Hyp-RL : Hyperparameter Optimization by Reinforcement Learning*

Hadi S. Jomaa[1], Josif Grabocka[1], and Lars Schmidt-Thieme[1]

University of Hildesheim, Universitätsplatz 1, 31141 Hildesheim, Germany
{jomaah,josif,lars}@ismll.uni-hildesheim.de

Abstract. Hyperparameter tuning is an omnipresent problem in machine learning as it is an integral aspect of obtaining the state-of-the-art performance for any model. Most often, hyperparameters are optimized just by training a model on a grid of possible hyperparameter values and taking the one that performs best on a validation sample (grid search). More recently, methods have been introduced that build a so-called surrogate model that predicts the validation loss for a specific hyperparameter setting, model and dataset and then sequentially select the next hyperparameter to test, based on a heuristic function of the expected value and the uncertainty of the surrogate model called acquisition function (sequential model-based Bayesian optimization, SMBO).
In this paper we model the hyperparameter optimization problem as a sequential decision problem, which hyperparameter to test next, and address it with reinforcement learning. This way our model does not have to rely on a heuristic acquisition function like SMBO, but can learn which hyperparameters to test next based on the subsequent reduction in validation loss they will eventually lead to, either because they yield good models themselves or because they allow the hyperparameter selection policy to build a better surrogate model that is able to choose better hyperparameters later on. Experiments on a large battery of 50 data sets demonstrate that our method outperforms the state-of-the-art approaches for hyperparameter learning.

Keywords: Hyperparameter Optimization · Reinforcement Learning · Transfer Learning.

1 Introduction

Hyperparameter tuning plays a significant role in the overall performance of machine learning models and can be the main factor in deciding whether a trained model turns out to be the state-of-the-art or simply moderate [35]. The growing sizes of data sets and the complexities of the models result in an increased training time and require more resources, which makes hyperparameter optimization an even harder task. When automating hyperparameter tuning, practical solutions must be robust, scalable and maintain low computational budget, while

* The authors would like to thank Nicolas Schilling for his valuable advice.

ensuring a strong final performance [9]. Several solutions have been proposed to tune hyperparameters over the years which vary in terms of computational complexity and scalability. Some more traditional approaches include manual search, grid search, and random search [4], whereas others, such as Bayesian optimization techniques [25, 15], rely on probabilistic models, most commonly Gaussian processes, to optimize the expected improvement of the surrogate model, by treating the response function (e.g. validation loss) as a black box function.
The search for the best hyperparameter configuration is a sequential decision process [37] in which initial values are set, and later adjusted, through a mixture of intuition and trial-and-error, to optimize an observed performance on typically the hold-out validation set, i.e. to maximize the accuracy or minimize the loss. Hence, finding a scalable and efficient policy for hyperparameter tuning is critical, especially for high-dimensional function estimators, such as deep neural networks that require longer time to evaluate a single hyperparameter response. To achieve an automatic hyperparameter tuning, one must also take into account how *well* certain configurations performed on other data sets and carry this knowledge over to new data sets. A well trained policy would then be able to navigate the response surface of a learning algorithm based on previous experiences, in order to converge rapidly to a global optimum.
In this work, we pose hyperparameter tuning within a reinforcement learning (RL) framework, where the agent learns to explore the hyperparameter space of a fixed network topology. The task in RL is to optimize an agent's return over a fixed budget of episodes, by learning to estimate the expected return at a given state, and optimizing the action selection process in a way that maximizes the return [38]. RL research has witnessed rapid progress within the last decade [12], showing promising results in several areas such as robotic control [18, 24], game playing [21, 27, 26], e-commerce [6, 40], and more. The core of hyperparameter optimization boils down to selecting the optimal configuration, given an observable history of configurations and corresponding rewards, in a way that increases future rewards, which falls seemlessly into a RL framework. In this paper, we learn a controller that can tune the hyperparameters of a fixed topology by defining a state representation, set of actions, and a transition function which allow an agent to navigate the response surface and maximize its reward.
We summarize the contributions of the paper as follows:

- This is the first paper, to the best of our knowledge, that formulizes hyperparameter tuning as a reinforcement learning problem, and provides empirical evidence on its feasibility
- A novel policy based on Q-learning, Hyp-RL, that can navigate high-dimensional hyperparameter spaces
- Formulation of hyperparameter response surface as an environment with a defined reward function and action space
- Experiments on a meta-data set created using 50 randomly selected UCI classification data sets that highlight the strong final performance of the controller and scalability across varying hyperparameter response surfaces
- Empirical evaluation that showcases the consistent gain in classification performance based on the initial hyperparameter trials selected by our controller

2 Related Work

Hyperparameter optimization is still considered an open problem within the machine learning community, despite being widely used in practice to improve the performance of any learning algorithm. Perhaps the simplest approaches include manual search, selecting configurations based on intuition, grid search, which allocates equal weights to unimportant parameters, and random search [4] that suffers from the lack of guidance.

