The Neural Support Vector Machine

M.A. Wiering [a] M.H. van der Ree [a] M.J. Embrechts [b] M.F. Stollenga [c]

A. Meijster [a] A. Nolte [d] L.R.B. Schomaker [a]

[a] *Institute of Artificial Intelligence and Cognitive Engineering, University of Groningen, Groningen, The Netherlands*
[b] *Department of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA*
[c] *Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland*
[d] *Bernstein Center for Computational Neuroscience, Berlin, Germany*

Abstract

This paper describes a new machine learning algorithm for regression and dimensionality reduction tasks. The Neural Support Vector Machine (NSVM) is a hybrid learning algorithm consisting of neural networks and support vector machines (SVMs). The output of the NSVM is given by SVMs that take a central feature layer as their input. The feature-layer representation is the output of a number of neural networks that are trained to minimize the dual objectives of the SVMs. Because the NSVM uses a shared feature layer, the learning architecture is able to handle multiple outputs and therefore it can also be used as a dimensionality reduction method. The results on 7 regression datasets show that the NSVM in general outperforms a standard SVM and a multi-layer perceptron. Furthermore, experiments on eye images show that the NSVM autoencoder outperforms state-of-the-art dimensionality reduction methods.

1 Introduction

Multi-layer perceptrons (MLPs) [16, 10] are universal in the sense that they can approximate any continuous nonlinear function arbitrary well on a compact interval [3]. However, one of their drawbacks is that in training neural networks one usually tries to solve a nonlinear optimization problem that has many local minima. Furthermore, neural networks tend to overfit on small datasets. Support vector machines (SVMs) [13, 2, 11] are a more robust method for both classification and regression tasks, and usually have excellent generalization performance. This makes them very well suited for small datasets with many features. However, SVMs also have some drawbacks: (1) Their performance heavily relies on the chosen kernel function, but most kernel functions are not very flexible. (2) SVMs are in principle "shallow" architectures, whereas deep architectures [8, 1] have been shown to be very promising alternatives to these shallow models. (3) The standard SVM is unable to handle multiple outputs in a single architecture, which restricts their usage for possible learning tasks such as dimensionality reduction.

In this paper, we introduce the Neural Support Vector Machine (NSVM), a hybrid machine learning algorithm consisting of both neural networks and SVMs. The output of the NSVM is given by support vector machines that take a small central feature layer as their input. This feature layer is in turn the output of a number of neural networks, trained through backpropagation of the derivatives of the dual objectives of the SVMs with respect to the feature-node values. The NSVM aims to overcome the problems of the standard SVM. First, the NSVM adds more layers to the SVM, making it "deeper". Furthermore, the neural networks can learn arbitrary features, making the kernel functions much more flexible. Finally, by combining multiple SVMs with a shared feature layer into one learning architecture, the NSVM extends an SVM's generalization capability to multiple outputs.

The NSVM is related to some other methods. In [14], a neural support vector network (NSVN) architecture is proposed. In this NSVN, the neural networks are trained using a boosting algorithm [5] and the

support vectors are chosen beforehand. Compared to the NSVM, this method updates the neural networks in a more heuristical manner and chooses the support vectors beforehand, which make its theory less elegant than the NSVM. The NEUROSVM [6] uses MLPs to learn a representation of the data in a feature space. This representation is subsequently used as the input of an ensemble of support vector classifiers. Unlike the method proposed here, the stages of feature learning and classification training are separated in the NEUROSVM. Therefore, this method acts more like an ensemble method like stacking [18]. In the training scheme of the NSVM, the optimization of the neural network weights is intricately linked to the optimization of the SVM objective. Another related method was suggested by Suykens and Vandewalle [12]. Their modified support vector method trains the weight matrix between the input layer and the feature layer by minimizing an upper bound on the VC dimension. The NSVM uses another optimization method and we put more emphasis on how it can be extended to handle multiple outputs. Because of this ability to handle multiple outputs the NSVM can also be used as an autoencoder [4, 8]: a non-linear model that can be used to find an efficient coding of data.

