicit parallelism, it brings two critical issues as follows: 1) Since the Spark kernel divides small jobs into threads, it precisely hardens to establish a second parallel architecture. However, a second parallelization is only possible thanks to an asynchronous execution [15]. 2) The objective functions are prepared based on speedup. Therefore, increasing the types of objective functions becomes almost impossible.

There are two common targets in tuning parameters of Spark: storage and execution improvements. While configuring some operations like sort and shuffle means coping with execution issues, storage problems are focused on the caching parameters of HO. Besides, the parameters associated with shuffle and partition are of great importance to achieve optimal performance in the big data processing. In rule-based tuning, a user defines the execution strategies and the performance is improved without applying expert knowledge in Spark [16]. But that operation is very effort intensive and it is very difficult to find the optimal configuration hyperparameter set when there is not any expert opinion. It is thus immensely challenging to find the best combination of a set of specific hyperparameters.

To yield minimum execution time, the values of hyperparameters can be represented as bytecode definitions [17]. However, if the size of the search space is very large, search time increases tremendously. In order to shorten that time, the less effective hyperparameters can be ignored in small iterations [18]. However, the optimal hyperparameter set may be lost if the threshold of trial execution is not set well. Therefore, detecting the most effective hyperparameters in HO is an alternative way to ignoring the less effective ones. The target hyperparameter set is produced leveraging different log files of the execution thanks to a block diagram [19]. In this way, some of the hyperparameters constituting the search space are very difficult to be held out of scope in which there are a great number of setting options. In adaptive methods, resource sharing [20] is reconsidered in each iteration of HO and the changes in workload are observed [21–23]. However, the possibility of adverse effects of concurrent workloads arises from online systems. Instead, the optimal performance can be achieved by leveraging the hyperparameters detected during the training of machine learning methods [16, 24]. But that approach is a black-box model that is strongly related to the internal settings of Spark and hardware. Retrieving training data is very costly in machine learning-based models and it necessitates different execution trials to avoid misleading results. If we define the basic execution steps of Spark as a group of interrelated processes, reinforcement learning (RL), which is one of the popular machine learning techniques, quite fits that definition. RL traces the major principles of the Markov Decision Process (MDP) and it has been tested thanks to a Factor Analysis, thereby utilizing a dynamic configuration that makes reward values more informative in terms of performance metrics [25]. But multi-objective optimization is not considered in that study. Metareasoning techniques are of great importance for deciding optimal stopping criteria and they are very useful to plan HO. They have been tried to increase the comprehensiveness of the approaches achieving optimal solutions using quality and time features [26, 27]. Despite the fact that metareasoning techniques could have yielded promising results in robot path planning and classification issues, interpreting model-based techniques is very difficult when the environment dynamics are quite complex. The aforementioned studies have some drawbacks as follows:

1. They are mostly designed with objective functions based on *speedup*, 2. Seeking the optimal hyperparameters is not easy when the search space is very large, 3. The training time increases if the features of Spark are involved in machine learning processes, 3.

The models, which are developed for HO, are designed by disregarding the internal dynamics of Spark along with a black-box approach. Therefore, the relationship between the configuration of the hyperparameter set and the obtained performance can not be explicitly defined.

The disadvantages of existing works lead to a waste of computational resources or result in longer training times. This study aims at developing a multi-objective HO method, thereby utilizing the major principles of model-free reinforcement learning [28]. In this way, we provide new insight on how internal dynamics of Spark affect HO efficiency thanks to a white-box approach. The method produces optimal hyperparameters for three objective functions: memory, time, and accuracy thanks to an ensemble technique. Hence, the tradeoffs are found for three objective functions. To this end, a dynamic set of hyperparameters modeled with 14 grids, is retrieved from a search space created with grid search. Thereafter, the HO is completed with the ultimate rewards updated in each iteration of RL. Before configuring the hyperparameters of $ml\_multilayer\_perceptron\_classifier$, Direct Search is performed for binary classification data sets. One of the distinctive properties of the proposed method is that it does not include the properties of the environment in the training data thanks to the model-free design. In this respect, the computational burden of determining which feature does contribute to the performance of HO is alleviated. Multi-objective design is a requirement for hyperparameter optimization since traditional optimizers generally choose one target that does not seek a tradeoff between various purposes.

The study claims the contributions as follows: 1. Unlike the literature, MFRLMO is the first model-free adaptive HO method developed for Spark,
2. MFRLMO is devised not just for tuning machine learning algorithms, it can also configure the hyperparameters of HiBench benchmarks,
3. When comparing MFRLMO with the-state-of-the-art methods, It could produce the highest $speedup$,
4. If MFRLMO is used for tuning CNN, Accuracy is improved up to 150 iterations in the observation of training,
5. The execution memory and the number of cores are the most effective hyperparameters for MFRLMO.
6. Compared to the state-of-the-art, MFRLMO is much more suitable for large-scale tuning setups when the limit of budget HO is not low.

The rest of the study comprises six sections. Section 2 summarizes the related works. The method is elucidated in Section 3. The details of the experiment are presented in Section 4. Section 5 discusses the results obtained from the comparison methods. The paper is concluded in Section 6.

## 2. Related works

Robotics manipulation is one of the real-world application areas of RL. In particular, RL is utilized for the problems encountered during the guidance of moves of robots. Generally, robot navigation studies are focused on two approaches: heuristics and learning-based techniques. Concretely, RL is very useful for collecting or squeezing some objects with robots. In the case of a limited budget, the constraints should be regarded in real-time scenarios [29]. RL may be combined with traditional control schemes for unprecedented circumstances. In this context, the results produced in the simulation environment are very promising for real-world problems with such techniques [30]. Cooperative exploration strategies help reduce the time allocated for training RL [31]. Hence, the robots move more reliably in which there is little or no information about the obstacles. Oliff et al. investigated the benefits of RL in manufacturing techniques [32]. The optimal policy achieved with the changes of conditions of the environment may lead to faster degradation of performance. In the meantime, RL has been tested for the coordination multi-robots [33, 34]. To that end, the data trained with CNN and the instances collected with RL consist of the elements of the experiment. Although simulation environments contribute to finding solutions for real-world problems, RL requires sophisticated HO methods to improve the reliability and the generalizability of training performance. If RL is employed for finding ways to robots in a cluttered terrain, the time and range spent for reaching the target can be reduced. However, diversifying the conditions of the environment chosen for testing robots enhances the generalizability of the RL techniques [35].

Game theory is an active research field that utilizes RL. Model-based RL is commonly applied in game theory due to the use of environmental features. The policy parameter of RL is optimized to support the models of game theory [36]. The coalition formula of game modeling is devised for reducing the error of convergence. To this end, approximated RL is one of the solutions for fixing problems that may arise in training. The anticipation of levels of opposed agents in game theory has unresolved issues. For instance, how to proceed with the interaction of agents having various levels is a complex process and it requires coping with a high computational demand [37]. Establishing a learning-based scheme for mobile social networks without knowing the details of network parameters may lead to security deficiencies. To solve that problem, a novel Q-learning-based edge caching strategy was designed [38]. Despite the fact that the method tested on the Stackelberg game has shown resistance to degradation of the quality of caching in the increase of

the number of cost parameters, it should be combined with special techniques that have the potential to reduce training time. Unifying game theory with deep learning resulted in a remarkable increase in the success of simulation-based anticipation studies [39]. However, cross-validation may be tried to avoid overfitting when there are independent experimental data sets. Ahad et al. [40] utilized RL to choose a way to 5G-based network packet transmission. RL helped to reduce energy consumption during the transfer of data. Bui et al. [41] proposed an RL-based technique to plan the budget of trading costs in grid systems. They stressed that the number of episodes of RL should be high to compare the congestion of internal trading and external trading.

