Chapter 1

Multi-Layer Support Vector Machines

Marco A. Wiering
Institute of Artificial Intelligence and Cognitive Engineering, University of Groningen

Lambert R.B. Schomaker
Institute of Artificial Intelligence and Cognitive Engineering, University of Groningen

1.1 Introduction

Support vector machines (SVMs) [24, 8, 20, 22] and other learning algorithms based on kernels have been shown to obtain very good results on many different classification and regression datasets. SVMs have the advantage of generalizing very well, but the standard SVM is limited in several ways. First, the SVM uses a single layer of support vector coefficients and is therefore a shallow model. Deep architectures [17, 14, 13, 4, 25, 6] have been shown to be very promising alternatives to these shallow models. Second, the results of the SVM rely heavily on the selected kernel function, but most kernel functions have limited flexibility in the sense they they are not trainable on a dataset. Therefore, it is a natural step to go from the standard single-layer SVM to

the multi-layer SVM (ML-SVM). Just like the invention of the backpropagation algorithm [26, 19] allowed to construct multi-layer perceptrons from perceptrons, this chapter describes techniques for constructing and training multi-layer SVMs consisting only of SVMs.

There is a lot of related work in multiple kernel learning (MKL) [16, 3, 21, 18, 31, 10]. In these approaches, some combination functions of a set of fixed kernels are adapted to the dataset. As has been shown by a number of experiments, linear combinations of base kernels do not often help to get significantly better performance levels. Therefore, in [7] the authors describe the use of non-linear (polynomial) combinations of kernels and their results show that this technique is more effective. An even more recent trend in MKL is the use of multi-layer MKL. In [9], a general framework for two-layer kernel machines is described, but unlike the current study no experimental results were reported in which both layers used non-linear kernels. In [32], multi-layer MKL is described where mixture coefficients of different kernels are stored in an exponential function kernel. These coefficients in the second layer of the two-layer MKL algorithm are trained using a min-max objective function. In [5] a new type of kernel is described, which is useful for mimicking a deep learning architecture. The neural support vector machine (NSVM) [28] is also related to the multi-layer SVM. The NSVM is a novel algorithm that uses neural networks to extract features which are given to a support vector machine for giving the final output of the architecture. Finally, the current chapter extends the ideas in [27] by describing a classification and autoencoder method using multi-layer support vector machines.

Contributions. We describe a simple method for constructing and training multi-layer SVMs. The hidden-layer SVMs in the architecture learn to extract relevant features or latent variables from the inputs and the output-layer SVMs learn to approximate the target function using the extracted features from the hidden-layer SVMs. We can easily make the association with multi-layer perceptrons (MLPs) by letting a complete SVM replace each individual neuron. However, in contrast to the MLP, the ML-SVM algorithm is trained using a min-max objective function: the hidden-layer SVMs are trained to minimize the dual-objective function of the output-layer SVMs and the output-layer SVMs are trained to maximize their dual-objective functions. This min-max optimization problem is a result of going from the primal objective to the dual objective. Therefore, the learning dynamics of the ML-SVM are entirely different compared to the MLP in which all model parameters are trained to minimize the same error function. When compared to other multi-layer MKL approaches, the ML-SVM does not make use of any combination weights, but trains support vector coefficients and the biases of all SVMs in the architecture. Our experimental results show that the ML-SVM significantly outperforms state-of-the-art machine learning techniques on regression, classification and dimensionality reduction problems.

We have organized the rest of this chapter as follows. Section 1.2 describes the ML-SVM algorithm for regression problems. In Section 1.3, the ML-SVM

algorithm is introduced for classification problems. In Section 1.4, the autoencoding ML-SVM is described. In Section 1.5, experimental results on 10 regression datasets, 8 classification datasets, and a dimensionality reduction problem are presented. Finally, Section 1.6 discusses the findings and describes future work.

1.2 Multi-layer Support Vector Machines for Regression Problems

We will first describe the multi-layer SVM for regression problems. We use a regression dataset: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$, where $\mathbf{x}_i$ are input vectors and y_i are the scalar target outputs. The architecture of a two-layer SVM is shown in Figure 1.1.

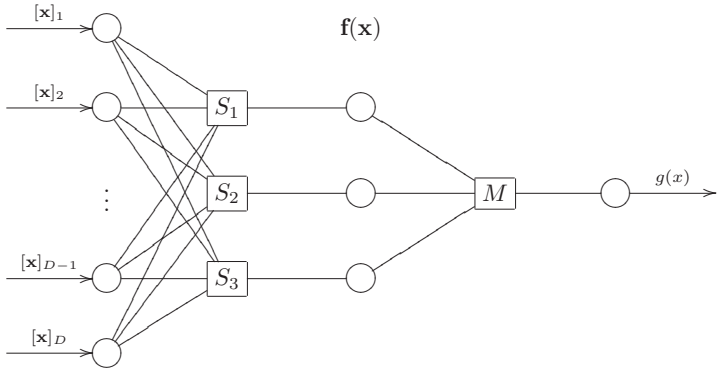

FIGURE 1.1: Architecture of a two-layer SVM. In this example, the hidden layer consists of three SVMs S_a.

