1 Using Support Vector Machines for Time Series Prediction

K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen
GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany.
{klaus, smola, raetsch, bs, jek }@first.gmd.de.
http://candy.first.gmd.de.

V. Vapnik
Image Processing Services Research Lab, AT&T Labs - Research
100 Schulz Drive, Red Bank, NJ 07701-7033, USA
vlad@research.att.com.
http://www.research.att.com

Support Vector Machines are used for time series prediction and compared to radial basis function networks. We make use of two different cost functions for Support Vectors: training with (i) an ε insensitive loss and (ii) Huber's robust loss function and discuss how to choose the regularization parameters in these models. Two applications are considered: data from (a) a noisy Mackey-Glass system (normal and uniform noise) and (b) the Santa Fe Time Series Competition (set D). In both cases, Support Vector Machines show an excellent performance. In case (b), the Support Vector approach improves the best known result on the benchmark by 29%.

1.1 Introduction

Support Vector Machines have become a subject of intensive study (see e.g. [3, 22]). They have been applied successfully to classification tasks as OCR [22, 17] and more recently also to regression [5, 23].
In this contribution[1] we use Support Vector Machines in the field of time series

1. This paper is an extended version of [12].

prediction and we find that they show an excellent performance.
In the following sections we will give a brief introduction to support vector regression (SVR) and we discuss the use of different types of loss functions. Furthermore, the basic principles of state space reconstruction are introduced in section 1.4. The experimental section considers a comparison of SVR and radial basis function (RBF) networks (introduced in section 1.3) with adaptive centers and variances. Both approaches show similarly excellent performance with an advantage for SVR in the high noise regime for Mackey Glass data. For benchmark data from the Santa Fe Competition (data set D) we get the best result achieved so far, which is 37% better than the winning approach during the competition [25] and still 29% better than our previous result [14]. A brief discussion concludes the chapter.

1.2 Support Vector Regression

In SVR the basic idea is to map the data $\mathbf{x}$ into a high-dimensional feature space $\mathcal{F}$ via a nonlinear mapping Φ, and to do linear regression in this space (cf. [3, 22])

$$f(\mathbf{x}) = (\omega \cdot \Phi(\mathbf{x})) + b \quad \text{with } \Phi : \mathbf{R}^n \to \mathcal{F},\ \omega \in \mathcal{F}, \tag{1.1}$$

where b is a threshold. Thus, *linear* regression in a *high* dimensional (feature) space corresponds to *nonlinear* regression in the *low* dimensional input space $\mathbf{R}^n$. Note that the dot product in Eq.(1.1) between $\omega \cdot \Phi(\mathbf{x})$ *would have* to be computed in this high dimensional space (which is usually intractable), if we were not able to use the kernel trick – described in the following – that finally leaves us with dot products that can be implicitly expressed in the low dimensional input space $\mathbf{R}^n$. Since Φ is fixed, we determine ω from the data by minimizing the sum of the empirical risk $R_{emp}[f]$ and a complexity term $\|\omega\|^2$, which enforces *flatness* in feature space

$$R_{reg}[f] = R_{emp}[f] + \lambda \|\omega\|^2 = \sum_{i=1}^{l} \mathcal{C}(f(\mathbf{x}_i) - y_i) + \lambda \|\omega\|^2, \tag{1.2}$$

where l denotes the sample size $(\mathbf{x}_1, \ldots, \mathbf{x}_l)$, $\mathcal{C}(.)$ is a cost function and λ is a regularization constant. For a large set of cost functions, Eq. (1.2) can be minimized by solving a quadratic programming problem, which is *uniquely* solvable [18, 19]. It can be shown that the vector ω can be written in terms of the data points

$$\omega = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \Phi(\mathbf{x}_i) \tag{1.3}$$

with α_i, α_i^* being the solution of the aforementioned quadratic programming problem [22]. α_i, α_i^* have an intuitive interpretation (see Fig. 1.1b) as forces pushing and pulling the estimate $f(\mathbf{x}_i)$ towards the measurements y_i (cf. [4]). Taking (1.3) and (1.1) into account, we are able to rewrite the whole problem in terms of dot products in the *low* dimensional input space (a concept introduced in [1])

$$f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)(\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})) + b = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)k(\mathbf{x}_i,\mathbf{x}) + b. \tag{1.4}$$

In Eq.(1.4) we introduced a kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = (\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j))$. It can be shown that any symmetric kernel function k satisfying Mercer's condition corresponds to a dot product in some feature space (see [3] for details). A common kernel is e.g. a RBF kernel

$$k(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)).$$

For extensive discussion about kernels see [19].