If we want continuous improvement with respect to the feasibility of chosen values of hyperparameters, RL can be considered as an auxiliary method. In Dong et al.'s work, CNN-based algorithms, which trace target objects defined by a user, were tuned with RL and the results of it were compared with those of the default configuration. Specifically, RL contributed to the acceleration of object tracking. If the use of the model is driven dynamically, the complexity of the related algorithm increases depending on the iteration of RL [42]. In such cases, model-free RL is an alternative way to alleviate the computational burden posed by a high number of iterations. Preceding works revealed that if RL is preferred in HO, the budget allocated for computing is reduced [43]. But if the features of the environment are not involved in the training data set, a high number of training iterations is needed. RL was investigated in the multi-objective optimization problem of image classification data to improve the success of CNN [44]. Some performance metrics such as accuracy are of vital importance as to which hyperparameter set is optimal or not. For instance, latency refers to the time passed between two consecutive iterations. Although [44] was able to achieve significant success for these two objectives, ignoring the consumption of memory, which comprises the scalarization function of reward, limits the generalizability of results.

## 3. Background

In this section, basic definitions and proofs are presented along with the formulations explaining the relationship between HO and RL. We then elaborate on the structural properties of model-free reinforcement learning that differentiates it from model-based approaches.

### 3.1. Hyperparameter optimization

Let $SS$ be a search space representing the hyperparameters of Spark and MLlib. Assuming $h_1, h_2, ..., h_m$ is the optimal hyperparameter configuration chosen from that search space, there are $m$ number of hyperparameters. Applying one function to a search space $f \rightarrow SS$ means performing single objective optimization. $f$ may be related to classification accuracy, memory consumption, and time. On the other hand, a set of functions $f \rightarrow SS$ denotes multi-objective optimization and should satisfy $n \geq 2$. $f$ decides whether the optimization meets the minimization or maximization criteria. For instance, if the objective function is associated with time, minimization is preferred in satisfying the criterion.

**Definition 1.** Hyperparameter: Let $Tr$ be training, $V$ be validation, and $T$ be testing, hyperparameter is a parameter applied between $hf \rightarrow S$ and $Tr$, thereby observing the effects of it are observed in $V$ along with the results at $T$.

**Definition 2.** Search space: $S$ is a set of values obtained from experiences gained in the preceding studies of HO.

### 3.2. Model-free reinforcement learning

Contrary to model-based learning, model-free RL does not require the use of the agent, thereby modeling the details of the environment. Instead, new actions are decided by controlling the rewards produced in the old episodes. Since there is no modeling of the environment, the computational cost of feature selection is alleviated as well. However, to achieve the ultimate solution, model-free RL necessitates a great number of training iterations. As such, unlike black-box approaches, a distinct mechanism is adapted for improving traditional HO techniques.

Assuming an agent is associated with action $A_t$, state $S_t$, and reward $R$ as sequentially at discrete time-points $t = 0, 1, 2, 3, ..., S_1A_1R_1, S_2A_2R_2, S_3A_3R_3, ....$ is produced. When the learning process is formulated with Markov Decision Process (MDP), the transition function is denoted with a tuple $(S, A, R, \sigma)$ in which $\sigma : SxAxR \rightarrow S$ generates a new state in the episodes. A probabilistic policy manages the agent for the states $p : S \rightarrow A$ that $S - A$ is yielded as in Equation 1. Those are called Q-values.

$$Q_p(S_2, A_2) = E_p(\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = S_2, A_0 = A_2) \qquad (1)$$

where $\gamma$ is the discount factor quantifying how much importance will be given for future rewards.

**Definition 3.** Exploitation: is a mathematical indicator showing agents have produced the best results up to the current episode.

**Definition 4.** Exploration: is inversely proportional to exploitation and refers to long term to find the optimal strategy.

The balance between exploitation and exploration is determined with $\epsilon$ and $\epsilon = 0.3$ means that 30% of the total actions are reserved for exploration and the rest of them are used for exploitation.

**Theorem 1.** New Q-values can be formulated with a constant $a \in (0,1)$ as $(1-a)Q_1 + \sum_{i=1}^{n} a(1-a)^{n-i}R_i$ where $n$ is the number of iterations. In the weighted average, $(1-a)^n + \sum_{i=1}^{n} a(1-a)^{n-i} = 1$.

**Proof.** The weight $a(1-a)^{n-i}$ given in Theorem 1 depends on the time when $R_i$ is produced. $1-a$ is less than 1 in which $R_i$ decreases as the number of rewards increases. So that summing $R_i \rightarrow 0$ and $(1-a)$ goes to 1.

**Theorem 2.** Given a state set $S_1, S_2, ..., S_u$ and action set $A_1, A_2, ..., A_v$, $v \leq= u$ in which a gridworld environment is defined with limited computation resources.

**Proof.** Assume the number of training=50, the number of states=y, and the number of actions=z. For 1000 bootstrapping random walks, state transitions overlap at least $x/(z^y - y^y)$. For (x=50, y=3, z=4), the overlapping occurs 2.85 times. To prevent that problem, the number of actions should be tried to make as large as possible.

**Theorem 3.** If $\epsilon - greedy$ is applied to find a tradeoff between exploration and exploitation, $\epsilon = (1/x) + v$ is a general formula for $\epsilon = v$ in which $x$ is the number of iterations.

**Proof.** In increasing iterations, $1/x \rightarrow 0$ for $x \rightarrow \infty$. For instance, if $v$ is 0.2, the left part of the formula goes to 0 after a specific number of iterations and $v$ becomes the ultimate value.

## 4. MFRLMO

The main steps of MFRLMO are given in Figure 1. Different from model-based approaches, it manages the HO process according to the results of action-reward interactions. In that process, the experimental data is converted to RDD after feature selection. Once the cross-validation partitions are generated, a dynamic cell update is done in RL depending on the obtained reward values. Subsequently, the reliability of the RL model is evaluated with dispersion and risk analyzes. In the last step, the success of optimal configuration is interpreted with various performance parameters such as accuracy.

MFRLMO comprises the details of a definition process since the experimental data is related to a Spark connection object that metadata contextual information including start time, identifiers, and session. The steps of MFRLMO are presented in Algorithm 1. The label column (0/1) of classification data sets is converted to factor values (true/false) in Step 1 thanks to the *Preprocessing* function. In Step 2, the experimental data sets are exposed to feature selection with Direct Search algorithm that works as follows: 1) Determines a $K$ value that shows the number of features to be

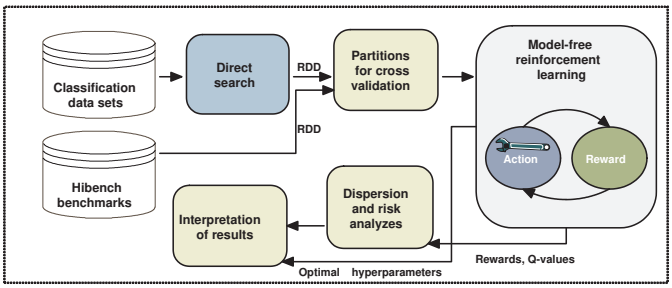



**Figure 1.** Overview of the proposed RL-based approach.

chosen, 2) $K$ features constitute the ultimate feature set, thereby employing a threshold value that eliminates some features. To perform feature selection, the formula given in Equation 2 is applied to the data sets.