The two-layer architecture contains an input layer of D inputs. Then, there are a total of d SVMs S_a, each one learning to extract one latent variable $\mathbf{f}(\mathbf{x}|\theta)_a$ from an input pattern $\mathbf{x}$. Here θ denotes the trainable parameters in the hidden-layer SVMs (which are the support vector coefficients and the biases). Finally, there is the main support vector machine M that learns to approximate the target function using the extracted feature vector as input. For computing the hidden-layer representation $\mathbf{f}(\mathbf{x}|\theta)$ of input vector $\mathbf{x}$, we use:

$$\mathbf{f}(\mathbf{x}|\theta)_a = \sum_{i=1}^{\ell} (\alpha_i^*(a) - \alpha_i(a)) K_1(\mathbf{x}_i, \mathbf{x}) + b_a, \tag{1.1}$$

which is iteratively used by each SVM S_a to compute the element $\mathbf{f}(\mathbf{x}|\theta)_a$.

In this equation, $\alpha_i^*(a)$ and $\alpha_i(a)$ are support vector coefficients for SVM S_a, b_a is its bias, and $K_1(\cdot,\cdot)$ is a kernel function for the hidden-layer SVMs. For computing the output of the whole ML-SVM, the main SVM maps the extracted hidden-layer representation to an output:

$$g(\mathbf{f}(\mathbf{x}|\theta)) = \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i)K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}|\theta)) + b. \tag{1.2}$$

Here, $K_2(\cdot,\cdot)$ is the kernel function in the output layer of the multi-layer SVM. The primal objective for a linear regression SVM M can be written as:

$$\min_{\mathbf{w},\theta,\boldsymbol{\xi},\boldsymbol{\xi}^*,b} J(\mathbf{w},\theta,\boldsymbol{\xi},\boldsymbol{\xi}^*,b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*) \tag{1.3}$$

subject to constraints:

$$y_i - \mathbf{w}\cdot\mathbf{f}(\mathbf{x}_i|\theta) - b \leq \varepsilon + \xi_i \;\;;\;\; \mathbf{w}\cdot\mathbf{f}(\mathbf{x}_i|\theta) + b - y_i \leq \varepsilon + \xi_i^* \tag{1.4}$$

and $\xi_i, \xi_i^* \geq 0$. Here C is a metaparameter, ϵ is an error tolerance value used in the Hinge (ϵ-insensitive) loss function, and ξ_i and ξ_i^* are slack variables that tolerate errors larger than ϵ, but which should be minimized. The dual-objective function for the regression problem for the main SVM M is:

$$\begin{aligned}\min_{\theta}\max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} J(\theta,\boldsymbol{\alpha},\boldsymbol{\alpha}^*) = -\varepsilon\sum_{i=1}^{\ell}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i)y_i \\ -\frac{1}{2}\sum_{i,j=1}^{\ell}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta))\end{aligned} \tag{1.5}$$

subject to: $0 \leq \alpha_i^*, \alpha_i \leq C$ and $\sum_{i=1}^{\ell}(\alpha_i - \alpha_i^*) = 0$. The second constraint in generally known as the bias constraint.

Our learning algorithm adjusts the SVM coefficients of all SVMs through the min-max formulation of the dual-objective function $J(\cdot)$ of the main SVM. Note that the min-max optimization problem is a result of going from the primal objective to the dual objective. In the primal objective, it is a joint minimization with respect to θ and the α coefficients. However, by dualizing the primal objective of the main SVM, it is turned into a min-max problem.

We have implemented a simple gradient ascent algorithm to train the SVMs. The method adapts all SVM coefficients α_i^* and α_i toward a (local) maximum of $J(\cdot)$, where λ is the learning rate. The resulting gradient ascent learning rule for α_i is:

$$\alpha_i \leftarrow \alpha_i + \lambda(-\epsilon - y_i + \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j)K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta))) \tag{1.6}$$

The resulting gradient ascent learning rule for α_i^* is:

$$\alpha_i^* \leftarrow \alpha_i^* + \lambda(-\epsilon + y_i - \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j)K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta))) \tag{1.7}$$

The support vector coefficients are set to 0 if they become less than 0, and set to C if they become larger than C. We also added a penalty term to respect the bias constraint, so actually the gradient ascent algorithm trains the support vector coefficients to maximize the objective $J'(\cdot) = J(\cdot) - c_1 \cdot (\sum_i(\alpha_i - \alpha_i^*))^2$, with c_1 some metaparameter. Although this simple strategy works well, this ad-hoc optimization strategy could also be replaced by a gradient projection method for which convergence properties are better understood.

In the experiments we will make use of radial basis function (RBF) kernels in both layers of a two-layer SVM. Preliminary results with other often used kernels were somewhat worse. For the main SVM and hidden-layer SVMs the RBF kernel is defined respectively by:

$$K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}|\theta)) = \exp(-\sum_{a=1}^{d}\frac{(\mathbf{f}(\mathbf{x}_i|\theta)_a - \mathbf{f}(\mathbf{x}|\theta)_a)^2}{\sigma_2}) \tag{1.8}$$

$$K_1(\mathbf{x}_i, \mathbf{x}) = \exp(-\sum_{a=1}^{D}\frac{(\mathbf{x}_i^a - \mathbf{x}^a)^2}{\sigma_1}) \tag{1.9}$$