1.2.1 Vapnik's ε-insensitive Loss Function

For this special cost function the Lagrange multipliers α_i, α_i^* are often sparse, i.e. they result in non-zero values after the optimization (1.2) only if they are on or outside the boundary (see Fig. 1.1b), which means that they fulfill the Karush-Kuhn-Tucker conditions (for more details see [22, 18]). The ε–insensitive cost function is given by

$$\mathcal{C}(f(\mathbf{x})-y) = \begin{cases} |f(\mathbf{x})-y| - \varepsilon & \text{for } \ |f(\mathbf{x})-y| \geq \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{1.5}$$

(cf. Fig. 1.1a); the respective quadratic programming problem is defined as

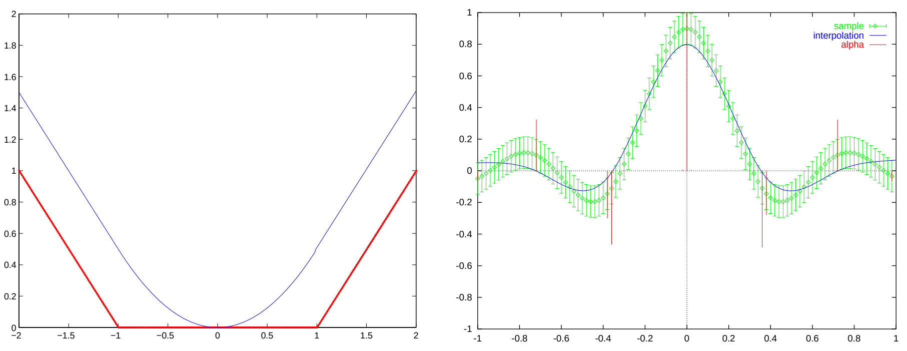
$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(\mathbf{x}_i,\mathbf{x}_j) - \sum_{i=1}^{l}\alpha_i^*(y_i - \varepsilon) - \alpha_i(y_i + \varepsilon) \\ &\text{subject to} \sum_{i=1}^{l}\alpha_i - \alpha_i^* = 0, \quad \alpha_i, \alpha_i^* \in [0, \frac{1}{\lambda}]. \end{aligned} \tag{1.6}$$

Note, that the less noisy the problem, the sparser are the α_i, α_i^* for Vapnik's ε-insensitive loss function. Note that the cost from Eq.(1.5) introduces a systematic bias, since we tend to underfit if ε is too large, e.g. in the extreme case of very large ε the resulting regression will be a constant.

1.2.2 Huber's Loss Function

Other cost functions like the robust loss function in the sense of [6] can also be utilized (cf. Fig. 1.1a) [18]. This cost function has the advantage of not introducing additional bias (like the ε-insensitive one does), at the expense, however, of sacrificing sparsity in the coefficients α_i, α_i^*.

$$\mathcal{C}(f(\mathbf{x})-y) = \begin{cases} \varepsilon|f(\mathbf{x})-y| - \frac{\varepsilon^2}{2} & \text{for } \ |f(\mathbf{x})-y| \geq \varepsilon \\ \frac{1}{2}(f(\mathbf{x})-y)^2 & \text{otherwise} \end{cases} \tag{1.7}$$



(a) (b)

Figure 1.1 (a) ε-insensitive and Huber's loss for $\varepsilon = 1$. (b) The shown regression for the ε-insensitive case (kernel: B-splines [18]) of the sinc function is the flattest within the ε tube around the data. α, α^* are drawn as positive and negative forces respectively. All points on the margin, where $f(\mathbf{x}_i) - y_i = \varepsilon \operatorname{sign}(\alpha_i - \alpha_i^*)$, are used for the computation of b.

