Hyperparameter Optimization with Factorized Multilayer Perceptrons

Nicolas Schilling, Martin Wistuba, and Lucas Drumond and Lars Schmidt-Thieme

Information Systems and Machine Learning Lab
Universitätsplatz 1, 31141 Hildesheim, Germany
{schilling,wistuba,ldrumond,schmidt-thieme}@ismll.uni-hildesheim.de

Abstract. In machine learning, hyperparameter optimization is a challenging task that is usually approached by experienced practitioners or in a computationally expensive brute-force manner such as grid-search. Therefore, recent research proposes to use observed hyperparameter performance on already solved problems (i.e. data sets) in order to speed up the search for promising hyperparameter configurations in the sequential model based optimization framework.
In this paper, we propose multilayer perceptrons as surrogate models as they are able to model highly nonlinear hyperparameter response surfaces. However, since interactions of hyperparameters, data sets and metafeatures are only implicitly learned in the subsequent layers, we improve the performance of multilayer perceptrons by means of an explicit factorization of the interaction weights and call the resulting model a factorized multilayer perceptron. Additionally, we evaluate different ways of obtaining predictive uncertainty, which is a key ingredient for a decent tradeoff between exploration and exploitation. Our experimental results on two public meta data sets demonstrate the efficiency of our approach compared to a variety of published baselines. For reproduction purposes, we make our data sets and all the program code publicly available on our supplementary webpage.

Keywords: Hyperparameter Optimization, Sequential Model-Based Optimization

1 Introduction

Unfortunately, machine learning models are very rarely parameter-free, as they usually contain a set of hyperparameters which have to be chosen appropriately on validation data. As a simple example, the number of latent variables in a matrix factorization cannot be determined using gradient descent as firstly, it is not explicitly given in the objective function and secondly is not a continuous but a discrete parameter. Additionally, the choice of kernel function for an SVM can also be understood as hyperparameter, where gradient descent approaches fail. Besides being a parameter of learned model, hyperparameters can also be part of the objective function, such as regularization constants. Moreover, they can also be part of the learning algorithm that is used to optimize the model for the objective function, for example the steplength of a gradient based technique or the threshold of a stopping criterion. Finally, even the choice of preprocessing can be viewed as a hyperparameter. Some of these hyperparameters are continuous, some are categorical, but what they all have in common is that there is no efficient learning algorithm for them. Therefore many researchers rely on searching them on a grid, which is computationally very expensive, as with growing data and growing complexity of models the optimization part usually requires a lot of time.

The performance of a model on test data trained with specific hyperparameters depends on the data set where the machine learning model should be learned, and therefore hyperparameter optimization is usually started from the scratch for each new data set. Thus, possibly valuable information of past hyperparameter performance on other data sets is ignored. Recent work proposes to use this information to be able to perform a more efficient and faster hyperparameter optimization than before [2]. To accomplish this, the sequential model-based optimization framework is applied, where a surrogate model is learned to predict hyperparameter performances in a first step. Then an acquisition function is queried to choose the next hyperparameter to test while maintaining a reasonable tradeoff between exploration and exploitation. As the prediction of the surrogate model can be done in constant time, hyperparameters can be optimized in a controlled way, resulting in less runs of the actual learning algorithm until a promising configuration is found.

This paper targets the problem of hyperparameter learning and more generally model selection across different data sets. We propose to use a multilayer perceptron as surrogate model and show how it can be learned to also include hyperparameter performances of data sets observed in the past. Additionally, we propose a factorized multilayer perceptron that contains a factorization part in the first layer of the network to directly model interactions of hyperparameters and datasets. For both of these surrogates, we propose different ways of assessing their uncertainty which is a key ingredient for hyperparameter optimization in the SMBO framework. Finally, we conduct three different experiments, where the first shows the capability of a surrogate model to predict the response surface. The second experiment compares different ways of estimating prediction uncertainty, and the last demonstrates surrogate performance in a standard SMBO setting against a variety of published baselines.

2 Related Work

In the recent years, the field of hyperparameter optimization has attracted more and more interest from the research community. The current state-of-the-art can be roughly classified into four different method categories.

At first, there are *exhaustive* methods that search the hyperparameter space exhaustively and therefore are usually conducted on a compute cluster as they are computationally expensive. The most simple and most widely used method is a grid search. Another exhaustive method was proposed by [3], where hyperparameters are not sampled on a grid but using probability distributions and work well in cases of low effective dimensionality, i.e. the case where one hyperparameter does not affect the final performance as much as others.

Secondly, there are the *model-specific* methods that optimize hyperparameters for a specific model choice, such as [1] and [7], which is tailored to least squares SVM. For a regression with small sample size, the work of [5] can be applied. Furthermore, [10] deal with hyperparameter optimization in the case of semi-supervised learning. There is a plethora of model-specific methods, but their common downside is that they are tailored to a chosen model class and therefore cannot be applied in general.

A third class of methods to optimize hyperparameter is based on evolutionary algorithms, for instance [11] optimizes kernel hyperparameters of an SVM and therefore can be seen as also a *model-specific* method.