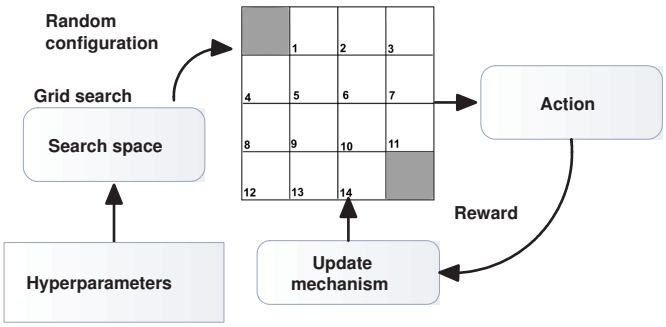



**Figure 2.** Dynamic update mechanism of 4x4 grid in model-free RL.

$$R^2 = 1 - \frac{SS_r}{SS_t}[45] \tag{2}$$

where $R^2$ is a coefficient used for continuous features. While $SS_r$ refers to the sum of the square of residuals, $SS_t$ is the sum of squares. Direct Search has been executed with the help of $FSinR$ library of R package [46]. Step 3 adds new form of data to Spark connection and Step 4 $D_2$ is divided into 10 parts to assign $D_3$ in 10x10 cross-validation. *hss* is composed of some values retrieved from a search space generated by grid search. *count* given in Step 5 refers to the number of updates of 4x4 grid *env*. Initially, the cells of the 4x4 grid are randomly generated values from hyperparameter search space. They are renewed checking rewards obtained from consequent actions. For instance, let (0.1,0.3,0.4) be a hyperparameter tuple of a hyperparameter set of a specific cell. If a remarkable increase is observed in the reward after that tuple is updated as (0.1,0.4,0.4), (0.1,0.5,0.4) is taken from the search space generated with grid search to update the RL grid having 14 cells. Hence, a dynamic grid is defined as model-free RL. The update mechanism of the 4x4 grid is

presented in Figure 2. Steps 8-14 change the rules of *env* depending on the yielded rewards. The herein described objective functions include three types: *sparkAccu*, *sparkMem*, and *sparkTime*. *sparkAccu* is the success of *ml_multilayer_perceptron_classifier* in terms of accuracy and it is calculated leveraging the elements of the confusion matrix as given in Equation 3. The optimal value gives the highest reward at the end of 50 iterations that were determined after conducting various trials on the experimental data. On the contrary, the values representing the lowest rewards are the optimal configuration of hyperparameters for *sparkMem* and *sparkTime*. This is because they are associated with memory and time in which a maximization-based model is established. In Step 14, the rewards are saved depending on the random progressive 1000 transitions. Step 15 defines learning rate *alpha*, discount factor *gamma*, and $\epsilon$. Since the model is established relying on a large number of iterations of actions, *alpha* is kept to a minimum. The exploration potential of the agent may be limited due to a high *alpha* in a modeled environment. Step 16 records $Q$ values depending on the state-action interaction for 50 iterations of 14 grids. In Steps 17-18, the mean hyperparameter values of three objective functions are calculated along with the rewards. Step 20 returns the average reward and optimal hyperparameter configuration.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Being able to measure the time difference on two grid types, a mini-experiment is performed for various iterations. Time-iteration results given in Figure 3 show that the type of grid is negligible although the difference is clear after 200 iterations. The more cells we have, the more design of HO is updated so that a 4x4 grid is preferred in the experiment. Notably, the trends observed in the two grid types are quite similar in the way they rise and fall.

## 5. Experiment

In the experiment, the structure of 4x4 gridworld is given in Figure 4. Here shaded cells can be called terminal states that halts the iterations. Otherwise, four actions including "up", "down", "right", and "left" are considered for each state. If Accuracy is achieved above 97% optimal configuration of hyperparameters is returned by Algorithm 1 regardless of we have attained the end of iterations.

### 5.1. Settings

The machine used for establishing the experiment has been configured as able to run Spark and Yarn

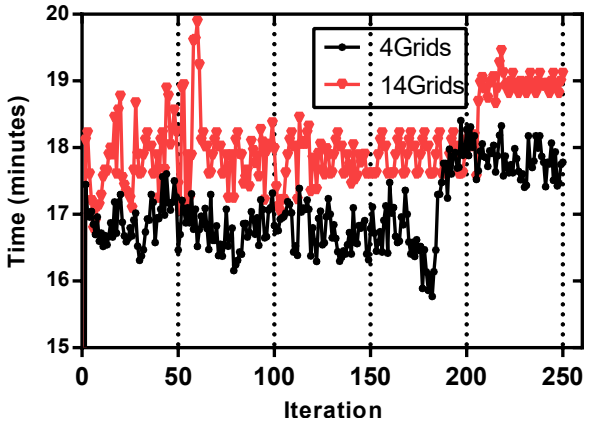
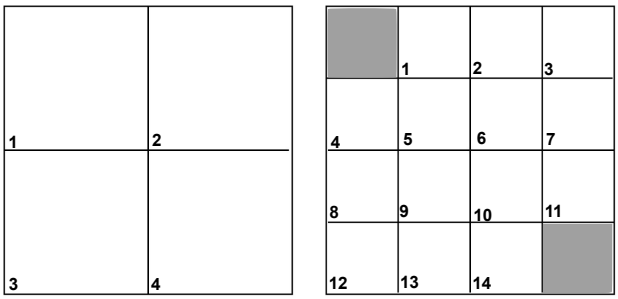



**Figure 3.** 2x2 gridworld vs. 4x4 gridworld for increasing iterations.




(a) 2x2 grid (b) 4x4 grid

**Figure 4.** Design properties of 2x2 gridworld and 4x4 gridworld.

along with the following features: CentOS Linux, 64-bit, Intel(R) Xenon(R) 2.9 GHz, 24 CPU Cores server with 222 GB RAM. To generate a Yarn cluster, the machine has allocated four workers each of which has four cores. In total, 30GB of memory has been spent on the Yarn experiment for each worker. The algorithm and the details of the experiment have been coded with R version 4.0.2.

### 5.2. Search space

The search space of the experiment is generated with the hyperparameters of *ml_multilayer_perceptron_classifier* and the parameters of Spark. Although MLlib can run a great number of algorithms featuring supervised and unsupervised learning, the chosen algorithm for classification is the feedforward artificial neural network constituting the fundamentals of deep learning. The hyperparameter *layers* is not involved in the tuning process. The reason is that the number of features does not change after feature selection. Further, it is not possible to change the number of features in the output layer since the type of classification is binary. In the first layer,

**Algorithm 1** MFRLMO algorithm

**Input:** Classification and benchmark data sets $D$, hyperparameter search space $hss$, spark connection $sc$
**Output:** Optimal hyperparameter configuration

$D_2 \leftarrow Preprocessing(D)$
$D_2 \leftarrow directSearch(D_2)$ ▷ Feature selection
$D_3 \leftarrow copy\_to(sc, D_2)$
$partitions \leftarrow createFolds(D_3)$
$count \leftarrow 50$
$env \leftarrow gridWorldEnvironment$
$generalResult \leftarrow list("reward","optimalHyperparameter")$
**while** count≥1 **do**
    $Accuracy \leftarrow sparkAccu(partitions, hss)$
    $memResult \leftarrow sparkMem(partitions, hss)$
    $timeResult \leftarrow sparkTime(partitions, hss)$
    $count \leftarrow count - 1$
    $env \leftarrow updateReward(env, Accuracy, memResult, timeResult)$
    $data \leftarrow sampleExperience(env)$ ▷ Sample N = 1000 random sequences from the environment
    $control \leftarrow list(alpha = 0.1, gamma = 0.1, epsilon = 0.2)$
    $model \leftarrow ReinforcementLearning(data, control, Q_p(S_2, A_2) = E_p(\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = S_2, A_0 = A_2))$
    $averageReward \leftarrow getEnsembleReward(model)$
    $meanHyperparameter \leftarrow getEnsembleHyperparameter(model)$
    $generalResult \leftarrow generalResult \cup (averageReward, meanHyperparameter)$
**return** $generalResult$ ▷ Optimal hyperparameter configuration

all the features are set as inputs. For instance, if we have four features, the size of layers is set to $layers = (4, 3, 2)$. The configurable hyperparameters of $ml\_multilayer\_perceptron\_classifier$ and parameters of Spark are presented in Table 1. Moreover, the hyperparameters of CNN are given in that table since one of the comparison methods was tested on CNN using image classification data sets. Therefore, the settings of the range of hyperparameters have been set regarding the baselines. Even though Spark has a large number of configurable parameters, an optimal hyperparameter group has been created considering pioneering studies [10, 47]. On the other hand, since multi-objective optimization takes a lot of time compared to single-objective optimization, the number of hyperparameters has been kept to a minimum to alleviate the computational burden. 17 hyperparameters and parameters of Spark are subjected to HO. The hyperparameter ranges have been set as large as possible due to the need for high iterations in model-free RL. As such, employing constantly changing cells of the grid has bolstered the comprehensiveness of the hyperparameter configurations.