Bayesian optimization techniques [25] model the hyperparameter space in a probabilistic framework and obtain smooth uncertainty estimates when modelling the validation error. Despite the demonstrated powerful performance on tuning convolutional [28, 17] and fully-connected [20] neural networks, these methods are based on Gaussian processes that require an increasing number of data points with higher dimensional spaces and setting hyperpriors [9]. Other approaches have also been proposed as a combination of existing methods, building on top of their existing merits [9, 29].

Meta-knowledge is also leveraged to accelerate hyperparameter tuning across data sets. The straightforward way is through a warm-start initialization which can extend to all methods. This has led to a significant improvement for Bayesian optimization techniques based on Gaussian processes [33, 11, 19] and on neural networks [22]. The transfer of surrogate models, limited to the sequential model-based optimization [16] framework is also explored, where independent surrogate models are either combined in an ensemble [36, 35, 10], or a surrogate model is trained across data sets by modeling a common response surface [39].

Hyperparameter optimization is also addressed within the scope of reinforcement learning, specifically for architectural network design. In [41], an RNN-based policy network is proposed which iteratively generates new architectures based on the gradient information from its child networks. At the end of the training process, the model that resulted in the best validation error is selected for further training. A meta-modeling approach is also proposed within a Q-learning setting [2]. The final architecture produces by these agents achieved state-of-the-art performance on several image classification data sets. For online learning rate adjustments, gradient information are used as state representation at every step [37]. Perhaps the most similar work to our proposed approach is hyperparameter optimization for tracking applications [8]. Using a continuous deep Q-learning network and a sequence of images as a state representation, hyperparameters are updated to maximize the performance of the tracker.

In this paper, we cast the problem of hyperparameter tuning into a reinforcement learning task. We define an environment where an agent learns to navigate the hyperparameter space and arrives at the best configuration, through a series of actions representing the hyperparameter configuration to be evaluated, and observes the reward provided by the response function. Hyperparameter configurations are sequentially chosen by a policy function that replaces the acquisition function utilized in Bayesian optimization based approaches and balances between exploration and exploitation. Posing the problem in this fashion allows to exploit all the bells and whistles of reinforcement learning, and to the best

of our knowledge, we are the first to formulate hyperparameter tuning in this manner.

3 Problem Definition: Hyperparameter Optimization

The objective of a machine learning algorithm $\mathcal{A} : \mathcal{D} \times \Lambda \rightarrow \mathcal{M}$ is to estimate a model $M_\lambda \in \mathcal{M}$, from the space of all models $\mathcal{M}$ with hyperparameters $\lambda \in \Lambda$, that optimizes an objective function, for example a loss function $\mathcal{L}$, over a data set distribution $D \in \mathcal{D}$, where $\mathcal{D}$ is the set of all data sets, such that:

$$\mathcal{A}(D, \lambda) = \arg\min_{M_\lambda \in \mathcal{M}} \mathcal{L}(M_\lambda, D). \tag{1}$$

We define $\Lambda = \Lambda_1 \times \cdots \times \Lambda_P$ as the P-dimensional hyperparameter space that can include continuous or discrete values. The resulting model M_λ is optimized with respect to its own parameters, if it is parametric, through a series of updates given a specific hyperparameter configuration λ. However, the trained model M_λ might suffer from a generalization error if λ is not carefully optimized.
The task of hyperparameter optimization is to identify an optimal hyperparameter configuration $\lambda^* \in \Lambda$ that results in a model M_{λ^*} , such that the generalization error on the validation set is minimized:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \mathcal{L}(\mathcal{A}(D^{\text{train}}, \lambda), D^{\text{valid}}), \tag{2}$$

and $D^{\text{train}} \bigcap D^{\text{valid}} = \emptyset$. Treating M_λ as a black box does not offer insight into the importance of hyperparameters, and makes tuning the model exhaustive and time-consuming as it must be trained for every configuration before arriving at the optimal one.
By utilizing the uncertainty that comes with Gaussian modeling, the acquisition function maintains a tradeoff between exploitation and exploration, i.e between selecting hyperparameters in an already well-explored are with a high confidence, which might represent a local optimum, or from a previously unexplored area with low confidence but a potentially lower validation loss.
The sequential decision nature of hyperparameter tuning and the defined objective of maximizing a reward allows us to cast the problem in a reinforcement learning framework, where we replace the acquisition function by Hyp-RL, the proposed reinforcement learning policy.

4 Hyp-RL: Hyperparameter Tuning in a RL Framework

In this section, we formulate the sequential decision-making task of hyperparameter tuning as a Markov Decision Process (MDP) and describe the model architecure used.