Lastly, a more recent class of hyperparameter optimization methods is based on the *sequential model-based optimization* (SMBO) [9] framework which stems from black-box optimization. The choice of this framework is quite reasonable, as the function that maps hyperparameters for a given model on a given data set to the final validation performance is certainly a black-box. All SMBO methods learn a surrogate model on given hyperparameter choices to infer the performance of unknown hyperparameters, where the next hyperparameter to test is chosen based on the prediction of the surrogate model and its uncertainty. Gaussian processes are used as surrogate model in [19], but are not used to include hyperparameter performances on other data sets. Moreover, SMAC [8] employs random forests as surrogate model, but also does not learn across data sets. The first paper that proposed to include past hyperparameter performances for SMBO-based hyperparameter optimization is [2], their method SCoT employs an SVM Rank as surrogate and uses a second stage Gaussian process that is learned on the output of SVM Rank to allow for uncertainty in the prediction. Another work uses past hyperparameter performances to come up with a good initialization for Bayesian hyperparameter optimization [6]. The work in [12] chooses hyperparameters and models by using active testing on the past observations, which can be seen as SMBO with a very specific choice of surrogate and acquisition function.

Finally, [21] uses a Gaussian process with a more sophisticated choice of kernel function, which is able to generalize over past performances on other data sets, which is very close to the multi-task Gaussian process approach used by [20].

Compared to exhaustive methods, SMBO algorithms are more efficient in the overall number of hyperparameters that have to be evaluated; compared to model-specific methods, they may be applied for every model choice. Moreover, SMBO algorithms learn a model for the hyperparameter space, which itself is very interesting as it gives a deeper understanding of hyperparameter interactions.

3 Background

In this section, we will first introduce the problem setting, to then discuss important properties of surrogate functions. Afterwards, we propose three new surrogates and finally show how to assess their prediction uncertainty.

3.1 Problem Setting

Let us define by $\mathcal{D}$ the space of all data sets. Furthermore, for a fixed model class $\mathcal{M}$, let us denote by $\mathcal{A}_\lambda$ a machine learning algorithm as a mapping $\mathcal{A}_\lambda : \mathcal{D} \longrightarrow \mathcal{M}$ that maps training data $D^{\text{train}} \in \mathcal{D}$ to a learned model $M_\lambda \in \mathcal{M}$ for a given hyperparameter configuration $\lambda \in \Lambda$ by searching through $\mathcal{M}$ and finding a model that minimizes:

$$\mathcal{A}_\lambda(D^{\text{train}}) := \underset{M_\lambda \in \mathcal{M}}{\arg\min}\ \mathcal{L}(M_\lambda, D^{\text{train}}) \ . \tag{1}$$

Usually, $\Lambda = \Lambda_1 \times \ldots \times \Lambda_p$, where Λ_i may be a continuous or discrete space. Having learned a model for a given hyperparameter configuration λ, the *hyperparameter optimization problem* can be stated as choosing the $\lambda^\star$, for which the associated model $M_{\lambda^\star}$ has a minimal error on a validation set

$$\lambda^\star := \underset{\lambda \in \Lambda}{\arg\min}\ \mathcal{L}(\mathcal{A}_\lambda(D^{\text{train}}), D^{\text{val}}) := \underset{\lambda \in \Lambda}{\arg\min}\ f(\lambda) \ . \tag{2}$$

Thus, the problem of hyperparameter optimization can be stated as minimizing computationally expensive black-box function f over Λ. As discussed earlier, these hyperparameters cannot be optimized using standard means, as there is no knowledge of f, and therefore exhaustive search methods such as grid search partition Λ into a discrete subset $G \subset \Lambda$ and optimize f over G, which takes a lot of time as many hyperparameter configurations have to be tested.

A more recent class of hyperparameter optimization methods follows the SMBO framework, where on known hyperparameter responses of f on a discrete subset G, a surrogate model $\Psi(\lambda)$ is learned to most accurately predict f. Once this is accomplished, Ψ is then used to predict promising hyperparameter configurations to choose next, while maintaining a tradeoff between exploration and exploitation. Exploration drives the choice of choosing *distant* hyperparameter configurations, where the surrogate model Ψ is very uncertain. Exploitation chooses hyperparameters in well-known regions of f, which might find local but not necessarily global optima. Therefore, a decent tradeoff between exploration and exploitation is desired.

As [2] proposed, this procedure is not limited to only one data set and can therefore be expanded in a way that Ψ learns the response for given hyperparameters across many data sets $D \in \{D_1, \ldots, D_m\}$ where the response surface has already been observed, to then use the gained knowledge to optimize hyperparameters for a new data set D^{new}. In order to learn such a surrogate, we now denote the input of Ψ and f by x, which also contains dataset information.

$$x = (\lambda, d, m) \ , \tag{3}$$

where d is a binary dataset indicator and for a given data set D_j defined as

$$d(D_j) = (d_1, ..., d_m) \quad d_i = \delta(i = j) \ , \tag{4}$$

for δ being the indicator function. By m or more formally $m(D_j)$, we denote descriptive features for data set D_j. They are usually called meta features and can be simple statistics, such as number of attributes, number of instances [2] [21] or more complex features such as the classification accuracy of a decision tree or a linear SVM [15]. Finally, an observation history $\mathcal{H}$ is built to contain all hyperparameter responses for $\lambda \in G$ for all data sets D where hyperparameter optimization has already been accomplished.

