Multilayer Kerceptron[*]

Zoltán Szabó, András Lőrincz

Department of Information Systems, Faculty of Informatics
Eötvös Loránd University
Pázmány Péter sétány 1/C
H-1117, Budapest, Hungary
e-mail: szzoli@cs.elte.hu, andras.lorincz@elte.hu

Abstract

Multilayer Perceptrons (MLP) are formulated within Support Vector Machine (SVM) framework by constructing multilayer networks of SVMs. The coupled approximation scheme can take advantage of generalization capabilities of the SVM and the combinatory feature of the hidden layer of MLP. The network, the Multilayer Kerceptron (MLK) assumes its own backpropagation procedure that we shall derive here. Tuning rule will be provided for quadratic cost function, with regularization capability as well. A further appealing property of our approach is that by the aid of the so called *kernel trick* the MLK computations can be performed in the dual space.

1 Introduction

Multilayer Perceptrons (MLP) and Support Vector Machines (SVM) have been extensively studied in the literature. For an excellent review, see [3, 2] and references therein. Here we extend SVMs to multi-layer structures and provide the backpropagation tuning rules for this system. By applying the so called kernel-trick, we embed the problem into a space having scalar product. For other approaches using the same trick, see, e.g., [4, 6, 7].

2 Network Architecture

2.1 Notations

Numbers (a), vectors ($\mathbf{a}$), and matrices ($\mathbf{A}$) are denoted by different letter types. $\mathbf{A}^T$ denotes the transpose of matrix $\mathbf{A}$. Extension of vector $\mathbf{a}$ by component a is

[*]Journal of Applied Mathematics, 24:209-222, 2007.

written as $[\mathbf{a}; a]$. $\mathbb{R}$ stands for real numbers. $\|\cdot\|_2$ indicates the L_2 norm induced by the scalar product $\langle \cdot, \cdot \rangle$ of Euclidean space E, i.e., $\|\mathbf{e}\|_2 = \sqrt{\langle \mathbf{e}, \mathbf{e} \rangle} \quad (\mathbf{e} \in E)$.

2.2 Building Blocks

2.2.1 SVM

SVMs are popular approximation tools [9, 10, 8, 6, 5]. SVMs approximate $\{\mathbf{x}(t), d(t)\}_{t=1..T}$ sample pairs, where each $\mathbf{x}(t)$ input is in *input space* $\mathcal{X}$ and $d(t) \in \mathbb{R}$. The approximation is linear, but it occurs in *feature space* $\mathcal{H}$. Inputs $\mathbf{x}(t)$ are mapped to feature space by

$$\boldsymbol{\varphi} : \mathbf{x} \in \mathcal{X} \to \mathcal{H}. \tag{1}$$

One can interpret $\boldsymbol{\varphi}(\mathbf{x})$ as the *representation* of input $\mathbf{x}$. The form of the SVM approximation is

$$f_{\mathbf{w}} : \mathbf{x} \in \mathcal{X} \mapsto \langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{x}) \rangle_{\mathcal{H}} \quad (\mathbf{w} \in \mathcal{H}). \tag{2}$$

Formally, the SVM-task is defined as

$$\min_{\mathbf{w}} H[\mathbf{w}] := C \cdot \sum_{t=1}^{T} V[d(t), f_{\mathbf{w}}(\mathbf{x}(t))] + \frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 \quad (C > 0), \tag{3}$$

where $V[\cdot, \cdot]$ is the so called loss function, which can assume quadratic, ϵ-insensitive, or other forms [4]. That is, SVMs are regularized linear approximators [2].

Instead of using the explicit $\boldsymbol{\varphi}$ mapping, feature space $\mathcal{H}$ can be exploited through kernel k, that is $\mathcal{H} = \mathcal{H}(k)$ [11], where $\boldsymbol{\varphi}(\mathbf{x}) = k(\cdot, \mathbf{x})$. Kernel k assumes the reproducing property [1, 11]

$$\langle f(\cdot), k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = f(\mathbf{x}) \quad (\mathbf{x} \in \mathcal{X}, \forall f \in \mathcal{H}), \tag{4}$$

and $\mathcal{H}$ is called Reproducing Kernel Hilbert Space (RKHS). Scalar product of any function with kernel $k(\cdot, \mathbf{x})$ evaluates the function at $\mathbf{x}$ in RKHS $\mathcal{H}$. Scalar product in feature space can be computed implicitly by means of the kernel

$$k(\mathbf{u}, \mathbf{v}) = \langle \boldsymbol{\varphi}(\mathbf{u}), \boldsymbol{\varphi}(\mathbf{v}) \rangle_{\mathcal{H}} \quad (\mathbf{u}, \mathbf{v} \in \mathcal{X}). \tag{5}$$