where σ_2 and σ_1 determine the widths of the RBF kernels in the output and hidden layers. The ML-SVM constructs a new dataset for each hidden-layer SVM S_a with a backpropagation-like technique for making examples: $(\mathbf{x}_i, \mathbf{f}(\mathbf{x}_i|\theta)_a - \mu \cdot \partial J(\cdot)/\partial \mathbf{f}(\mathbf{x}_i|\theta)_a)$, where μ is some metaparameter, and $\partial J(\cdot)/\partial \mathbf{f}(\mathbf{x}_i|\theta)_a$ for the RBF kernel is given by:

$$\frac{\partial J(\cdot)}{\partial \mathbf{f}(\mathbf{x}_i|\theta)_a} = (\alpha_i^* - \alpha_i)\sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j)\frac{\mathbf{f}(\mathbf{x}_i|\theta)_a - \mathbf{f}(\mathbf{x}_j|\theta)_a}{\sigma_2} \cdot K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta)). \tag{1.10}$$

We constrain the target values for hidden-layer features between -1 and 1, so if some target output is larger than 1 for a feature we simply set the target value to 1. To allow the hidden-layer SVMs to extract different features, symmetry breaking is necessary. For this, we could randomly initialize the trainable parameters in each hidden-layer SVM. However, we discovered that a better way to initialize the hidden-layer SVMs is to let them train on different perturbed versions of the target outputs. Therefore we initially construct a dataset $(\mathbf{x}_i, y_i + \gamma_i^a)$, with γ_i^a some random value $\in [-\gamma, \gamma]$ for the hidden-layer SVM S_a, where γ is another metaparameter. In this way, the ML-SVM resembles a stacking ensemble approach [30], but due to the further training with the min-max optimization process, these approaches are still very different. The complete algorithm is given in Algorithm 1.

In the algorithm alternated training of the main SVM and hidden-layer

Algorithm 1 The multi-layer SVM algorithm

```
Initialize output SVM
Initialize hidden-layer SVMs
Compute kernel matrix for hidden-layer SVMs
Train hidden-layer SVMs on perturbed dataset
repeat
  Compute kernel matrix for output-layer SVM
  Train output-layer SVM
  Use backpropagation to create training sets for hidden-layer SVMs
  Train hidden-layer SVMs
until maximum number of epochs is reached
```

SVMs is executed a number of epochs. An epoch here is defined as training the main SVM and the hidden-layer SVM a single time on their respective datasets with our gradient ascent technique that uses a small learning rate and a fixed number of iterations. The bias values of all SVMs are set by averaging over the errors on all examples.

Theoretical insight. Due to the min-max optimization problem and the two layers with non-linear kernel functions, the ML-SVM loses the property that the optimization problem is convex. However, similar to multiple-kernel learning, training the output-layer SVM given the outputs of the hidden layer remains a convex learning problem. Furthermore, the datasets generated with the backpropagation technique explained above, are like normal training datasets. Since training an SVM on a dataset is a convex learning problem, these newly created datasets are also convex learning problems for the hidden-layer SVMs. By using the pre-training of hidden-layer SVMs on perturbed versions of the target outputs, the learning problem of the output-layer SVM becomes much simpler. In fact, this resembles a stacking ensemble approach [30], but unlike any other ensemble approach, the ML-SVM is further optimized using the min-max optimization process. This is interesting, because it is different from other approaches in which the same error function is minimized by all model parameters. Still, it could also be seen as a disadvantage, because min-max learning is not yet well understood in the machine learning community.

1.3 Multi-layer Support Vector Machines for Classification Problems

In the multi-layer SVM classifier, the architecture contains multiple support vector classifiers in the output layer. To deal with multiple classes, we

use a binary one vs. all classifier M_c for each class c. We do this even with 2 classes for convenience. We use a classification dataset for each classifier M_c: $\{(\mathbf{x}_1, y_1^c), \ldots, (\mathbf{x}_\ell, y_\ell^c)\}$, where $\mathbf{x}_i$ are input vectors and $y_i^c \in \{-1, 1\}$ are the target outputs that denote if the example $\mathbf{x}_i$ belongs to class c or not. All classifiers M_c share the same hidden-layer of regression SVMs. M_c determines its output on an example $\mathbf{x}$ as follows:

$$g_c(\mathbf{f}(\mathbf{x}|\theta)) = \sum_{i=1}^{\ell} y_i^c \alpha_i^c K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}|\theta)) + b_c. \tag{1.11}$$

Here $\mathbf{f}(\mathbf{x}_i|\theta)$ is computed with the hidden-layer SVMs as before. The values α_i^c are the support vector coefficients for classifier M_c. The value b_c is its bias. After computing all output values for all classifiers, the class with the highest output is assumed to be the correct class label (with ties being broken randomly). The primal objective for a linear support vector classifier M_c can be written as:

$$\min_{\mathbf{w}^c, \boldsymbol{\xi}, b, \theta} J_c(\mathbf{w}^c, \boldsymbol{\xi}, b, \theta) = \frac{1}{2}||\mathbf{w}^c||^2 + C \sum_{i=1}^{\ell} \xi_i \tag{1.12}$$

subject to: $y_i^c(\mathbf{w}^c \cdot \mathbf{f}(\mathbf{x}_i|\theta) + b_c) \geq 1 - \xi_i$, and $\xi_i \geq 0$. Here C is a metaparameter and ξ_i are slack variables that tolerate errors, but which should be minimized. The dual-objective function for the classification problem for classifier M_c is:

$$\min_{\theta} \max_{\boldsymbol{\alpha}^c} J_c(\theta, \boldsymbol{\alpha}^c) = \sum_{i=1}^{\ell} \alpha_i^c - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i^c \alpha_j^c y_i^c y_j^c K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta)) \tag{1.13}$$

subject to: $0 \leq \alpha_i^c \leq C$, and $\sum_{i=1}^{\ell} \alpha_i^c y_i^c = 0$. Whenever the ML-SVM is presented a training pattern $\mathbf{x}_i$, each classifier in the multi-layer SVM uses gradient ascent to adapt its α_i^c values towards a local maximum of $J_c(\cdot)$ by:

$$\alpha_i^c \leftarrow \alpha_i^c + \lambda(1 - \sum_{j=1}^{\ell} \alpha_j^c y_j^c y_i^c K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta))) \tag{1.14}$$

where λ is a metaparameter controlling the learning rate of the values α_i^c. As before the support vector coefficients are kept between 0 and C. Because we use a gradient ascent update rule, we use an additional penalty term $c_1(\sum_{j=1}^{\ell} \alpha_j^c y_j^c)^2$ with metaparameter c_1 so that the bias constraint is respected.

As in the regression ML-SVM, the classification ML-SVM constructs a new dataset for each hidden-layer SVM S_a with a backpropagation-like technique for making examples. However, in this case the aim of the hidden-layer SVMs is to minimize the sum of objectives $\sum_c J_c(\cdot)$. Therefore, the algorithm constructs a new dataset using: $(\mathbf{x}_i, \mathbf{f}(\mathbf{x}_i|\theta)_a - \mu \sum_c \partial J_c(\cdot)/\partial \mathbf{f}(\mathbf{x}_i|\theta)_a)$, where

μ is some metaparameter, and $\partial J_c(\cdot)/\partial \mathbf{f}(\mathbf{x}_i|\theta)_a$ for the RBF kernel is:

$$\frac{\partial J_c(\cdot)}{\partial \mathbf{f}(\mathbf{x}_i|\theta)_a} = \alpha_i^c y_i^c \sum_{j=1}^{\ell} \alpha_j^c y_j^c \frac{\mathbf{f}(\mathbf{x}_i|\theta)_a - \mathbf{f}(\mathbf{x}_j|\theta)_a}{\sigma_2} \cdot K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}_j|\theta)) \quad (1.15)$$

The target outputs for hidden-layer features are again kept between -1 and 1. The datasets for hidden-layer SVMs are made so that the sum of the dual-objective functions of the output SVMs is minimized. All SVMs are trained with the gradient ascent algorithm on their constructed datasets. Note that the hidden-layer SVMs are still regression SVMs, since they need to output continuous values. For the ML-SVM classifier, we use a different initialization procedure for the hidden-layer SVMs. Suppose there are d hidden-layer SVMs and a total of c_{tot} classes. The first hidden-layer SVM is first pre-trained on inputs and perturbed target outputs for class 0, the second on the perturbed target outputs for class 1, and the k^{th} hidden-layer SVM is pre-trained on the perturbed target outputs for class k modulo c_{tot}. The bias values are computed in a similar way as in the regression ML-SVM, but for the output SVMs only examples with non-bound support vector coefficients (which are not 0 or C) are used.

1.4 Multi-layer Support Vector Machines for Dimensionality Reduction

The architecture of the ML-SVM autoencoder differs from the single-output regression ML-SVM in two respects: (1) The output layer consists of D nodes, the same number of nodes the input layer has. (2) It utilizes a total of D support vector regression machines M_c, which each take the entire hidden-layer output as input and determine the value of one of the outputs.

The forward propagation of a pattern $\mathbf{x}$ of dimension D determines the representation in the hidden layer. The hidden layer is then used as input for each support vector machine M_c that determines its output with:

$$g_c(\mathbf{f}(\mathbf{x}|\theta)) = \sum_{i=1}^{\ell} (\alpha_i^{c*} - \alpha_i^c) K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}|\theta)) + b_c. \quad (1.16)$$

Again we make use of RBF kernels in both layers. The aim of the ML-SVM autoencoder is to reconstruct the inputs in the output layer using a bottleneck of hidden-layer SVMs, where the number of hidden-layer SVMs is in general much smaller than the number of inputs. The ML-SVM autoencoder tries to find the SVM coefficients θ such that the hidden-layer representation $\mathbf{f}(\cdot)$ is most useful for accurately reconstructing the inputs, and thereby codes the features most relevant to the input distribution. This is similar to neural

network autoencoders [23, 12]. Currently popular deep architectures [14, 4, 25] stack these autoencoders one by one, which is also possible for the ML-SVM autoencoder.