The resulting procedure can be seen in Algorithm 1. At the beginning of one trial, we fit the surrogate model Ψ to the given observation history. Then we query an acquisition and choose its maximum to be the next hyperparameter configuration to test.

The most widely used acquisition function is the expected improvement (EI) [9], which given a currently best hyperparameter configuration x^{best} is defined as

$$EI(x) := \int_0^\infty I \cdot p(I \mid \Psi, x^{\text{best}}) \, \mathrm{d}I \ . \tag{5}$$

Afterwards, f is evaluated for the proposed hyperparameter configuration and the tuple $(x, f(x))$ is then added into the observation history $\mathcal{H}$.

Algorithm 1 Sequential Model-based Optimization Across Data Sets

```
Input: Hyperparameter space $\Lambda$, observation history $\mathcal{H}$, target data set $D^{\text{new}}$, number of iterations $T$, acquisition function $a$, surrogate model $\Psi$.
Output: Best hyperparameter configuration $x^{\text{best}}$ for $D^{\text{new}}$
1: for $t = 1$ to $T$ do
2:     Fit $\Psi$ to $\mathcal{H}$
3:     $x^{\text{new}} = \arg\max_x a(x, \Psi(x))$
4:     Evaluate $f(x^{\text{new}})$
5:     if $f(x^{\text{new}}) > f(x^{\text{best}})$ then
6:         $x^{\text{best}} = x^{\text{new}}$
7:     $\mathcal{H} = \mathcal{H} \cup (x^{\text{new}}, f(x^{\text{new}}))$
8: return $x^{\text{best}}$
```

3.2 Requirements for a Surrogate Model

We have identified three main ingredients for a surrogate model to be able to accurately predict hyperparameter responses across data sets.

Nonlinearity. Usually, the hyperparameter response f is highly nonlinear and therefore dictates a surrogate model to also adapt this property. We will see later in our experiments, that even nonlinear models can fail to reproduce the response surface, if the employed basis functions are not well chosen and thus the model does not offer enough complexity.

Prediction Uncertainty. If we fully trust the surrogate model Ψ in its predictions, i.e. use the identity as acquisition function and therefore always query the hyperparameter configuration with the best predicted performance, we are doomed to fail because only exploitation of the model is done, meaning that we always stay in a region of the hyperparameter space Λ where we have started. This is due to the fact that the surrogate model is learned on a few observations of f and therefore will not accurately predict every hyperparameter performance. To circumvene this issue, acquisition functions such as the EI are employed, that try to balance exploration and exploitation. In order for EI to work, the surrogate model needs a predictive posterior, i.e. a probability distribution on $\Psi(x)$ that can be queried for how uncertain the prediction is, thus forming the second key ingredient for a decent surrogate model.

Shared and Data Set Specific Parameters. To successfully learn surrogate model across different problem aspects (i.e. data sets), it should be able to distinguish between these to learn specific data set characteristics. A natural way is to add binary dataset indicators as it was done above. However, to be able to learn more than only a data set bias with these features, we aim to learn factorization models that can also model the interactions of hyperparameters with datasets, hyperparameters with model choices and so on. In this way, we automatically learn latent characteristics of a data set.

Another way to let the surrogate learn across problems is to add meta features that describe the problems, where for data sets, many meta features have already been proposed. If we think one step further and want to generalize over other problem aspects such as preprocessing, choice of model, etc. we have to come up with meta features describing these problem aspects, which does not seem reasonable to us anymore.

3.3 Proposed Models

Factorization Machines. The first surrogate model we propose is a factorization machine which was introduced in [16]. It works as a generalization of factorization models and can mimic all different kind of models if the features are preprocessed in a certain way. To every given feature, i.e. in our case hyperparameters and binary data set indicators, the model associates a vector of $K \in \mathbb{N}$ latent features. The final prediction is then given through

$$\Psi(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \quad . \tag{6}$$

The model is also sometimes called a factorized polynomial regressor, as in its essence it is a polynomial regression of degree two, if one sets $w_{i,j} := \langle v_i, v_j \rangle$, though by factorizing this weight the model can be fitted more effectively in sparse settings as the parameters have more instances to learn from. Moreover, by applying a factorization machine, we are also able to learn interactions of data sets and hyperparameters. Ultimatively, we are even able to use continuous features, such as meta features, which a standard matrix factorization model would not allow us to do.

Multilayer Perceptron. The next model we propose to use as a surrogate is the multilayer perceptron, which may be more commonly named as feedforward neural network. A multilayer perceptron consists of L many layers, where each layer comprises N many nodes and is fully connected to the next layer, forming the structure of a directed acyclic graph. At the beginning, $x = x^0$ is used as input for the first layer. The k-th output of a layer l is then defined as

$$x_k^l = \sigma^{l-1}\Big(w_{0,k}^{l-1} + \sum_{i=1}^{n} w_{i,k}^{l-1} x_i^{l-1}\Big) = \sigma(s_k^l) \quad , \tag{7}$$

thus acts as input for the subsequent layer, where σ^{l-1} is a sigmoid function, in our case we used the hyperbolic tangent, and w are the weights, i.e. parameters of the model. In this way, the information is propagated forward until predictions are made in the final layer. As our task is regression, the final prediction will be one-dimensional and σ^{L-1} is defined as the identity function

$$\Psi(x) = w_0^{l-1} + \sum_{i=1}^{n} w_i^{l-1} x_i^{l-1} \ . \tag{8}$$

Let us have a closer look at what the model does with binary data set indicators. In the input layer, the multilayer perceptron learns exactly N many weights per each data set, which act as a data set bias, and therefore can be used by the model to generalize across data sets. From the second layer onwards, the model acts independently from the data set as all features then are fitted globally. Nevertheless, interactions can still implicitly be modeled throughout the learning process of the network. The question to answer is whether an explicit modelling of these interactions such as in a factorization machine is better than an implicit one.