In particular, for $\mathbf{w} = \sum_{j=1}^{N} \alpha_j \cdot \boldsymbol{\varphi}(\mathbf{z}_j) \quad (\alpha_j \in \mathbb{R}, \mathbf{z}_j \in \mathcal{X})$ we have

$$f_{\mathbf{w}}(\mathbf{x}) \quad = \quad \langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{x}) \rangle_{\mathcal{H}} = \sum_{j=1}^{N} \alpha_j \cdot \langle \boldsymbol{\varphi}(\mathbf{z}_j), \boldsymbol{\varphi}(\mathbf{x}) \rangle_{\mathcal{H}} = \sum_{j=1}^{N} \alpha_j \cdot k(\mathbf{z}_j, \mathbf{x}). \tag{6}$$

Thus, function $f_{\mathbf{w}}$ can be evaluated by means of coefficients α_j, samples $\mathbf{z}_j$ and the kernel k without explicit reference to representation $\boldsymbol{\varphi}(\mathbf{x})$. This is called the *kernel trick*.

2.2.2 MLP

An MLP network has multiple layers, each performing non-linear mapping

$$\mathbf{x} \mapsto \mathbf{g}(\mathbf{W} \cdot \mathbf{x}). \tag{7}$$

Here, $\mathbf{g}$ is a differentiable non-linear function. In the MLP task, we tune matrix $\mathbf{W}$ for all layers so that the network approximates the sampled input-output mapping given by input-output training pairs $\{\mathbf{x}(t), \mathbf{d}(t)\}$. That is, the objective is to minimize the squared error

$$\varepsilon^2(t) := \|\mathbf{d}(t) - \mathbf{y}(t)\|_2^2 \to \min_{\mathbf{W}_1, \mathbf{W}_2, \ldots}, \tag{8}$$

where the output of the network at time t is $\mathbf{y}(t)$, for all times. The MLP task is solved by the well-known backpropagation algorithm.

2.3 The MLK Architecture

The mapping of a general MLP layer (i.e., Eq. (7)) can be written as

$$\mathbf{x} \mapsto \mathbf{g}\left(\begin{bmatrix} \vdots \\ \langle \mathbf{w}_i, \mathbf{x} \rangle \\ \vdots \end{bmatrix}\right), \tag{9}$$

where $\mathbf{w}_i^T$ denotes the i^{th} row of matrix $\mathbf{W}$. The SVM can also be inserted into the MLP: Let a general layer of the network assume the form[1]

$$\mathbf{x} \mapsto \mathbf{g}\left(\begin{bmatrix} \langle \mathbf{w}_1, \boldsymbol{\varphi}(\mathbf{x}) \rangle_{\mathcal{H}} \\ \vdots \\ \langle \mathbf{w}_N, \boldsymbol{\varphi}(\mathbf{x}) \rangle_{\mathcal{H}} \end{bmatrix}\right). \tag{10}$$

A network made of such layers, see Fig. 1, will be called Multilayer Kerceptron (MLK). The input ($\mathbf{x}^l$) of each layer is the output of the preceding layer ($\mathbf{y}^{l-1}$). The external world is the 0^{th} layer providing input to the first layer of the MLK. $\mathbf{x}^l = \mathbf{y}^{l-1} \in \mathbb{R}^{N_\mathrm{I}^l}$, where N_I^l is the dimension of the l^{th} layer. Inputs $\mathbf{x}^l$ to layer l are mapped by features $\boldsymbol{\varphi}^l$ and are multiplied by the weights $\mathbf{w}_i^l$. This two-step process can be accomplished implicitly by making use of kernel k^l and the expansion property for $\mathbf{w}_i^l$s. The result is vector $\mathbf{s}^l \in \mathbb{R}^{N_\mathrm{S}^l}$, which undergoes non-linear processing $\mathbf{g}^l$, where function $\mathbf{g}^l$ is differentiable. The output of this non-linear function is the input to the next layer, i.e., layer $\mathbf{x}^{l+1}$. The output of the last layer (layer L, the output of the network) will be referred to as $\mathbf{y}$. Given that $\mathbf{y}^l = \mathbf{x}^{l+1} \in \mathbb{R}^{N_\mathrm{O}^l}$, the output dimension of layer l is N_O^l.

Below, we show that (i) a backpropagation rule can be derived for MLKs, which (ii) requires the kernels only, so computations can be accomplished in dual space.

[1] For the sake of simplicity let us choose sample space $\mathcal{X}$ as finite dimensional Euclidean space, i.e., $\mathbb{R}^n$.