## 5.3. Data sets

The data sets presented in Table 2 have three types: Spark benchmark, classification, and Keras. After executing Apache Hadoop YARN, the benchmarks of HiBench are included in the experiment to evaluate the success of Algorithm 1 in terms of resource management. The second type consists of classification data sets that are suitable to be exposed to the algorithms of MLlib. Image processing data sets have been retrieved from the Keras library of R packages [48].

WordCount is the largest benchmark that was designed for counting a word in a text corpus. TeraSort performs sorting on big data leveraging Hadoop. Kmeans is an iterative clustering algorithm that uses some distributions such as Gaussian and it is available at MLlib. Bayes is also a multi-label classification algorithm that works via MLlib. That algorithm is involved in the experiment since it is based on various mathematical inference techniques. *Dense* was originally devised for developing neural network models in classification [1]. *Microsoft* consists of instances collected from a security intrusion contest and it has 1805 features [2]. *Payload* data set, which has 32 features, was generated from a research project conducted by Politecnico di Milano University [3]. *Santander* data set includes completely numeric values comprising 199 features that were retrieved from customer transaction prediction [4]. *Payload* can be accessed through Zenodo open source code sharing

[1] https://www.kaggle.com/c/dense-network/data?select=train.csv
[2] https://www.kaggle.com/muhammad4hmed/malwaremicrosoftbig
[3] https://zenodo.org/record/5731597
[4] https://www.kaggle.com/datasets/lakshmi25npathi/santandercustomer-transaction-prediction-dataset

**Table 1.** Search spaces of hyperparameters and spark parameters.

| Type | Name | Description | Range | Step size |
|---|---|---|---|---|
| *ml_multilayer_perceptron_classifier* | max_iter | maximum number of iterations | 5-100 | 1 |
| *ml_multilayer_perceptron_classifier* | step_size | step size of optimization | 1-9 | 1 |
| *ml_multilayer_perceptron_classifier* | tol | convergence tolerance of algorithm | (1e-06)-(1e-02) | 0.0001 |
| Apache spark | spark.task.cpus | Number of cores to allocate for each task | 1-3 | 1 |
| Apache spark | spark.executor.cores | number of cores to use on each executor | 1-24 | 1 |
| Apache spark | spark.executor.memory | memory to use per executor process (GB) | 1-130 | 1 |
| Apache spark | spark.memory.fraction | a fraction used for execution and storage | 0.2-0.8 | 0.1 |
| Apache spark | spark.memory.storageFraction | storage memory immune to eviction | 0.2-0.8 | 0.1 |
| Apache spark | spark.default.parallelism | number of partitions in RDDs returned by transformations like join, reduceByKey, and parallelize when not set by user | 20-400 | 10 |
| Apache spark | spark.shuffle.compress | whether to compress map output files | true/false | - |
| Apache spark | spark.shuffle.spill.compress | whether to compress data spilled during shuffles | true/false | - |
| Apache spark | spark.broadcast.compress | whether to compress broadcast variables before sending them | true/false | - |
| Apache spark | spark.rdd.compress | whether to compress serialized RDD partitions | true/false | - |
| Apache spark | spark.io.compression.codec | codec used to compress internal data such as RDD partitions, event log, broadcast variables and shuffle outputs | lz4, lzf, snappy, and zstd | - |
| Apache spark | spark.reducer.maxSizeInFlight | maximum size of map outputs to fetch simultaneously from each reduce task (MB) | 24-96 | 1 |
| Apache spark | spark.shuffle.file.buffer | size of the in-memory buffer for each shuffle file output stream (KB) | 32-64 | 1 |
| Apache spark | spark.serializer | class to use for serializing objects that will be sent over the network or need to be cached in serialized form | JavaSerializer/KryoSerializer | - |
| CNN | batch size | maximum number of iterations | 5-100 | 1 |
| CNN | alpha | Minimum learning rate value as a fraction of initial_learning_rate | 0.001-0.05 | 0.01 |
| CNN | pool size | Max pooling operation for spatial data | 2 | 5 |
| CNN | strides | the strides of the convolution along the width and height | 1 | 5 |
| CNN | steps_per_execution | The number of batches to run during each tensor flow object | 1 | 4 |

platform and the other data sets were retrieved from *Kaggle* [5]. We employ the same data set group chosen in our preceding study since in that study Pareto-based multi-objective optimization produced promising results [49].

**Table 2.** Spark benchmarks and MLlib data sets preferred in this work.

| Type | Name | Size range | Category |
|---|---|---|---|
| Spark benchmark | WordCount | 120-200 (GB) | Workload |
| Spark benchmark | TeraSort | 20-100 (GB) | Workload |
| Spark benchmark | Kmeans | 60-100 (million instances) | Machine Learning |
| Spark benchmark | Bayes | 12-20 (million pages) | Machine Learning |
| Classification | Dense | 175000 (instances) | Machine Learning |
| Classification | Microsoft | 10868 (instances) | Machine Learning |
| Classification | Payload | 130529 (instances) | Machine Learning |
| Classification | Santander | 200000 (instances) | Machine Learning |
| Keras | Cifar10 | 50000 (instances) | Image processing |
| Keras | FashionMnist | 60000 (instances) | Image processing |
| Keras | Mnist | 60000 (instances) | Image processing |

## 5.4. Performance parameters

There exist various measures to evaluate the reliability of RL methods [50]. They are generally employed for interpreting variability during training. Two types of RL evaluation measures were chosen in the experiment: Dispersion and risk. Different from distribution, dispersion shows to what extent the data is aggregated [51]. To measure dispersion, the reward values were exposed to the Coefficient of quartile variation (CQV) designed by improving the Inter-quartile range (IQR) as defined in Equation 4.

$$CQV = (\frac{Q_3 - Q_1}{Q_3 + Q_1})x100 \tag{4}$$

where $Q_3$ and $Q_1$ denote the third and first quartiles giving general dispersion of data sets. CQV is preferred in the experiment since it is much more suitable for asymmetric distributions. *cqv_versatile* function of R *cvcqv* library has been combined with Adjusted bootstrap percentile (BCa) statistical analysis [52].

A complementary method called Conditional Value at Risk (CVaR) for dispersion is utilized to measure how heavy is the low tail of the distribution. CVaR is a numerical measure defining the risk of the occurrence of the worst scenario with respect to dispersion and it is calculated as in Equation 5. CVaR has been recorded as having varying iterations.

$$CVaR_{\phi}(V) = E[V|V \geq VaR_{\phi}(V)] \tag{5}$$

where $\phi \in (0,1)$ and $VaR_{\phi}(V)$ represents the value of risk in the related quarter. It is also used for measuring risks in various research fields including finance [53], robotics [54], and health [55]. The results were

[5] https://zenodo.org/record/5731597

yielded with *cvar* library of R, thereby setting standard deviation and quadratic form (QF) distributions.

The efficacy of tuning processes in Hibench benchmarks was evaluated by *speedup*. While $perf_0$ of *speedup* given in Equation 6 is the time recorded in the default settings, $perf_1$ denotes the time after applying the tuning process. Since the machine utilized in the experiment enable us to increase the number of cores up to 24, *speedup* was recorded between 4 and 24 cores.

$$Speedup = perf_0/perf1[56] \quad (6)$$

## 5.5. Competitors

MFRLMO is compared with Effective Multi-Objective Reinforcement Learning (EMORL) [44] and Hyperparameter Optimization by Reinforcement Learning (Hyp-RL) [43]. Two criteria are used for choosing the baselines: 1) The comparison method should have been revealed in the recent past, and 2) The baselines regard RL as a tool to solve a HO problem rather than aim at configuring the hyperparameters of RL.