Factorized Multilayer Perceptron. Finally, the third surrogate model proposed by us is a mixture of both previous models and is therefore called a factorized multilayer perceptron. Closely related to a multilayer perceptron, it also consists of L many layers, where each layer comprises N many nodes, also the final prediction is the same as given in Equation 8. The only difference is that here we explicitly model feature interactions in the input layer, by using the prediction of a factorization machine instead of a linear model. Thus, the k-th output of the first layer is defined as

$$x_k = \sigma(s_k^1) = \sigma\Big(w_{0,k} + \sum_{i=1}^{n} w_{i,k} x_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle v_{i,k}, v_{j,k}\rangle x_i x_j\Big) \ , \tag{9}$$

where $v_{i,k} \in \mathbb{R}^K$ are the latent characteristics of feature i for the output k. Note that we only do this in the first layer and therefore dropped the layer dependencies to avoid unnecessary clutter.

In this way, we explicitly model the feature interactions of a factorization machine into the first layer of a multilayer perceptron, as the binary data set indicators are naturally only given in the input layer. This model can be learned straightforward using backpropagation [17], the only difference is that we have to consider the update for the latent feature vectors as well. The resuling procedure can be viewed in Algorithm 2, where the updates are denoted for a stochastic gradient descent approach. We dropped the usual momentum term to avoid clutter, the implementation of such a term is straightforward.

3.4 Estimating Prediction Uncertainty

The proposed surrogate models are still lacking the ability to predict under uncertainty, which is a key ingredient for running SMBO with a decent tradeoff between exploration and exploitation. SMAC uses a random forest, i.e. a bagged ensemble of decision trees, and is thus able to compute a mean and a standard deviation by assuming that the

Algorithm 2 SGD-Backpropagation for Factorized Multilayer Perceptron

```
Input: Data Set $D$, Loss function $\mathcal{L}$, step length $\eta > 0$.
1: repeat
2:     Draw $(x, y) \in D$
3:     Predict $\hat{y}(x)$
4:     Compute $\delta_k^l = \frac{\partial \mathcal{L}(\hat{y}(x), y)}{\partial s_k^l} \cdot \frac{\mathrm{d}\sigma}{\mathrm{d}s_k^l}$ for all layers $l$ and nodes $k$
5:     Update $w_{i,k}^l = w_{i,k}^l - \eta \delta_k^l x_i$
6:     Precompute $\mu_j^k = \sum_{i=1}^n v_{i,j}^k x_i$
       Update $v_{i,j}^k = v_{i,j}^k - \eta \delta_k^l (x_i \mu_j^k - v_{i,j}^k x_i^2)$
7: until Convergence
```

prediction of the ensemble is Gaussian distributed. Alternatively, SCoT uses a ranking approach and learns a Gaussian process on the ranked output, thus obtaining prediction uncertainty through the Gaussian process.

By treating the abovely proposed surrogate models in a Bayesian setting as it is described in [13], it is possible to deduce prediction uncertainty using a Taylor approximation of the objective function. Let us denote by w a vector of all parameters of Ψ, including biases, weights and possibly latent characteristics. Assuming a Gaussian prior with covariance α^{-1} of the form

$$p(w) = \mathcal{N}(w \,|\, \mathbf{0}, \alpha^{-1}\mathbf{I}) \ , \tag{10}$$

the posterior distribution of the parameters w given the data D, α and data set noise σ^2 can be estimated by using a second-order Taylor decomposition on the objective function. The resulting parameter posterior is approximated as

$$p(w \,|\, D, \alpha, \sigma^2) \approx \mathcal{N}(w \,|\, w^\star, A^{-1}) \ , \tag{11}$$

where $A = \beta H + \alpha I$, and H is the Hessian matrix of the loss on the data set. The densitiy of the predictive posterior can then be written as

$$p(y \,|\, x, D, \alpha, \sigma^2) = \int \mathcal{N}(y \,|\, \Psi(x, w), \sigma^2)\mathcal{N}(w \,|\, w^\star, A^{-1})\mathrm{d}w \ . \tag{12}$$

As [13] argues, this integral is not feasible to compute because of the nonlinearity of Ψ, thus a first order approximation is done around $w^\star$ yields

$$\Psi(x, w) \approx \Psi(x, w^\star) + g^\top (w - w^\star) \quad \text{where} \quad g = \nabla_w \Psi(x, w)\big|_{w=w^\star} \ . \tag{13}$$

Finally, the predictive posterior can be written as Gaussian

$$p(y \,|\, x, D, \alpha, \sigma^2) = \mathcal{N}(y \,|\, \Psi(x, w^\star), \sigma^2 + g^\top A^{-1} g) \ . \tag{14}$$

In conclusion, to predict the uncertainty of Ψ for an instance x, we need to estimate the Hessian of the loss of Ψ on D, and a gradient g depending on x. The latter is easy at it only involves a computation of the gradient, which is for a multilayer perceptron a forward and a backward pass through the network.