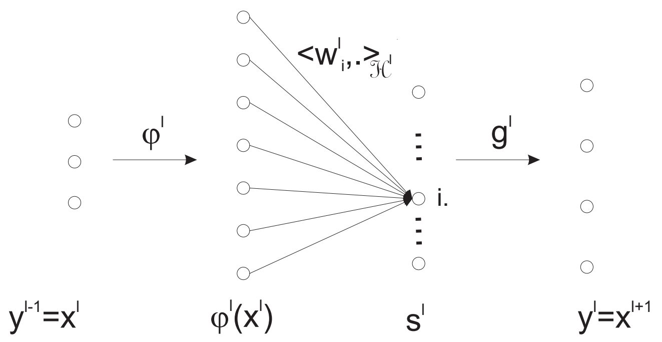

Figure 1: The l^{th} layer of the MLK, $l = 1, 2, \ldots L$. The input ($\mathbf{x}^l$) of each layer is the output of the preceding layer ($\mathbf{y}^{l-1}$). The external world is the 0^{th} layer providing input to the first layer of the MLK. Inputs $\mathbf{x}^l$ to layer l are mapped by features mapping $\boldsymbol{\varphi}^l$ undergo scalar multiplication by the weights ($\mathbf{w}_i^l$) of the layer in RKHS $\mathcal{H}^l = \mathcal{H}^l(k^l)$. The result is vector $\mathbf{s}^l$, which undergoes non-linear processing $\mathbf{g}^l$, with a differentiable function. The output of this non-linear function is the input to the next layer, layer $\mathbf{x}^{l+1}$. The output of the network is the output of the last layer.

3 MLK Backpropagation

A slightly more general task, which incorporates regularizing terms, too, is formalized below:

$$c(t) := \varepsilon^2(t) + r(t) \longrightarrow \min_{\{\mathcal{H}^l \ni \mathbf{w}_i^l:\ l=1,\ldots,L;\ i=1,\ldots,N_S^l\}}, \tag{11}$$

where $\varepsilon^2(t) = \|\mathbf{d}(t) - \mathbf{y}(t)\|_2^2$ and $r(t) = \sum_{l=1}^{L}\sum_{i=1}^{N_S^l} \lambda_i^l \cdot \left\|\mathbf{w}_i^l(t)\right\|_{\mathcal{H}^l}^2 \quad (\lambda_i^l \geq 0)$ are the approximation and the regularization terms of the cost function, respectively, and $\mathbf{y}(t)$ denotes the output of the network for the t^{th} input. Parameters λ_i^l control the trade-off between approximation and regularization. For $\lambda_i^l = 0$ the best approximation is searched like in the MLP task (Eq. (8)). Increasing the λ_i^l values, the smoothness of the approximation will increase.

With the notations introduced above, the following statements can be proven.

Theorem 1 (explicit case)**.** *Let us suppose that the* $\mathbf{x} \mapsto \left\langle \mathbf{w}, \boldsymbol{\varphi}^l(\mathbf{x})\right\rangle_{\mathcal{H}^l}$ *and the* $\mathbf{g}^l$ *functions are all differentiable* ($l = 1, \ldots, L$)*. Then, backpropagation rule can be derived for MLK if the cost function has the form*

$$c(t) = \varepsilon^2(t) + \sum_{l=1}^{L}\sum_{i=1}^{N_S^l} \lambda_i^l \cdot \left\|\mathbf{w}_i^l(t)\right\|_{\mathcal{H}^l}^2 \quad (\lambda_i^l \geq 0). \tag{12}$$

Theorem 2 (implicit case)**.** *Assume that the following holds*

1. *Constraint on differentiability: Kernels k^l are differentiable with respect to both arguments, functions $\mathbf{g}^l$ are also differentiable ($l = 1, \ldots, L$).*
2. *Expansion property: The initial weights $\mathbf{w}_i^l(1)$ of the network can be expressed in the dual representation, i.e.,*

$$\mathcal{H}^l \ni \mathbf{w}_i^l(1) = \sum_{j=1}^{N_i^l(1)} \alpha_{i,j}^l(1) \cdot \boldsymbol{\varphi}^l(\mathbf{z}_{i,j}^l(1)) \quad (l = 1, \ldots, L; i = 1, \ldots, N_\mathrm{S}^l). \tag{13}$$

Then backpropagation applies for MLK if the cost function has the form

$$c(t) = \varepsilon^2(t) + \sum_{l=1}^{L} \sum_{i=1}^{N_\mathrm{S}^l} \lambda_i^l \cdot \left\| \mathbf{w}_i^l(t) \right\|_{\mathcal{H}^l}^2 \quad (\lambda_i^l \geq 0). \tag{14}$$

This procedure spares the expansion property (13)*, which then remains valid for the tuned network. The algorithm is implicit in the sense that it can be realized in the dual space.*

The pseudo codes of the MLK backpropagation algorithms are provided in Table 1 and Table 2, respectively. Derivations of these algorithms, both for the explicit and for the implicit forms, are provided in the next subsection.

MLK-backpropagation can be envisioned as follows (see Table 1 and 2 simultaneously):

1. backpropagated error $\boldsymbol{\delta}^l(t)$ starts from $\boldsymbol{\delta}^L(t)$ and is computed by a backward recursion via the differential expression $\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]}$.
2. expression $\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]}$ can be determined by means of feature mapping φ^{l+1}, or, in an implicit fashion, through kernels k^{l+1}.
3. two components play roles in the tuning of **w**s:
   (a) *forgetting* is accomplished by scaling the weights $\mathbf{w}_i^l$ with multiplier $\left(1 - 2\mu_i^l(t) \cdot \lambda_i^l\right)$, where λ_i^l is the regularization coefficient.
   (b) *adaptation* occurs through the backpropagated error. Weights at layer l are tuned by feature space representation of $\mathbf{x}^l(t)$, the actual input arriving at layer l. Tuning is weighted by the backpropagated error.

