Kernel Function Tuning for Single-Layer Neural Networks

Petra Vidnerová and Roman Neruda

***Abstract*—This paper describes an unified learning framework for kernel networks with one hidden layer, including models like radial basis function networks and regularization networks. The learning procedure consists of meta-parameter tuning wrapping the standard parameter optimization part. Several variants of learning are described and tested on various classification and regression problems. It is shown that meta-learning can improve the performance of models for the price of higher time complexity.**

***Index Terms*—Radial basis function networks, shallow neural networks, kernel methods, hyper-parameter tuning.**

I. Introduction

The family of kernel networks encompasses models like radial basis function (RBF) networks – an example of a more general class of generalized regularization networks defined by Poggio and Girosi – on one hand, to SVM models, on the other [1], [2].

One of the advantages of kernel networks is their relatively fast training. Considering particular type of kernel and other dependent parameters of the network, the resulting training algorithm is usually based on linear optimization. On the other hand, fixing the model in such a way creates a rather strong inductive bias and may relevantly influence the model performance. Still, the choice of kernel function, together with other options, referred to as meta-parameters here, remains largely an empirical choice at best [3]-[5].

In this paper we gather our work on meta-parameters setting for several types of kernel networks, including radial basis function networks and regularization networks, both possibly with multi-kernel units. The results show that using additional level of the algorithm to search the space of meta-parameters can improve the performance of the model in a dramatic way for the price of increased time complexity.

The structure of the paper is as follows. In Section II we introduce various types of kernel networks. In Section III we present several meta-parameter search methods. These methods are tested on several datasets representing classification and regression problems in Section IV. A brief comparison with other machine learning methods is also presented. Finally, the paper is concluded in Section V.

Manuscript received May 21, 2018; revised July 20, 2018. This work was partially supported by the Czech Science foundation under project no. 15-18108S, and institutional support of the Institute of Computer Science project no. RVO 67985807.

P. Vidnerová is with the Czech Academy of Sciences, Institute of Computer Science, Prague (e-mail: petra@cs.cas.cz).

R. Neruda is with the Czech Academy of Sciences, Institute of Computer Science and Charles University, Prague (e-mail: roman@cs.cas.cz).

II. Kernel Networks

Considering number of applications, RBF neural networks represent a relatively less common and alternative neural network architecture. In contrast with the more classical models (such as multilayer perceptron) the RBF network contains local units, which was probably motivated by the presence of many local response units in human brain. Other motivation came from computational mathematics, radial basis functions were first introduced as a solution of real multivariate interpolation problems [6].

An RBF network is a feed-forward neural network with one hidden layer of RBF units and a linear output layer (see Fig. 1). By an RBF unit we mean a neuron with n real inputs and one real output, realizing a radial basis function (1), such as Gaussian. Instead of the most commonly used Euclidean norm it is possible to use the *weighted norm* $\|\cdot\|_C$ where $\|x\|_C = (Cx)^T(Cx) = x^T C^T Cx$.

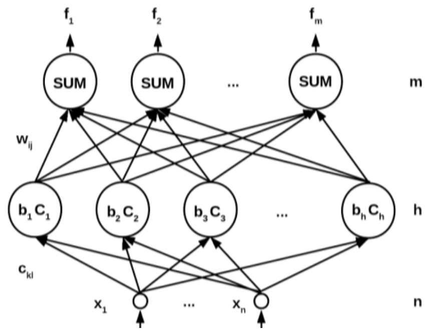

Fig. 1. RBF network architecture.

$$y(x) = \varphi\left(\frac{\|x-c\|_C}{b}\right) \tag{1}$$

$$f_s(x) = \sum_{j=1}^{h} w_{js}\,\varphi\left(\frac{\|x-c\|_C}{b}\right)$$

There are many learning algorithms for RBF networks, ranging from gradient approaches to linear optimization accompanied by heuristics [7].

To introduce the concept, consider a problem of learning from examples by means of regularization theory. We are given a set of examples $\{(\vec{x}_i, y_i) \in R^d \times R\}_{i=1}^{N}$ obtained by random sampling of some real function $y = f(\vec{x})$, and we would like to find this function.

Since this problem is ill-posed, we have to consider some a priori knowledge about the function *f*. It is usually assumed that the function is *smooth*, in the sense that two similar inputs corresponds to two similar outputs and the function does not oscillate much. This is the main idea of the regularization theory, where the solution is found by minimizing the functional *H[f]* containing both the data term and the smoothness information.

$$H[f] = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2 + \gamma\Phi[f],$$

where Φ is called a *stabilizer* and $\gamma > 0$ is *the regularization parameter* controlling the trade off between the closeness to data and the smoothness of the solution. The above regularization scheme was first introduced by Tikhonov [8] and therefore it is often called a Tikhonov regularization.

Poggio, Girosi and Jones in [1] proposed a form of a smoothness functional based on Fourier transform:

$$\Phi[f] = \int_{R^d} ds \, \frac{|\tilde{f}(s)|^2}{\tilde{G}(s)}, \qquad (2)$$

where $\tilde{f}$ indicates the Fourier transform of f , $\tilde{G}$ is some positive function that goes to zero for $||s|| \to \infty$ (i.e. $1/\tilde{G}$ is a high-pass filter). The stabilizer (2) measures the energy in the high frequency and so penalizes the functions with high oscillations.

It was shown that for a wide class of stabilizers in form of eq. (2) the solution has a form of feed-forward neural network with one hidden layer, called *regularization network*, and that different types of stabilizers lead to different types of Regularization Networks [1], [9].

From the regularization framework point of view, RBF networks belong to the family of generalized *regularization networks (RN)*. Generalized regularization networks are RN with lower number of kernels than the number of data points, and also it is not necessarily uniform kernels (thus, for example the network with Gaussian kernels may use kernels with different widths).

Poggio and Smale in [9] studied the regularization networks derived using a Reproducing Kernel Hilbert Space (RKHS) as the hypothesis space. Let $\boldsymbol{H}_K$ be an RKHS defined by a symmetric, positive-definite kernel function

$$K_x(x') = K(x, x')$$

Then if we define the stabilizer by means of norm in $\boldsymbol{H}_K$ and minimize the functional:

$$\min_{f \in H_K} H[f]$$

where

$$H[f] = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2 + \gamma ||f||^2{}_K,$$

over the hypothesis space $\boldsymbol{H}_K$, the solution of such a minimization is unique and has the form:

$$f(x) = \sum_{i=1}^{N} c_i K_{x_i}(x),$$

$$(N\gamma I + K)c = y$$

where I is the identity matrix, K is the matrix $K_{i,j} = K(x_i,x_j)$, and $y = (y_1,...,y_N)$. Girosi in [10] showed that for positive definite functions of the form $K(x-y)$ (such as Gaussian function) the norm in RKHS defined by K is equivalent to stabilizer:

$$||f||_K^2 = \int_{R^d} ds \, \frac{|\tilde{f}(s)|^2}{\tilde{G}(s)}.$$

This means, that using such a norm as a regularization term indeed penalizes highly oscillating functions *f*.

The kernel function used in a particular application of regularization network is typically supposed to be given in advance, for instance chosen by a user. It reflects our prior knowledge or assumption about the problem and its solution. Therefore its choice is crucial for the quality of the solution and should be always done according to the given task.

What is common to all kernel methods is the way the data are seen by the learning algorithm. Data are not represented individually, but through a set of pair-wise comparisons, realized by the kernel function [11]. The data set $T = \{(x_i, y_i) \in R^d \times R\}_{i=1}^{N}$ is presented by $N \times N$ matrix of pair-wise comparisons $K_{ij} = K(x_i,x_j)$ (i.e., the matrix K in the solution equation above).

In this context, kernels are often understood as measures of similarity, i.e. the higher $K(x,y)$ is, the more similar data points x and y are. The prior knowledge of the problem may suggest suitable similarity measure.

In theory, mostly symmetric and positive-definite functions are considered as kernels. In practice, wider range of functions can be considered, for example in [12], it was demonstrated that kernels which are only conditionally positive definite can possibly outperform classical kernels.

Examples of kernel functions (that are used further in our experiments) are:

- Gaussian: $f(x, y) = e^{-\gamma ||x-y||^2}$
- multiquadric: $f(x, y) = \sqrt{(1 + (\gamma * ||x - y||)\ )}$
- polyspline:

$$f(x, y) = \begin{cases} ||x - y||^{\gamma}; & \text{if } \gamma \text{ is even} \\ ||x - y||^{\gamma} \ln(||x - y||); & \text{if } \gamma \text{ is odd.} \end{cases}$$

Here, γ is a parameter of the kernel.

All kernels mentioned above work with numerical data. It was pointed out in [13] that kernels make it possible to work with non-vectorial data. This is due to the fact that kernels automatically provide a vectorial representation of the data in the feature space. Some examples on nonvectorial kernels may be found in [14].

In recent years, several methods have been proposed to combine multiple kernels instead of using a single one [15]. These multi-kernel algorithms are mainly designed for SVM learning.

One motivation for multi-kernel approach stems from the multi-modal nature of data. Each set of these features may require a different notion of similarity (i.e., a different kernel). Instead of building a specialized kernel for such applications, is it possible to define just one kernel for each of these modes and linearly combine them.

In our previous work [16]-18], we have also proposed composite types of kernel functions to be used in RN learning. Following the reasoning of Aronszajn's breakthrough paper [19], where products, sums and linear combinations of reproducing kernels are considered, it can be easily shown that these types of kernels can be used as activation functions in the regularization networks (cf. [16]). Moreover, kernel functions that are created as a combination of simpler kernel functions might better reflect the character of data.

Let $K_1,...,K_k$ be kernel functions defined on $\Omega_1,...,\Omega_k$ ($\Omega_i \subset R^{di}$), respectively. Let $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$. The kernel function K defined on Ω that satisfies:

$$K(\sim x_1,\sim x_2,\ldots,\sim x_k,\sim y_1,\sim y_2,\ldots,\sim y_k) = \\ = K_1(\sim x_1,\sim y_1)K_2(\sim x_2,\sim y_2)\cdots K_k(\sim x_k,\sim y_k), \qquad (3)$$

where $\sim x_i \in \Omega_i$, we call a *product kernel*.

By a *sum kernel* we mean a kernel function K that can be expressed as $K(x,y) = K_1(x,y) + K_2(x,y)$, where K_1 and K_2 are kernel functions.

In eq. (3), we can combine different kernel functions or two kernel functions of the same type but with different parameters, such as two Gaussians of different widths (note that in this case the Gaussians have the same center).

III. Meta-Parameters Selection

All of the additional parameters of the model are called *meta-parameters* in this work. Their proper choice is crucial for the performance of the learning algorithm. Their setup requires a hands on experience with the learning algorithm and it often leads to wasting time of the user by the trial and error method (cf. Fig. 2).

In this section we explore methods for automatic meta-parameters setup. A standard way is to optimize the cross validation error, and then learn the whole training set with the best parameters found.

By cross-validation we mean the process in which the data sets are divided in K sub samples. The learning process is executed K-times, where one sub sample is taken as the testing data set and the rest as the training data set, each time. The average evaluation of each run will express our estimate of the true machine-learning model generalization abilities over the given data.

Our scenario of applying a kernel network to data typically consists of the following parts:

- a kernel network model
- a meta-parameter space (possible widths of Gaussians, number of units, other parameters)
- a method for searching for optimal meta-parameters (genetic algorithm, grid search, etc.)
- a cross-validation scheme (e.g. K-fold)
- a score or error function (such as precision, recall, or κ metrics)

Now we briefly describe optimization methods that can be used for the search for optimal meta-parameters.

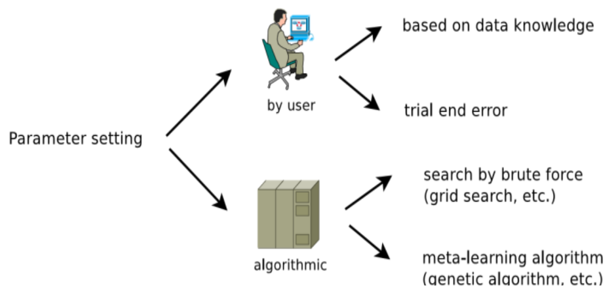

Fig. 2. Possibilities of parameter setting.

A. Grid Search

By a *grid search* we understand a technique where various couples of parameters are tried and the one with the best cross validation accuracy is picked. Though grid search can be easily parallelized, it can be quite time-consuming. Therefore it is using a coarse grid first. After identifying a better region on the grid, a finer grid search on that region can be conducted.

The grid search is also recommended in the libSVM library [20].

B. Genetic Search

Genetic algorithms (GA) [21] represent a sound and robust technique used to find approximate solutions to optimization and search problems. The use of GA gives us versatility, thus with suitable solution encoding, we can search for different type of kernel units within the framework of one algorithm.

The genetic algorithms typically work with a population of *individuals* embodying abstract representations of feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. The better the solution, the higher the fitness value.

The population evolves towards better solutions. The evolution starts from a population of completely random individuals and iterates in generations. In each generation, the fitness of each individual is evaluated. Individuals are stochastically selected from the current population (based on their fitness), and modified by means of operators *mutation* and *crossover* to form a new population. The new population is then used in the next iteration of the algorithm.

In our case, the individuals are encoding the type of kernel function, its additional parameters, and the parameters of the algorithm (C in case of SVM, γ in case of RN).

For example for the case of SVM:

I = {type of kernel function, kernel parameters, C}

I = {Gaussian, width = 0.5, C = 0.001}

The fitness value is given by the *cross-validation error*. The lower the cross-validation error is, the higher the fitness value is.

C. Simulated Annealing

Another available optimization method is the simulated annealing [22], [23]. Simulated annealing assigns a small probability even to moves which make the evaluation worse. The acceptance rate of the worse solutions depends on the difference between the actual solution and the neighbor, and a parameter called temperature. With a higher temperature, the acceptance rate is higher, and with a low temperature, the algorithm accepts only a small proportion of the non-improving moves. In the procedure, the temperature slowly declines with time. In the beginning, the algorithm explores larger parts of the search space and in the end later after lowering the temperature it converges toward only better evaluated solutions. There are various cooling strategies [24] which ensure such a decrease of the temperature variable.

IV. Experimental Results

This section reports on two experiments performed with our approach. The first one is a classification task on two different datasets – wine quality [25] and wilt [26] from the UCI Machine learning repository. The second is a regression task from real world air pollution sensor data. The experiments were realized using python *scikit-learn* [27] package with our own extensions.

A. Classification Data

Both classification datasets represent a middle-sized task with less than 5000 thousands data items and six, or twelve, respectively, attributes. A quadratic weighted kappa metrics (from python *ml-metrics* package) have been used as a measure of success for the model. The κ is a number between -1 and 1, where larger values represent a better performance. Each individual result represent a mean value from 5-fold stratified cross-validation. The meta-parameter search was performed by a grid search algorithm. We have used networks both with simple kernel functions (Gaussian, multiquadric and polyharmonic spline kernels) and multi-kernels (Sum Kernels). First, we tried various network sizes (see Figs. 3, 4, 5, 6, 7, 8). Then, we decided to use networks with 450 units and performed a finer grid search. The centers were set up at random, the output layer is optimized by linear optimization. This is done for three times, and the centers with the best fit are picked up (having higher number of tries does not help, since it leads to over-fitting and makes the cross-validated scores worse). These results have been compared with other models, whose optimal parameters have also been set by a grid search. The comparison models were support vector classification (SVC), logistic regression (LR), Gaussian naive Bayes (GNB), and decision tree (DT).

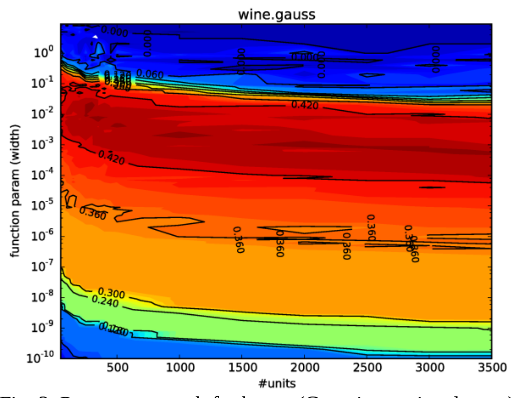

Fig. 3. Parameter search for best κ (Gaussians, wine dataset).

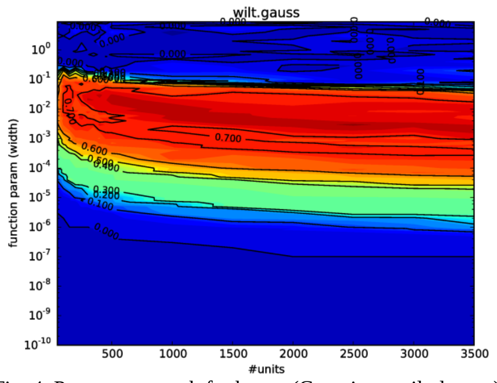

Fig. 4. Parameter search for best κ (Gaussians, wilt dataset).

The results are gathered in Table I. The following figures illustrate the grid search situation for individual methods. It can be seen that in general kernel networks perform very well for both tasks, always competing for the best results. The sum kernel network is the best, while Gaussian kernel network is a close runner-up.

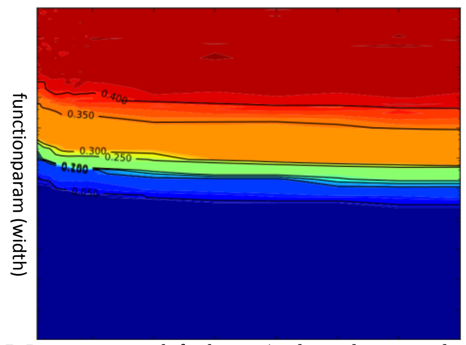

Fig. 5. Parameter search for best κ (multiquadric, wine dataset).

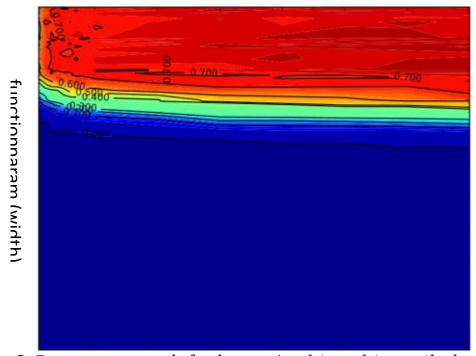

Fig. 6. Parameter search for best κ (multiquadrics, wilt dataset).

B. Regression Data

The dataset used for our regression experiments consists of real-world data from the application area of sensor networks for air pollution monitoring. The data contain tens of thousands measurements of gas multi-sensor MOX array devices recording concentrations of several gas pollutants collocated with a conventional air pollution monitoring station that provides labels for the data. The data are recorded in 1 hour intervals, and there is quite a large number of gaps due to sensor malfunctions. There are altogether 5 sensors as inputs and we have chosen three target output values representing concentrations of CO, NO_2 and NOx.

For the experiment we have divided the whole time period into five distinct intervals. These are used to perform a special kind of five-fold crossvalidation learning procedure. To describe it in detail – for five different runs we always chose one part for training, while the rest is again utilized for testing. This task is more difficult for two reasons. First, the training data is rather small compared to testing data. Second, the prediction is performed also in different parts of the year than the learning, e.g. the model trained on data obtained during winter may perform worse during summer (as was suggested by experts in the application area).

TABLE I: PERFORMANCE COMPARISON KERNEL NETWORKS WITH OTHER CLASSIFIERS. THE NUMBER REPRESENTS A MEAN VALUE OF QUADRATIC WEIGHTED κ FROM 5-FOLD STRATIFIED CROSSVALIDATION

dataset	Wine quality tuned	Wilt tuned
SVC	0.3359	0.8427
LR	0.3812	0.6341
GNB	0.4202	0.2917
DT	0.4283	0.8229
Gaussian network	0.438	0.849
Polyspline network	0.399	0.826
Multiquadric network	0.425	0.845
Best kernel network	0.438	0.849
Sum-kernel network	0.440	0.857

TABLE II: OVERVIEW OF DATA SETS SIZES

Task	train set	test set
CO i1-5	1469	5875
NO2 i1-5	1479	5914
NOx i1-5	1480	5916

Table II brings overview of data sets sizes. All tasks have 5 input values and 1 output (predicted value). All values are normalized between ⟨0,1⟩.

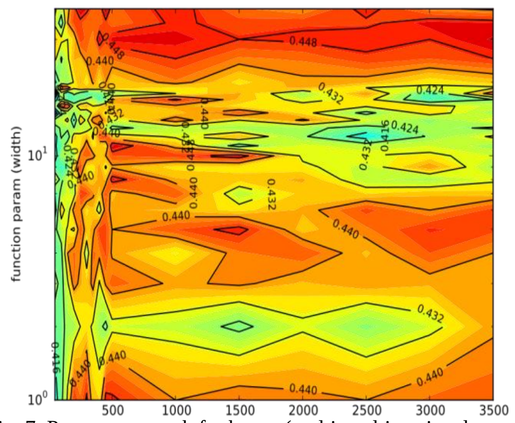

Fig. 7. Parameter search for best κ (multiquadric, wine dataset).

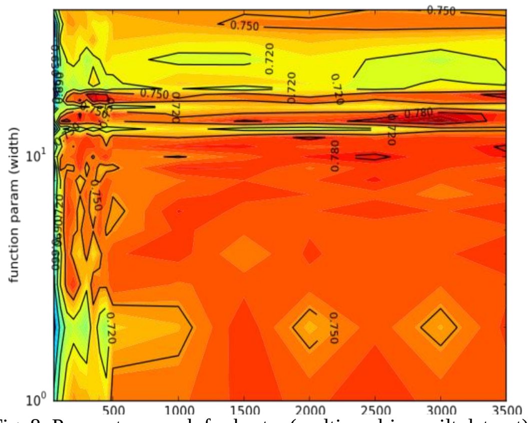

Fig. 8. Parameter search for best κ (multiquadrics, wilt dataset).

The genetic algorithm described in the previous section was applied in this experiment. For our models, the Gaussian and product kernels were evolved. Among the possible sub-kernels, there were Gaussian, multiquadric, inverse multiquadric, and sigmoid functions. The genetic algorithm was run for 300 generations, with population of 20 individuals. We were also using the elite mechanism heuristics with the size of the elite set to 2 individuals, i.e. 10 per cent. For fitness evaluation, the 10 folds cross-validation is used to ensure good generalization performance and to prevent overfitting of the evolved models.

Errors are computed as follows:

$$E = 100\frac{1}{N}\sum_{i=1}^{N}\|y_i - f(x_i)\|^2.$$

Each computation was repeated 10 times, and then the average errors and their standard deviations were computed and recorded.

The results of the experiment can be found in Table III, Table IV, where training and test errors are listed, and Fig. 9. The results present comparison between the performance of Gaussian and product kernels that were evolved. In terms of training errors, the product kernels were better in 10 cases from 15. In terms of testing errors, the product kernels were superior only in 7 cases. Note that if we compare the minimal values instead of average values, the product kernels are winning in almost all cases (except 1 in case of training errors, and except 3 in case of test errors). That may indicate there is still space for improvement for the optimal product kernel search, because the search algorithm provides results with high variance.

It is also interesting to note that the resulting product networks are mainly combinations of Gaussian and inverse multiquadric functions, indicating that hybrid models combining different kernels can be better in practice.

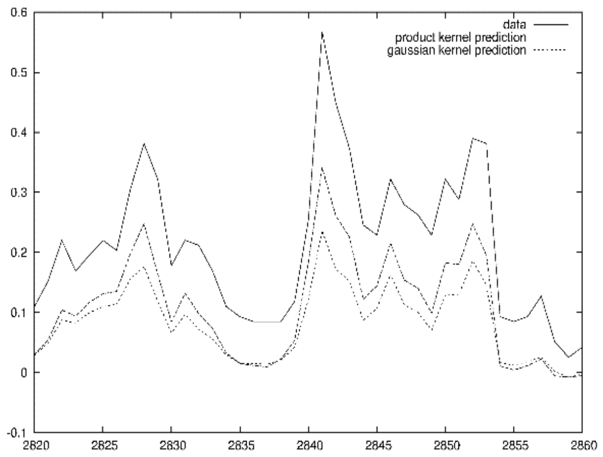

Fig. 9. Task 2: Example of prediction of CO concentration during 40 hours (in period not used for training).

TABLE III: COMPARISON OF TRAINING ERRORS FOR NETWORKS TRAINED ON SINGLE EPOCHS

	Gaussian kernel				Product kernels			
Task	Eavg	stddev	min	max	Eavg	stddev	Min	max
CO-i1	**0.050**	0.000	0.050	0.050	0.051	0.002	0.049	0.055
CO-i2	0.049	0.000	0.049	0.049	**0.046**	0.002	0.043	0.050
CO-i3	**0.054**	0.000	0.053	0.054	0.056	0.003	0.054	0.065
CO-i4	**0.333**	0.001	0.332	0.334	0.347	0.016	0.325	0.378
CO-i5	0.133	0.000	0.132	0.133	**0.097**	0.018	0.077	0.142
NO2-i1	**0.096**	0.002	0.093	0.101	0.100	0.015	0.091	0.141
NO2-i2	0.133	0.001	0.131	0.134	**0.122**	0.014	0.105	0.148
NO2-i3	0.388	0.001	0.384	0.389	**0.314**	0.077	0.214	0.434
NO2-i4	0.297	0.002	0.295	0.299	**0.287**	0.012	0.265	0.307
NO2-i5	**0.375**	0.001	0.374	0.376	0.389	0.032	0.330	0.435
NOx-i1	0.018	0.000	0.018	0.018	**0.017**	0.001	0.016	0.020
NOx-i2	0.026	0.000	0.026	0.027	**0.025**	0.002	0.021	0.028
NOx-i3	0.156	0.001	0.154	0.158	**0.152**	0.019	0.121	0.184
NOx-i4	0.231	0.002	0.229	0.234	**0.230**	0.017	0.203	0.258
NOx-i5	0.106	0.023	0.087	0.132	**0.095**	0.011	0.083	0.122

TABLE IV: COMPARISON OF TEST ERRORS FOR NETWORKS TRAINED ON SINGLE EPOCHS

	Gaussian kernel				**Product kernels**			
Task	**Eavg**	**stddev**	**min**	**max**	**Eavg**	**stddev**	**min**	**max**
CO-i1	**0.210**	0.005	0.205	0.217	0.214	0.020	0.192	0.248
CO-i2	1.134	0.007	1.116	1.142	**0.878**	0.088	0.709	0.988
CO-i3	0.233	0.009	0.221	0.254	**0.228**	0.019	0.197	0.267
CO-i4	**0.326**	0.002	0.323	0.329	0.749	0.512	0.433	1.921
CO-i5	**0.296**	0.005	0.287	0.301	0.321	0.050	0.204	0.374
NO2-i1	**2.151**	0.052	2.096	2.267	2.263	0.540	1.189	2.997
NO2-i2	5.260	0.045	5.161	5.319	**3.928**	1.447	2.661	6.874
NO2-i3	**0.718**	0.004	0.709	0.721	1.033	0.218	0.764	1.351
NO2-i4	0.735	0.011	0.726	0.757	**0.734**	0.069	0.669	0.908
NO2-i5	**0.678**	0.024	0.655	0.735	0.913	0.183	0.709	1.302
NOx-i1	2.515	0.015	2.495	2.538	**2.409**	0.159	2.093	2.658
NOx-i2	3.113	0.019	3.081	3.139	**2.495**	0.068	2.416	2.592
NOx-i3	1.105	0.008	1.088	1.114	**0.956**	0.267	0.730	1.689
NOx-i4	**0.952**	0.008	0.941	0.970	1.256	0.520	0.774	2.610
NOx-i5	**0.730**	0.102	0.642	0.850	0.748	0.091	0.544	0.856

V. CONCLUSION

We have proposed a general meta-parameter setting algorithm for kernel networks that can be applied to various flavors of kernel networks, from RBF to networks with multi-kernel units. Different search algorithms have been tested, including genetic algorithm and grid search. The results on several classification and regression tasks show that kernel networks perform competitively with other methods such as support vector classifiers or classification trees. Moreover, with meta-parameter tuning, they usually outperformed other methods (also tuned for optimal meta-parameter values). It is interesting to note that sum and product kernel units provided superior solution in many cases.

In the future work we will focus on comparison of search algorithms for different network architectures. Since the meta-parameter search problem is naturally very time consuming, involving many evaluations of the model, the efficiency of search is crucial. While methods such as grid search or simulated annealing perform well on a smaller number of real valued parameters, the genetic algorithm has the advantage to master more complex parameters such as a type of kernel or its combination. It would be also interesting to study the applicability of our approach to the area of deep convolutional networks that have achieved surprisingly good results in several applications recently.

ACKNOWLEDGMENT

This work was partially supported by the Czech Science Foundation project no. GA15-18108S, and by the institutional support of the Institute of Computer Science, Czech Academy of Sciences RVO 67985807.

REFERENCES

[1] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," *Cambridge, MA, USA, Tech. Rep.*, a. I. Memo No. 1140, C.B.I.P. Paper NO. 31, 1989.

[2] S. Salcedo-Sanz, J. L. Rojo-Alvarez, M. Martinez-Ramon, and G. Camps-Valls, "Support vector machines in engineering: An overview," *Wiley Int. Rev. Data Min. and Knowl. Disc.*, vol. 4, no. 3, pp. 234-267, May 2014.

[3] M. Kim, "Accelerated max-margin multiple kernel learning," *Applied Intelligence*, vol. 38, no. 1, pp. 45-57, Jan. 2013.

[4] P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen, "Tuning and evolution of support vector kernels," *Evolutionary Intelligence*, vol. 5, no. 3, pp. 153-170, 2012.

[5] L. Diosan, A. Rogozan, and J.-P. Pecuchet, "Improving classification performance of support vector machine by genetically optimising kernel shape and hyper-parameters," *Applied Intelligence*, vol. 36, no. 2, pp. 280-294, 2012.

[6] M. Powel, "Radial basis functions for multivariable interpolation: A review," in *Proc. IMA Conference on Algorithms for the Approximation of Functions and Data*, RMCS, Shrivenham, England, 1985, pp. 143-167.

[7] R. Neruda and P. Kudová, "Learning methods for radial basis functions networks," *Future Generation Computer Systems*, vol. 21, pp. 1131-1142, 2005.

[8] A. Tikhonov and V. Arsenin, *Solutions of Ill-posed Problems*, W.H. Winston, Washington, D.C, 1977.

[9] T. Poggio and S. Smale, "The mathematics of learning: Dealing with data," *Notices of the AMS*, vol. 50, pp. 536-544, 5 2003.

[10] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and Neural Networks architectures," *Neural Computation*, vol. 2, pp. 219-269, 7 1995.

[11] J. P. Vert, K. Tsuda, and B. Scholkopf, "A primer on kernel methods," *Kernel Methods in Computational Biology*, pp. 35-70.

[12] S. Boughorbel, J.-P. Tarel, and N. Boujemaa, "Conditionally positive definite kernels for svm based image recognition," in *Proc. ICME*, 2005, pp. 113-116.

[13] B. Schoelkopf and A. J. Smola, *Learning with Kernels*, MIT Press, Cambridge, Massachusetts, 2002.

[14] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

[15] M. Goenen and E. Alpaydin, "Multiple kernel learning algorithms," *Journal of Machine Learning Research*, vol. 12, pp. 2211–2268, Jul. 2011.

[16] P. Kudová and T. Šámalová, "Sum and product kernel regularization networks," *Artificial Intelligence and Soft Computing*, 2006, pp. 56-65.

[17] P. Vidnerová and R. Neruda, "Evolutionary learning of regularization networks with product kernel units," in *Proc. SMC*, IEEE, 2011, pp. 638-643.

[18] P. Vidnerov á and R. Neruda, "Evolving sum and composite kernel functions for regularization networks," in *Proc. ICANNGA*, vol. 6593, Springer, 2011, pp. 180-189.

[19] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the AMS*, vol. 68, pp. 337-404, 1950.

[20] C. Hsu, C. Chang, and C. Lin, *A Practical Guide to Support Vector Classification*, 2000.

[21] M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press, 1996.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.

[23] V. Černý, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41-51, 1985.

[24] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, pp. 8373-8385, 1998.

[25] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, Elsevier, vol. 47, no. 4, pp. 547-553, 2009.

[26] B. A. Johnson, R. Tateishi, and N. T. Hoan, "A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees," *International Journal of Remote Sensing*, vol. 34, no. 20, pp. 6969-6982, Oct. 2013.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

Petra Vidnerová is a researcher at the Czech Academy of Sciences, Institute of Computer Science in Prague, where she has worked since 2001. She received her Ph.D. in 2007, the topic of her thesis was learning with regularization networks. Currently she works at the Department of Machine Learning. Her research interests include machine learning, namely kernel methods, deep learning, hyper-parameter search and meta-learning, evolutionary and hybrid approaches.

Roman Neruda is a researcher at the Czech Academy of Sciences, Institute of Computer Science, and a lecturer at Charles University in Prague, Faculty of mathematics and physics. He received his Ph.D. in 1999 in theoretical computer science at Czech Academy of Sciences, the topic of his thesis was Genetic learning of RBF Networks. Currently, he works at the Department of Machine Learning. His research interests include machine learning, deep learning, hyper-parameter search and meta-learning, evolutionary and hybrid approaches, and multi-agent systems.