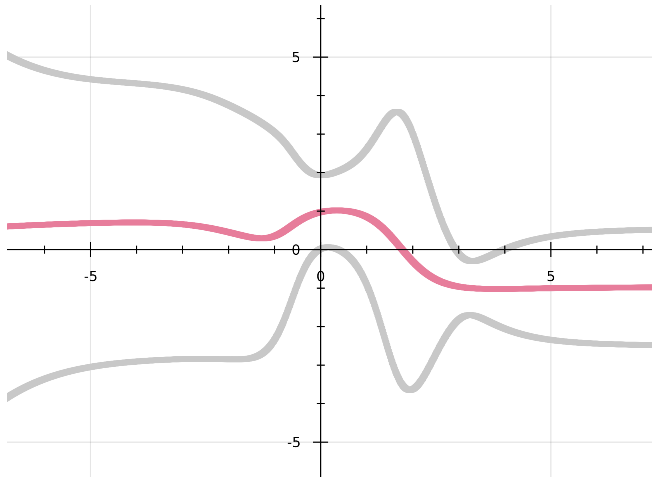

Fig. 1. Predictive Posterior of a multilayer perceptron learned on $(x_1, y_1) = (0, 1)$ and $(x_2, y_2) = (\pi, -1)$. The red line shows the mean, the grey line shows one standard deviation (Best viewed in color)

To compute the inverse Hessian in an analytic fashion is usually not feasible as computing one entry of the Hessian involves a pass over the whole data and then inverting the resulting matrix has an effort that is cubical in the number of parameters, i.e. the dimensionality of w. Out of this reason, we seek to approximate the inverse of the Hessian directly by using a sum of outer products as it is exposed in [4]. As the target loss is least squares, the Hessian can be written as

$$H = \sum_{(x,y) \in D} \nabla \Psi(x, w) \nabla \Psi(x, w)^\top + \sum_{(x,y) \in D} (y - \Psi(x, w)) \nabla\nabla \Psi(x, w) \ . \tag{15}$$

As [4] outlines, for a carefully learned model the second sum can be neglected as the quantity $(y - \Psi(x, w))$ is close to zero. Thus, H can be approximated using only the first term which is a sum of outer products. The inverse of H can directly be computed in an iterative fashion over the data set using the Sherman-Woodbury formula and starting with an initialization of $H^{-1} = \alpha^{-1} I$. In this way, we we effectively compute the inverse of $H + \alpha I$, which is exactly the matrix we seeked to invert. A one dimensional example can be seen in Figure 1, where a multilayer perceptron is learned on two data points.

As an alternative approach, we simply compute an ensemble of surrogates and predict the uncertainty using the estimated mean and standard deviation of all the predictions, as it is also done by SMAC. The resulting variance then stems from differently learned models, which in the case of SMAC results from bagging. As the most simple approach, we propose to learn an average ensemble, where the resulting variance stems only from different initializations of the surrogate model, which is reasonable if the whole optimization problem is not convex and therefore yields different solutions.

4 Experiments

To assess the performance of our proposed surrogate models we will conduct three different experiments on two meta data sets that we have created on our own.

4.1 Meta Data Set Creation

For 25 randomly chosen classification data sets of the UCI repository[1], we merged existing splits into one data set, then shuffled the data set and created one split where 80% of the data was used for training, and the remaining 20% for testing. To create the first data set, we learned AdaBoost[2] by employing decision products as weak learners of the ensemble. This involves two hyperparameters, the number of iterations I and the number of product terms M. For all 25 classification datasets, the resulting test accuracy was recorded when learning AdaBoost with hyperparameters $I \in \{2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$ and $M \in \{2, 3, 4, 5, 7, 10, 15, 20, 30\}$ which yields 108 meta instances per data set.

The second meta data set was created by learning an SVM[3] with four involved hyperparameters, namely the choice of kernel between linear, polynomial and Gaussian, the tradeoff parameter C, the degree of the polynomial d and the width γ of the Gaussian kernel. If a hyperparameter is not involved, for example the polynomial degree for the SVM with Gaussian Kernel, we set it to zero in the meta instances. Again, the test accuracy was precomputed on a grid consisting of hyperparameters $C \in \{2^{-5}, \dots, 2^{6}\}$, $d \in \{2, \dots, 10\}$ and $\gamma \in \{0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000\}$, which results in a meta dataset of 288 instances per data set. By including also the choice of kernel for the SVM, this meta data set can already be viewed as cross-model, since we try to learn not only the hyperparameters but also a model choice.

As they are an indispensable part of the competing methods, we also added the meta features used by [2] and [21] to our meta data sets. These encompass the number of classes c, the logarithm of the number of predictors $\log(p)$ and finally the logarithm of the quotient of dataset instances and number of predictors $\log(|D|/p)$. Finally, we scaled the meta features to have values in $[0, 1]$.