3.1 Derivation of the Backpropagation Algorithm for MLKs

Gradient $\frac{d[c(t)]}{d[\mathbf{w}_i^l(t)]}$ is derived first. Then it is embedded into steepest descent tuning.[2] The $c(t)$ error has two terms, the approximation and the regularization

[2] Steepest descent is used to illustrate the concepts. Other types of gradient optimizations beyond steepest descent may be utilized. For example, different versions of the momentum method or the conjugate gradient procedure could have their respective advantages.

Table 1: Pseudocode of the explicit MLK backpropagation algorithm

```
Inputs
    sample points: ${\{\mathbf{x}(t), \mathbf{d}(t)\}_{t=1,\ldots,T}}, T$
    cost function: $\lambda_i^l \geq 0$ $(l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l)$
    learning rates: $\mu_i^l(t) > 0$ $(l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l; t = 1, \ldots, T)$
Network initialization
    size: $L$ (number of layers), $N_{\mathrm{I}}^l$, $N_{\mathrm{S}}^l$, $N_{\mathrm{o}}^l$ $(l = 1, \ldots, L)$
    parameters: $\mathbf{w}_i^l(1)$ $(l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l)$
Start computation
    Choose sample $\mathbf{x}(t)$
    Feedforward computation
        $\mathbf{x}^l(t)$ $(l = 2, \ldots, L+1)$, $\mathbf{s}^l(t)$ $(l = 2, \ldots, L)^a$
    Backpropagation of error
        $l = L$
        while $l \geq 1$
            if $(l = L)$
                $\boldsymbol{\delta}^L(t) = 2 \cdot [\mathbf{y}(t) - \mathbf{d}(t)]^T \cdot \left(\mathbf{g}^L\right)' (\mathbf{s}^L(t))$
            else
                $\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]} = \begin{bmatrix} \vdots \\ \left.\frac{d\left[\left\langle \mathbf{w}_i^{l+1}(t), \boldsymbol{\varphi}^{l+1}(\mathbf{u}) \right\rangle_{\mathcal{H}^{l+1}}\right]}{d[\mathbf{u}]}\right|_{\mathbf{u}=\mathbf{x}^{l+1}(t)} \\ \vdots \end{bmatrix} \cdot \left[\left(\mathbf{g}^l\right)' (\mathbf{s}^l(t))\right]^b$
                $\boldsymbol{\delta}^l(t) = \boldsymbol{\delta}^{l+1}(t) \cdot \frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]}$
            Weight update
                for all $i$: $1 \leq i \leq N_S^l$
                    $\mathbf{w}_i^l(t+1) = (1 - 2\mu_i^l(t) \cdot \lambda_i^l) \cdot \mathbf{w}_i^l(t) - \mu_i^l(t) \cdot \delta_i^l(t) \cdot \boldsymbol{\varphi}^l(\mathbf{x}^l(t))$
            $l = l - 1$
End computation
```

[a] The output of the network, i.e., $\mathbf{y}(t) = \mathbf{x}^{L+1}(t)$ is also computed.

[b] Here: $i = 1, \ldots, N_{\mathrm{S}}^{l+1}$.