The dual objective of each support vector machine M_c is:

$$\min_{\theta} \max_{\alpha^{c*},\alpha^{c}} J_c(\theta, \alpha_i^{c(*)}) = -\varepsilon \sum_{i=1}^{\ell}(\alpha_i^{c*} + \alpha_i^{c}) + \sum_{i=1}^{\ell}(\alpha_i^{c*} - \alpha_i^{c})y_i^{c}$$
$$-\frac{1}{2}\sum_{i,j=1}^{\ell}(\alpha_i^{c*} - \alpha_i^{c})(\alpha_j^{c*} - \alpha_j^{c})K_2(\mathbf{f}(\mathbf{x}_i|\theta), \mathbf{f}(\mathbf{x}|\theta)) \qquad (1.17)$$

subject to: $0 \leq \alpha_i^{c}, \alpha_i^{c*} \leq C$, and $\sum_{i=1}^{\ell}(\alpha_i^{c*} - \alpha_i^{c}) = 0$. The minimization of this equation with respect to θ is a bit different from the single-node ML-SVM. Since all SVMs share the same hidden layer, we cannot just minimize $J(\cdot)$ for every SVM separately. It is actually this shared nature of the hidden layer which enables the ML-SVM to perform autoencoding. Therefore the algorithm creates new datasets for the hidden-layer SVMs by backpropagating the sum of the derivatives of all dual objectives $J_c(\cdot)$. Thus, the ML-SVM autoencoder uses: $(\mathbf{x}, \mathbf{f}(\mathbf{x}|\theta)_a - \mu \sum_{c=1}^{D} \frac{\partial J_c(\cdot)}{\partial \mathbf{f}(\mathbf{x}|\theta)_a})$ to create new datasets for the hidden-layer SVMs.

1.5 Experiments and Results

We first performed experiments on regression and classification problems to compare the multi-layer SVM (we used 2 layers) to the standard SVM and also to a multi-layer perceptron. Furthermore, we performed experiments with an image dataset where it was the goal to obtain the smallest reconstruction error with a limited number of hidden components.

1.5.1 Experiments on Regression Problems

We experimented with 10 regression datasets to compare the multi-layer SVM to an SVM, both using RBF kernels. We note that both methods are trained with the simple gradient ascent learning rule, adapted to also consider the penalty for obeying the bias constraint, although standard algorithms for the SVM could also be used. The first 8 datasets are described in [11] and the other 2 datasets are taken from the UCI repository [1]. The number of examples per dataset ranges from 43 to 1049, and the number of input features is between 2 and 13. The datasets are split into 90% training data and 10% test data. For optimizing the metaparameters we have used particle swarm optimization (PSO) [15]. There are in total around 15 metaparameters for the

ML-SVM such as the learning rates for the two layers, the values for the error tolerance ϵ, the values for C, the number of gradient ascent iterations in the gradient ascent algorithm, the values for respecting the bias constraint c_1, the RBF kernel widths σ_1 and σ_2, the number of hidden-layer SVMs, the value for the perturbation value γ used for pre-training the hidden-layer SVMs, and the maximal number of epochs. PSO saved us from laborious manual tuning of these metaparameters. We made an effective implementation of PSO that also makes use of the UCB bandit algorithm [2] to eliminate unpromising sets of metaparameters. We always performed 100,000 single training-runs to obtain the best metaparameters that took at most 2 days on a 32-CPU machine on the largest dataset. For the gradient ascent SVM algorithm we also used 100,000 evaluations with PSO to find the best metaparameters, although our implementation of the gradient ascent SVM has 7 metaparameters, which makes it easier to find the best ones. Finally, we used 1000 or 4000 new cross validation runs with the best found metaparameters to compute the mean squared error and its standard error of the different methods for each dataset.

TABLE 1.1: The mean squared errors and standard errors of the gradient ascent SVM, the two-layer SVM, and results published in [11] for an MLP on 10 regression datasets. N/A means not available.

Dataset	Gradient ascent SVM	ML-SVM	MLP
Baseball	0.02413 ± 0.00011	$\mathbf{0.02294} \pm 0.00010$	0.02825
Boston Housing	0.006838 ± 0.000095	$\mathbf{0.006381} \pm 0.000091$	0.007809
Concrete Strength	0.00706 ± 0.00007	$\mathbf{0.00621} \pm 0.00005$	0.00837
Diabetes	0.02719 ± 0.00026	$\mathbf{0.02327} \pm 0.00022$	0.04008
Electrical Length	0.006382 ± 0.000066	0.006411 ± 0.000070	0.006417
Machine-CPU	0.00805 ± 0.00018	$\mathbf{0.00638} \pm 0.00012$	0.00800
Mortgage	0.000080 ± 0.000001	0.000080 ± 0.000001	0.000144
Stock	0.000862 ± 0.000006	$\mathbf{0.000757} \pm 0.000005$	0.002406
Auto-MPG	6.852 ± 0.091	6.715 ± 0.092	N/A
Housing	$\mathbf{8.71 \pm 0.14}$	9.30 ± 0.15	N/A

In Table 1.1 we show the results of the standard SVM trained with gradient ascent and the results of the two-layer SVM. The table also shows the results for a multi-layer perceptron (MLP) reported in [11] on the first 8 datasets. The MLP used sigmoidal hidden units and was trained with backpropagation. We note that Graczyk et al. [11] only performed 10-fold cross validation and did not report any standard errors.

The results show that the two-layer SVM significantly outperforms the other methods on 6 datasets ($p < 0.001$) and only performs worse than the standard SVM on the Housing dataset from the UCI repository. The average gain over all datasets is 6.5% error reduction. The standard errors are very small because we performed 1000 or 4000 times cross validation. We did this because we observed that with less cross validation runs the results were less

trustworthy due to their stochastic nature caused by the randomized splits into different test sets. We also note that the results of the gradient ascent SVM are a bit better than the results obtained with an SVM in [11]. We think that the PSO method is more capable in optimizing the metaparameters than the grid search employed in [11]. Finally, we want to remark that the results of the MLP are worse than those of the two other approaches.