Table 1. Confidence intervals of the resulting RMSE of experiment 1 for all models when reconstructing the response surface

	RF	SVR	FM	MLP	FMLP
SVM	0.0997±0.028	0.1110±0.020	0.1041±0.029	0.0596±0.013	**0.0550**±0.016
AdaBoost	0.0462±0.012	0.0840±0.009	0.0579±0.015	0.0380±0.008	**0.0377**±0.009

[1] http://archive.ics.uci.edu/ml/index.html

[2] http://www.multiboost.org

[3] http://svmlight.joachims.org

4.2 Experiment 1: Reconstruction of the Response Surface

As a first experiment, we seek to learn models to reconstruct the hyperparameter response surface in order to determine their usefulness for hyperparameter optimization in the sequential model-based optimization framework. The evaluation protocol is designed in a leave-one-out fashion, where we learn a surrogate model on 24 response surfaces plus a few observations of hyperparameter responses of the new dataset to then predict the full response surface. For the test data, we took 4% of the responses as training data, 10% as validation data for hyperparameter optimization of the surrogate model and used the remaining 86% as test data.

As surrogate models, we used a random forest (RF), a support vector regression (SVR), a factorization machine (FM), a multilayer perceptron (MLP) and a factorized multilayer perceptron (FMLP). For the RF, we used the implementation in MLTK[4], for the support vector regression we used the implementation by Joachims [5] . All remaining models were implemented in Java by ourselves and optimized for minimal root mean squared error (RMSE). Hyperparameters of all models have been optimized using grid search, for more detail on the grids, we refer to our supplementary webpage [18]. The resulting 95% confidence intervals of the leave-one-out cross validation are reported in Table 1. Clearly, both neural network models outperform the other models by a considerable margin, where the FMLP tends to achieve the best performance, although the lift to a normal MLP is marginal and not statistically significant. It is observable that results on AdaBoost are much better, therefore indicating that the hyperparameter optimization problem for this specific model is easier than for an SVM. We acknowledge that also the lift of our models compared to the RF is not statistically significant.

Unexpectedly, a factorization machine fails to reconstruct the response surface as its RMSE is clearly worse than of the MLP, for instance. This stems from the fact that its expressivity in this setting is rather limited as a standalone model, which can be demonstrated by a small example. If we consider an instance out of the SVM meta data set with RBF kernel, then, leaving out the meta features, the prediction can be shown to have the form:

$$\Psi(x) = w_0 + w_C C + w_\gamma \gamma + w_{C,\gamma} C\gamma \ , \tag{16}$$

which has the geometrical form of a hyperbolic paraboloid. This clearly fails to reproduce any complex response surface and therefore a plain Factorization Machine is not a good candidate for a surrogate model.

4.3 Experiment 2: Uncertainty Estimation in SMBO

In this experiment we compare the a multilayer perceptron in two different scenarios. Before we proceed, we first introduce the evaluation metrics that we applied in the SMBO setting.

Evaluation Metrics for SMBO. We use two different evaluation metrics, at first the average rank of the individual models, where the second metric is the average hyperparameter rank.

[4] http://www.cs.cornell.edu/ yinlou/projects/mltk/

[5] http://svmlight.joachims.org/

Average Rank. The average rank among different tuning strategies ranks all tuning strategies by the best hyperparameter configuration they have found so far, where ties are solved by granting the average rank. If we for example have four different tuning strategies where one obtains an accuracy of 0.9, two others obtain 0.8 and the third obtains an accuracy of 0.7, we associate the ranks 1, 2.5, 2.5, 4.

Average Hyperparameter Rank. By average hyperparameter rank we do not compare between methods but between hyperparameters found. For a fixed data set D, the hyperparameter responses are ranked according to their performance, then the average hyperparameter rank is simply the average over all folds.

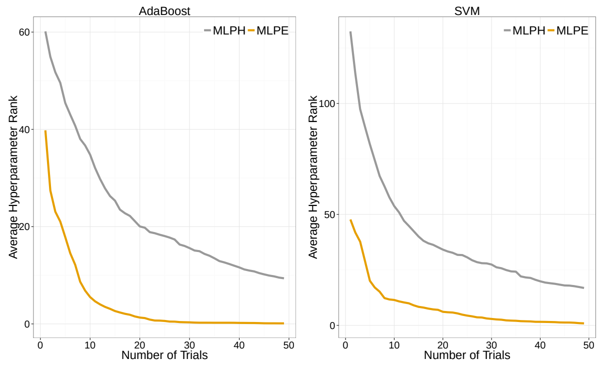

Fig. 2. Development of the average hyperparameter rank with increasing numbers of trials. Clearly, the convergence of the ensemble MLP is much faster than using the inverse Hessian (Best viewed in color)

Experiment Setup and Results. We evaluate at first the MLP when computing predictive uncertainty by means of an inverse Hessian matrix (MLPH) as proposed in section 3.4, opposed to the approach where the uncertainty is assessed by using an average ensemble (MLPE). The development of the average hyperparameter rank for both the SVM and the AdaBoost data set is plotted in Figure 2, where results are averaged over 10 runs for both methods. As the figure indicates, the convergence of MLPE is much faster, which is due to several reasons. At first, having an ensemble yields a better approximation of the response surface itself. Secondly, the predicted uncertainty does not seem to help in exploring the hyperparameter space, which then results in an overall small convergence. This is due to the fact, that the inverse Hessian is only approximated in many ways, if we consider Equation 15, which is already a Tailor approximation, the second term was neglected for a carefully trained model. By adding the new target data set to our overall loss, this assumption is likely not valid anymore as the surrogate has almost no knowledge of the new data set and therefore cannot be perfectly trained for it. Moreover, if we consider Figure 1, it still shows quite a bit uncertainty around points which have already been evaluated which might lead to exploitation. The uncertainty only decreases if more points in this region are queried, a luxury that we are not permitted in the SMBO scenario.