Table 2: Pseudocode of the implicit MLK backpropagation algorithm

```
**Inputs**
    sample points: $\{\mathbf{x}(t), \mathbf{d}(t)\}_{t=1,\ldots,T}, T$
    cost function: $\lambda_i^l \geq 0 \ (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l)$
    learning rates: $\mu_i^l(t) > 0 \ (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l; t = 1, \ldots, T)$
**Network initialization**
    size: $L$ (number of layers), $N_{\mathrm{I}}^l$, $N_{\mathrm{S}}^l$, $N_{\mathrm{o}}^l$ $(l = 1, \ldots, L)$
    parameters: $\mathbf{w}_i^l(1)$-expansions $(l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l)$
        coefficients: $\boldsymbol{\alpha}_i^l(1) \in \mathbb{R}^{N_i^l(1)}$
        ancestors: $\mathbf{z}_{i,j}^l(1)$, where $j = 1, \ldots, N_i^l(1)$
**Start computation**
    **Choose sample** $\mathbf{x}(t)$
    **Feedforward computation**
        $\mathbf{x}^l(t)$ $(l = 2, \ldots, L+1)$, $\mathbf{s}^l(t)$ $(l = 2, \ldots, L)$[a]
    **Backpropagation of error**
        $l = L$
        while $l \geq 1$
            if $(l = L)$
                $\boldsymbol{\delta}^L(t) = 2 \cdot [\mathbf{y}(t) - \mathbf{d}(t)]^T \cdot \left(\mathbf{g}^L\right)' (\mathbf{s}^L(t))$
            else
                $\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]} = \begin{bmatrix} \vdots \\ \sum_{j=1}^{N_i^{l+1}(t)} \alpha_{ij}^{l+1}(t) \cdot [k^{l+1}]'_y(\mathbf{z}_{ij}^{l+1}(t), \mathbf{x}^{l+1}(t)) \\ \vdots \end{bmatrix} \cdot \left[\left(\mathbf{g}^l\right)' (\mathbf{s}^l(t))\right]$[b]
                $\boldsymbol{\delta}^l(t) = \boldsymbol{\delta}^{l+1}(t) \cdot \frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]}$
            **Weight update**
                for all $i$: $1 \leq i \leq N_S^l$
                    $N_i^l(t+1) = N_i^l(t) + 1$
                    $\boldsymbol{\alpha}_i^l(t+1) = \left[\left(1 - 2\mu_i^l(t) \cdot \lambda_i^l\right) \cdot \boldsymbol{\alpha}_i^l(t); -\mu_i^l(t) \cdot \delta_i^l(t)\right]$
                    $\mathbf{z}_{i,j}^l(t+1) = \mathbf{z}_{i,j}^l(t) \quad (j = 1, \ldots, N_i^l(t))$
                    $\mathbf{z}_{i,j}^l(t+1) = \mathbf{x}^l(t) \quad (j = N_i^l(t+1))$
            $l = l - 1$
**End computation**
```

[a] The output of the network, i.e., $\mathbf{y}(t) = \mathbf{x}^{L+1}(t)$ is also computed.

[b] $i = 1, \ldots, N_{\mathrm{S}}^{l+1}$. Note also that $(k^l)'_y$ denotes the derivative of kernel k^l according to its second argument.

terms:

$$c(t) = \varepsilon^2(t) + r(t). \tag{15}$$

3.1.1 Gradient of the Approximation Term

First, we list basic relations, involved by the MLK structure. For the case of simplicity, below, index t shall be dropped [precise form: $\mathbf{x}^l = \mathbf{x}^l(t)$, $\mathbf{y}^l = \mathbf{y}^l(t)$, $\mathbf{s}^l = \mathbf{s}^l(t)$,$\mathbf{w}_i^l = \mathbf{w}_i^l(t)$].

$$\begin{aligned}
\mathbf{x}^l &= \mathbf{y}^{l-1} \in \mathbb{R}^{N_{\mathrm{I}}^l} \quad (l = 1, \ldots, L+1) && (16)\\
\mathbf{x}^{l+1} &= \mathbf{g}^l(\mathbf{s}^l) \quad (l = 1, \ldots, L) && (17)\\
\mathbf{s}^l &= \begin{bmatrix} \langle \mathbf{w}_1^l, \boldsymbol{\varphi}^l(\mathbf{x}^l) \rangle_{\mathcal{H}^l} \\ \vdots \\ \langle \mathbf{w}_i^l, \boldsymbol{\varphi}^l(\mathbf{x}^l) \rangle_{\mathcal{H}^l} \\ \vdots \end{bmatrix} \quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l) && (18)\\
&= \begin{bmatrix} \langle \mathbf{w}_1^l, \boldsymbol{\varphi}^l(\mathbf{g}^{l-1}(\mathbf{s}^{l-1})) \rangle_{\mathcal{H}^l} \\ \vdots \\ \langle \mathbf{w}_i^l, \boldsymbol{\varphi}^l(\mathbf{g}^{l-1}(\mathbf{s}^{l-1})) \rangle_{\mathcal{H}^l} \\ \vdots \end{bmatrix} \quad (l = 2, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l) && (19)\\
\mathbf{s}^{l+1} &= \begin{bmatrix} \langle \mathbf{w}_1^{l+1}, \boldsymbol{\varphi}^{l+1}(\mathbf{g}^l(\mathbf{s}^l)) \rangle_{\mathcal{H}^{l+1}} \\ \vdots \\ \langle \mathbf{w}_i^{l+1}, \boldsymbol{\varphi}^{l+1}(\mathbf{g}^l(\mathbf{s}^l)) \rangle_{\mathcal{H}^{l+1}} \\ \vdots \end{bmatrix} && (20)\\
& (l = 1, \ldots, L-1; i = 1, \ldots, N_{\mathrm{S}}^{l+1})
\end{aligned}$$

Let the backpropagated error for layer l be defined as

$$\boldsymbol{\delta}^l(t) := \frac{d[\varepsilon^2(t)]}{d[\mathbf{s}^l(t)]} \quad (l = 1, \ldots, L). \tag{21}$$

The special case of the last layer is as follows:

$$\begin{aligned}
\boldsymbol{\delta}^L(t) &= \frac{d[\varepsilon^2(t)]}{d[\mathbf{s}^L(t)]} = \frac{d\left[\left\|\mathbf{d}(t) - \mathbf{g}^L(\mathbf{s}^L(t))\right\|_2^2\right]}{d[\mathbf{s}^L(t)]} && (22)\\
&= 2 \cdot \left[\mathbf{g}^L\left(\mathbf{s}^L(t)\right) - \mathbf{d}(t)\right]^T \cdot \left(\mathbf{g}^L\right)' \left(\mathbf{s}^L(t)\right) && (23)\\
&= 2 \cdot [\mathbf{y}(t) - \mathbf{d}(t)]^T \cdot \left(\mathbf{g}^L\right)' (\mathbf{s}^L(t)). && (24)
\end{aligned}$$

Here we used the chain rule and made use of the rule valid for vectors

$$\frac{d[\|\mathbf{d} - \mathbf{y}\|_2^2]}{d\mathbf{y}} = 2(\mathbf{y} - \mathbf{d})^T, \tag{25}$$

and inserted the relation

$$\mathbf{y}(t) = \mathbf{g}^L\left(\mathbf{s}^L(t)\right), \tag{26}$$

imposed by the MLK architecture.

Expression

$$\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]} \quad (l = 1, \ldots, L-1) \tag{27}$$

can be computed by using Eq. (20). It is sufficient to consider terms like

$$\frac{d[\langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{g}(\mathbf{s}))\rangle_{\mathcal{H}}]}{d[\mathbf{s}]} \tag{28}$$

and then to 'compile' the full derivative from them. The value of (28) is computed by means of the following lemma.

Lemma 1. *Let $\mathbf{w} \in \mathcal{H} = \mathcal{H}(k)$ be a point in the RKHS. Let us assume the following*

1. *Let kernel k be differentiable w.r.t. both arguments and let k'_y denote the derivative of the kernel according to its second argument.*
2. *In the implicit case we also assume that $\mathbf{w}$ is within the image space of the feature space representation of a finite number of points $\mathbf{z}_i$. That is*

$$\mathbf{w} \in Im\left(\boldsymbol{\varphi}(\mathbf{z}_1), \boldsymbol{\varphi}(\mathbf{z}_2), \ldots, \boldsymbol{\varphi}(\mathbf{z}_N)\right) \subseteq \mathcal{H}. \tag{29}$$

   Let this expansion be $\mathbf{w} = \sum_{j=1}^{N} \alpha_j \cdot \boldsymbol{\varphi}(\mathbf{z}_j)$, where $\alpha_j \in \mathbb{R}$.

Then we have two cases:

1. *Explicit case:*

$$\frac{d[\langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{g}(\mathbf{s}))\rangle_{\mathcal{H}}]}{d[\mathbf{s}]} = \left.\frac{d\left[\langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{u})\rangle_{\mathcal{H}}\right]}{d[\mathbf{u}]}\right|_{\mathbf{u}=\mathbf{g}(\mathbf{s})} \cdot \mathbf{g}'(\mathbf{s}) \tag{30}$$

2. *Implicit case:*

$$\frac{d[\langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{g}(\mathbf{s}))\rangle_{\mathcal{H}}]}{d[\mathbf{s}]} = \sum_{j=1}^{N} \alpha_j \cdot k'_y(\mathbf{z}_j, \mathbf{g}(\mathbf{s})) \cdot \mathbf{g}'(\mathbf{s}) \tag{31}$$

Proof.

1. Explicit case: the statement follows from the chain rule.

2. Implicit case:

$$\frac{d[\langle \mathbf{w}, \boldsymbol{\varphi}(\mathbf{g}(\mathbf{s}))\rangle_{\mathcal{H}}]}{d[\mathbf{s}]} = \frac{d\left[\left\langle \sum_j \alpha_j \cdot \boldsymbol{\varphi}(\mathbf{z}_j), \boldsymbol{\varphi}(\mathbf{g}(\mathbf{s}))\right\rangle_{\mathcal{H}}\right]}{d[\mathbf{s}]} \tag{32}$$

$$= \frac{d\left[\sum_j \alpha_j \cdot \langle \boldsymbol{\varphi}(\mathbf{z}_j), \boldsymbol{\varphi}(\mathbf{g}(\mathbf{s}))\rangle_{\mathcal{H}}\right]}{d[\mathbf{s}]} \tag{33}$$

$$= \frac{d\left[\sum_j \alpha_j \cdot k\left(\mathbf{z}_j, \mathbf{g}(\mathbf{s})\right)\right]}{d[\mathbf{s}]} \tag{34}$$

$$= \sum_j \alpha_j \cdot k'_y(\mathbf{z}_j, \mathbf{g}(\mathbf{s})) \cdot \mathbf{g}'(\mathbf{s}). \tag{35}$$

The first equation has the expansion of $\mathbf{w}$ and the linear property of the scalar product was utilized. Then, the relation

$$k(\mathbf{u}, \mathbf{v}) = \langle \boldsymbol{\varphi}(\mathbf{u}), \boldsymbol{\varphi}(\mathbf{v})\rangle_{\mathcal{H}} \tag{36}$$

between feature mapping and the kernel was applied. The last step follows from the chain rule.

$\square$

Let us turn back to the computation of Eq. (27):

1. Explicit case: According to the lemma we have

$$\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]} = \begin{bmatrix} \vdots \\ \left.\frac{d\left[\left\langle \mathbf{w}_i^{l+1}(t), \boldsymbol{\varphi}^{l+1}(\mathbf{u})\right\rangle_{\mathcal{H}^{l+1}}\right]}{d[\mathbf{u}]}\right|_{\mathbf{u}=\mathbf{g}^l(\mathbf{s}^l(t))} \cdot \left(\mathbf{g}^l\right)'\left(\mathbf{s}^l(t)\right) \\ \vdots \end{bmatrix} \tag{37}$$

$$= \begin{bmatrix} \vdots \\ \left.\frac{d\left[\left\langle \mathbf{w}_i^{l+1}(t), \boldsymbol{\varphi}^{l+1}(\mathbf{u})\right\rangle_{\mathcal{H}^{l+1}}\right]}{d[\mathbf{u}]}\right|_{\mathbf{u}=\mathbf{x}^{l+1}(t)} \\ \vdots \end{bmatrix} \cdot \left[\left(\mathbf{g}^l\right)'\left(\mathbf{s}^l(t)\right)\right] \tag{38}$$

$$(l = 1, \ldots, L-1; i = 1, \ldots, N_{\mathrm{S}}^{l+1}).$$

In the second equation (i) we used identity (17) and (ii) pulled out the term $\left(\mathbf{g}^l\right)'\left(\mathbf{s}^l(t)\right)$ according to the matrix multiplication rules.

2. Implicit case: For terms $\mathbf{w}_i^{l+1}(t)$ we have the expansion property expressed by Eq. (13). This was our starting assumption. In subsection 3.1.3, we shall see that this feature is 'inherited' from time to time. Thus,

$$\mathbf{w}_i^{l+1}(t) = \sum_{j=1}^{N_i^{l+1}(t)} \alpha_{ij}^{l+1}(t) \cdot \boldsymbol{\varphi}^{l+1}(\mathbf{z}_{ij}^{l+1}(t)) \quad (l = 1, \ldots, L-1; i = 1, \ldots, N_{\mathrm{S}}^{l+1}) \tag{39}$$

and the derivative (27) we need assumes the form

$$\frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^{l}(t)]} =$$

$$= \begin{bmatrix} \vdots \\ \sum_{j=1}^{N_i^{l+1}(t)} \alpha_{ij}^{l+1}(t) \cdot [k^{l+1}]'_y(\mathbf{z}_{ij}^{l+1}(t), \mathbf{g}^l(\mathbf{s}^l(t))) \cdot \left(\mathbf{g}^l\right)' \left(\mathbf{s}^l(t)\right) \\ \vdots \end{bmatrix} \tag{40}$$

$$= \begin{bmatrix} \vdots \\ \sum_{j=1}^{N_i^{l+1}(t)} \alpha_{ij}^{l+1}(t) \cdot [k^{l+1}]'_y(\mathbf{z}_{ij}^{l+1}(t), \mathbf{x}^{l+1}(t)) \\ \vdots \end{bmatrix} \cdot \left[\left(\mathbf{g}^l\right)' \left(\mathbf{s}^l(t)\right)\right] \tag{41}$$

$$(l = 1, \ldots, L-1; i = 1, \ldots, N_{\mathrm{S}}^{l+1}).$$

Here, the second equation is based on identity (17). Matrix term $\left(\mathbf{g}^l\right)' \left(\mathbf{s}^l(t)\right)$ was pulled out according to the matrix multiplication rules.

Applying the chain rule and the definition of $\boldsymbol{\delta}^{l+1}(t)$, we have

$$\boldsymbol{\delta}^l(t) = \frac{d[\varepsilon^2(t)]}{d[\mathbf{s}^l(t)]} = \frac{d[\varepsilon^2(t)]}{d[\mathbf{s}^{l+1}(t)]} \cdot \frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]} = \boldsymbol{\delta}^{l+1}(t) \cdot \frac{d[\mathbf{s}^{l+1}(t)]}{d[\mathbf{s}^l(t)]} \quad (l = 1, \ldots, L-1). \tag{42}$$

One can apply the chain rule once again and can make use the definitions of $\boldsymbol{\delta}^l(t)$ and $\mathbf{s}^l(t)$ to show that

$$\frac{d[\varepsilon^2(t)]}{d[\mathbf{w}_i^l(t)]} = \frac{d[\varepsilon^2(t)]}{d[s_i^l(t)]} \cdot \frac{d[s_i^l(t)]}{d[\mathbf{w}_i^l(t)]} = \delta_i^l(t) \cdot \boldsymbol{\varphi}^l(\mathbf{x}^l(t)) \quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l), \tag{43}$$

which is the desired derivative. Note that the derivative can be expressed by using number $\delta_i^l(t)$ and by the feature representation of the input $\mathbf{x}^l(t)$ arriving to the l^{th} layer, i.e., by $\boldsymbol{\varphi}^l(\mathbf{x}^l(t))$.

3.1.2 Regularization Term

This term is relatively simple:

$$\frac{d[r(t)]}{d[\mathbf{w}_i^l(t)]} = \frac{d\left[\sum_{l=1}^{L} \sum_{i=1}^{N_{\mathrm{S}}^l} \lambda_i^l \cdot \left\|\mathbf{w}_i^l(t)\right\|_{\mathcal{H}^l}^2\right]}{d[\mathbf{w}_i^l(t)]} = 2\lambda_i^l \cdot \mathbf{w}_i^l(t) \quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l). \tag{44}$$

Note that the respective terms of the derivative are scaled actual weights $[\mathbf{w}_i^l(t)]$. This form enables our implicit tuning.

3.1.3 Cost Term

Using identity

$$\frac{d[c(t)]}{d[\mathbf{w}_i^l(t)]} = \frac{d[\varepsilon^2(t)]}{d[\mathbf{w}_i^l(t)]} + \frac{d[r(t)]}{d[\mathbf{w}_i^l(t)]} \quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l) \tag{45}$$

as well as our results on the approximation and the regularization terms [i.e., Eqs. (43), and (44)], we arrive to the steepest descent form

$$\mathbf{w}_i^l(t+1) = \mathbf{w}_i^l(t) - \mu_i^l(t) \cdot \frac{d[c(t)]}{d[\mathbf{w}_i^l(t)]} \quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l). \tag{46}$$

So we have

$$\begin{aligned} \mathbf{w}_i^l(t+1) &= \mathbf{w}_i^l(t) - \mu_i^l(t) \cdot \left(\delta_i^l(t) \cdot \boldsymbol{\varphi}^l(\mathbf{x}^l(t)) + 2\lambda_i^l \cdot \mathbf{w}_i^l(t)\right) & (47)\\ &= (1 - 2\mu_i^l(t) \cdot \lambda_i^l) \cdot \mathbf{w}_i^l(t) - \mu_i^l(t) \cdot \delta_i^l(t) \cdot \boldsymbol{\varphi}^l(\mathbf{x}^l(t)) & (48)\\ &\quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l). \end{aligned}$$

The same in dual form is as follows

$$\boldsymbol{\alpha}_i^l(t+1) = [\left(1 - 2\mu_i^l(t) \cdot \lambda_i^l\right) \cdot \boldsymbol{\alpha}_i^l(t); -\mu_i^l(t) \cdot \delta_i^l(t)] \quad (l = 1, \ldots, L; i = 1, \ldots, N_{\mathrm{S}}^l). \tag{49}$$

In turn, the expansion property of the weight vectors of the network [i.e., Eq. (13)] is inherited from time to time. In particular, the expansion is valid for parameter set $\mathbf{w}_i^l$ received at the end of the computation. In summing up, MLK can be tuned by the backpropagation procedure. The derived explicit and implicit procedures are summarized in Table 1 and Table 2, respectively.

4 Conclusions

Theoretical description of a novel multilayer model, the Multilayer Kerceptron was provided. This network unifies the advantages of Multilayer Perceptrons and Support Vector Machines: (i) It learns the weights and learning is subject to regularization. (ii) MLK allows for feature representations. (iii) MLK computes the output quickly through learned weights. (iv) MLK can have *hidden layers*, and thus, it can *combine* SVM partitionings. Advantages and disadvantages of the approach for different databases remain to be seen.

References

[1] N. Aronszajn. Theory of Reproducing Kernels. *Trans. of Am. Math. Soc.*, 68:337–404, 1950.

[2] T. Evgeniou, M. Pontil, and T. Poggio. Regularization Networks and Support Vector Machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[3] S. Haykin. *Neural Networks.* Prentice Hall, New Jersey, USA, 1999.

[4] R. Herbrich. *Learning Kernel Classifiers.* MIT Press, 2002.

[5] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting Time Series with Support Vector Machines. In *Advances in Kernel Methods*, pages 243–254. MIT Press, 1999.

[6] B. Schölkopf and A.J. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.

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

[8] V. Vapnik, S. Golowich, and A. Smola. *Support Vector Method for Function Estimation, Regression Estimation and Signal Processing*, volume Vol. 9. MIT Press, Cambridge, MA, neural information processing systems edition, 1997.

[9] V.N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., 1995.

[10] V.N. Vapnik. *Statistical Learning Theory.* Wiley, Chichester, GB, 1998.

[11] G. Wahba. Support Vector Machines, Reproducing Kernel Hilbert Spaces, and Randomized GACV. In *Advances in Kernel Methods*, pages 69–88. MIT Press, 1999.