The corresponding quadratic programming problem takes the following form

$$\begin{aligned} &\text{minimize } \frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)y_i + \frac{1}{2\lambda}(\alpha_i^2 + {\alpha_i^*}^2) \\ &\text{subject to } \sum_{i=1}^{l}\alpha_i - \alpha_i^* = 0, \quad \alpha_i, \alpha_i^* \in [0, \frac{\varepsilon}{\lambda}]. \end{aligned} \tag{1.8}$$

So basically all patterns become support vectors.

1.2.3 How to compute the threshold b?

Eqs. (1.6) and (1.8) show how to compute the variables α_k, α_k^*. For the proper choice of b, however, one has to make more direct use of the Karush-Kuhn-Tucker conditions that lead to the quadratic programming problems stated above. The key idea is to pick those values α_k, α_k^* for which the prediction error $\delta_k = f(\mathbf{x}_k) - y_k$ can be determined uniquely. In the ε-insensitive case this means picking points $\mathbf{x}_k$ on the margin, by requiring that one of the corresponding α_k or α_k^* be in the open interval $(0, \frac{1}{\lambda})$. In that case we know the exact value

$$\delta_k = \varepsilon \operatorname{sign}(\alpha_k - \alpha_k^*)$$

of the prediction error. Already one $\mathbf{x}_k$ would in principle be sufficient to compute b but for stability purposes it is recommended to take the average over *all points on the margin* with

$$b = \text{average}_k\{\delta_k + y_k - \sum_i(\alpha_i - \alpha_i^*)k(\mathbf{x}_i, \mathbf{x}_k)\}.$$

For the Huber case b is computed along the same lines with

$$\delta_k = \lambda(\alpha_k - \alpha_k^*)$$

for α_k or $\alpha_k^* \in [0, \frac{\varepsilon}{\lambda})$, i.e. for points where the quadratic part of the cost function is active.

Finally, we note that when we solve the quadratic programming problem with an optimizer which computes the double dual (e.g. [21]), we can directly recover the value of the primal variable b as the corresponding one of the double dual [19].

1.3 RBF networks with adaptive centers and widths

The RBF nets used in the experiments are an extension of the method of Moody and Darken [10], since centers and variances are also adapted (see also [2]). The output of the network is computed as a linear superposition

$$f(\mathbf{x}) = \sum_{k=1}^{K} w_k g_k(\mathbf{x}) \ , \tag{1.9}$$

where $w_k (k = 1, \ldots, K)$ denotes the weights of the output layer. The Gaussian basis functions g_k are defined as

$$g_k(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mu_k\|^2}{2\,\sigma_k^2}\right), \tag{1.10}$$

where μ_k and σ_k^2 denote means and variances, respectively. In a first step, the means μ_k are initialized with k-means clustering and the variances σ_k are determined as the distance between μ_k and the closest μ_i $(i \neq k)$. Then in the following steps we perform a gradient descent in the regularized error function (weight decay)

$$R_{reg} = \frac{1}{2}\sum_{i=1}^{l}(y_i - f(\mathbf{x}_i))^2 + \frac{\lambda}{2l}\sum_{k=1}^{K}(w_k)^2. \tag{1.11}$$

Note that in analogy to Eq.(1.2), we used $\lambda > 0$ to denote the regularization parameter. It is easy to derive the gradients $\partial R_{reg}/\partial \mu_k$ and $\partial R_{reg}/\partial \sigma_k$ (see Appendix). Numerically we minimize Eq.(1.11) by a conjugate gradient descent with line search, where we always compute the optimal output weights in every evaluation of the error function during the line search. The optimal output weights $\mathbf{w} = [w_1, \ldots, w_K]^\top$ in matrix notation can be computed in closed form by

$$\mathbf{w} = (G^T G + 2\frac{\lambda}{l}\mathbf{I})^{-1} G^T \mathbf{y}, \quad \text{where} \quad G_{ik} = g_k(\mathbf{x}_i) \tag{1.12}$$

and $\mathbf{y} = [y_1, \ldots, y_l]^\top$ denotes the output vector, and I an identity matrix. For $\lambda = 0$, this corresponds to the calculation of a pseudo-inverse of G.