Consequently, as it also takes more time to compute the inverse Hessian than learning an ensemble model, we propose to follow the latter strategy. This is also done in the next experiment, where MLP and FMLP are used in an ensemble fashion.

4.4 Experiment 3: Sequential Model Based Optimization

As a final experiment, we test our surrogate models, the MLP and FMLP in the SMBO setting, where we again perform a leave-one-out cross validation over data sets. Hyperparameters of baseline models have been specifically optimized for the average hyperparameter rank, for the models proposed by us we used the optimal hyperparameters of the first experiment. To consider initialization variance, all results are averaged over 10 runs, except for the random search where 1000 runs were executed. We will briefly describe the competing methods in the following.

Tuning Strategies.

Random Search. This is a tuning strategy that neither uses a surrogate model nor uses an acquisition function. It was first proposed in [3], and has proven to work well in scenarios of low effective dimensionality.

Independent Gaussian Process (I-GP). This tuning stategy uses a Gaussian Process with a Gaussian kernel as surrogate model. It does not employ any information of hyperparameter responses on other datasets, therefore does not learn across data sets.

Sequential Model-based Algorithm Configuration++ (SMAC++). SMAC [8] uses a random forest as surrogate model, we denote by SMAC++ a random forest that also incorporates meta features and therefore is able to take hyperparameter performance of other datasets into account.

Surrogate-based Collaborative Tuning (SCoT). This is the tuning strategy proposed by [2]. Its surrogate model is based on a two stage approach, as it first learns a ranking using SVM$^{\text{RANK}}$ with an RBF Kernel. Then, a Gaussian Process is learned on the output of the ranking. As indicated by [21], learning an RBF Kernel takes too much time, we followed their suggestion and learned a linear kernel instead.

Gaussian Process with MKL (MKL-GP). As proposed by [21], this tuning strategy is based on a Gaussian Process as surrogate model where the kernel is a mixture of an SE-ARD Kernel combined with a kernel modelling the distances between data sets, which is estimated based on the meta features.

Multilayer Perceptron (MLP). Our tuning strategy based on a multilayer perceptron that associates weights to binary data set indicators. We learn an average-ensemble of 100 models to assess uncertainty in the prediction. The weights of the network are initialized using the Nguyen-Widrow [14] initialization for faster convergence of the model.

Factorized Multilayer Perceptron (FMLP). The final tuning strategy that we propose. It is similar to the multilayer perceptron, but uses an additional factorization part in the first layer to directly model all interactions of hyperparameters, data sets and meta features. As for the MLP, we again learn an ensemble of 100 models to predict uncertainties, the network weights are being initialized as for the MLP, the latent factors are initialized using a Gaussian prior.

Optimal. This is an artificial surrogate model that always predicts the best hyperparameter configurations and is plotted for orientation purposes.

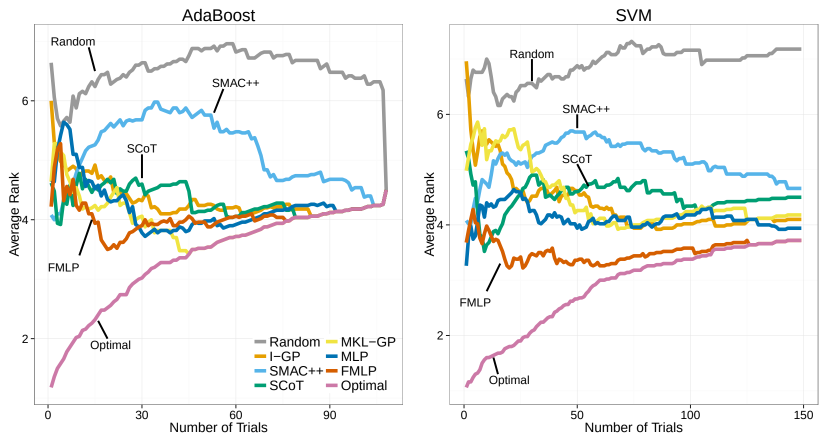

Fig. 3. Development of the average rank with increasing numbers of trials (Best viewed in color or online [18])

Results. Figure 3 shows the development of the average rank with increasing number of trials. For both meta data sets the first ten trials of SMBO encompass some noise as basically all competing methods start out equally good (or bad). Afterwards, we see that the FMLP performs best in the arguably most interesting region, where there is a proper tradeoff between the optimal hyperparameter found so far and the overall used percentage of the grid, as in the beginning, there is a lot of noise involved and in the end the improvement in hyperparameter performance degrades. Note that the MLP also is very competitive and therefore empirically already a decent tuning strategy.