This paper attempts to answer the following research questions: (1) How does the single-output regression NSVM compare to other state-of-the-art machine learning algorithms on regression problems? (2) How does the NSVM autoencoder compare to other dimensionality reduction methods?

Outline. In section 2 we will discuss the theory of support vector regression. Next, we present a single-output NSVM, discussing its architecture and the modified support vector objectives it utilizes. In section 4 we show how the NSVM can be adapted to handle multiple outputs by presenting the NSVM autoencoder. Section 5 will cover the setup and the results of the experiments conducted with the single-output NSVM on several benchmark regression datasets and the results of the NSVM autoencoder on a dataset of images of left eyes of people. A conclusion and future work will be presented in section 6.

2 Support Vector Regression

In linear ε-insensitive support vector regression (SVR), training consists of solving the following constrained optimization problem:

$$\min_{\mathbf{w},\boldsymbol{\xi}^{(*)},b}\left[\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*)\right] \tag{1}$$

subject to constraints:

$$y_i - \mathbf{w}\cdot\mathbf{x}_i - b \leq \varepsilon + \xi_i, \quad \mathbf{w}\cdot\mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^*, \quad \text{and} \quad \xi_i,\ \xi_i^* \geq 0. \tag{2}$$

Here, $\mathbf{w}$ is a weight vector, b is a bias value, $(\mathbf{x}_i, y_i)$ is a training sample and its target value, ξ_i and ξ_i^* are so-called "slack variables" enabling the model to allow deviations between the model output and the target value of training examples larger than ε, C is a parameter controlling the extent to which such deviations are allowed and ℓ is the total number of training samples. Equation (1) is called the *primal objective*, and its variables *primal variables*. Introduction of Lagrange multipliers $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ and solving for the coordinates of a saddle point allow us to reformulate the primal objective and its constraints in the following way[1]:

$$\max_{\boldsymbol{\alpha}^{(*)}}\left[-\varepsilon\sum_{i=1}^{\ell}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i)y_i - \frac{1}{2}\sum_{i,j=1}^{\ell}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_i\cdot\mathbf{x}_j)\right] \tag{3}$$

subject to constraints:

$$0 \leq \alpha_i^{(*)} \leq C \quad \text{and} \quad \sum_{i=1}^{\ell}(\alpha_i - \alpha_i^*) = 0. \tag{4}$$

Here, we use $\alpha_i^{(*)}$ to denote both α_i and α_i^*, and $\boldsymbol{\alpha}^{(*)}$ to denote the vectors containing all $\alpha_i^{(*)}$ values. Equation (3) is called the *dual objective*. The second constraint in (4) is called the bias constraint. Once the

[1] See [11] for a more elaborate derivation.

$\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ maximizing the dual objective are found, a linear regression SVM determines its output using:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i)(\mathbf{x}_i \cdot \mathbf{x}) + b \tag{5}$$

The presented SVR model assumes that the relation between $\mathbf{x}_i$ and y_i is a linear one. Obviously, we want to make the SVR model nonlinear. This could be achieved by preprocessing the training patterns $\mathbf{x}_i$ by a map $\Psi : \mathcal{X} \to \mathcal{F}$ into some higher-dimensional feature space $\mathcal{F}$ and then applying the standard SVR algorithm [11]. However, such an approach can become computationally infeasible. Since both the dual objective (3) and the regression estimate (5) only depend on inner products between patterns $\mathbf{x}_i$, it suffices to know $K(\mathbf{x}_i, \mathbf{x}) := \Psi(\mathbf{x}_i) \cdot \Psi(\mathbf{x})$, rather than Ψ explicitly. It is this *kernel function* $K(\cdot, \cdot)$ that is often used in SVR to make the algorithm nonlinear. A number of kernel functions are widely used, including polynomial functions, radial basis functions and sigmoidal functions.