So, we simultaneously adjust the output weights and the RBF centers and variances (see Appendix for pseudo-code of this algorithm). In this way, the network fine-tunes itself to the data after the initial clustering step, yet, of course, overfitting has to be avoided with careful tuning of the regularization parameter (cf. [2]).

1.4 How to predict?

Let $\{x(t)\}$, $t = 1, \ldots, T$, be a time series that was generated by a dynamical system. For convenience, consider $x(t)$ to be scalar, but note that the treatment of multi-scalar time series is straightforward. We assume that $\{x(t)\}$ is a projection of a dynamics operating in a high-dimensional state space. If the dynamics is deterministic, we can try to predict the time series by reconstructing the state space. A way to reconstruct the state space was introduced by Packard *et al.* [13] and mathematically analyzed by Takens [20]. A state vector is defined as

$$\mathbf{x}_t = (x(t),\ x(t-\tau),\ \ldots,\ x(t-(d-1)\tau)), \tag{1.13}$$

with time-delay τ and embedding dimension d. If the dynamics runs on an attractor of dimension D, a necessary condition for determining $\mathbf{x}_t$ is

$$d \geq D. \tag{1.14}$$

If the embedding dimension is big enough, such that $\mathbf{x}_t$ unambiguously describes the state of the system at time t, then there exists an equation for points on the attractor, which is of the form

$$x(t+p) = f^*(\mathbf{x}_t). \tag{1.15}$$

In this equation, f^* is a function that allows to predict future values of the time series $\{x(t)\}$ given past values, with p being the prediction horizon. Takens [20] showed that there is an upper bound

$$d \leq 2D + 1 \tag{1.16}$$

for the embedding dimension d, such that a continuous function f^* can be found within this bound. Regression techniques like SVR or RBF nets can therefore be used to estimate the prediction function on the basis of time-delay coordinates according to Eq. (1.13). For stationary dynamical systems the embedding parameters τ and d can be found e.g. by the method of Liebert, Pawelzik and Schuster [8].

1.5 Experiments

We fix the following experimental setup for our comparison: (a) RBF nets and (b) SVR are trained using a simple cross validation technique. We stop training the RBF networks at the minimum of the one step prediction error measured on a randomly chosen validation set. For SVR the parameters (λ, ε) are also determined at the minimum of the one step prediction error on the same validation set. Other methods, e.g. bootstrap can also be used to assess λ and ε. For SVR we distinguish between a training with Huber loss and ε-insensitive loss. Gaussian kernels with

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / (2\sigma^2)) \quad \text{and} \quad \sigma^2 = 0.75$$

are used in the SVR experiments. Note again that the RBF networks employed can adapt their variances σ_k to the data individually. Furthermore, in contrast to SVMs the means μ_k do not need to coincide with data points. As forecasting experiments we consider (i) a toy problem to understand and control the experimental set-up and (ii) a benchmark problem from the Santa Fe Competition (data set D).

1.5.1 Mackey Glass Equation

Our first application is a high-dimensional chaotic system generated by the Mackey-Glass delay differential equation

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t - t_\Delta)}{1 + x(t - t_\Delta)^{10}}, \tag{1.17}$$

with delay $t_\Delta = 17$. Eq. (1.17) was originally introduced as a model of blood cell regulation [9] and became quite common as an artificial forecasting benchmark. After integrating (1.17), we added noise to the time series. We obtained training (1000 patterns) and validation (the following 194 patterns) sets using an embedding dimension $d = 6$ and a step size $\tau = 6$. The test set (1000 patterns) is noiseless to measure the true prediction error. We conducted experiments for different signal to noise ratios (SNR) using Gaussian and uniform noise (Table 1.1).