1.5.2 Experiments on Classification Problems

We compare the multi-layer classification SVM to the standard SVM and a multi-layer perceptron trained with backpropagation with one hidden layer with sigmoid activation functions. Early stopping was implemented in the MLP by optimizing the number of training epochs. For the comparison we use 8 datasets from the UCI repository. In these experiments we have used SVMLight as standard SVM and optimized the metaparameters (σ and C) with grid search (also with around 100,000 evaluations). We also optimized the metaparameters (number of hidden units, learning rate, number of epochs) for the multi-layer perceptron. The metaparameters for the multi-layer SVM are again optimized with PSO.

TABLE 1.2: The accuracies and standard errors on the 8 UCI classification datasets. The results are shown of an MLP, a support vector machine (SVM), and the two-layer SVM.

Dataset	MLP	SVM	ML-SVM
Hepatitis	84.3 ± 0.3	81.9 ± 0.3	**85.1** ± 0.1
Breast Cancer W.	97.0 ± 0.1	96.9 ± 0.1	97.0 ± 0.1
Ionosphere	91.1 ± 0.1	94.0 ± 0.1	**95.5** ± 0.1
Ecoli	87.6 ± 0.2	87.0 ± 0.2	87.3 ± 0.2
Glass	64.5 ± 0.4	70.1 ± 0.3	**74.0** ± 0.3
Pima Indians	77.4 ± 0.1	77.1 ± 0.1	77.2 ± 0.2
Votes	96.6 ± 0.1	96.5 ± 0.1	96.8 ± 0.1
Iris	97.8 ± 0.1	96.5 ± 0.2	**98.4** ± 0.1
Average	87.0	87.5	88.9

We report the results on the 8 datasets with average accuracies and standard errors. We use 90% of the data for training data and 10% for test data. We have performed 1000 new random cross validation experiments per method with the best found metaparameters (and 4000 times for Iris and Hepatitis, since these are smaller datasets). The results are shown in Table 1.2. The multi-layer SVM significantly ($p < 0.05$) outperforms the other methods on 4 out of 8 classification datasets. On the other problems the multi-layer SVM performs equally well as the other methods. We also performed experiments with the gradient ascent SVM on these datasets, but its results are very similar to those obtained with SVMLight, so we do not show them here. On some

datasets such as Breast Cancer Wisconsin and Votes, all methods perform equally well. On some other datasets, the multi-layer SVM reduces the error of the SVM a lot. For example, the error on Iris is 1.6% for the multi-layer SVM compared to 3.5% for the standard SVM. The MLP obtained 2.2% error on this dataset. Finally, we also optimized and tested a stacking ensemble SVM method, which uses an SVM to directly map the outputs of the pretrained hidden-layer SVMs to the desired output without further min-max optimization. This approach obtained 2.3% error on Iris and is therefore significantly outperformed by the multi-layer SVM.

1.5.3 Experiments on Dimensionality Reduction Problems

The used dataset in the dimensionality reduction experiment contains a total of 1300 instances of gray-scaled images of the left eyes manually cropped from pictures in the 'Labeled faces in the wild' dataset. The images, shown in figure 1.2, are normalized and have a resolution of 20 by 20 pixels, and thus have 400 values per image. The aim of this experiment is to see how well the autoencoder ML-SVM performs compared to some state-of-the-art methods. The goal of the used dimensionality reduction algorithms is to accurately encode the input data using fewer dimensions than the number of inputs. A well known, but suboptimal technique for doing this is the use of principal component analysis.

FIGURE 1.2: Examples of some of the cropped gray-scaled images of left eyes that are used in the dimensionality reduction experiment.

We compared the ML-SVM to principal component analysis (PCA) and a neural network autoencoding method. We used a state-of-the-art neural network autoencoding method, named a denoising autoencoder [25], for which we optimized the metaparameters. The autoencoders were trained using stochastic gradient descent with a decreasing learning rate. In each epoch, all samples in the training set were presented to the network in a random order. To improve generalization performance of the standard neural network autoencoder [23], in the denoising autoencoder each input sample is augmented with Gaussian noise, while the target stayed unaltered. We also added l_1 regularization on the hidden layer of the network to increase sparsity. These additions improved the performance of this non-linear autoencoder.

We also compared the ML-SVM to principal component analysis using a multi-variate Partial-Least Squares (PLS) regression model with standardized inputs and outputs [29]. It can easily be shown that the standard PLS algo-

rithm in autoencoder mode is actually equivalent with a principal component projection (with symmetric weights in the layer from the latent variable bottleneck layer to the output layer). The attractiveness of applying the PLS autoencoder in this case is the elegant and efficient implementation of the standard PLS algorithm to compute the principal components.

For these experiments, random cross validation is used to divide the data in a training set containing two thirds (867 examples) of the dataset, and a test set containing one third. The methods are compared by measuring the reconstruction error for different numbers of (non-linear) principal components: we used 10, 20, and 50 dimensions to encode the eye images. The root mean square error of 10 runs and standard errors are computed for the comparison.