Hyp-RL works as follows: 1) Asks users to choose a data set $D$ and the inputs of hyperparameters such as $\Lambda$, 2) RL initiates Q-networks, 3) The reward and Q-values are updated along with the data of minibatch, 4) The Q-network is re-defined for reward and discount factor having the maximum Q-value, 5) The set of Q-values is returned at the end of the iteration. Hyp-RL was compared against five alternatives in estimating the life of large batteries.

EMORL has five steps to perform tuning and it can be summarized as follows: 1) The user is asked to give a target hyperparameter set after assigning the first state of RL, 2) The accuracy and latency changes are observed, thereby sampling new hyperparameters during the state transitions. 3) Q-value is updated with the help of the PPO-clip method [57] to return the optimal value for each hyperparameter.

The reason we compare the proposed method against Hyp-RL and EMORL can be ordered as follows: 1. EMORL is a multi-objective tuning technique and it is compatible with big data processes in updating optimal hyperparameter sets as a way of RL. It thus includes a formula namely latency indicating the magnitude of the reward signal. EMORL performs a two-objective (accuracy-latency) optimization by using that formula. The complexity of EMORL is $O(n_h * k_{LAT} * k_{accuracy})$ wherein $n_h$ is the number of hyperparameters, $k_{LAT}$ is the constraint of latency, and $k_{accuracy}$ is the constraint of accuracy. Since EMORL is defined considering various constraints, dynamic changes are observed depending on the type of optimization problem. 2. Hyp-RL prefers to update reward values in each episode of actions. In this context, Hyp-RL shows similarities in design variations to the proposed method. The complexity of Hyp-RL is $O(N * D * T)$ where $N$ denotes the number of episodes, $D$ is the number of data sets, and $T$ is the number of tasks. The complexity of MFRLMO is $O(count * N_{rs})$ where *count* is the number of updates of *env* and $N_{rs}$ is the number sequences in sampling defined in Algorithm 1-Step 14.

**Table 3.** Mean State-Action function Q values of Microsoft data set in optimizing for time.

| | **right** | **up** | **down** | **left** |
|---|---|---|---|---|
| **s14** | -1.07 | -1.11 | -1.06 | -1.06 |
| **s1** | -1.06 | -1.04 | -1.08 | -1.1 |
| **s2** | -1.06 | -0.97 | -1.04 | -0.98 |
| **s3** | -0.87 | -1.06 | -0.59 | -0.8 |
| **s4** | -1.1 | -1.04 | -0.96 | -0.93 |
| **s5** | -1.01 | -1.09 | -1.03 | -1.07 |
| **s6** | -1.1 | -1.04 | -0.58 | -1.05 |
| **s7** | -0.97 | -1.06 | -0.96 | -1.05 |
| **s8** | -0.98 | -1.09 | -1.04 | -1.06 |
| **s9** | -0.3 | -1.06 | -0.47 | -0.89 |
| **s10** | -0.93 | -1 | **8.54** | -0.3 |
| **s11** | -0.99 | -1.04 | -1.04 | -0.4 |
| **s12** | -0.76 | -1.08 | -1.08 | -1.02 |
| **s13** | **5.26** | -0.82 | -0.47 | -0.76 |

Q-values have been produced for each objective function of MFRLMO as presented in Table 3. The three-objective optimization is established for 50 iterations of each data set that results in 3x50=150 Q-values. In that table, the bold-faced values represent the closest state action to 97% of accuracy. It is worth noting that the minus values in the transitions of other cells are yielded with a model-free approach. Since the experiment exploits 11 data sets in total, 11x150=1650 tables should be evaluated. However, the performance metrics are only associated with the changes in rewards, Q-values have only been used for controlling the internal dynamics of the cells.

# 6. Results

The optimal values of hyperparameters have been found in the low number of iterations in classification as given in Tables 4-5. We can conclude that the variability is relatively high for the hyperparameters *tol* and *step_size*. However, since one algorithm of the MLlib library has been tested in the experiment, a more general result can be achieved, thereby increasing the number of algorithms. On the other hand, the parameters that are directly related to the size of data such as *spark.memory.fraction* differs in Hibench benchmarks. The largest value of *spark.executor.memory*, which defines the memory allocated for the execution, has been detected in WordCount (120GB). The storage memory was set to

0.4 since TeraSort requires large memory. However, the compression rate of RDD is set by MFRLMO without considering the type of Benchmark. Although KyroSerializer performs faster than JavaSerializer, it was not chosen for finding the optimal configuration due to the need for registering classes.

**Table 4.** Optimal configurations (I) detected by MFRLMO.

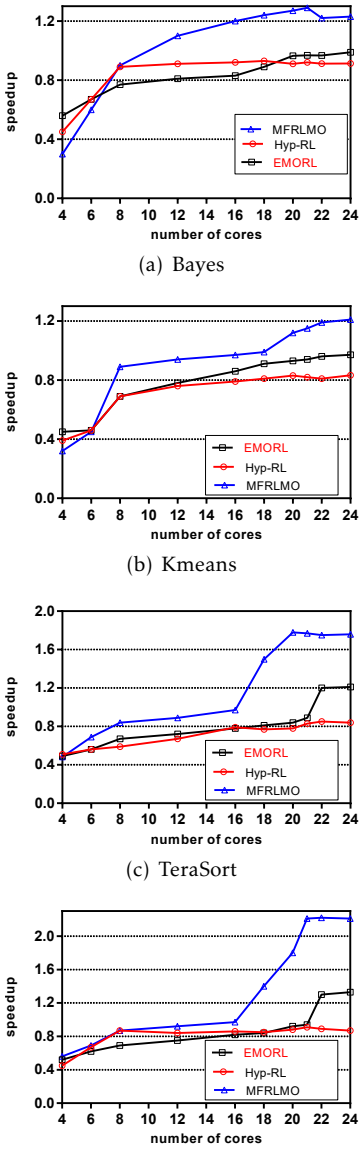
| | Dense | | Microsoft | | Payload | | Santander | |
|---|---|---|---|---|---|---|---|---|
| Hyperparameter | max_iter | 0.35 | max_iter | 15 | max_iter | 20 | max_iter | 20 |
| | tol | 0.00337 | tol | 0.000067 | tol | 0.000034 | tol | 0.000001 |
| | step_size | 4 | step_size | 3 | step_size | 4 | step_size | 7 |

**Table 5.** Optimal configurations (II) detected by MFRLMO.

| | Bayes | | Kmeans | TeraSort | WordCount |
|---|---|---|---|---|---|
| Hyperparameter | spark.task.cpus | 1 | 1 | 1 | 2 |
| | spark.executor.cores | 12 | 4 | 8 | 12 |
| | spark.executor.memory | 90 | 100 | 80 | 120 |
| | spark.memory.fraction | 0.2 | 0.3 | 0.2 | 0.6 |
| | spark.memory.storageFraction | 0.2 | 0.3 | 0.4 | 0.2 |
| | spark.default.parallelism | 210 | 180 | 260 | 240 |
| | spark.shuffle.compress | true | true | true | true |
| | spark.shuffle.spill.compress | true | true | true | true |
| | spark.broadcast.compress | true | true | true | true |
| | spark.rdd.compress | true | true | true | true |
| | spark.io.compression.codec | lzf | lzf | lzf | lzf |
| | spark.reducer.maxSizeInFlight | 32 | 32 | 32 | 32 |
| | spark.shuffle.file.buffer | 64 | 64 | 64 | 64 |
| | spark.serializer | JavaSerializer | JavaSerializer | JavaSerializer | JavaSerializer |

When it comes to interpreting the results of *speedup*, as MFRLMO traces a relatively more linear line in machine learning processes, TeraSort and WordCount, which are workloads, have low acceleration as given in Figure 5. Since Hyp-RL has a high complexity, it has a stable *speedup* though showing a remarkable improvement in accuracy. It is thus not effective in cases of high cores above 8. In particular, EMORL differs from Hyp-RL in a high number of cores. However, MFRLMO is similar to Hyp-RL in terms of convergence. Regardless of the type of benchmark, it is clear that each algorithm has a specific threshold. For instance, the threshold of MFRLMO for TeraSort is 20. Exceeding that value does contribute to *speedup* that leads to the waste of computational resources. Despite the fact that the threshold of Hyp-RL is very high for Workload types of benchmarks, it has a lower *speedup* than that of MFRLMO since Hyp-RL is devised based on a two-objectives optimization. That finding validates that Hyp-RL is much more suitable for low-budget HO operations. These results confirm the inference that the threshold for the number of cores is 16 regardless of the type of benchmark when the optimization is devised based on RL.