We define the SNR in this experiment as the ratio between the variance of the noise and the variance of the Mackey Glass data.

noise	normal				uniform					
SNR	22.15%		44.3%		6.2%		12.4%		18.6%	
test error	1S	100S	1S	100S	1S	100S	1S	100S	1S	100S
ε-insensitive	0.017	0.218	0.040	0.335	0.006	0.028	0.012	0.070	0.017	0.142
Huber	0.017	0.209	0.040	0.339	0.008	0.041	0.014	0.065	0.019	0.226
RBF	0.018	0.109	0.044	0.266	0.009	0.062	0.014	0.083	0.028	0.282

Table 1.1 1S denotes the 1-step prediction error (RMS) on the test set. 100S is the 100-step iterated autonomous prediction. "SNR" is the ratio between the variance of the respective noise and the underlying time series. E.g. parameter choices for normal noise with SNR 22.15% is $\varepsilon = 0.01$ and $\lambda = 0.56$ for ε-insensitive loss and $\varepsilon = 0.1334$ and $\lambda = 0.0562$ for Huber loss. The respective RBF network uses 30 centers and $\lambda = 0.1$ choosen according to the validation set.

RBF networks and SVR achieve similar results for normal noise. It is to be expected that the method using the proper loss function (squared loss) wins for Gaussian noise, so we would actually expect the RBF nets to perform best followed by SVR trained with Huber loss, which is for large ε close to the squared loss and finally followed by SVR using an ε-insensitive loss. Table 1.1 confirms this intuition partially. For uniform noise, the whole scenario should be reversed, since ε-insensitive loss is the more appropriate noise model (cf. [6]). This is again confirmed in the experiment. The use of a validation set to assess the proper parameters λ and ε, however, is suboptimal and so the low resolution with which the (λ, ε) space is scanned is partly responsible for table entries that do not match the above intuition.

1.5.2 Data Set D from the Santa Fe Competition

Data set D from the Santa Fe competition is artificial data generated from a nine-dimensional periodically driven dissipative dynamical system with an asymmetrical four-well potential and a slight drift on the parameters [24]. The system has the property of operating in one well for some time and then switching to another well with a different dynamical behavior. Therefore, we first segment the time series into regimes of approximately stationary dynamics. This is accomplished by applying the *Annealed Competition of Experts* (ACE) method described in [14, 11] (no assumption about the number of stationary subsystems was made). Moreover, in order to reduce the effect of the continuous drift, only the last 2000 data points of the training set are used for segmentation. After applying the ACE algorithm, the data points are individually assigned to classes of different dynamical modes. We then select the particular class of data that includes the data points at the end of Data Set D as the training set for the RBF networks and the SVR[2]. This allows us to train the RBF networks and the SVR on quasi-stationary data and we avoid having to predict the average over all dynamical modes hidden in the full training set (see also [14] for further discussion). However, at the same time we are left with a rather small training set requiring careful regularization, since there are only 327 patterns in the extracted training set. As in the previous section we use a validation set (50 patterns of the extracted quasi-stationary data) to determine the stopping point and (λ, ε) respectively. The embedding parameters used, $d = 20$ and $\tau = 1$, are the same for all the methods compared in table 1.2.

Table 1.2 shows that our 25 step iterated prediction of the SVR is 37% better than the one achieved by Zhang and Hutchinson [25], who used a specialized network architecture. It is still 29% better than our previous result [14] that used the same ACE preprocessing as above and simple RBF nets (however at that time with non-adaptive centers and variances). As expected, the results are inferior, if we

2. Hereby we assume that the class of data that generated the last points in the training set is the one that is also responsible for the first couple of steps of the iterated continuation that we aim to predict.

train on the full, non-stationary training set without prior segmentation. However, ε-insensitive SVR is still better than the previous results on the full set.

experiment	ε-ins.	Huber	RBF	ZH [25]	PKM [14]
full set	0.0639	0.0653	0.0677	0.0665	–
segmented set	0.0418	0.0425	0.0569	–	0.0596

Table 1.2 Comparison (under competition conditions) of 25 step iterated predictions (root mean squared errors) on Data set D. "–" denotes: no prediction available. "Full set" means, that the full training set of set D was used, whereas "segmented set" means that a prior segmentation according to [11, 14] was done as preprocessing.

1.6 Discussion and Outlook

The chapter showed the performance of SVR in comparison to tuned RBF networks. For data from the Mackey-Glass equation we observed that also for SVR it pays to choose the *proper* loss function for the respective noise model (cf. [18, 19]). In both SVR cases training consisted in solving a – uniquely solvable – *quadratic optimization* problem, unlike the RBF network training, which requires non-linear optimization with the danger of getting stuck in local minima. Note that a stable prediction is a difficult problem since the noise level applied to the chaotic dynamics was rather high. For the data set D benchmark we obtained excellent results for SVR – 37% above the best result achieved during the Santa Fe competition [25]. Clearly, this remarkable difference is mostly due to the segmentation used as preprocessing step to get stationary data [11, 14], nevertheless still 29% improvement remain compared to a previous result using the same preprocessing step [14]. This underlines that we need to consider non-stationarities in the time series *before* the actual prediction, for which we can then use SVR or RBF nets (see also [11, 14, 15, 7] for discussion).

Our experiments show that SVR methods work particularly well if the data is *sparse* (i.e. we have little data in a high-dimensional space). This is due to their good inherent regularization properties.

Inspecting the RBF network approach more closely, we can see that a variety of individual variances σ_k appear as a result of the learning process. Clearly, in this sense RBF nets are the more flexible model, since multi-scaling information is extracted and taken into account. Of course the higher flexibility must be counter-balanced with a careful regularization. It now appears tempting to keep the principled regularization approach of SVR and to also allow for multiple variance SV kernels in Support Vector machine training. This way we would not to be obliged to determine a single scale to look at the data before learning[3].

Other things that remain are: determining the proper parameters λ and ε. This is still suboptimal and computationally intensive (if not clumsy). Both, some improved theoretical bounds and/or a simple heuristics to choose them would enhance the usability of SVR, since (λ, ε) are powerful means for *regularization* and *adaptation to the noise in the data*. Bootstrap methods or methods using a validation set are only a first step.

Acknowledgements

A.S. and G.R. are supported by DFG (# Ja 379/51). We thank C.Burges for valuable discussions. Moreover, we gratefully acknowledge travel grants from DAAD and NSF.

Appendix

Taking the derivative of Eq.(1.11) with respect to RBF means and variances we obtain

$$\frac{\partial R_{reg}}{\partial \mu_q} = \sum_{i=1}^{l} (f(\mathbf{x}_i) - y_i) \frac{\partial}{\partial \mu_q} f(\mathbf{x}_i), \text{ with } \quad \frac{\partial}{\partial \mu_q} f(\mathbf{x}_i) = w_q \frac{x_i - \mu_q}{(\sigma_q)^2} g_q(\mathbf{x}_i) \tag{1.18}$$

and

$$\frac{\partial R_{reg}}{\partial \sigma_q} = \sum_{i=1}^{l} (f(\mathbf{x}_i) - y_i) \frac{\partial}{\partial \sigma_q} f(\mathbf{x}_i), \text{ with } \quad \frac{\partial}{\partial \sigma_q} f(\mathbf{x}_i) = w_q \frac{\|\mu_q - \mathbf{x}_i\|^2}{(\sigma_q)^3} g_q(\mathbf{x}_i)\ . \tag{1.19}$$

These two derivatives are employed in the following algorithm (in pseudo-code):

```
Algorithm RBF-Net
    Input:
        Sequence of labeled training patterns $\mathbf{Z} = \langle(\mathbf{x}^1, y^1), \cdots, (\mathbf{x}^l, y^l)\rangle$
        Number of RBF centers $K$
        Regularization constant $\lambda$
        Number of iterations T
    Initialize:
        Run $K$-means clustering to find initial values for $\mu_k$ and determine $\sigma_k$
        (k=1,...,K) as the distance between $\mu_k$ and the closest $\mu_i$ $(i \neq k)$.
    Do for $t = 1 : T$,
        1. Compute optimal output weights $\mathbf{w} = \left(G^\top G + \frac{\lambda}{l} I\right)^{-1} G\mathbf{y}^\top$
```

3. The ability of processing multiscaling information could also be the reason to the often more stable 100 step prediction of RBF nets that was observed in the experiments.