TABLE 1.3: The RMSE and standard errors for different numbers of principal components for principal component analysis, a denoising autoencoder (DAE), and a multi-layer support vector machine (ML-SVM)

#dim	PCA	DAE	ML-SVM
10	0.1242 ± 0.0004	0.1211 ± 0.0002	$\mathbf{0.1202} \pm 0.0003$
20	0.0903 ± 0.0003	0.0890 ± 0.0002	$\mathbf{0.0875} \pm 0.0003$
50	0.0519 ± 0.0002	0.0537 ± 0.0001	$\mathbf{0.0513} \pm 0.0002$

The results of these experiments can be found in Table 1.3. These results show a significantly better ($p<0.05$) performance for autoencoding with the use of a multi-layer support vector machine compared to the denoising autoencoder and PCA. As known from literature, the difference to PCA decreases when more principal components are used.

1.5.4 Experimental Analysis of the Multi-layer SVM

We also studied why the multi-layer SVM outperforms the SVM in many cases. For this we will examine the Iris dataset again, but in more detail. For this dataset the multi-layer SVM and the MLP perform much better than the standard SVM with an RBF kernel (see Table 1.2). We performed the experiments again with the previous best found metaparameters, but set the C-values to 3.0 for all methods so that the dual-objectives of different methods can be easily compared. This did not significantly change the performances. Furthermore, we set the number of epochs to 14.

Figure 1.3 shows the evolution of the training and test errors for three methods. The reported errors are averaged over 1000 simulations. We compare the standard SVM trained with gradient ascent, the multi-layer SVM, and a multi-layer SVM in which the hidden-layer SVMs were not pre-trained on perturbed class labels, but completely randomly initialized. In the case of the standard SVM, the epochs refer to the number of repetitions of the gradient ascent algorithm. For the multi-layer SVMs, the epoch counter is increased after only training the output SVMs or after only training the hidden-layer

SVMs. The training times on a single training set are less than one second for the Iris dataset for all methods. In epoch 0, the output layer SVMs were initialized with constant positive support vector coefficients and by PSO optimized kernel widths. Therefore, they immediately work quite well since the SVM and the ML-SVM behave like a k-nearest neighbor or locally weighted learning method in this case.

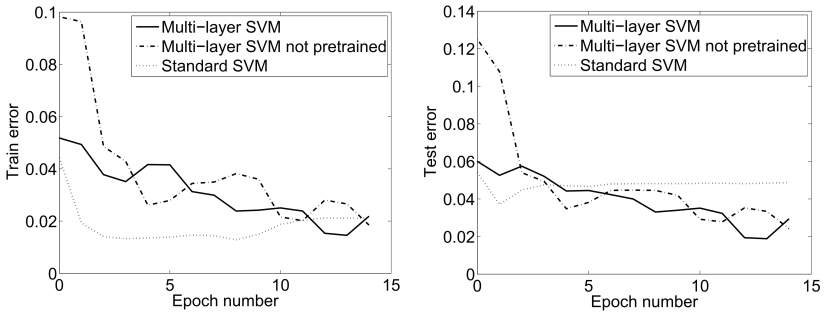

FIGURE 1.3: (A) Training error results on the Iris dataset. (B) Test error results on the Iris dataset.

The results show that the standard SVM quicker obtains a low training error than the other methods, but that its test error is higher (its best test error is 3.7%). The best test error is obtained by the (pre-trained) multi-layer SVM after 13 epochs, when it obtains a test error of 1.9%. The error of the multi-layer SVM that is not pre-trained starts much higher than with the other methods, but this method is still able to obtain a test error of 2.8% and significantly outperforms the standard SVM. The standard SVM obtains its best performance after a single training epoch with the gradient ascent algorithm during which 10 training iterations of the support vector coefficients were executed. Figures 1.3(A) and 1.3(B) show that for the multi-layer SVMs the test errors are very close to the training errors, except for the beginning. This behavior is due to the strong regulatization power of the output-layer SVMs. Even with many hidden-layer SVMs generalization performance can be excellent by setting the regularization parameter C to a small value.

We also plotted the evolution of the average values of the dual-objective function that correspond to the evolution of the training and test errors shown before. Again this plot shows averages of 1000 simulations. Figure 1.4 shows that the gradient ascent SVM monotonically increases the dual-objective function (between epochs 1 and 14, the dual-objective value increases from 33 to 70). As can be seen in Figure 1.3(B), this does not lead to always improving test errors. This may have to do with not exactly fulfilling the bias constraint. However, when PSO is used it optimizes the number of epochs to overcome this problem (it found the best value of 1 for the number of epochs). The multi-layer SVMs alternate between minimizing and maximizing the dual-objective

function. The min-max optimization process is quite complex, because multiple metaparameters influence the learning updates. Therefore, the dual objective does not just increase, then in the next epoch decreases, etc. Instead, the dual-objective function increases for some epochs, then decreases, etc., without any signs of convergence. The three figures show that the dual-objective should be minimized to obtain the lowest test errors. However, standard SVMs can only maximize the dual-objective function. Therefore, the flexibility of the hidden layer in the ML-SVM is especially fruitful to minimize the dual-objective function and thereby obtain lower test errors.

