Neural Networks 49 (2014) 39–50

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Biologically relevant neural network architectures for support vector machines

Magnus Jändel *

Swedish Defence Research Agency, SE 164 90 Stockholm, Sweden

ARTICLE INFO

Article history:
Received 8 December 2011
Received in revised form 5 June 2013
Accepted 18 September 2013

Keywords:
Support vector machine
Neural network
Competitive queuing memory
Perceptual learning

ABSTRACT

Neural network architectures that implement support vector machines (SVM) are investigated for the purpose of modeling perceptual one-shot learning in biological organisms. A family of SVM algorithms including variants of maximum margin, 1-norm, 2-norm and ν-SVM is considered. SVM training rules adapted for neural computation are derived. It is found that competitive queuing memory (CQM) is ideal for storing and retrieving support vectors. Several different CQM-based neural architectures are examined for each SVM algorithm. Although most of the sixty-four scanned architectures are unconvincing for biological modeling four feasible candidates are found. The seemingly complex learning rule of a full ν-SVM implementation finds a particularly simple and natural implementation in bisymmetric architectures. Since CQM-like neural structures are thought to encode skilled action sequences and bisymmetry is ubiquitous in motor systems it is speculated that trainable pattern recognition in low-level perception has evolved as an internalized motor programme.

1. Introduction

Many living organisms learn new behaviors from one single exposure to significant sensor inputs. For example, snails learn food aversion from a single experience (Teyke, 1995). This form of learning is called one-shot learning or one-trial learning (Guthrie, 1935) and is common in nature but is hard to explain in neural network models since Hebbian learning requires many repetitions of appropriate stimuli before a new pattern is added to an existing repertoire of recognizable patterns. A mechanism where the brain remembers noteworthy sensory inputs and runs a background loop of simulated experiences would help to provide the repetitive training that is required for Hebbian learning. Such regurgitation could perhaps be performed in sleep when normal sensor inputs are disabled. A driving motivation for the present work is to find viable neural architectures for modeling biological one-shot learning according to this concept.

Support vector machines (SVMs) are pattern recognition algorithms with a firm foundation in optimization and generalization theory and a good track record of practical applications. SVMs are easy to use for non-experts and the performance of a standard SVM often rival that of expertly hand-crafted ANN. An interesting property of SVMs is that the learning state consists of a special set of training examples called support vectors. Significant new training examples may join the set of support vectors. Therefore, it appears that neural implementations of SVMs could be interesting as models of one-shot perceptual learning in nature.

The reasons for considering neural network implementations of SVMs are (1) finding efficient hardware implementations of SVMs, in particular as analogue circuits, and (2) using SVMs as models of biological neural functions. This paper follows the latter approach but we will here briefly review the literature of both branches. The SVM classification function is a weighted sum of kernel function values in which the input vector is one of the arguments to the kernel function. Neural networks can implement any continuous multivariate function with arbitrary accuracy (Cybenko, 1989) so it is not surprising that the SVM classification function readily is expressed as neural networks (Schölkopf & Smola, 2002). As pointed out by Yang, He, and Hu (2012), SVM training is a quadratic programming problem and recurrent neural networks are able to solve such problems. Hence it is feasible to implement both SVM classification and supervised iterative SVM training as artificial neural networks.

Anguita, Ridella, and Rovetta (1998) showed that SVMs can be realized as recurrent electronic circuits. Anguita and Boni (2003) reviewed VLSI implementations of SVMs. A two-layer artificial neural network that implements a 1-norm SVM was defined by Tan, Xia, and Wang (2000) and simplified with respect to the bias calculation by Anguita and Boni (2002). Xia and Wang (2004) demonstrated a one-layer recurrent ANN implementing a 1-norm SVM for which Perfetti and Ricci (2006) as well as Liu and Liu (2009) and Yang et al. (2012) proposed improvements intended to further optimize electronic circuit implementations. For comparison to the present work, it should be noted that these hardware-oriented implementations are in the context of supervised learning

* Tel.: +46 709277264; fax: +46 855503700.
E-mail addresses: magjan@foi.se, magnus@jaendel.se, magjan76@yahoo.se.

0893-6080/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.neunet.2013.09.006

in which an iterative update rule acts on a series of externally provided training examples and is hence not intended for modeling biological one-shot learning.

Support vector machines are used extensively for automating classification in computational biology (for a review see e.g. Noble (2004)). The literature on support vector machines as models of biological phenomena is, however, scant. Galán, Sachse, Galizia, and Herz (2003, 2004) analyzed the olfactory code of the honey bee and noted that the interaction between the antenna lobe and the mushroom body can be regarded as a biological realization of the classification function of a support vector machine. Odours trigger neural attractors in the antenna lobe and the pattern of activated attractors is classified by the mushroom body according to an SVM-like process. Viéville and Crahay (2004) introduced a biologically plausible SVM-like neural network classifier aiming at explaining the fast (100–150 ms) classification process in the visual cortex. The key idea is to classify based on distance to a set of known class prototypes. The prototypes are similar to support vectors and the Hebbian learning mechanism is guided by Vapnik learning theory (Vapnik, 1998). The present paper differs from Viéville and Crahay (2004) in that it explores neural network architectures for standard SVMs and that it focuses on explaining biological one-shot learning rather than the speed of visual perception.

Previous work by the author of this paper shows that a particular SVM algorithm (zero-bias ν-SVM) can be expressed as a biologically plausible ANN that is capable of one-shot learning (Jändel, 2010a). The architecture and dynamics of this model have been compared to the olfactory system (Jändel, 2010a) and also to the burst dynamics of the thalamocortical system (Jändel, 2009). Jändel (2011) point to an evolutionary path along which a neural SVM could emerge from ubiquitous neural components.

Section 2 of the present paper defines an ensemble of SVM algorithms. Section 3 introduces the major neural building blocks, derives training rules and finally analyzes several different neural architectures for each SVM algorithm. Discussion and conclusions are found in Section 4.

2. Support vector machine algorithms

We consider SVMs for binary classification that execute in two different modes: classification and learning. In the classification mode, they receive a test vector $\mathbf{x}$ and output the predicted valence $y \in \{1, -1\}$ of the test vector (bold letters signify vectors). In the learning mode, they use a set of training examples for optimizing internal parameters called weights. Each training example $\{\mathbf{x}, y\}$ consists of a training vector and the associated known valence. All input vectors are considered to be real-valued vectors of equal length. The goal of training is to achieve optimal robust classification performance (see Cristianini and Shawe-Taylor (2000) for details about SVMs).

For future reference, we describe eight well known types of SVMs where each definition consists of the following four parts.

(I) The classification function,

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b, \tag{1}$$

where K is the positive definite kernel function, α_i are weights and b is a real-valued bias factor. The input vector is classified to be of positive valence if $f(\mathbf{x}) \geq 0$ and to be of negative valence otherwise.

(II) The (dual) objective function $W(\boldsymbol{\alpha})$ where $\boldsymbol{\alpha}$ is the weight vector.

(III) A set of constraints. The positivity condition,

$$\forall i : \alpha_i \geq 0 \tag{2}$$

is satisfied for all SVMs. Biased ($b > 0$) SVMs always include the constraint,

$$\sum_{i=1}^{m} y_i \alpha_i = 0. \tag{3}$$

The constraints (2) and (3) are understood to hold even if they are not repeated in the following SVM definitions.

(VI) An algorithm for computing the value of the bias factor b.

The optimal set of weights is found by maximizing $W(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$ under the relevant set of constraints. After computing the bias factor the classification function is used for predicting the valence of test vectors.

Support vectors are training examples with weights $\alpha_i > 0$. The remaining *trivial examples* with $\alpha_i = 0$ do not contribute to the classification function. Support vectors are either *regular* support vectors or *outliers* where the former are support vectors that are correctly classified with sufficient margin. The precise definition of regular support vectors differs between the various SVM types.

For future use we define the classification margin of a training example $(\mathbf{x}_i, y_i)$,

$$M_i = y_i f(\mathbf{x}_i). \tag{4}$$

The margin is positive if the example is correctly classified and negative otherwise. Some of the SVM algorithms specify a target margin M_T for support vectors.