The changes in mean rewards are given in Figure 6. The mean reward is initially very low but it increases remarkably with high iterations thanks to setting $\epsilon = 0.2$. It is worth noting that there is a gradual decline in the churn of rewards. That churn is not dependent on the type of data set that the tradeoff between exploration and exploitation is achieved after a specific number of iterations. The beginning of the rise and decline of average reward for Santander and Dense are




(a) Bayes

(b) Kmeans

(c) TeraSort

(d) WordCount

**Figure 5.** Speedup results of HiBench benchmarks.

very similar for mean rewards. On the other hand, the analysis confirms the opposite trend for Microsoft and Santander data sets after 20 iterations.

The CQV results of all the data sets are given in Figure 7. The data set having the lowest dispersion is Santander which has no reasonable churn although it could not yield the best average reward. The payload data set has progressed a narrow area in yielding average reward. Likewise, the distance between the extreme outliers of the CQV boxplot is very short. WordCount has the best dispersion among the benchmarks. Note that it is also the largest experimental data set. WordCount may have produced the best result thanks to the need for operation increasing its size linearly depending on the number

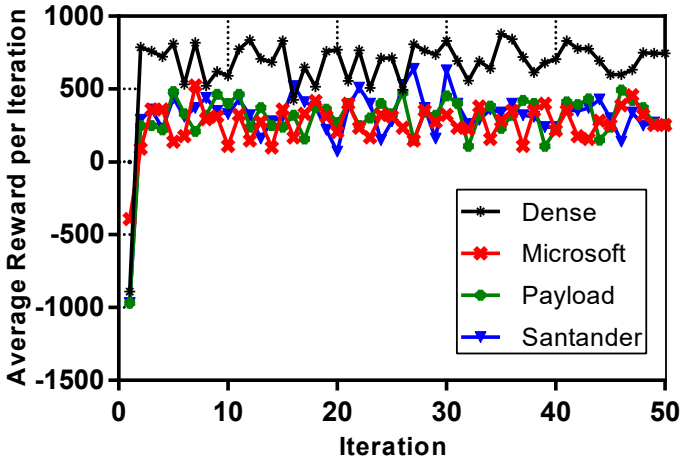



**Figure 6.** Average reward of classification data sets.

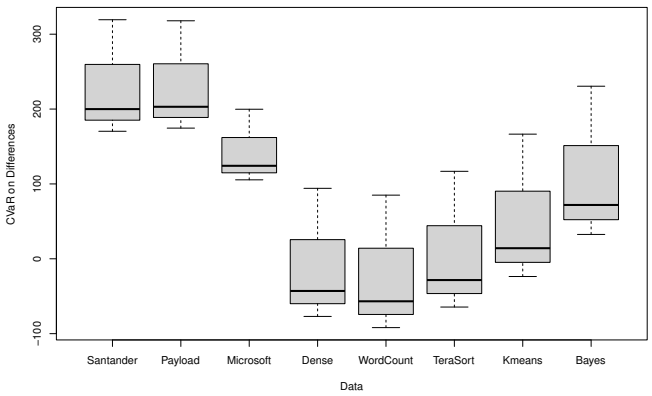



**Figure 8.** Short-term CVaR results of the experimental data sets.

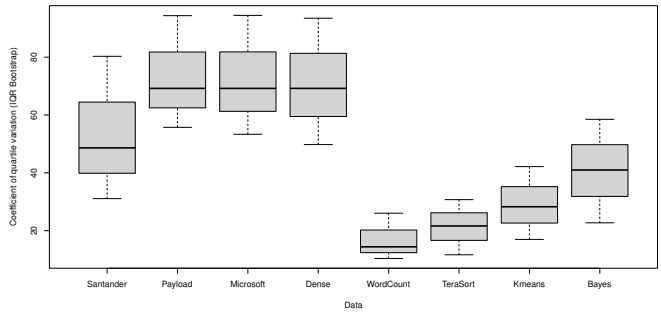



**Figure 7.** CQV results of the experimental data sets.

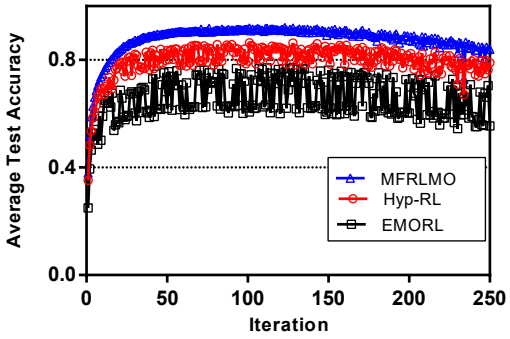
of instances. Although Bayes yielded the worst CQV, it showed promising results in classification data sets. The results given in Figure 8 claim the exact opposite that the inferences we draw from dispersion are not valid for CVaR. For instance, Dense, which is a classification data set, produced worse CQV results than those of CVaR. The ordering has not changed in the benchmarks but it has expanded along with the outliers of the boxplot.

The performance of three competitive methods is compared for tuning CNN with respect to the average test accuracy as presented in Figure 9. The comparison comprising three image processing data sets revealed that convergence is achieved after 50 iterations for Cifar10 and FashionMnist data sets. MFRLMO outperformed the other two methods for three data sets. Although Hyp-RL has a high complexity, it has been observed that Hyp-RL has the lowest fluctuation. EMORL has the highest fluctuation and produced the lowest accuracy. This is because achieving good policy in RL hardens when the search space of HO is very large. In addition to this, EMORL has only been tested for tuning image recognition in the preceding studies. It is worthwhile to note that although MFRMO is a three-objective optimization technique, the tradeoff found during the tuning process resulted in high accuracy. Despite the fact that FashionMnist and Minst include similar data instances, they have different fluctuations. The findings obtained from Figure 9 support the claim that the similarity across the data sets has no impact on achieving a fast convergence.

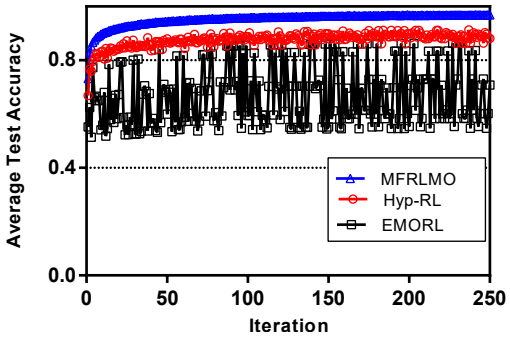
**Table 6.** The performance of optimizing *ml_multilayer_perceptron_classifier* with the baselines.

| | Dataset | Accuracy (Default) | Accuracy with optimal Hyperparameters | Change% |
|---|---|---|---|---|
| **Hyp-RL** | Dense | 0.68 | 0.79 | 16 |
| | Payload | 0.7 | 0.77 | 10 |
| | Microsoft | 0.72 | 0.81 | 12.5 |
| | Santander | 0.73 | 0.78 | 6.84 |
| | WordCount | 0.79 | 0.83 | 5.06 |
| | Bayes | 0.84 | 0.89 | 5.95 |
| | Kmeans | 0.82 | 0.87 | 6.09 |
| | TeraSort | 0.83 | 0.90 | 8.43 |
| **EMORL** | Dense | 0.68 | 0.74 | 8.82 |
| | Payload | 0.7 | 0.73 | 4.28 |
| | Microsoft | 0.72 | 0.79 | 9.72 |
| | Santander | 0.73 | 0.77 | 5.47 |
| | WordCount | 0.79 | 0.81 | 2.53 |
| | Bayes | 0.84 | 0.88 | 4.76 |
| | Kmeans | 0.82 | 0.85 | 3.65 |
| | TeraSort | 0.83 | 0.89 | 7.22 |
| **MFRLMO** | Dense | 0.68 | **0.84** | 8.82 |
| | Payload | 0.7 | **0.87** | 24.28 |
| | Microsoft | 0.72 | **0.85** | 18.05 |
| | Santander | 0.73 | **0.86** | 17.8 |
| | WordCount | 0.79 | **0.92** | 16.45 |
| | Bayes | 0.84 | **0.91** | 8.33 |
| | Kmeans | 0.82 | **0.93** | 10.97 |
| | TeraSort | 0.83 | **0.91** | 9.63 |