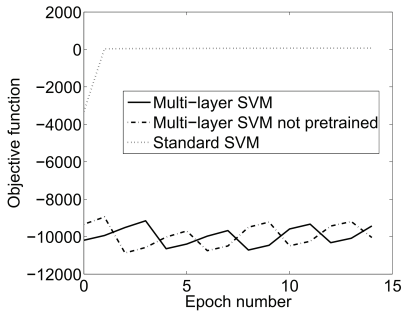

FIGURE 1.4: The evolution of the dual-objective value on the Iris dataset.

1.6 Discussion and Future Work

The multi-layer SVM consists of a hidden-layer of SVMs and an output layer of SVMs that learn to approximate the target function using the outputs of the hidden-layer SVMs. The results show that the ML-SVM can outperform other state-of-the-art machine learning algorithms. By going from a single SVM to the multi-layer SVM, we have made the SVM a deeper architecture. Compared to other deep neural network architectures, the ML-SVM has the advantage that due to the strong regularization power of the output-layer SVMs, the system does not easily overfit the data. Therefore, the ML-SVM could potentially perform very well with very large input vectors and few training examples. On the other hand, training an SVM with many examples is more computationally demanding than training a deep neural network architecture.

There are several advantages of the ML-SVM algorithm. First, the method is very flexible in adapting the kernel functions compared to other multiple-kernel learning algorithms. Second, the algorithm is straightforward to implement by using the gradient ascent algorithm and the backpropagation

technique. Finally, the training method uses a min-max optimization process, which is interesting and not (yet) applicable to neural network training.

There remains future work to be done in order to increase the power of the ML-SVM. First of all, the current implementation uses many metaparameters. Instead of using PSO to optimize the metaparameters, many different real-coded optimization algorithms can be employed. Second, the ML-SVM becomes very large for large datasets and then needs a lot of training time. To deal with large datasets, we want to explore stochastic gradient ascent techniques instead of the batch gradient ascent method we used in this chapter. We can also include more diversity in the hidden-layer SVMs, for example by letting them use different subsets of inputs, different examples, or different kernels. Finally, we want to develop more rigorous theory to explain why the ML-SVM performs so well and test the ML-SVM on challenging handwriting and image recognition datasets.

Acknowledgements

Thanks to Mark Embrechts, Adrian Millea, Arnold Meijster, Aleke Nolte, Egbert van der Wal, Marten Schutten, Rick van der Mark, Michiel van der Ree, and Marijn Stollenga for helping with the experiments.

Bibliography

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, May 2002.

[3] F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 6–15, 2004.

[4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, volume 13, pages 153–160, 2007.

[5] Y. Cho and L. K. Saul. Kernel methods for deep learning. *Advances in Neural Information Processing Systems*, 22:342–350, 2009.

[6] D.C. Ciresan, U. Meier, L.M. Gambardella, and J. Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.

[7] C. Cortes, M. Mohri, and A. Rostamizadeh. Learning non-linear combinations of kernels. *Advances in Neural Information Processing Systems*, 22:396–404, 2009.

[8] N. Cristianini and J. Shawe-Taylor. *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[9] F. Dinuzzo. Kernel machines with two layers and multiple kernel learning. *CoRR*, 2010.

[10] M. Gönen and E. Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, pages 2211–2268, July 2011.

[11] M. Graczyk, T. Lasota, Z. Telec, and B. Trawinski. Nonparametric statistical analysis of machine learning algorithms for regression problems. In *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 111–120. 2010.

[12] G.E. Hinton. Training product of experts by minimizing constrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[13] G.E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, pages 1527–1554, 2006.

[14] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[15] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[16] G.R.G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M.I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learnine Research*, 5:27–72, 2004.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Back-propagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[18] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 775–782, 2007.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[20] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[21] S. Sonnenburg, G. Rätsch, and C. Schäfer. A general and efficient multiple kernel learning algorithm. In *Advances in Neural Information Processing Systems 18*, pages 1273–1280, Cambridge, MA, 2006.

[22] J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific Pub, 2002.

[23] M. Turk and A. Pentland. Eigenfaces for recognition. *Cognitive Neuroscience*, 3(1):71–86, 1991.

[24] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[25] P. Vincent, H. Larochelle, Y. Bengio, and P-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

[26] P. J. Werbos. Advanced forecasting methods for global crisis warning and models of intelligence. In *General Systems*, volume XXII, pages 25–38, 1977.

[27] M.A. Wiering, M. Schutten, A. Millea, A. Meijster, and L.R.B. Schomaker. Deep support vector machines for regression problems. In *Proceedings of the International Workshop on Advances in Regularization, Optimization, Kernel Methods, and Support Vector Machines: theory and applications*, 2013.

[28] M.A. Wiering, M.H. van der Ree, M.J. Embrechts, M.F. Stollenga, A. Meijster, A. Nolte, and L.R.B. Schomaker. The neural support vector machine. In *Proceedings of the 25th Benelux Artificial Intelligence Conference (BNAIC)*, 2013.

[29] S Wold, M. Sjöström, and L. Eriksson. PLS-regression: a basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, 58:109–130, 2001.

[30] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[31] Z. Xu, R. Jin, I. King, and M.R. Lyu. An extended level method for efficient multiple kernel learning. In *Advances in Neural Information Processing Systems 20*, pages 1825–1832. Curran Associates, Inc., 2008.

[32] J. Zhuang, I.W. Tsang, and S.C.H. Hoi. Two-layer multiple kernel learning. In *AISTATS*, pages 909–917, 2011.