2a. Compute gradients $\frac{\partial}{\partial \mu_k} R_{reg}$ and $\frac{\partial}{\partial \sigma_k} R_{reg}$ as in (1.19) and (1.18) with optimal $\mathbf{w}$ and form a gradient vector $\mathbf{v}$

2b. Estimate the conjugate direction $\overline{\mathbf{v}}$ with Fletcher-Reeves-Polak-Ribiere CG-Method [16]

3a. Perform a line search to find the minimizing step size δ in direction $\overline{\mathbf{v}}$; in each evaluation of R_{reg} compute the optimal output weights $\mathbf{w}$ as in line 1

3b. update μ_k and σ_k with $\overline{\mathbf{v}}$ and δ

Output: Optimized RBF net

References

1. M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821 – 837, 1964.

2. C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

3. B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

4. C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

5. H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, Cambridge, MA, 1997. MIT Press.

6. P. J. Huber. Robust statistics: a review. *Ann. Statist.*, 43:1041, 1972.

7. J. Kohlmorgen, K.-R. Müller, and K. Pawelzik. Analysis of drifting dynamics with neural network hidden markov models. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, Cambridge, MA, 1998. MIT Press. In press.

8. W. Liebert, K. Pawelzik, and H. G. Schuster. Optimal embeddings of chaotic attractors from topological considerations. *Europhys. Lett.*, 14:521 – 526, 1991.

9. M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

10. J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

11. K.-R. Müller, J. Kohlmorgen, and K. Pawelzik. Analysis of switching dynamics with competing neural networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E78–A(10):1306–1315, 1995.

12. K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks — ICANN'97*, pages

999 – 1004, Berlin, 1997. Springer Lecture Notes in Computer Science, Vol. 1327.

13. N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Phys. Rev. Lett.*, 45:712–716, 1980.

14. K. Pawelzik, J. Kohlmorgen, and K.-R. Müller. Annealed competition of experts for a segmentation and classification of switching dynamics. *Neural Computation*, 8(2):342–358, 1996.

15. K. Pawelzik, K.-R. Müller, and J. Kohlmorgen. Prediction of mixtures. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 127–133, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

16. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.).* Cambridge University Press, Cambridge, 1992. ISBN 0-521-43108-5.

17. B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining.* AAAI Press, Menlo Park, CA, 1995.

18. A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. Technical Report 1064, GMD, 1997.

19. A. Smola, B. Schölkopf, and K.-R. Müller. General cost functions for support vector regression. In T. Downs, M. Frean, and M. Gallagher, editors, *Proc. of the Ninth Australian Conf. on Neural Networks*, pages 79 – 83, Brisbane, Australia, 1998. University of Queensland.

20. F. Takens. Detecting strange attractors in fluid turbulence. In D. Rand and L.S. Young, editors, *Dynamical Systems and Turbulence*, pages 366–381. Springer-Verlag, Berlin, 1981.

21. R. J. Vanderbei. LOQO user's manual – version 3.10. Technical Report SOR-97-08, Princeton University, Statistics and Operations Research, 1997. Code available at http://www.princeton.edu/~rvdb/.

22. V. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, New York, 1995.

23. V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 281–287, Cambridge, MA, 1997. MIT Press.

24. A. S. Weigend and N. A. Gershenfeld (Eds.). *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley, 1994. Santa Fe Institute Studies in the Sciences of Complexity.

25. X. Zhang and J. Hutchinson. Simple architectures on fast machines: practical issues in nonlinear time series prediction. In A. S. Weigend and N. A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past.* Santa Fe Institute, Addison-Wesley, 1994.