4.1 MDP Formulation

A standard reinforcement learning setting is based on an MDP represented by a tuple $\langle S, A, R, \tau \rangle$, with S as the set of all states, A as the action space, a reward function $R : S \times A \to \mathbb{R}$ and the transition function $\tau : S \times A \times \mathbb{R} \to S$ that generates a new state in a possibly stochastic or deterministic environment $\mathcal{E}$. The agent interacts with the environment $\mathcal{E}$, by executing an action from $A = \{1, \ldots, |A|\}$ with the task of maximizing the expected discounted reward. The agent's behaviour is governed by a stochastic policy, $\pi : S \to A$, which computes the true state-action value, as:

$$Q_\pi(s, a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \;\mid\; S_0 = s, A_0 = a \right], \tag{3}$$

where $\gamma \in [0, 1]$ represents the discount factor balancing between immediate and future rewards.
To solve this sequential decision problem, the optimal policy selects the action that maximizes the discounted cumulative reward, $\pi_*(s) \in \arg\max_a Q_*(s, a)$, where $Q_*(s, a)$ denotes the optimal action value. One of the most prominent value-based methods for solving reinforcement learning problems is Q-learning [32], which directly estimates the optimal value function and obeys the fundamental identity, known as the Bellman equation [3]:

$$Q_*(s, a) = E_\pi \left[r + \gamma \max_{a'} Q_*(s', a') \;\mid\; S_0 = s, A_0 = a \right] \tag{4}$$

where $s' = \tau(s, a)$. Hyperparameter optimization can also be formulated as a MDP. As mentioned previously Λ is the space of all hyperparameter configurations and the hyperparameter response surface can be defined by the function f as:

$$f : \mathcal{D} \times \Lambda \to \mathbb{R} \tag{5}$$

The response can be any meaningful performance metric, so without loss of generality, we consider it to be the validation loss of the model M trained with hyperparameters λ:

$$f(D, \lambda) = \mathcal{L}\left(M_\lambda(D^{\text{train}}), D^{\text{valid}}\right) \tag{6}$$

The agent navigates the hyperparameter response space through a series of actions, which are simply the next hyperparameter configurations to be evaluated, and thus the total number of actions correponds to the total number of possible hyperparameter configurations, i.e. :

$$A = \Lambda \tag{7}$$

The state of the environment is defined as the data set metafeatures $\mathbb{D} = \mathbb{R}^w$, with $w = \mathbf{dim}(\mathbb{D})$ as the number of metafeatures, described in Section 5, plus

the history of evaluated hyperparameter configurations and their corresponding response:

$$S = \mathbb{D} \times (\Lambda \times R)^* \tag{8}$$

The reward function is set as the hyperparameter response function, and depends on the data set D and the action selected, as shown below:

$$R(D, a) = -f(D, \lambda = a) \tag{9}$$

considering that the agent's task is to maximize the reward. The observed reward depends solely on the data set and the hyperparameter configuration selected. Once an action is selected, a new hyperparameter configuration is evaluated. The transition function then generates a new state, s', by appendinding the newly evaluated hyperparameter configuration, λ, and the corresponding reward r observed to the previous state s:

$$s' = \tau(s, a, r) = (s, (\lambda = a, r))) \tag{10}$$

for $s = (d, (\lambda_0, r_0), \ldots, (\lambda_t, r_t))$ and $d = \text{metafeatures}(D) \in \mathbb{D}$. Inherently, each state s encompasses all previously seen states. The agent reaches a terminal state in two cases, either the agent exceeds a pre-allocated budget T, for example running time, or the same action is selected twice in a row. We impose the second condition to ensure that the agent keeps on exploring the hyperparameter space and does not settle on a specific rewarding configuration. The proposed algorithm is summarized in Algorithm 1. We alternate between data sets, D by randomly sampling the initial state s_0 as $s_0 = (\text{metafeatures}(D), (\{0\}^{\mathbf{dim}(\Lambda)}, 0)), \quad D \sim \text{Unif}(\mathcal{D})$.

Wrapping this formulation in a reinforcement learning framework, the agent starts at a random location in the hyperparameter space of a random data set and is required to navigate the response surface of a given model type. The agent explores the environment by selecting the next best hyperparameter configuration, and receives a reward with every step until a terminal state reached, i.e until the budget is exceeded or the same hyperparameter configuration is selected twice in a row. When the agent runs out of episodes, it is relocated to the response surface of another data set. For the sake of comparison to RL in games, we can think of the hyperparameter tuning as a maze-like environment where every data set represents a different level in the game.