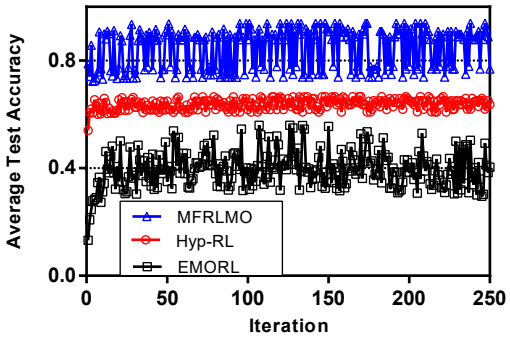
MFRLMO has the highest accuracy among the baselines as given in Table 6. The payload was able to produce the highest improvement (24%). The results of EMORL and Hyp-RL are dependent on the type of data set. Concretely, although Hyp-RL could yield better improvement for the Microsoft data set, EMORL is much more feasible for Kmeans which is also a benchmark having medium size compared to Microsoft. These rates at which accuracy improves are very difficult to be grouped. The algorithmic design




(a) Cifar10




(b) FashionMnist




(c) Mnist

**Figure 9.** Comparison of baselines in tuning CNN (Optimal configurations: alpha:0.02, pool size:3, strides:2, steps_per_execution:2).

properties given in Section 5.5 may have led to that churn due to their effects on the complexity.

## 7. Threats to Validity

Firstly, the proposed method was only evaluated by using the *ml_multilayer_perceptron_classifier* classifier of MLlib. Therefore, the results may not be generalizable for the other algorithms of MLlib. However, the experimental data sets chosen for the classification have various sizes and they are compatible with the other types of classifiers. Hence, the results may incur the loss of generality of the proposed method but it can be avoided by performing a replication study. Moreover, some hyperparameters related to the input and output layers are out of the scope of the experiment due to the stability of the number of features after performing Direct Search for *ml_multilayer_perceptron_classifier*. The experiment can be replicated by utilizing the major steps of MFRLMO for all hyperparameters of some classifiers such as the Gradient-boosted tree.

The second threat is about the chosen metrics of data sets. *Accuracy* is the sole formula used for evaluating the success of classification. However, the width of the distribution of reward values is calculated with CQV and its risk is interpreted with CVaR. Hence, the reliability of the results could have been examined with the help of three different evaluation metrics. Nevertheless, these measures are not always sufficient, given that Q-values are of great importance in some RL experiments. In this respect, new formulas would be designed to find out the underlying mechanism of Q-value changes during the state transition of the iterations.

## 8. Conclusion

Although Apache Spark is an open-source big data processing platform, it is a must to perform tuning regarding various criteria, resulting in a significant reduction in computational cost. In this study, a multi-objective RL-based optimization method called MFRLMO is proposed to solve the HO problem of Apache Spark. Unlike black-box approaches, some dynamic internal improvements have been made by considering the objective functions as a set of sequential HO problems. Specifically, the study is the first to use model-free RL in tuning parameters of a big data processing platform. To this end, the update of rewards has been performed in each iteration of MFRLMO by checking action-reward results. Though a tradeoff between memory, time, and accuracy has been achieved thanks to an ensemble technique, MFRLMO outperforms the baselines in accuracy. The proposed method does not suffer from a grid-based dimensionality that may affect the computation time. It can be deduced that to obtain consistent results of dispersion and risk analyzes, the number of instances should be as large as possible. We can conclude from the comparison that there is not a monotonic relationship between *speedup* and the number of cores. Although the model of the environment is not involved in the feature set of data sets when the type of RL is model-free, potential consequences of actions should be rigorously analyzed. Therefore, in future works, the exact number of random sequences we need to obtain a reliable RL depending on the state transitions could be investigated. Further, it is planned to analyze to what extent the interaction between the number of grid cells

and $\epsilon$ changes the tradeoff of objective functions.

## Declarations

**Funding** Not applicable.

**Conflict of interest** The authors declare that they have no confict of interest.

**Availability of data and material** The data required to replicate the experiment is presented in Section 5.2.

**Code availability** The link required to access the replication packages is presented in Appendix A.

**Authors' contributions** Not applicable.

**Ethics approval** This article dos not contain any studies with human participants or animals performed by any of the authors.

**Consent to participate** Informed consent was obtained from all individual participants included in the study.

**Consent for publication** The authors affirm that human research participants provided informed consent for publication.

## References

[1] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J. *et al.* (2016) Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* **17**(1): 1235–1241.

[2] Xie, F. (2023) Monitoring and quality evaluation method of english teaching in machine manufacturing based on machine learning and internet of things. *EAI Endorsed Transactions on Scalable Information Systems* **10**(6).

[3] Morfino, V. and Rampone, S. (2020) Towards near-real-time intrusion detection for iot devices using supervised learning and apache spark. *Electronics* **9**(3): 444.

[4] Cheng, Y., Yu, N., Foggo, B. and Yamashita, K. (2022) Online power system event detection via bidirectional generative adversarial networks. *IEEE Transactions on Power Systems* .

[5] de Souza Neto, J.B., Martins Moreira, A., Vargas-Solar, G. and Musicante, M.A. (2022) Transmut-spark: Transformation mutation for apache spark. *Software Testing, Verification and Reliability* : e1809.

[6] Wang, G., Xu, J. and He, B. (2016) A novel method for tuning configuration parameters of spark based on machine learning. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (IEEE): 586–593.

[7] Baldacci, L. and Golfarelli, M. (2018) A cost model for spark sql. *IEEE Transactions on Knowledge and Data Engineering* **31**(5): 819–832.

[8] Zhu, Y., Liu, J., Guo, M., Bao, Y., Ma, W., Liu, Z., Song, K. *et al.* (2017) Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*: 338–350.

[9] Ge, Y.F., Wang, H., Bertino, E., Zhan, Z.H., Cao, J., Zhang, Y. and Zhang, J. (2023) Evolutionary dynamic database partitioning optimization for privacy and utility. *IEEE Transactions on Dependable and Secure Computing* .

[10] Cheng, G., Ying, S. and Wang, B. (2021) Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model. *Journal of Systems and Software* **180**: 111028.

[11] Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z. and Guyon, I. (2021) Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track* (PMLR): 3–26.

[12] Shekar, B. and Dagnew, G. (2019) Grid search-based hyperparameter tuning and classification of microarray cancer data. In *2019 second international conference on advanced computational and communication paradigms (ICACCP)* (IEEE): 1–8.

[13] Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T. *et al.* (2022) Smac3: A versatile bayesian optimization package for hyperparameter optimization. *J. Mach. Learn. Res.* **23**: 54–1.

[14] Binder, M., Moosbauer, J., Thomas, J. and Bischl, B. (2020) Multi-objective hyperparameter tuning and feature selection using filter ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*: 471–479.

[15] Meister, M., Sheikholeslami, S., Payberah, A.H., Vlassov, V. and Dowling, J. (2020) Maggy: Scalable asynchronous parallel hyperparameter search. In *Proceedings of the 1st Workshop on Distributed Machine Learning*: 28–33.