3 The Neural Support Vector Machine

Here we introduce the Neural Support Vector Machine. This section concerns a single-output regression NSVM. First, we will discuss its architecture. Next we describe the modifications made to the SVR objectives of section 2. The section is closed by a description of the procedure by which the system is trained.

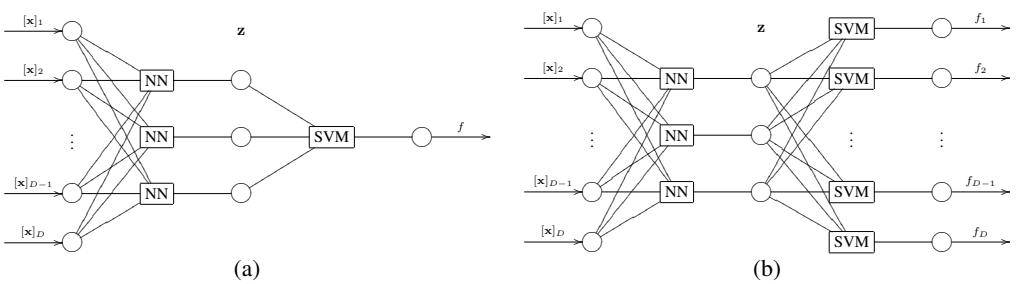
3.1 Architecture

The NSVM (see Figure 1a) consists of: (1) an input layer consisting of D nodes; (2) a central feature layer $\mathbf{z}$ consisting of d nodes; (3) a total of d two-layer neural networks (MLPs) N, which each take the entire input layer as their input and produce one of the feature values as their output, and (4) a main support vector regression model M that takes the entire feature layer as its input and determines the value of the output node. When a pattern $\mathbf{x}$ of dimension D is presented to the NSVM, it is propagated through the neural networks, determining the values of the feature layer. We use $\Phi(\mathbf{x}|\theta)$ to denote the mapping performed by the neural networks, i.e. $\mathbf{z} = \Phi(\mathbf{x}|\theta)$. Here, $\Phi : \mathbb{R}^D \to \mathbb{R}^d$ and θ is a vector containing all the weights of the neural networks. The representation in the feature layer is used as input for the support vector machine M that determines the value of the output node. The regression NSVM computes its output using:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) K(\Phi(\mathbf{x}_i|\theta), \Phi(\mathbf{x}|\theta)) + b. \tag{6}$$

Where $K(\cdot, \cdot)$ is the kernel function of the main SVM.

Figure 1: Architecture of the NSVM regression estimator (a), and the NSVM autoencoder (b). In both examples the feature layer consists of three neural networks each one extracting one feature.



3.2 Modified Objectives

To obtain a suitable f, the system must find a representation of the input data in $\mathbf{z}$ that codes the features most relevant for estimating the desired output. The NSVM adapts the primal objective of SVR by replacing the training samples $\mathbf{x}_i$ with their representation in the feature layer, $\Phi(\mathbf{x}_i|\theta)$. Consequently, the weightvector of the neural networks θ is introduced as an additional primal variable. This yields the following primal objective for an NSVM with a linear SVR model M:

$$\min_{\mathbf{w},b,\theta}\left[\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{\ell}(\xi_i+\xi_i^*)\right] \tag{7}$$

subject to constraints: $y_i - \mathbf{w}\cdot\Phi(\mathbf{x}_i|\theta) - b \le \varepsilon+\xi_i,\ \ \mathbf{w}\cdot\Phi(\mathbf{x}_i|\theta)+b-y_i \le \varepsilon+\xi_i^*,$ and $\xi_i^{(*)} \ge 0$.