4.2 Model Architecture

The state representation is decomposed into two parts: the static data set features $s_{\text{static}} = d$, and the sequence of selected hyperparameter configurations and their correponding rewards, $s_{\text{dynamic}} \in (\Lambda \times R)^*$, which we model as a dynamic multivariate time series distribution at time T as $s_{\text{dynamic}} \in (\Lambda \times R)^T$. Each channel hence in s_{dynamic} represents a hyperparameter value, with the final channel including the reward. For general purpose sequence modeling, long short term memory (LSTM) cells [14] have proven to be powerful structures to

Algorithm 1 Hyp-RL

Input: $\mathcal{D}$ - data sets, Λ - hyperparameter grid, γ - discount factor, N_u - target update frequency, N_b - replay buffer size, N_e - number of episodes per data set, T - number of actions per episode
initialize $\hat{Q}$ network parameters θ randomly; $\theta^- = \theta$; replay buffer $\mathcal{B} = \emptyset$
for $N_e \cdot |\mathcal{D}|$ iterations **do**
  $s_0 = (\text{metafeatures}(D), (\{0\}^{\mathbf{dim}(\Lambda)}, 0)), \quad D \sim \text{Unif}(\mathcal{D})$
  for $t \in 0, \ldots, T$ and while s_t is not terminal **do**
    Determine next action a_t:

$$a_t = \begin{cases} \sim \text{Unif}(\Lambda), & \text{if } p \sim \text{Unif}([0,1]) < \epsilon \\ \max_a \hat{Q}(s_t, a; \theta), & \text{otherwise} \end{cases}$$

    Receive reward $r_t = R(D, \lambda = a_t)$ ▷ eq. 9
    Generate new state $s_{t+1} = \tau(s_t, \lambda_t, r_t)$ ▷ eq. 10
    Store $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, s_{t+1}, a_t, r_t)\}$ and replace oldest tuple if $|\mathcal{B}| > N_b$
    Sample and relabel a minibatch B of experiences from the replay buffer:

$$B = \{(s, a, Q(s, s', a, r)) \mid (s, s', a, r) \sim \text{Unif}(\mathcal{B})\}$$
$$\text{with } Q(s, s', a, r) = \begin{cases} r, & \text{if } s' \text{ is terminal} \\ r + \gamma \max_{a'} \hat{Q}_{\text{target}}(s', a'; \theta^-), & \text{otherwise} \end{cases}$$

    Update the $\hat{Q}$ network by minimizing

$$\theta = \arg\min_{\theta'} \sum_{(s,a,Q) \in B} \left(Q - \hat{Q}(s, a; \theta') \right)^2$$

    Replace target parameters $\theta^- \leftarrow \theta$ every N_u steps
return θ

model long-range dependencies. A simple LSTM cell includes a memory cell, c_t that accumulates information across time steps. The amount of information carried over by c_t is regulated by means of an input gate, i_t, and a forget gate, f_t, whereas an output gate o_t controls the final state h_t. The key equations that are executed by an LSTM are presented in Equation 11:

$$\begin{aligned} f_t &= \sigma_g(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma_g(W_i \cdot [h_{t-1}, x_t] + b_i) \\ o_t &= \sigma_g(W_o \cdot [h_{t-1}, x_t] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot (\tanh(W_c[h_{t-1}, x_t] + b_c)) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{11}$$

with $b_\cdot$ and $\odot$ denote the bias and the Hadamard product respectively, and $x_t = (\lambda_t, r_t) \in s_{\text{dynamic}}$. The hyperparameter response surface differs across data sets, and therefore it is important that the policy is conditioned on the data set itself. We take that into consideration by modifying the initialization

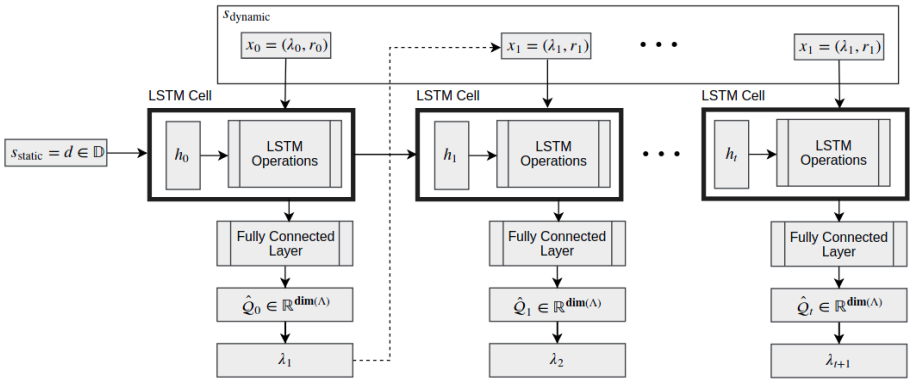


Fig. 1. The schematic illustration of Hyp-RL

of the LSTM, where the initial state, h_0, is commonly initialized as a the zero vector, i.e. $h_0 \in \{0\}^{N_h}$, with N_h as the number of hidden units in the cell. More specifically, we set h_0 as:

$$h_0 = W_0 \cdot s_{\text{static}} \tag{12}$$

where $W_0 \in \mathbb{R}^{N_h \times \mathbf{dim}(\mathbb{D})}$. Through this formulation, the agent is able to start navigating the hyperparameter response surface intelligently from the very start, as opposed to Bayesian-based approaches that require the evaluation of several randomly selected configurations to initally fit the surrogate model. A schematic illustration of our model is presented in Figure 1.