Figure 4 demonstrates the development of the average hyperparameter rank. This chart gives an impression of how fast the actual performance of proposed hyperparameter configurations converges to the optimal configuration on the grid. Again, we observe that the FMLP works best for both the AdaBoost and the SVM data set, which none of the baselines accomplish, as for example SCoT only works well on the SVM data set. We acknowledge the good results of an independent Gaussian Process on the AdaBoost data, which degrades on the SVM data set. This may be due to the higher complexity of the SVM data as it not only contains different hyperparameters but also model choices. On AdaBoost, the MKL-GP also performs really well, but does not show the same performance when applied to the SVM data set.

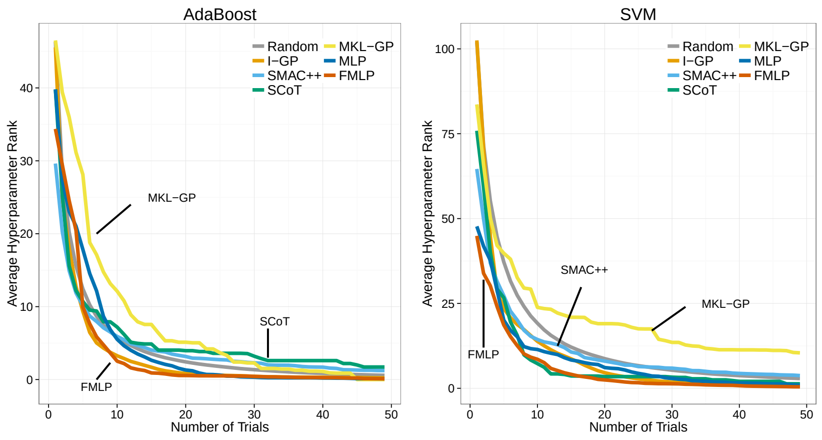

Fig. 4. Development of the average hyperparameter rank with increasing numbers of trials (Best viewed in color or online [18])

5 Conclusions

We proposed to use multilayer perceptrons as surrogate model and improved them by using a factorization approach in the first layer. Our experimental results on two public meta data sets show that the FMLP outperforms current state of the art surrogate models in hyperparameter optimization using the SMBO framework. Moreover, we evaluated two different strategies of assessing prediction uncertainty and showed empirically, that the simpler and faster strategy works better. For future work, we want to extend our meta data sets to a cross-model problem by using a plethora of base models and then try to learn a common latent feature space for datasets and models. We argue that this is the next step to be made in hyperparameter optimization.

Acknowledgments The authors gratefully acknowledge the co-funding of their work by the German Research Foundation (DFG) under grant SCHM 2583/6-1.

References

1. Adankon, M.M., Cheriet, M.: Model selection for the LS-SVM. Application to handwriting recognition. Pattern Recognition 42(12), 3264–3270 (2009)

2. Bardenet, R., Brendel, M., Kegl, B., Sebag, M.: Collaborative hyperparameter tuning. In: Dasgupta, S., Mcallester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML-13). vol. 28, pp. 199–207. JMLR Workshop and Conference Proceedings (May 2013)
3. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, 281–305 (Feb 2012)
4. Bishop, C.M., et al.: Pattern recognition and machine learning, vol. 4. springer New York (2006)
5. Chapelle, O., Vapnik, V., Bengio, Y.: Model selection for small sample regression. Machine Learning 48(1-3), 9–23 (2002)
6. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
7. Guo, X.C., Yang, J.H., Wu, C.G., Wang, C.Y., Liang, Y.C.: A novel ls-svms hyper-parameter selection based on particle swarm optimization. Neurocomput. 71(16-18), 3211–3215 (Oct 2008)
8. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization. pp. 507–523. LION'05, Springer-Verlag, Berlin, Heidelberg (2011)
9. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. of Global Optimization 13(4), 455–492 (Dec 1998)
10. Kapoor, A., Ahn, H., Qi, Y., Picard, R.W.: Hyperparameter and kernel learning for graph based semi-supervised classification. In: Advances in Neural Information Processing Systems. pp. 627–634 (2005)
11. Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., Konen, W.: Tuning and evolution of support vector kernels. Evolutionary Intelligence 5(3), 153–170 (2012)
12. Leite, R., Brazdil, P., Vanschoren, J.: Selecting classification algorithms with active testing. In: Machine Learning and Data Mining in Pattern Recognition, pp. 117–131. Springer (2012)
13. Murphy, K.P.: Machine learning: a probabilistic perspective. MIT press (2012)
14. Nguyen, D., Widrow, B.: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Neural Networks, 1990., 1990 IJCNN International Joint Conference on. pp. 21–26. IEEE (1990)
15. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: In Proceedings of the Seventeenth International Conference on Machine Learning. pp. 743–750. Morgan Kaufmann (2000)
16. Rendle, S.: Factorization machines. In: Data Mining (ICDM), 2010 IEEE 10th International Conference on. pp. 995–1000. IEEE (2010)
17. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cognitive modeling 5 (1988)
18. Schilling, N.: Supplementary website: `http://hylap.org/publications/hyper-opt-with-factorized-multilayer-perceptrons`
19. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 2951–2959. Curran Associates, Inc. (2012)
20. Swersky, K., Snoek, J., Adams, R.P.: Multi-task bayesian optimization. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 26, pp. 2004–2012. Curran Associates, Inc. (2013)
21. Yogatama, D., Mann, G.: Efficient transfer learning method for automatic hyperparameter tuning. In: International Conference on Artificial Intelligence and Statistics (AISTATS 2014) (2014)