Correspondingly, the new 'dual objective' (the primal variable θ has not been eliminated!) for the NSVM when the SVR model M uses a kernel function $K(\cdot,\cdot)$ is:

$$\min_{\theta}\max_{\boldsymbol{\alpha}^{(*)}} W(\boldsymbol{\alpha}^{(*)},\theta) = -\varepsilon\sum_{i=1}^{\ell}(\alpha_i^*+\alpha_i)+\sum_{i=1}^{\ell}(\alpha_i^*-\alpha_i)y_i-\frac{1}{2}\sum_{i,j=1}^{\ell}(\alpha_i^*-\alpha_i)(\alpha_j^*-\alpha_j)K(\Phi(\mathbf{x}_i|\theta),\Phi(\mathbf{x}_j|\theta)) \tag{8}$$

subject to constraints: $0 \le \alpha_i, \alpha_i^* \le C$, and $\sum_{i=1}^{\ell}(\alpha_i^*-\alpha_i)=0$.

3.3 Training Procedure

The two goals of training the NSVM regression estimator are: (1) Find the $\boldsymbol{\alpha}^{(*)}$ that maximizes Equation (8), and (2) Find the weights of the neural networks θ such that each network N_a contributes to the $\Phi(\mathbf{x}|\theta)$ that minimizes Equation (8). The two goals are not obtained independently, i.e. an adjustment of $\boldsymbol{\alpha}$ requires an adjustment of the neural networks and vice versa. Whenever the system is presented a training pattern $\mathbf{x}_i$, the NSVM adjusts each α_i and α_i^* towards a local maximum of $W^+(\cdot)$ with a gradient ascent algorithm:

$$\alpha_i^{(*)} \leftarrow \alpha_i^{(*)} + \lambda\frac{\partial W^+(\cdot)}{\partial\alpha_i^{(*)}} \quad \text{with} \quad W^+(\cdot) = W(\cdot) - P_1(\sum_{j=1}^{\ell}(\alpha_j^*-\alpha_j))^2 - P_2\alpha_i^*\alpha_i \tag{9}$$

where λ is a metaparameter controlling the learning rate and P_1, P_2 are parameters chosen beforehand. By using the derivative of $W^+(\cdot)$ in Equation (9) we ensure that $\alpha_i^{(*)}$ is adjusted towards a satisfaction of the bias constraint, and towards a pair of (α_i^*,α_i) of which at least one of the values equals zero. The two derivatives of our gradient ascent algorithm are given by:

$$\frac{\partial W^+(\cdot)}{\partial\alpha_i^*} = -\varepsilon + y_i - \sum_{j=1}^{\ell}(\alpha_j^*-\alpha_j)K(\Phi(\mathbf{x}_i|\theta),\Phi(\mathbf{x}_j|\theta)) - 2P_1\sum_{j=1}^{\ell}(\alpha_j^*-\alpha_j) - P_2\alpha_i \tag{10}$$

and

$$\frac{\partial W^+(\cdot)}{\partial\alpha_i} = -\varepsilon - y_i + \sum_{j=1}^{\ell}(\alpha_j^*-\alpha_j)K(\Phi(\mathbf{x}_i|\theta),\Phi(\mathbf{x}_j|\theta)) + 2P_1\sum_{j=1}^{\ell}(\alpha_j^*-\alpha_j) - P_2\alpha_i^*.$$

We keep all $\alpha^{(*)}$-coefficients between 0 and C. We want to adjust the weights of each neural network N_a such that its output *minimizes* $W^+(\cdot)$. Let us denote the output of N_a given training pattern $\mathbf{x}_i$ (i.e. the a-th entry of $\mathbf{z}_i = \Phi(\mathbf{x}_i|\theta)$) by z_i^a. Then, using gradient descent, we aim to decrease z_i^a by:

$$\frac{\partial W^+(\cdot)}{\partial z_i^a} = -(\alpha_i^*-\alpha_i)\sum_{j=1}^{\ell}(\alpha_j^*-\alpha_j)\frac{\partial K(\Phi(\mathbf{x}_i|\theta),\Phi(\mathbf{x}_j|\theta))}{\partial z_i^a}. \tag{11}$$

By considering Equation (11) the error of N_a given pattern $\mathbf{x}_i$, we can adjust its weights to decrease $W^+(\cdot)$ using backpropagation [16, 10]. The complete training algorithm is given in Algorithm 1. It shows that training the main SVM M is alternated for a number of epochs with training the neural networks. The bias value of the SVR model M is learned by using the average error.

Algorithm 1 The NSVM training algorithm

Initialize main SVM M
Initialize neural networks

repeat
  Compute kernel matrix for main SVM M
  Train main SVM M
  Use backpropagation on the dual objective of M to train the neural networks
until stop_condition (maximal number of epochs)

The computational time and space complexity of the NSVM is linear in the number of inputs and also linear in the number of feature-extracting neural networks. Just as SVMs, its complexity depends quadratically on the number of training examples.

4 The Neural Support Vector Machine Autoencoder

This section will show how the single-output regression NSVM can be adapted to handle multiple outputs by presenting the NSVM autoencoder. The architecture of the NSVM autoencoder differs from the single-output regression NSVM 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 feature layer as input and determine the value of one of the output nodes.

Figure 1b depicts the architecture graphically. Just like the method described in section 3, the forward-propagation of a pattern $\mathbf{x}$ of dimension D determines the representation in the feature layer $\mathbf{z}$. The feature layer is then used as input for each support vector machine M_c that determines its output according to:

$$f_c(\mathbf{x}) = \sum_{i=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i)(\Phi(\mathbf{x}_i|\theta) \cdot \Phi(\mathbf{x}|\theta)) + b_c. \tag{12}$$

Here, we use $[\boldsymbol{\alpha}_c]_i$ to denote the α_i used by M_c. In this case we use a linear kernel for the SVR models.

Like the single-output NSVM, the NSVM autoencoder tries to find the neural network weights θ such that $\Phi(\cdot)$ codes the features most relevant to the properties corresponding with the desired output. Since the desired output of an autoencoder is the same as the input data, it is trained to learn structural features of the data in general. The dual objective of each support vector machine M_c is:

$$\begin{aligned}\min_{\theta} \max_{\boldsymbol{\alpha}_c^*, \boldsymbol{\alpha}_c} W_c(\mathbf{x}, \boldsymbol{\alpha}_c^{(*)}) = &-\varepsilon \sum_{i=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i + [\boldsymbol{\alpha}_c]_i) + \sum_{i=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i)[\mathbf{y}_i]_c \\ &- \frac{1}{2}\sum_{i,j=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i)([\boldsymbol{\alpha}_c^*]_j - [\boldsymbol{\alpha}_c]_j)(\Phi(\mathbf{x}_i|\theta) \cdot \Phi(\mathbf{x}_j|\theta))\end{aligned} \tag{13}$$

subject to constraints: $0 \leq \alpha_i, \alpha_i^* \leq C$, and $\sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i) = 0$. Recall from section 3.3 that the first goal of training the NSVM regression estimator is finding the $\boldsymbol{\alpha}^{(*)}$ that maximizes $W_c(\cdot)$ shown in Equation (8). Similarly, in the NSVM autoencoder we want to find the $\boldsymbol{\alpha}_c^{(*)}$ that maximizes Equation (13) for every support vector machine M_c. Like for the single-output NSVM, this is done through gradient ascent.

However, the minimization of Equation (13) with respect to θ is different from the single-node NSVM. Since all SVMs share the same feature layer, we cannot just minimize $W(\cdot)$ for every SVM separately. It is actually this shared nature of the feature layer which enables the NSVM to handle multiple outputs in one architecture. Let us again denote the output of N_a given $\mathbf{x}_i$ by z_i^a. We can adjust the weights of each N_a towards a minimum of (13) with respect to z_i^a by backpropagation of the *sum* of the derivatives of all dual objectives $W_c(\cdot)$ with respect to z_i^a. Therefore, for training the MLPs the NSVM autoencoder uses:

$$-\sum_{c=1}^{D} \frac{\partial W_c(\cdot)}{\partial z_i^a} \quad \text{with} \quad \frac{\partial W_c(\cdot)}{\partial z_i^a} = -([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i) \sum_{j=1}^{\ell} ([\boldsymbol{\alpha}_c^*]_j - [\boldsymbol{\alpha}_c]_j) z_j^a. \tag{14}$$

This situation resembles that of a neural network with multiple outputs, in which a single hidden node's error is determined by summing the proportional errors of all the neurons it directly contributes to [16, 10].

5 Experimental Results

5.1 Experimental Results on Regression Datasets

We experimented with 7 regression datasets to compare the NSVM 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. The 7 datasets are described in [7]. Some information about these datasets is shown in Table 1. The datasets are split into 90% training data and 10% testing data.

The NSVM uses two-layer MLPs with Tanh activation functions in the output and hidden layer. The NSVM contains a fairly large number of metaparameters. To optimize the metaparameters, particle swarm optimization (PSO) [9] has been utilized. Finally, we used 1000 or 4000 times cross validation runs with the best found metaparameters to compute the mean squared errors and standard errors of the different methods for each dataset. In Table 1 we show the results of the NSVM, the results of a standard SVM trained with gradient ascent, and the results for an SVM and a multi-layer perceptron (MLP) obtained in [7]. We note that Graczyk et al. [7] only performed 10-fold cross validation and did not report any standard errors.

Dataset	#Exams.	#Feat.	#Runs	NSVM	Gradient ascent SVM	SVM [7]	MLP [7]
Baseball	337	6	4000	0.02343 ± 0.00010	$0.02413 \pm 0.00011^{-}$	0.02588^{-}	0.02825^{-}
Boston Housing	461	4	1000	0.006782 ± 0.000091	0.006838 ± 0.000095	0.007861^{-}	0.007809^{-}
Concrete Strength	72	5	4000	0.00782 ± 0.00006	$0.00706 \pm 0.00007^{+}$	0.00851^{-}	0.00837^{-}
Diabetes	43	2	4000	0.02655 ± 0.00023	0.02719 ± 0.00026	0.02515^{+}	0.04008^{-}
Electrical Length	495	2	1000	0.006492 ± 0.000064	0.006382 ± 0.000066	0.006352	0.006417
Machine-CPU	188	6	1000	0.00740 ± 0.00014	$0.00805 \pm 0.00018^{-}$	0.00777	0.00800^{-}
Stock	950	5	1000	0.000825 ± 0.000005	$0.000862 \pm 0.000006^{-}$	0.002385^{-}	0.002406^{-}
Wins/Losses					1 - 3	1 - 4	0 - 6

Table 1: The 7 datasets (name, number of examples, number of features), the number of runs for cross validation, and the mean squared errors and standard errors of the NSVM, the gradient ascent SVM, and results from [7] for an SVM and an MLP. We denote with +/- a significant win/loss ($p < 0.01$) compared to the NSVM.

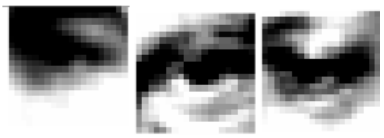
The results show that the NSVM in general outperforms the other methods. It wins against the different SVMs in 3 or 4 times, and only loses one time. We want to remark that the win of the SVM from [7] could be explained by the small (10) number of cross validation runs on a a small dataset. Furthermore, compared to the results obtained with the MLP, the NSVM obtains much better performances on almost all datasets.

5.2 Experimental Results on Dimensionality Reduction of Eye Images

The dataset we used contains 1300 gray-scale images of left eyes of 20 by 20 pixels, manually cropped from the "Labeled Faces in the Wild Dataset". Figure 2 shows three examples of the data used. The 400 pixel values have been normalized such that their average value is $\mu = 0.0$ with a standard deviation of $\sigma = 0.3$.

We used $\frac{2}{3}$ of the data as trainingdata and the rest for testing. All experiments were repeated 10 times. For the autoencoder experiments, we implemented an adaptive learning rate of the neural networks of the NSVM. Furthermore, we used a linear kernel for the SVMs M_c. The two-layer MLPs used Tanh and linear activation functions in the hidden and output layer, respectively.

Figure 2: Three examples of images of left eyes of people used in the autoencoder experiment.



We made a comparison to a state-of-the-art non-linear neural network autoencoding method, named a denoising autoencoder [15], 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. Each input sample was 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 generalization performance of the autoencoder.

We also compared the NSVM to principal component analysis (PCA) using a multi-variate Partial-Least Squares (PLS) regression model with standardized inputs and outputs [17]. It can easily be shown that the standard PLS algorithm 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.

As results we compute the reconstruction error on the test images using the root of the mean squared error (RMSE). The results of the three methods using 10, 20, and 50 dimensions (or features) are shown in Table 2. The results show that the best method is the NSVM autoencoder. It significantly outperforms the denoising autoencoder when 50 features are used, and outperforms PCA with all sizes of the feature layer.

Dimensionality	RMSE NSVM	RMSE NN	RMSE PCA
10	0.1218 ± 0.0004	0.1211 ± 0.0002	$0.1242 \pm 0.0004^{-}$
20	0.0882 ± 0.0004	0.0890 ± 0.0002	$0.0903 \pm 0.0003^{-}$
50	0.0503 ± 0.0002	$0.0537 \pm 0.0001^{-}$	$0.0519 \pm 0.0002^{-}$
Wins/losses		0 - 1	0 - 3

Table 2: The test RMSE and its standard error obtained by the NSVM autoencoder, the denoising neural network autoencoder, and principal component analysis, for different sizes of the feature layer. The symbol +/- indicates a significant ($p < 0.01$) win/loss of the method compared to the NSVM autoencoder.

6 Conclusion and Future Work

In this paper we have described the Neural Support Vector Machine, a new machine learning algorithm that learns through a complex interaction between support vector machines and neural networks. Because neural networks try to minimize a function that the SVMs try to maximize, the learning dynamics follow a chaotic behavior. Although the NSVM has many trainable parameters, the experimental results have shown that the NSVM often performs better than state-of-the-art machine learning techniques. But, because the NSVM consists of a large number of adaptive parameters, it takes more computational time than the other methods.

There is a lot of possible future work. First of all, the current implementation uses around 12 different metaparameters. We want to study if they are all important and if we cannot set some of them to constant (universal) values. Second, the NSVM becomes very large for large datasets and then needs a lot of training

time. To deal with large datasets, we want to research faster optimization techniques. We can also include more diversity in the feature-extracting neural networks, for example by letting them use different inputs or different examples. Finally, we want to test the NSVM on some challenging image classification datasets.

References

[1] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. *Large Scale Kernel Machines, MIT Press*, 2007.

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

[3] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2:303–314, 1989.

[4] D. DeMers and G. Cottrell. Non-linear dimensionality reduction. *Advances in neural information processing systems*, pages 580–587, 1993.

[5] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the thirteenth International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

[6] P. Ghanty, S. Paul, and N.R. Pal. NEUROSVM: An architecture to reduce the effect of the choice of kernel on the performance of SVM. *The Journal of Machine Learning Research*, 10:591–622, 2009.

[7] 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.

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

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

[10] 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.

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

[12] J.A.K. Suykens and J. Vandewalle. Training multilayer perceptron classifiers based on a modified support vector method. *IEEE Transactions on Neural Networks*, 1(4):907–911, 1999.

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

[14] P. Vincent and Y. Bengio. A neural support vector network architecture with adaptive kernels. In *Proceedings of IJCNN 2000*, volume 5, pages 187–192. IEEE, 2000.

[15] 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.

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

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

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