5 Meta Data Set Description: nnMeta

Meta features are descriptive statistical information about a data set distribution, and are important factors in learning algorithms that are trained across data sets. These indicators can be as simple as a one-hot encoded vector, however this does not entirely capture the latent relationships between different data sets. For this reason, we look into the observations within each data set and compute different statistics, such as skewness, kurtosis, dimensionality, etc., as a richer encoding of every distribution. These features are further scaled to a Gaussian distribution to reduce variations across data sets. The complete list of features are presented in Table 1, and have been initially presented in [34]. To generate the meta-data set, we randomly select 50 UCI classification data sets, and train a neural network architecture with different hyperparameter configurations in Keras [7]. The hyperparameter response is obtained by evaluating on 20% of the data after training on the remaining 80%. Three different groups of hyper-paramaters are investigated to generate the meta-data set: structure-based which can be summarized by the number of layers, number of neurons and activation function; optimization-based which include the type of optimizer and the number of epochs; and regularization-based which include dropout rate,

Table 1. The list of all metafeatures used to generate nnMeta

Number of Instances	Kurtosis Min
Log Number of Instances	Kurtosis Max
Number of Features	Kurtosis Mean
Log Number of Features	Kurtosis Standard Deviation
Data Set Dimensionality	Skewness Min
Log Data Set Dimensionality	Skewness Max
Inverse Data Set Dimensionality	Skewness Mean
Log Inverse Data Set Dimensionality	Skewness Standard Deviation

Table 2. Hyperparameter grid of nnMeta

		Values			Encoding
Structure	Activation Function	ReLU	leakyReLU	tanh	One-hot
	Number of Neurons	5	10	20	Scalar
	Number of Hidden Units	10	20	50	Scalar
Optimization	Optimizer	Adam	AdaDelta	AdaGrad	One-hot
	Number of Epochs	10	100		Scalar
Regularization	Dropout	0	0.2	0.4	Scalar
	L_p Regularization	L_1	L_2		One-hot
	Regularization Constant	0.01	0.001	0.0001	Scalar

regularization technique and the regularization constant. The values of the hyperparameters are summarized in Table 2. The experimental results represent the average of a 5-fold based cross-validation. The hyperparameter grid results in 2916 distinct experiments per data set, and 13 dimensions per configuration.

6 Experiments

The netowork is implemented in Tensorflow [1] and the reported results are the averaged over 5 environments representing 5-splits. The environments are wrapped in the OpenAI framework [5]. The code is available here [1].

6.1 Baselines

We compare our proposed approach with several adopted baselines that vary in terms of complexity and scalability.

1. Random Search [4] (RS): One of the simplest hyperparameter tuning methods, in which several configurations are evaluated at random from the hyperparameter search space, which have shown to outperform manual and grid search.

[1] https://github.com/hadijomaa/HypRL.git

2. Independent Gaussian Process [28] (I-GP): In this approach, the surrogate is modeled by a Gaussian process with a squared-exponential kernel and automatic relevance determination. Hyperparameter tuning is done on each data set independently.
3. Spearmint [28]: Similar to I-GP, the surrogate is modeled by a Gaussian process with a Matérn 5/2 kernel.
4. Factorized Multi-Layer Perceptron [23] (F-MLP): Instead of a Gaussian process, this approach models the surrogate model of all data sets by a neural network. To accomplish this task, binary data set indicators are appended to the hyperparameter configuration as the input vector. In the input layer, feature interactions are explicitly modeled as the prediction of a factorization machine instead of a linear model.
5. Hyp-RL: This is the approch proposed by this paper. We train a policy that helps an agent navigate the hyperparameter response surface of several data sets in a way that maximizes a future reward given previous hyperparameter configurations and observed rewards.

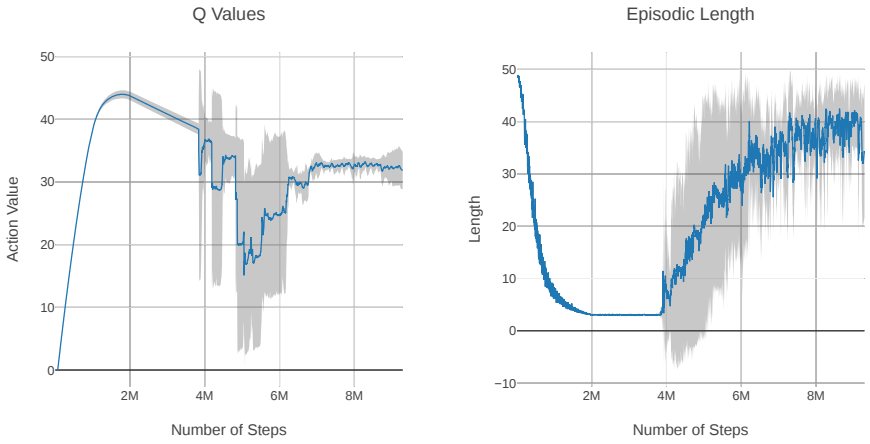
RS, I-GP, and Spearmint cannot carry over information from previous experiments, i.e. on other data sets, whereas F-MLP is a strong baseline that leverages meta-knowledge to reconstruct a scalable hyperparameter response surface. I-GP, Spearmint and F-MLP follow the sequential model-based optimization approach.
Our hyperparameter finding controller has its own set of hyperparameters, too, so we performed a limited grid search over the following parameters: the number of hidden units in the LSTM cell $N_{\text{cells}} \in \{16, 32\}$, the number of neurons for the single fully connected layer $N_{\text{layer}} \in \{32, 64\}$, replay buffer size $N_b \in \{10000, 50000\}$, target update frequency $N_u \in \{500, 1000\}$, training frequency $N_{\text{train}} \in \{4, 8\}$ and finally learning rate $lr \in \{0.001, 0.0001\}$ experiments.

6.2 Evaluation Metrics

We follow the evaluation metrics of the state-of-the-art papers [35]. For the average rank (AR), we rank the performance of the best hyperparameter configuration obtained by a tuning strategy with respect to other methods and then average over all data sets. This highlights the difference between different approaches, however it does not offer insight into how well the selected hyperparameters are with respect to the global optimum. For that, we use the average distance to the minimum (ADTM) as the second evaluation metric. After t trials, ADTM is defined as

$$\text{ADTM}((\Lambda_t^D)_{D\in\mathcal{D}}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{D\in\mathcal{D}} \min_{\lambda\in\Lambda_t^D} \frac{f(D,\lambda) - f(D)^{\min}}{f(D)^{\max} - f(D)^{\min}}$$

with Λ_t^D as the set of hyperparameters that have been selected by a hyperparameter optimization method for data set D in the first t trials and $f(D)^{\min}, f(D)^{\max}$ the range of the loss function on the hyperparameter grid Λ under investigation.



(a) Average Q Values across Splits (b) Average Number of Trials across Splits

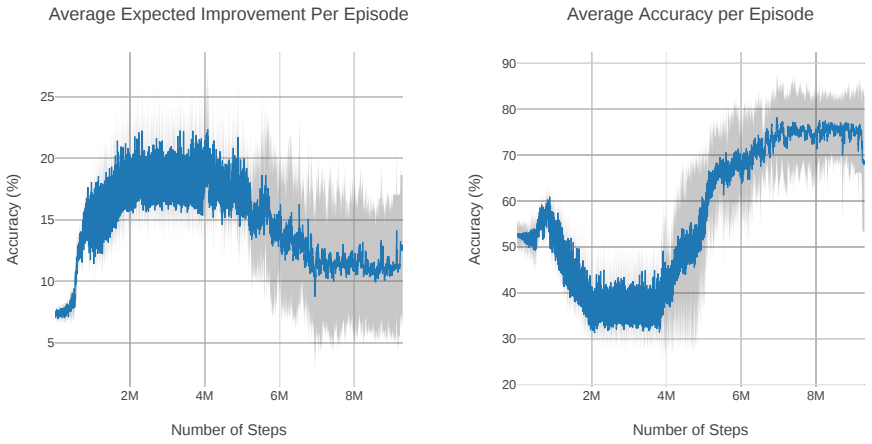
Fig. 2. Learning progress of the proposed policy

6.3 Results and Discussion

Before discussing the performance of the proposed approach for the task hyperparameter tuning, we investigate the learning progress of the RL policy.

Evaluating the RL Policy The learning curves of our RL policy are presented in Figures 2 and 3. For the first several hundred episodes, no actual learning takes place as the agent is left free to randomly explore, i.e. exploration ratio ϵ is relatively high, until the buffer $\mathcal{B}$ is completely filled. We can clearly see from the short episodic length in the early stages, Figure 2(b) that the agent tends to terminate quickly due to the selection of similar hyperparameter configurations consecutively, which can be attributed to the random initialization of the model. However as learning progresses, we notice the episode length, and respectively the episodic reward increases, which highlights the balance between exploration, i.e. selecting new (unseen) actions, hyperparameter configurations, and exploitation, i.e. selecting actions that result in a high reward. We also notice that the action-value increases, which reflects the growing confidence of the model with each action selection. The observed overestimation is a typical issue in function estimation based on Q-learning [30], and has been addressed by several approaches [13, 31] and can be considered future work for this type of formulation.

As the number of trials increases, the aggregated episode reward also increases, however a detailed look at Figure 3(b) shows the average reward per trial and respectively reflects that improved navigation of the policy across hyperparameter response surfaces. The expected improvement (EI) is also investigated. As mentioned earlier, Gaussian-based optimization techniques employ EI as an acquisition function that estimates the performance of hyperparameters before



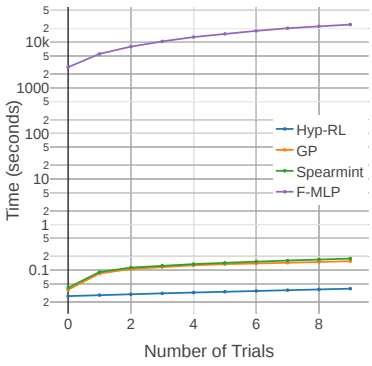
(a) Average EI per Episode (b) Average Observed Reward per Episode

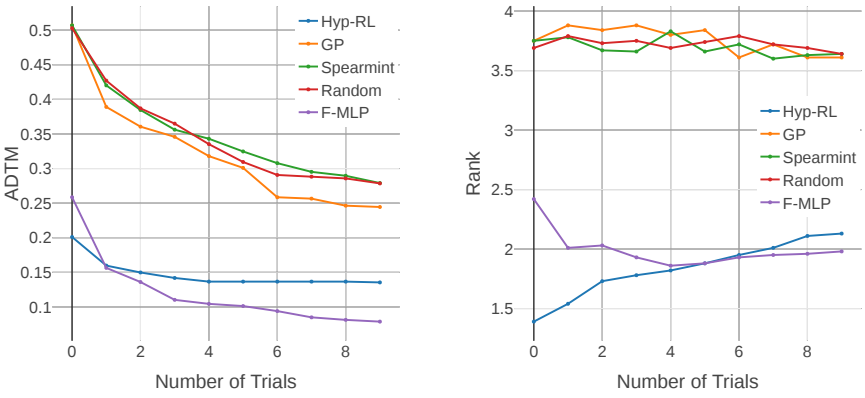
Fig. 3. Performance of the proposed policy

selecting the one with the highest potential. Our reinforcement-learning agent inherently models this effect, as initially the EI overshoots, around the time when the observed reward is below the global optimum, and then as the reward increases, EI naturally decreases as there can only be so much improvement to be achieved as the agent arrives closer at the best configuratiion. The policy network has 115369 parameters and required 24 GPU hours to train for 10 million frames. It is worth mentioning, however that online hyperparameter optimization is immediate as the configurations are selected through via forward pass.

Hyperparameter Tuning Now that we have established that the RL formulation is capable of learning a meaningful policy to navigate different hyperparameter response surfaces, we can move on to hyperparameter tuning on unseen data sets. We investigate the performance of the different optimization methods for the problem of hyperparameter tuning and summarize the results in Figure 5. The acquisition function of the Gaussian-based optimization techniques are usually initialized with randomly evaluated hyperparameter configurations before they start selecting configurations in a meaningful way. However, our policy, which is conditioned on the

Fig. 4. Average time (in seconds) to finish one trial; The y-axis is plotted in log-scale





(a) Average Distance to the Minimum (b) Average Rank

Fig. 5. Hyp-RL consistently outperforms the baselines that do not make use of knowledge transfer. Hyp-RL demonstrates competitive performance against F-MLP in a much more efficient approach.

data metafeatures does not suffer from this problem. From the very beginning, Hyp-RL selects hyperparameter configurations with better performance than the rest, which is indicative of the capability of the proposed policy to scale across data sets. The proposed method also demonstrates competetive performance with respect to F-MLP, which relies heavily on knowledge transfer. It is worth mentioning that since our agent does not learn a surrogate model, inference becomes significantly faster, as opposed to SMBO-based techniques which re-fit a surrogate model after every configuration selection. This additional step, along with the curse of dimensionality, which renders Gaussian processes' complexity cubic in the number of data points, are avoided using our formulation. Even F-MLP, which trains a neural network as a surrogate model, demands a large time budget as compared to other approaches, as seen in Figure 4.

7 Conclusion

In this paper, we presented a new approach for hyperparameter optimization within a reinforcement learning framework. We describe a novel environmental setup represented by the history of observed hyperparameter configurations and their corresponding rewards, and train an agent to select the next hyperparameter setting to be evaluated in a way that maximizes the total reward within a limited budget. The idea extends naturally from Bayesian-based approaches that use an acquisition function as a policy. The proposed approach does not suf-

fer from the cubic dimensionality problem that face Gaussian-based approaches, however the number of actions is still proportional to the grid size. Empirically, our method demonstrates competitive performance w.r.t. the state-of-the-art, especially with a smaller budget, where we notice that from the very first selection, the average distance to the minimum is small. We also show that the agent balances between exploration and exploitation by enforcing termination when an action in an episode is repeated.

Acknowledgment

We acknowledge the funding provided by the "Zentrales Innovationsprogramm Mittelstand" of the German Federal Ministry for Economic Affairs and Energy through the project ADDA.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)
2. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016)
3. Bellman, R.: Functional equations in the theory of dynamic programming–vii. a partial differential equation for the fredholm resolvent. Proceedings of the American Mathematical Society **8**(3), 435–440 (1957)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(Feb), 281–305 (2012)
5. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
6. Cai, Q., Filos-Ratsikas, A., Tang, P., Zhang, Y.: Reinforcement mechanism design for fraudulent behaviour in e-commerce. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
7. Chollet, F., et al.: Keras (2015)
8. Dong, X., Shen, J., Wang, W., Liu, Y., Shao, L., Porikli, F.: Hyperparameter optimization for tracking with continuous deep q-learning. In: CVPR. pp. 518–527. IEEE Computer Society (2018)
9. Falkner, S., Klein, A., Hutter, F.: Bohb: Robust and efficient hyperparameter optimization at scale. arXiv preprint arXiv:1807.01774 (2018)
10. Feurer, M., Letham, B., Bakshy, E.: Scalable meta-learning for bayesian optimization. CoRR **abs/1802.02219** (2018), http://arxiv.org/abs/1802.02219
11. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 1128–1135 (2015), http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10029
12. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)

13. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
14. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
15. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization. pp. 507–523. Springer (2011)
16. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optimization **13**(4), 455–492 (1998). https://doi.org/10.1023/A:1008306431147, https://doi.org/10.1023/A:1008306431147
17. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In: ICML workshop on AutoML. pp. 2825–2830. Citeseer (2014)
18. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
19. Lindauer, M., Hutter, F.: Warmstarting of model-based algorithm configuration. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 1355–1362 (2018), https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17235
20. Mendoza, H., Klein, A., Feurer, M., Springenberg, J.T., Hutter, F.: Towards automatically-tuned neural networks. In: Workshop on Automatic Machine Learning. pp. 58–65 (2016)
21. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
22. Perrone, V., Jenatton, R., Seeger, M.W., Archambeau, C.: Scalable hyperparameter transfer learning. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 6846–6856 (2018), http://papers.nips.cc/paper/7917-scalable-hyperparameter-transfer-learning
23. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II. pp. 87–103 (2015). https://doi.org/10.1007/978-3-319-23525-7_6, https://doi.org/10.1007/978-3-319-23525-7_6
24. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
25. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016)
26. Silva, M.P., do Nascimento Silva, V., Chaimowicz, L.: Dynamic difficulty adjustment on moba games. Entertainment Computing **18**, 103–123 (2017)

27. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484 (2016)
28. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems. pp. 2951–2959 (2012)
29. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable bayesian optimization using deep neural networks. In: International conference on machine learning. pp. 2171–2180 (2015)
30. Thrun, S., Schwartz, A.: Issues in using function approximation for reinforcement learning. In: Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum (1993)
31. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
32. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)
33. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Sequential model-free hyperparameter tuning. In: 2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015. pp. 1033–1038 (2015). https://doi.org/10.1109/ICDM.2015.20, https://doi.org/10.1109/ICDM.2015.20
34. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Hyperparameter optimization machines. In: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA). pp. 41–50. IEEE (2016)
35. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Two-stage transfer surrogate model for automatic hyperparameter optimization. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I. pp. 199–214 (2016). https://doi.org/10.1007/978-3-319-46128-1_13, https://doi.org/10.1007/978-3-319-46128-1_13
36. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Scalable gaussian process-based transfer surrogates for hyperparameter optimization. Machine Learning **107**(1), 43–78 (2018). https://doi.org/10.1007/s10994-017-5684-y, https://doi.org/10.1007/s10994-017-5684-y
37. Xu, C., Qin, T., Wang, G., Liu, T.Y.: Reinforcement learning for learning rate control. arXiv preprint arXiv:1705.11159 (2017)
38. Xu, Z., van Hasselt, H.P., Silver, D.: Meta-gradient reinforcement learning. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 2402–2413 (2018), http://papers.nips.cc/paper/7507-meta-gradient-reinforcement-learning
39. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014. pp. 1077–1085 (2014), http://jmlr.org/proceedings/papers/v33/yogatama14.html
40. Zhao, M., Li, Z., An, B., Lu, H., Yang, Y., Chu, C.: Impression allocation for combating fraud in e-commerce via deep reinforcement learning with action norm penalty. In: IJCAI. pp. 3940–3946 (2018)
41. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)