[16] Trotter, M., Wood, T. and Hwang, J. (2019) Forecasting a storm: Divining optimal configurations using genetic algorithms and supervised learning. In *2019 IEEE international conference on autonomic computing (ICAC)* (IEEE): 136–146.

[17] Liu, J., Ravi, N., Chakradhar, S. and Kandemir, M. (2012) Panacea: Towards holistic optimization of mapreduce applications. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*: 33–43.

[18] Lin, J.C., Lee, M.C., Yu, I.C. and Johnsen, E.B. (2018) Modeling and simulation of spark streaming. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)* (IEEE): 407–413.

[19] Petridis, P., Gounaris, A. and Torres, J. (2016) Spark parameter tuning via trial-and-error. In *INNS Conference on Big Data* (Springer): 226–237.

[20] Li, J.Y., Du, K.J., Zhan, Z.H., Wang, H. and Zhang, J. (2022) Distributed differential evolution with adaptive resource allocation. *IEEE transactions on cybernetics* .

[21] Fu, T.Z., Ding, J., Ma, R.T., Winslett, M., Yang, Y. and Zhang, Z. (2015) Drs: dynamic resource scheduling for real-time analytics over fast streams. In *2015 IEEE 35th International Conference on Distributed Computing Systems* (IEEE): 411–420.

[22] Petrov, M., Butakov, N., Nasonov, D. and Melnik, M. (2018) Adaptive performance model for dynamic scaling apache spark streaming. *Procedia Computer Science* **136**: 109–117.

[23] Venkataraman, S., Panda, A., Ousterhout, K., Armbrust, M., Ghodsi, A., Franklin, M.J., Recht, B. *et al.* (2017) Drizzle: Fast and adaptable stream processing at scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*: 374–389.

[24] Zacheilas, N., Kalogeraki, V., Zygouras, N., Panagiotou, N. and Gunopulos, D. (2015) Elastic complex event processing exploiting prediction. In *2015 IEEE International Conference on Big Data (Big Data)* (IEEE): 213–222.

[25] Vaquero, L.M. and Cuadrado, F. (2018) Auto-tuning distributed stream processing systems using reinforcement learning. *arXiv preprint arXiv:1809.05495* .

[26] Bhatia, A., Svegliato, J., Nashed, S.B. and Zilberstein, S. (2022) Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, **32**: 556–564.

[27] Liu, X., Wu, J. and Chen, S. (2022) A context-based meta-reinforcement learning approach to efficient hyperparameter optimization. *Neurocomputing* **478**: 89–103.

[28] Yu, W., You, J., Niu, X., He, J. and Zhang, Y. (2023) Rboira: Integrating rules and reinforcement learning to improve index recommendation. *EAI Endorsed Transactions on Scalable Information Systems* **10**(6).

[29] Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P. and Levine, S. (2021) How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research* **40**(4-5): 698–721.

[30] Fan, T., Long, P., Liu, W. and Pan, J. (2020) Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research* **39**(7): 856–892.

[31] Hu, J., Niu, H., Carrasco, J., Lennox, B. and Arvin, F. (2020) Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology* **69**(12): 14413–14423.

[32] Oliff, H., Liu, Y., Kumar, M., Williams, M. and Ryan, M. (2020) Reinforcement learning for facilitating human-robot-interaction in manufacturing. *Journal of Manufacturing Systems* **56**: 326–340.

[33] Wang, D., Deng, H. and Pan, Z. (2020) Mrcdrl: Multi-robot coordination with deep reinforcement learning. *Neurocomputing* **406**: 68–76.

[34] Wu, Y.H., Yu, Z.C., Li, C.Y., He, M.J., Hua, B. and Chen, Z.M. (2020) Reinforcement learning in dual-arm trajectory planning for a free-floating space robot. *Aerospace Science and Technology* **98**: 105657.

[35] Hu, H., Zhang, K., Tan, A.H., Ruan, M., Agia, C. and Nejat, G. (2021) A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain. *IEEE Robotics and Automation Letters* **6**(4): 6569–6576.

[36] Rajeswaran, A., Mordatch, I. and Kumar, V. (2020) A game theoretic framework for model based reinforcement learning. In *International conference on machine learning* (PMLR): 7953–7963.

[37] Albaba, B.M. and Yildiz, Y. (2019) Modeling cyber-physical human systems via an interplay between reinforcement learning and game theory. *Annual Reviews in Control* **48**: 1–21.

[38] Xu, Q., Su, Z. and Lu, R. (2020) Game theory and reinforcement learning based secure edge caching in mobile social networks. *IEEE Transactions on Information Forensics and Security* **15**: 3415–3429.

[39] Albaba, B.M. and Yildiz, Y. (2021) Driver modeling through deep reinforcement learning and behavioral game theory. *IEEE Transactions on Control Systems Technology* **30**(2): 885–892.

[40] Ahad, A., Tahir, M., Sheikh, M.A., Ahmed, K.I. and Mughees, A. (2021) An intelligent clustering-based routing protocol (crp-gr) for 5g-based smart healthcare using game theory and reinforcement learning. *Applied Sciences* **11**(21): 9993.

[41] Bui, V.H., Hussain, A. and Su, W. (2022) A dynamic internal trading price strategy for networked microgrids: A deep reinforcement learning based game-theoretic approach. *IEEE Transactions on Smart Grid* .

[42] Wu, J., Chen, S. and Liu, X. (2020) Efficient hyperparameter optimization through model-based reinforcement learning. *Neurocomputing* **409**: 381–393.

[43] Jomaa, H.S., Grabocka, J. and Schmidt-Thieme, L. (2019) Hyp-rl: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527* .

[44] Chen, S., Wu, J. and Liu, X. (2021) Emorl: Effective multi-objective reinforcement learning method for hyperparameter optimization. *Engineering Applications of Artificial Intelligence* **104**: 104315.

[45] García, S., Luengo, J. and Herrera, F. (2015) *Data preprocessing in data mining*, **72** (Springer).

[46] Aragón-Royón, F., Jiménez-Vílchez, A., Arauzo-Azofra, A. and Benítez, J.M. (2020) Fsinr: an exhaustive package for feature selection. *arXiv preprint arXiv:2002.10330* .

[47] Nguyen, N., Khan, M.M.H. and Wang, K. (2018) Towards automatic tuning of apache spark configuration. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (IEEE): 417–425.

[48] Arnold, T.B. (2017) kerasr: R interface to the keras deep learning library. *J. Open Source Softw.* **2**(14): 296.

[49] Ozturk, M.M. Tuning parameters of apache spark with gauss pareto based multi-objective optimization.

[50] Chan, S.C., Fishman, S., Canny, J., Korattikara, A. and Guadarrama, S. (2019) Measuring the reliability of reinforcement learning algorithms. *arXiv preprint arXiv:1912.05663* .

[51] Altunkaynak, B. and Gamgam, H. (2019) Bootstrap confidence intervals for the coefficient of quartile variation. *Communications in Statistics-Simulation and Computation* **48**(7): 2138–2146.

[52] Jung, K., Lee, J., Gupta, V. and Cho, G. (2019) Comparison of bootstrap confidence interval methods for gsca using a monte carlo simulation. *Frontiers in psychology* **10**: 2215.

[53] Chapman, M.P., Bonalli, R., Smith, K.M., Yang, I., Pavone, M. and Tomlin, C.J. (2021) Risk-sensitive safety analysis using conditional value-at-risk. *IEEE Transactions on Automatic Control* .

[54] Mhammedi, Z., Guedj, B. and Williamson, R.C. (2020) Pac-bayesian bound for the conditional value at risk. *Advances in Neural Information Processing Systems* **33**: 17919–17930.

[55] Soma, T. and Yoshida, Y. (2020) Statistical learning with conditional value at risk. *arXiv preprint arXiv:2002.05826* .

[56] Guo, Y., Shan, H., Huang, S., Hwang, K., Fan, J. and Yu, Z. (2021) Gml: Efficiently auto-tuning flink's configurations via guided machine learning. *IEEE Transactions on Parallel and Distributed Systems* **32**(12): 2921–2935.

[57] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017) Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .