Kernel Multilayer Perceptron

Thomas W. Rauber
Department of Computer Science
University of Espírito Santo
Av. Fernando Ferrari s/n, 29060–970, Vitória, ES, Brazil
Email: thomas@inf.ufes.br

Karsten Berns
Robotics Research Lab, Department of Computer Science
University of Kaiserslautern, Gottlieb-Daimler-Strasse,
Bldg. 48, 67663 Kaiserslautern, Germany
Email: berns@informatik.uni-kl.de

***Abstract*—We enhance the Multilayer Perceptron to map a feature vector not only from the original d-dimensional feature space, but from an intermediate implicit Hilbert feature space in which kernels calculate inner products. The kernel substitutes the usual inner product between weight vectors and the input vector (or the feature vector of the hidden layer). The objective is to boost the generalization capability of this universal function approximator even more. Classification experiments with standard Machine Learning data sets are shown. We are able to improve the classification accuracy performance criterion for certain kernel types and their intrinsic parameters for the majority of the data sets.**

***Keywords*-Multilayer Perceptron, kernel mapping**

I. Introduction

Ever since the error backpropagation training algorithm in conjunction with the multilayer feedforward neural network with nonlinear activation function was defined by Werbos [1] and presented to a larger audience by Rumelhart et al. [2], [3], it became one of the most powerful general purpose function approximators for regression and classification.

In recent years, the use of feature mapping into Hilbert space, together with kernel based inner product similarity measures[1] has gained considerable attention, because its introduces an additional, in general non-linear, mapping from the original feature space to an intermediate space which should improve regression (cf. for example, [4]) and/or classification quality of the new features. Especially the Support Vector Machine [5] has drawn much attention to kernel-based implicit feature mapping, although many other classification techniques can be enhanced by kernels, as will be reviewed later, and also constitutes the main motivation of this work, since we incorporate it into the standard feedforward artificial neural network. Our main contribution is to introduce an additional nonlinear feature space mapping by kernels before applying the MLP mapping, improving even more the general purpose function approximation capabilities of the MLP. An important characteristic of our approach is that the knowledge of the function approximator is explicitly stored in the weights w (besides the architecture of the network). This is not possible, for instance, in the kernel enhanced perceptron [6] that constitutes the basic architecture for the Support Vector Machine (SVM). The SVM additionally maximizes the margin but has the same architecture (a linear dichotomizer in the implicit Hilbert feature space). The SVM stores the weight vector implicitly in the Lagrange multipliers obtained from the solution of the convex quadratic programming problem defined by the margin maximization.

The rest of the paper is organized as follows. In section II we review the multilayer perceptron and backpropagation in the light of the enhancing its mapping by an additional nonlinear mapping, using kernels. Section III incorporates the kernel mapping into the Multilayer Perceptron (MLP) prior to the proper MLP mapping, followed by the review of related relevant work in section IV. Experimental results suggesting the convergence and potential to improve performance of the kernel enhanced MLP is presented in section V and finally the conclusions are drawn in section VI.

II. Multilayer Perceptron and the Error Backpropagation

We expect the reader to be familiar with the Multilayer Perceptron and the Error Backpropagation training algorithm in its basic form, described in standard textbooks, such as [7], [8]. We consider only a network with one hidden layer and the basic error backpropagation algorithm based on gradient descent, since further aspects are irrelevant regarding the conceptual enhancement of the kernel mapping introduced here. In the following we review the architecture and learning algorithm of the network, in order to be able to introduce our conceptual enhancement of kernel mapping.

A. Multilayer Perceptron mapping

Consider as input to the net a d-dimensional pattern $\mathbf{x}$ from an input domain $\mathcal{X}$ which is usually the Euclidean vector space $\mathbb{R}^d$. The hidden layer of the net has H neurons which all calculate the function $\theta : \mathcal{X} \to \mathbb{R}$ which is the inner product $\langle \mathbf{w}_h, \mathbf{x} \rangle = \mathbf{w}_h \cdot \mathbf{x} = \mathbf{w}_h^\mathsf{T} \mathbf{x}$ of a neuron specific d-dimensional weight vector $\mathbf{w}_h$ and the input $\mathbf{x}$, followed by an activation function $z : \mathbb{R} \to \mathbb{R}$, usually the logistic sigmoid function $z(a) = 1/(1 + \exp(-a))$ or hyperbolic tangent sigmoid function $z(a) = \tanh a$ to give

$$\theta_h = z(\langle \mathbf{w}_h, \mathbf{x} \rangle), \qquad h = 1, \ldots, H. \tag{1}$$

All H input-to-hidden mappings θ_h of the hidden layer are assembled into a H-dimensional feature vector $\boldsymbol{\theta} =$

[1]For the sake of simplicity, we will occasionally use the term "kernel mapping" to identify the mapping from the original feature space into Hilbert space where kernels can be used to calculate an inner product.

$[\ \theta_1 \quad \cdots \quad \theta_H\ ]^\mathsf{T}$ which constitutes a new pattern in the domain Θ. The calculus of the hidden layer feature vector $\boldsymbol{\theta}$ consequently defines a map (using the same symbol)

$$\begin{aligned}\boldsymbol{\theta} &: \mathcal{X} \to \Theta \\ \mathbf{x} &\mapsto \boldsymbol{\theta}(\mathbf{x}) = [\ z(\langle \mathbf{w}_{h_1}, \mathbf{x}\rangle) \quad \cdots \quad z(\langle \mathbf{w}_{h_H}, \mathbf{x}\rangle)\ ]^\mathsf{T}. \quad (2)\end{aligned}$$

The final mapping from the hidden to the output layer is replicating the same functional mapping from the input pattern to the hidden layer, calculating for all output neurons the function

$$y_i = z(\langle \mathbf{w}_i, \boldsymbol{\theta}\rangle), \qquad i = 1, \ldots, c, \tag{3}$$

where the $\mathbf{w}_i$ are H-dimensional weight vectors[2] specifically for each of the c output units. We can again merge the output into a c-dimensional feature vector $\mathbf{y} = [\ y_1 \quad \cdots \quad y_c\ ]^\mathsf{T}$ which constitutes a new pattern in the final output domain $\mathcal{Y}$. The calculus of the output layer feature vector $\mathbf{y}$ therefore defines another map (using the same symbol)

$$\begin{aligned}\mathbf{y} &: \Theta \to \mathcal{Y} \\ \boldsymbol{\theta} &\mapsto \mathbf{y}(\boldsymbol{\theta}) = [\ z(\langle \mathbf{w}_{i_1}, \boldsymbol{\theta}\rangle) \quad \cdots \quad z(\langle \mathbf{w}_{i_c}, \boldsymbol{\theta}\rangle)\ ]^\mathsf{T}. \quad (4)\end{aligned}$$

Resumingly we observe that the Multilayer Perceptron performs a mapping

$$\begin{aligned}\mathbf{y} &: \mathcal{X} \to \Theta \to \mathcal{Y} \\ \mathbf{x} &\mapsto \boldsymbol{\theta}(\mathbf{x}) \mapsto \mathbf{y}(\boldsymbol{\theta}(\mathbf{x})).\end{aligned} \tag{5}$$

Frequently an additional fixed input component $x_0 = 1$ and eventually a fixed hidden component $\theta_0 = 1$ is introduced which augments the dimension of the input, hidden and weight vectors in 1 and incorporates an offset into the calculated inner products.

B. Error Backpropagation weight optimization

An unconstrained optimization in the form of a gradient descent is the basic algorithm to optimize the weight vectors $\mathbf{w}_h$ to the hidden layer and $\mathbf{w}_i$ to the output layer. An error function E is established which describes the expected discrepancy between a desired c-dimensional output vector $\mathbf{t}$ and the vector $\mathbf{y}$ calculated by the MLP. We consider only the stochastic version of the optimization algorithm, aka. on-line learning which adjusts all weights after presenting a single training pattern $\mathbf{x}$ to the network which is labeled with its desired output $\mathbf{t} = [\ t_1 \quad \cdots \quad t_i \quad \cdots \quad t_c\ ]^\mathsf{T}$, such that the pattern specific mismatch (aka. delta) $\boldsymbol{\delta} \in \mathbb{R}^c$ is defined as

$$\boldsymbol{\delta} = \mathbf{t} - \mathbf{y}(\boldsymbol{\theta}(\mathbf{x})). \tag{6}$$

For the sake of simplicity we avoid pattern specific indices p for the $p = 1, \ldots, n$ training patterns $\mathbf{x}^{(p)}$. The pattern specific error is the square of the delta (multiplying by 0.5 for convenience)

$$E = \frac{1}{2}\boldsymbol{\delta}^2 = \frac{1}{2}\left[\mathbf{t} - \mathbf{y}(\boldsymbol{\theta}(\mathbf{x}))\right]^2 = \frac{1}{2}\sum_{i=1}^{c}\left[t_i - y_i(\boldsymbol{\theta}(\mathbf{x}))\right]^2, \tag{7}$$

or more specifically, emphasizing the dependencies of the error on the weights

$$\begin{aligned}&E(\{\mathbf{w}_h\}_{h=1}^H, \{\mathbf{w}_i\}_{i=1}^c) = \\ &\frac{1}{2}\left[\mathbf{t} - \mathbf{y}(\boldsymbol{\theta}(\mathbf{x}; \{\mathbf{w}_h\}_{h=1}^H); \{\mathbf{w}_i\}_{i=1}^c)\right]^2.\end{aligned} \tag{8}$$

We furthermore define δ_i as the i-th component of the discrepancy vector

$$\delta_i = t_i - y_i \qquad i = 1, \ldots, c. \tag{9}$$

In order to modify the hidden-to-output weights by the basic gradient descent, with learning rate $\eta \in \mathbb{R}^+$, we use the rule

$$\mathbf{w}_i^{\text{new}} = \mathbf{w}_i^{\text{old}} - \eta \left.\frac{\partial E}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i^{\text{old}}}, \qquad i = 1, \ldots, c. \tag{10}$$

In order to unify the formalisms for the presence and absence of kernel mapping, we abbreviate the inner product of weights and feature vectors as

$$k(\mathbf{w}, \mathbf{x}) := \mathbf{w} \cdot \mathbf{x}, \quad k_i := k(\mathbf{w}_i, \boldsymbol{\theta}), \quad k_h := k(\mathbf{w}_h, \mathbf{x}). \tag{11}$$

The gradient of the error with respect to a hidden-to-output weight vector, needed in (10), considering (6), (7) is obtained by the chain rule

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{w}_i} = \frac{\partial \frac{1}{2}\boldsymbol{\delta}^2}{\partial \mathbf{w}_i} = -\delta_i \frac{\partial \mathbf{y}}{\partial \mathbf{w}_i} = -\delta_i \frac{\partial z(k(\mathbf{w}_i, \boldsymbol{\theta}))}{\partial \mathbf{w}_i} = \\ -\delta_i z'(k_i)\frac{\partial k(\mathbf{w}_i, \boldsymbol{\theta})}{\partial \mathbf{w}_i} = -s_i \frac{\partial k(\mathbf{w}_i, \boldsymbol{\theta})}{\partial \mathbf{w}_i} = \quad (12)\\ -s_i \boldsymbol{\theta}, \quad (13)\end{aligned}$$

where $z'(k)$ is the derivative of the activation function, $z'(k) = z(k)(1 - z(k))$ for the logistic sigmoid and $z'(k) = 1 - z(k)^2$ for the hyperbolic tangent, and s_i is defined as the hidden-to-output sensitivity

$$s_i := \delta_i z'(k_i). \tag{14}$$

Similarly for the input-to-hidden gradient descent, using the same schema for the weights $\mathbf{w}_h$ as in (10) for the hidden-to-output weights $\mathbf{w}_i$ and additionally backpropagating the error deeper into the net, we obtain

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{w}_h} = -s_h \frac{\partial k(\mathbf{w}_h, \mathbf{x})}{\partial \mathbf{w}_h} = \quad (15)\\ -s_h \mathbf{x}, \quad (16)\end{aligned}$$

where the input-to-hidden sensitivity s_h is defined as

$$s_h := z'(k_h)\sum_{i=1}^{c} s_i w_{ih}. \tag{17}$$

[2]It would be necessary to introduce an additional index to distinguish the input-to-hidden weights $\mathbf{w}_h$ from the hidden-to-output weights $\mathbf{w}_i$, or rename one of the two, but for the sake of simplicity we expect the reader to recognize the layer of the weight vector by its context.

III. Multilayer Perceptron Kernel mapping

We will now extend the MLP model and the backpropagation algorithm by introducing an additional nonlinear mapping from the input domain $\mathcal{X}$ to an implicit intermediate Hilbert space $\mathcal{H}$ domain[3], which is then mapped to the hidden layer domain Θ. Thinking further we can also introduce this intermediate mapping in between the hidden and output layer, but this would create a different model, and is left for later consideration.

A. Implicit kernel map

We define a map $\boldsymbol{\phi}$

$$\begin{aligned} \boldsymbol{\phi} : \mathcal{X} &\rightarrow \mathcal{H} \\ \mathbf{x} &\mapsto \boldsymbol{\phi}(\mathbf{x}) \end{aligned} \tag{18}$$

which projects a feature vector $\mathbf{x}$ of dimension d to a new feature vector $\boldsymbol{\phi}(\mathbf{x})$ of dimension $\ell \leq \infty$. This map however may not be defined explicitly in general. The metric of similarities and consequently distances in the mapped space $\mathcal{H}$ must exclusively be expressed as an inner product $\boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}_j)$ of the mapped versions of the original feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ when implementable calculus [4] has to be done. This constraint enables the use of kernels k that implement the inner product in the transformed space as

$$k(\mathbf{x}_i, \mathbf{x}_j) := \boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}_j). \tag{19}$$

As examples consider the commonly employed Radial Basis Function kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2)$ with spread parameter γ or the inhomogeneous polynomial kernel $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p$ which maps to all possible monomials up to degree p, see e.g. [9]. The mapped space $\mathcal{H}$ is a vector space where a inner product is defined, whereas the features in the original pattern space $\mathcal{X}$ are not restricted to be continuous values. By virtue of an appropriate kernel, the original features can be of symbolic, discrete or continuous nature, or any mix of these. An extensive treatment of kernels can be found, for example, in [6].

B. Input-to-hidden kernel mapping

A crucial observation when considering the argument $\langle \mathbf{w}_h, \mathbf{x} \rangle = \mathbf{w}_h \cdot \mathbf{x} = \mathbf{w}_h^\mathsf{T} \mathbf{x}$ of the activation function z is that both arguments $\mathbf{w}_h$ and $\mathbf{x}$ of the inner product are from the same domain $\mathcal{X}$, in the case of the conventional MLP they are from $\mathbb{R}^d$. This becomes even clearer when we geometrically interpret the meaning of the inner product. The value $d_h := \mathbf{w}_h \cdot \mathbf{x}$, incorporating the offset (aka bias) w_{h0} and fixing $x_0 = 1$, is an unnormalized distance of $\mathbf{x}$ from the hyperplane defined by $\mathbf{w}_h$ (the absolute value could be obtained by dividing d_h by the modulus of the vector $[\ w_{h1} \quad \cdots \quad w_{hd}\ ]^\mathsf{T}$). Consider also the Support Vector Machine where the weight vector of the separating hyperplane is a linear combination of the feature vectors, and is consequently from the same domain.

This rationale leads us to the mapping of the elements of the input domain $\mathcal{X}$, in this case the input vector $\mathbf{x}$ and the input-to-hidden weight vector $\mathbf{w}_h$ to the implicit domain $\mathcal{H}$ by virtue of the mapping (18), obtaining the ℓ-dimensional images $\boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\phi}(\mathbf{w}_h)$. Note again that patterns in the mapped space do only exist implicitly and cannot be accessed directly (the dimension ℓ of $\mathcal{H}$ can even be infinite [10], [11]). The mapping (2), (1) to the hidden domain Θ now takes its arguments from the intermediate space $\mathcal{H}$ and is modified correspondingly to

$$\theta_h = z(\langle \boldsymbol{\phi}(\mathbf{w}_h), \boldsymbol{\phi}(\mathbf{x}) \rangle) = z(k(\mathbf{w}_h, \mathbf{x})), \qquad h = 1, \ldots, H. \tag{20}$$

$$\begin{aligned} \boldsymbol{\theta} : \mathcal{H} &\rightarrow \Theta \\ \boldsymbol{\phi}(\mathbf{x}) &\mapsto \boldsymbol{\theta}(\boldsymbol{\phi}(\mathbf{x})) = \\ &[\ z(\langle \boldsymbol{\phi}(\mathbf{w}_{h_1}), \boldsymbol{\phi}(\mathbf{x}) \rangle) \quad \cdots \quad z(\langle \boldsymbol{\phi}(\mathbf{w}_{h_H}), \boldsymbol{\phi}(\mathbf{x}) \rangle)\ ]^\mathsf{T}. \end{aligned} \tag{21}$$

Fortunately we can calculate the inner products $\langle \boldsymbol{\phi}(\mathbf{w}_h), \boldsymbol{\phi}(\mathbf{x}) \rangle$ needed in the MLP mapping function (20) by the kernel (19) as $k_h = k(\mathbf{w}_h, \mathbf{x})$. The consequence for the MLP feedforward mapping is that the abbreviations for the inner products defined in (11) are now really meaning kernel functions defined in (19). They specialize to the usual inner product and consequently to the well known MLP when the linear kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$ with the identity mapping $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$ is used.

The kernel enhanced mapping of the Multilayer Perceptron, considering (5) is correspondingly adapted to

$$\begin{aligned} \mathbf{y} : \mathcal{X} &\rightarrow \mathcal{H} \rightarrow \Theta \rightarrow \mathcal{Y} \\ \mathbf{x} &\mapsto \boldsymbol{\phi}(\mathbf{x}) \mapsto \boldsymbol{\theta}(\boldsymbol{\phi}(\mathbf{x})) \mapsto \mathbf{y}(\boldsymbol{\theta}(\boldsymbol{\phi}(\mathbf{x}))). \end{aligned} \tag{22}$$

We can state that we have modified the architecture of the Multilayer Perceptron by introducing an additional nonlinear Hilbert space halfway between the input vector domain and the hidden layer domain. The next step is necessarily the adaptation of the weight optimization since the chain rule to calculate the dependency of the approximation error on the weights now affect the weight specific gradients (13) and (16).

C. Error backpropagation weight optimization with kernel mapping

We have to review the term $\partial k_h / \partial \mathbf{w}_h$ in (15) since k in general does not stand for an inner product of original weight and input vectors anymore, but for an inner product of their mapped counterparts in the Hilbert space. The value of the partial derivative of a kernel relative to the weight vector depends on the nature of the kernel. In table I some important kernels are listed. The definition k is needed in the feedforward input-to-hidden mapping (21) and the partial derivative with respect to the weight vector $\partial k(\mathbf{w}_h, \mathbf{x}) / \partial \mathbf{w}_h$ is needed in the modified error function gradient (15) and specialized in (15) with respect to the type of kernel in table I. We can also reuse

[3] $\mathcal{H}$ stands for the *Hilbert* but also suggests *hidden*, since the patterns $\boldsymbol{\phi}(\mathbf{x})$ in this space in general are only implicitly available.

[4] Patterns $\boldsymbol{\phi}(\mathbf{x})$ in Hilbert space in general cannot be stored in computer memory explicitly. They can only be handled in analytical calculus.

Kernel type	**Definition** $k(\mathbf{w},\mathbf{x})$	**Derivative** $\frac{\partial k(\mathbf{w},\mathbf{x})}{\partial \mathbf{w}}$
Linear	$\mathbf{w}\cdot\mathbf{x}$	$\mathbf{x}$
Polynomial (inhomogeneous)	$(\mathbf{w}\cdot\mathbf{x}+1)^p$	$(\mathbf{w}\cdot\mathbf{x}+1)^{p-1}\mathbf{x}$
Radial Basis Function	$\exp(-\gamma\|\mathbf{w}\cdot\mathbf{x}\|^2)$	$-2k\gamma(\mathbf{w}-\mathbf{x})$
Hyperbolic tangent sigmoid	$\tanh(\gamma\mathbf{w}\cdot\mathbf{x})$	$\gamma(1-k^2)\mathbf{x}$
Logistic sigmoid	$\frac{1}{1+\exp(-\gamma(\mathbf{w}\cdot\mathbf{x}))}$	$k(1-k)(-\gamma\mathbf{x})$

TABLE I
KERNELS AND THEIR PARTIAL DERIVATIVES WRT TO WEIGHT VECTOR (FIRST ARGUMENT)

the value of k_h in all but the linear and polynomial kernels in the calculus of the derivative.

IV. Related Relevant Work

Kernel based clustering, regression and classification has gained a lot of attention, especially in the context of the Support Vector Machine (SVM) [5], [11]. Outside the SVM context, kernel mapping can also be used to nonlinearly transform the problem description to a new feature space and adapt classical classifier paradigms, for instance, Kernel Principal Component Analysis (Kernel PCA) [12], Fisher discriminant analysis [13], kernel K-means clustering [14], the Nearest-Neighbor classifier [15], Quadratic Gaussian based Bayes classifier [16], Least Mean Square Linear dichotomizer [16] or high-dimensional pattern visualization by the Sammon map [17]. To the best of our knowledge kernel mapping has not been applied within the architecture of the feedforward general purpose function approximator known as the Multilayer Perceptron with the ability to represent the knowledge explicitly in weights w. A special attention is given here to the kernel perceptron algorithm and the Support Vector related optimization algorithms for regression and classification described in [6]. If a map $\boldsymbol{\phi}(\mathbf{x})$ is used to map the original pattern $\mathbf{x}$ to the implicit feature space $\mathcal{H}$ by a kernel mapping, the consequence is that the weight vector $\mathbf{w}$ cannot be represented in an explicit manner anymore, since it is a linear combination of the mapped patterns $\boldsymbol{\phi}(\mathbf{x}_j)$ that do exist only implicitly in the kernel mapped feature space. The classificator/regressor must rely on the dual coefficients α_j obtained from the learning algorithm. On the contrary to that, in our approach, we have the weights always explicitly at hand, even if a feature mapping via kernels is applied. The weight adaptation is also directly embedded into the backpropagation algorithm with a particular additional derivation term for each kernel type. These are fundamental differences to the dual representation in kernel perceptron and SVM architecture.

V. Experimental Results

The aim of the experiments is to show, extending the conventional Multilayer Perceptron by kernel mapping that the network approximation converges. One should observe a considerable diminishing of the expected error $\mathbb{E}[E]$ of (7), estimating it as the mean over all training and/or test patterns.

If additionally an appropriate kernel, together with its intrinsic parameter(s) improves the performance criterion when comparing it to the conventional MLP, i.e. with a linear kernel, then the introduction of kernel mapping suggests a conceptual improvement of the general purpose architecture. The experiments do not claim that the kernel enhanced MLP performs better than other classifiers (or regressor in general). We intend to suggest that if the right kernel and its intrinsic parameters are chosen, the performance can increase compared to the conventional MLP.

A. Benchmark data sets

We want to provide a simple experimental design which easily allows reproducibility of the proposed method.[5] We restrict the experiments to classification of standard data sets from the UCI Machine Learning Repository [18]. Only continuous features with no missing values are allowed. The task must be classification, i.e. one attribute is the symbolic class label. For each of the c classes, one output neuron is created, coding the class membership as "1-out-of-c", i.e. a "1" at output y_i if pattern $\mathbf{x}$ belongs to class ω_i, and "0" otherwise.

Where there is an explicit separation of training and test data sets, for instance, in the 'pendigit' set, the MLP was trained with the training set and tested with the test set. For all other data sets, K-fold cross-validation with $K = 10$ is used. Table II shows the data sets that were randomly chosen from the repository. The sets provide a good mix of small and large number of classes, features and patterns.

B. Training method of the Multilayer Perceptron

The MLP is always trained with a basic pattern-based (stochastic) backpropagation algorithm that uses the pure gradient descent of (10). The learning rate is always set to $\eta = .2$. All weight vectors $\mathbf{w}_i, i = 1,\ldots,c$ of the output elements and $\mathbf{w}_h, h = 1,\ldots,H$ of the hidden elements were randomly initialized with a normalized modulus of one. All input features $x_j, j = 1,\ldots,d$ are independently standardized to the range $[-1,1]$ by the linear transformation $x^{\text{new}} = -1 + 2(x^{\text{old}} - x_{\min})/(x_{\max} - x_{\min})$. When there is an explicit training-test set division, the test set is submitted to the same standardization, based on the extremes $x_{\min}, x_{\max}$ of the training set. The number of inputs is the dimension d of the feature vector $\mathbf{x}$, plus one, i.e. fixing an additional input to $x_0 = 1$ which incorporates the offset (bias) w_{h0} of each of the hidden units into the weight vector $\mathbf{w}_h$. The number H of the hidden units is set to twice the number of inputs, plus one

[5] The Kernel MLP has been incorporated into the 'tooldiag' pattern recognition toolbox, written in C, and can be obtained at http://sites.google.com/site/tooldiag.

TABLE II
UCI MACHINE LEARNING REPOSITORY DATA SETS USED FOR THE CLASSIFICATION EXPERIMENTS. IN CASE OF AN EXPLICIT TRAIN-TEST SPLIT, THE NUMBER OF TEST SAMPLES IS MENTIONED. FOR A DETAILED DESCRIPTION C.F. THE UCI SITE.

Data set	# classes	# features	# training	# test
breast cancer	2	30	569	-
glass	6	8	214	-
haberman survival	2	3	41	-
pendigits	10	16	7494	3498
pima	2	8	768	-
satimage	6	36	4435	2000
shuttle	7	9	43500	14500
sonar	2	60	208	-
vehicle	4	18	846	-
wine	3	13	178	-

In the hidden layer, we also use a fixed input $\theta_0 = 1$, totaling $H + 1 = 2(d + 1) + 1$ hidden units $\theta_0, \ldots, \theta_H$. The number of the output neurons corresponds to the number of classes, using the 1-out-of-c coding mentioned earlier. The activation function is always the logistic sigmoid function. Training is stopped when the maximum number of by default 10000 steps is reached, or the mean of the approximation error (7) over all n training patterns falls below 10^{-7}. No heuristics to speed up learning are used, such as momentum parameters or more sophisticated gradients descent methods.

C. Parameter tuning

The intrinsic parameters of the kernels are of decisive importance for the function approximation quality of the proposed kernel Multilayer Perceptron. There is an obvious need to perform an additional validation phase in order to find the best parameter values. Whenever kernels are used, this tuning stage must be done. This constitutes a drawback and a limitation of the proposed method, although many other successful regression/classification paradigms share this burden. Consider, for example, the Support Vector Machine. Always, when using another than a linear kernel, the appropriate values of the parameters must be chosen carefully to obtain optimized results. We illustrate the influence of the γ-parameter of the RBF kernel, c.f. table I for the *pendigit* data set of table II. It has an explicit train-test separation with 7494 training patterns and 3498 test patterns. We expect a statistical significance from this relatively large number of samples and no arbitrary influence of a random train-test split because of the explicit separation. In our experiments we chose a small number of iterations (typically 100 to 1000) to find preliminary good values of the kernel parameters, around which the main experiments with the usually number of 10000 training steps. Fig. 1 emphasizes the need for the right choice of the intrinsic kernel parameters, since the accuracy decreases dramatically with inappropriate γ values.

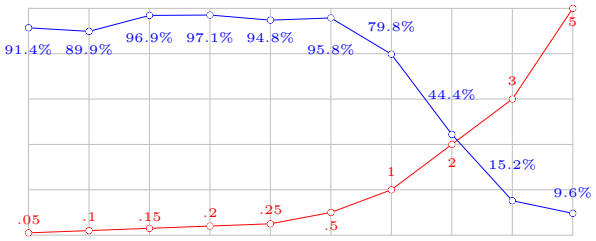

Fig. 1. Influence of the γ parameter of the Radial Basis Function kernel on the accuracy of the kernel MLP classifier for the *pendigit* data set. Ten-fold cross validation with 10000 learning steps was used to estimate the error. Horizontal axis stands for the different cases, vertical axis shows the parameter value and estimated accuracy for these cases.

D. Performance estimation in classification experiments

Table III shows the results of the classification experiments for the UCI data. For all kernels the accuracy and the mean squared error after 10000 iterations (or reaching the MSE limit before) are shown. In order to favor the linear kernel, we repeated the classification performance experiments ten times. The best estimated accuracy of these ten different experiments is shown, together with the standard deviation of these ten runs (in parentheses). For the nonlinear kernels the estimated accuracy is shown, together with the kernel parameter. For example in the 'glass' data set the polynomial kernel of degree five gave an estimated accuracy of 66.82%, or the RBF kernel with $\gamma = 0.4$ estimated and error of 97.06% for the 'pendigits' data set.

E. Evaluation of the experiments

The numerical results cannot justify any theoretical conclusions but point to some interesting observations which in our opinion justify further experiments with the kernel enhanced MLP.

1) Convergence: The values for the mean squared error obtained in the training phase suggest that the kernel enhanced versions do converge as well as the linear MLP. Qualitatively the error is comparable in both the linear and nonlinear case. For example, in the 'shuttle' base the training error reaches $2.04 \cdot 10^{-4}$ for the polynomial kernel and $1.00 \cdot 10^{-4}$ for the linear kernel. Also the test mean squared error in the case of an explicit train/test split is consistently in the same dimension in the linear and nonlinear case, although it is generally higher than the training error, for instance, in 'satimage' 0.179 for the linear kernel and 0.178 for the sigmoid hyperbolic tangent kernel.

2) Performance criterion: We can observe cases in which the introduction of a kernel mapping does not seem to improve the estimated classification accuracy, like in the case of 'breast cancer' and 'glass' where the linear MLP is as good (97.54%) or even better (70.09%) than the kernel enhanced MLPs. This might suggest that the linear kernel is the best for this kind of data set, that we did not find the optimal parameters

TABLE III

EXPLICIT OR 10-FOLD CROSS VALIDATION CLASSIFICATION ACCURACY [%]. FOR KERNEL ENHANCED CLASSIFIERS THE PARAMETER VALUE WITH THE BEST RESULT IS INCLUDED IN BRACKETS. FOR EXPLICIT TRAIN/TEST SEPARATED DATA SETS THE MEAN SQUARED ERROR (MSE) OF THE TRAINING DATA AND TEST DATA IS GIVEN, FOR CROSS VALIDATION THE VALUE IS THE MEAN OF THE 10 RUNS OF THE MSE OF THE TRAINING DATA IS GIVEN. FOR THE LINEAR KERNEL, ADDITIONALLY THE STANDARD DEVIATION σ OF THE ACCURACY FOR TEN EXPERIMENTS IS SHOWN IN PARENTHESES. BEST RESULTS FOR EACH DATA SET IN **boldface**.

Dataset	Kernel Type				
	Linear (Standard deviation 10), MSE	Poly. [degree], MSE	RBF [γ], MSE	Sigmoid tanh [γ], MSE	Sigmoid logistic [γ], MSE
breast cancer	**0.9754** (3.62e-3), 5.47e-2	0.9736 [3], 4.30-3	**0.9754** [0.009], 1.45e-2	**0.9754** [2.3], 3.79e-3	0.9736 [1.3], 5.09e-3
glass	**0.7009** (3.09e-2), 1.85e-2	0.6682 [5], 1.93e-2	0.6122 [0.2], 8.02e-2	0.6916 [2.3], 2.24e-2	0.6729 [1.0], 1.85e-2
survival	0.6585 (3.41e-2), 2.25e-2	0.6829 [7], 4.40e-2	**0.7073** [0.33], 1.46e-1	0.6585 [2.3], 5.16e-2	0.6829 [3.0], 5.92e-2
pendigits	0.9320 (1.96e-3), 3.58e-3 / 1.01e-1	0.9224 [5], 3.63e-3 / 1.01e-1	**0.9706** [0.4], 4.50e-3 / 1.00e-1	0.9691, [3.2], 9.45e-5 / 1.01e-1	0.9291, [0.05], 3.72e-3 / 1.01e-1
pima	0.7331 (2.17e-1), 6.66e-2	0.7448 [5], 7.69e-2	0.7604 [0.03], 1.69e-01	0.7682 [0.02], 1.50e-1	**0.7695** [0.02], 1.62e-1
satimage	0.9060 (4.85e-3), 6.28e-3 / 1.79e-1	0.8900 [3], 1.92e-3 / 1.78e-1	0.8905 [0.7], 2.85e-2 / 1.68e-1	0.8945 [0.2], 4.20e-3 / 1.78e-1	**0.9105** [0.075], 1.22e-2 / 1.74e-1
shuttle	0.9990 (5.44e-4), 1.00e-4 / 1.43e-1	0.9986 [2], 2.04e-4 / 1.43e-1	0.9826 [0.3], 1.78e-2 / 1.40e-1	**0.9994** [1.95], 1.02e-4 / 1.43e-1	0.9980 [0.6], 4.62e-4 / 1.43e-1
sonar	0.8461 (1.82e-2), 2.14e-2	0.8654 [3], 3.75e-3	0.8894 [0.15], 5.71e-4	**0.9038** [2.2], 1.0e-7	0.8750 [5.0], 1.0e-7
vehicle	0.8357 (7.84e-3), 3.00e-3	0.8286 [3], 1.40-3	0.7565 [0.5], 8.53e-2	0.8251 [2.3], 8.53e-2	**0.8463** [0.6], 1.81e-3
wine	0.9888 (6.51e-3), 1.0e-7	0.9831 [2,3,4,7], 1.0e-7	**0.9943** [0.008], 6.91e-3	0.9888 [1.9], 2.22e-5	0.9888 [2.0], 1.0e-7

for the nonlinear kernels used here, or that we did not find yet the appropriate kernel and its parameters to improve the results for the conventional, linear MLP. On the other hand, there are examples of considerable improvement, for example, 'survival' (65.85% to 70.73%), 'pendigits' (93.20% to 97.06%) or 'sonar' (84.61% to 90.38%). For each of the three data bases with explicit training/test split ('pendigits', 'satimage', 'shuttle') we are able to find kernels that improve the results of the linear kernel. This provides a potential justification for using our model. Again, it should be said that there probably are more appropriate classifiers for certain data sets, but we juxtapose the linear and non-linear kernel version of the Multilayer Perceptron and are able to improve performance for certain data sets, kernel types and kernel parameters.

VI. CONCLUSION AND FUTURE WORK

We have extended the general purpose function approximator Multilayer Perceptron in the sense that the argument $\mathbf{x} \in \mathcal{X}$ of the function that has to be learned can be first transformed to an intermediate hidden space, by virtue of a mapping $\boldsymbol{\phi}(\mathbf{x}) \in \mathcal{H}$. This flexibilizes this highly adaptive approximator even more, since an additional nonlinear mapping is preprocessing the original feature vector. One of the advantages of this proposal is that symbolic patterns from a space $\mathcal{X}$ can be processed by the MLP when an appropriate kernel maps the symbolic patterns to an Euclidean implicit space, where similarity metrics can be used. The price for the introduction of the kernel mapping that has to be paid is an additional meta-parameter, namely the type of kernel, together with new intrinsic parameters of this kernel, for instance, the spreads γ in table I.

As mentioned before, applying a hidden-to-output kernel mapping is straightforward. Considering (5) and (23) the mapping of the MLP with input-to-hidden and hidden-to-output kernel mapping becomes

$$\begin{aligned}\mathbf{y} &: \mathcal{X} \to \mathcal{H} \to \Theta \to \mathcal{H} \to \mathcal{Y} \\ &\mathbf{x} \mapsto \boldsymbol{\phi}(\mathbf{x}) \mapsto \boldsymbol{\theta}(\boldsymbol{\phi}(\mathbf{x})) \mapsto \boldsymbol{\phi}(\boldsymbol{\theta}(\boldsymbol{\phi}(\mathbf{x}))) \mapsto \mathbf{y}(\boldsymbol{\phi}(\boldsymbol{\theta}(\boldsymbol{\phi}(\mathbf{x}))))).\end{aligned} \tag{23}$$

The backpropagation step suffers a small modification in (12) where the partial derivative $\partial k_i / \partial \mathbf{w}_i$ has now to be specialized respecting the form of the kernel in table I. This modification would somehow create a new MLP, since the application of the kernel mapping in the input-to-hidden part, just applies the usual MLP algorithm to previously kernel mapped features. An additional hidden-to-output mapping modifies the MLP philosophy in the sense that an additional intermediate feature space is created. The input-hidden and hidden-output kernel types even can be different, for instance, a polynomial kernel and then a radial basis function kernel defining a first intermediate space $\mathcal{H}_1$ and then a second intermediate space $\mathcal{H}_2$. We leave this idea for future work. Additionally other benchmark data sets and kernels be be experimented which hopefully for particular applications improve the approximation capabilities of the Multilayer Perceptron. We also intend to apply the kernel MLP to regression problems.

Finally, our work can readily be simplified to obtain a kernel version of the Adaline [19], [20] and Perceptron [21], however with an explicit representation of the weight vector $\mathbf{w}$, different from the dual representation [6] that does only permit to express the separating hyperplane (for the Perceptron) implicitly in the dual variables $\boldsymbol{\alpha}$. We just have to exclude

the output layer for the Adaline and further post-process the results by the sgn function (for the Perceptron). This is also left for future work.

REFERENCES

[1] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974.

[2] C. PDP Research Group, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 2: psychological and biological models*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA, USA: MIT Press, 1986.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[4] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," Royal Holloway College, University of London, UK, NeuroCOLT Technical Report NC-TR-98-030, 1998, to appear in Statistics and Computing, 2001. [Online]. Available: /papers/tr-30-1998.ps.gz

[5] V. Vapnik, *The Nature of Statistical Learning Theory*. N.Y.: Springer, 1995.

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

[7] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Third Edition*. Orlando, FL, USA: Academic Press, Inc., 2006.

[8] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: John Wiley and Sons, 2001.

[9] B. Schölkopf, *Support Vector Learning*. Munich: R. Oldenbourg Verlag, 1997.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin: Springer, 2007.

[11] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Annals of Statistics*, vol. 36, pp. 1171–1220, 2008. [Online]. Available: http://arxiv.org/pdf/math/0701907

[12] B. Schölkopf, A. Smola, and K. R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, pp. 1299–1319, 1998, technical Report No. 44, 1996, Max Planck Institut für biologische Kybernetik, Tübingen. [Online]. Available: /papers/nlpca.ps.gz

[13] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller, "Fisher discriminant analysis with kernels," in *Neural Networks for Signal Processing IX*, Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, Eds. IEEE, 1999, pp. 41–48. [Online]. Available: /papers/upload_23366_MikRaeWesSchMue99.ps

[14] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 551–556. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014118

[15] K. Yu, L. Ji, and X. Zhang, "Kernel nearest-neighbor algorithm," *Neural Processing Letters*, vol. 15, pp. 147–156, 2002, 10.1023/A:1015244902967. [Online]. Available: http://dx.doi.org/10.1023/A:1015244902967

[16] A. Ruiz and P. López-de Teruel, "Nonlinear kernel-based statistical pattern analysis," *IEEE Transactions on Neural Networks*, vol. 12, no. 1, pp. 16–32, 2001.

[17] F. K. Inaba, E. O. T. Salles, and T. W. Rauber, "Kernel Sammon map," in *XXIV Sibgrapi Conference on Graphics, Patterns and Images*, T. Lewiner and R. Torres, Eds. IEEE Computer Society's Conference Publishing Services (CPS), 2011, pp. 1–7. [Online]. Available: http://sibgrapi.sid.inpe.br/

[18] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[19] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Electronics Lab., Stanford, California, Tech. Rep., June 1960.

[20] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, Aug. 2002. [Online]. Available: http://dx.doi.org/10.1109/5.58323

[21] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.