We further define the unbiased classification function,

$$h(\mathbf{x}) = f(\mathbf{x}) - b = \sum_{i=1}^{m} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i). \tag{5}$$

Introducing Kronecker delta functions,

$$\delta_+(y) = \begin{cases} 1 & \text{if } y = 1 \\ 0 & \text{if } y \neq 1 \end{cases}, \qquad \delta_-(y) = \begin{cases} 1 & \text{if } y = -1 \\ 0 & \text{if } y \neq -1, \end{cases} \tag{6}$$

and the set of regular support vectors SV_R, averages over positive and negative valence regular support vectors are defined according to,

$$\tilde{h}_* = \frac{1}{\tilde{m}_*} \sum_{i \in SV_R} \delta_*(y_i) h(\mathbf{x}_i), \tag{7}$$

where $\tilde{m}_* = \sum_{i \in SV_R} \delta_*(y_i)$ and the symbol $*$ refers to either $+$ or $-$.

The eighth types of SVMs to be defined come in four pairs where each pair includes a zero-bias ($b = 0$) and a biased SVM. The biased SVMs are first described.

2.1. Maximum-margin SVM

The maximum-margin SVM is a hard-margin SVM which means that all support vectors are regular with $M_T = 1$ and the remaining trivial examples have margins $M_T > 1$. Training will succeed only if these conditions are satisfied after optimization of the objective function,

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{8}$$

under the standard constraints (2) and (3). The bias factor can be computed according to,

$$b = -\frac{1}{2}(\tilde{h}_+ + \tilde{h}_-). \tag{9}$$

2.2. 1-norm SVM

The 1-norm SVM is a soft-margin SVM where support vectors may violate the target margin $M_T = 1$. A slack variable that measures how much the margin is surpassed is defined for each

training example. The objective function and the constraints are derived from a Lagrangian optimization problem where the sum of the slack variables is incorporated in the Lagrangian. The objective function is the same as for the maximum margin SVM (Eq. (8)) but there is an extra constraint,

$$\forall i : \alpha_i \leq C, \tag{10}$$

where C is a positive parameter. Regular support vectors have weights in the interval $0 < \alpha_i < C$. The bias factor is computed using the same expression (Eq. (9)) as for the maximum margin SVM.

2.3. The 2-norm SVM

The 2-norm SVM is a soft margin SVM where the sum of the squares of the slack variables is optimized in the underlying Lagrangian problem. The resulting objective function is,

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{m} y_i y_j \alpha_i \alpha_j \left(K(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C} \right) \tag{11}$$

where δ_{ij} is a Kronecker delta function and no extra constraint is applied. The parameter C is positive. All support vectors are regular. The bias factor can be computed according to,

$$b = -\frac{1}{2} \left(\tilde{h}_+ + \tilde{h}_- + \frac{\tilde{\alpha}_+ - \tilde{\alpha}_-}{C} \right) \tag{12}$$

where $\tilde{\alpha}_+$ and $\tilde{\alpha}_-$ are the average weights of positive and negative valence support vectors respectively.

2.4. The ν-SVM

The ν-SVM is a soft margin SVM where the trade-off between generalization and precision is governed by a single dimensionless parameter ν ($0 < \nu < 1$) (Schölkopf, Smola, Williamson, & Bartlett, 2000). The objective function is,

$$W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i=1,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{13}$$

There are two extra constraints

$$\forall i : \alpha_i \leq \frac{1}{m} \tag{14}$$

and

$$\sum_{i=1}^{m} \alpha_i = \nu. \tag{15}$$

The latter constraint is usually written as $\sum_{i=1}^{m} \alpha_i \leq \nu$ but Chang and Lin (2001) shows that the stricter constraint (15) applies to all solutions of the ν-SVM problem. Regular support vectors have weights in the interval $0 < \alpha_i < \frac{1}{m}$. The bias factor is given by Eq. (9).

2.5. Zero-bias SVMs

For each of the four SVMs defined in Sections 2.1–2.4 there is a zero-bias companion with $b = 0$. Cristianini and Shawe-Taylor (2000) demonstrates that zero-bias SVMs are valid pattern recognizers although there in general is a negative impact on convergence and generalization ability. Starting from a biased SVM it is always possible to absorb the bias factor in a redefined kernel. This is equivalent to clamping the bias to zero while adding a constant factor to the kernel function. The objective function of each of the zero-bias SVM is identical to that of the associated biased SVM. The set of constraints is also the same expect the constraint (3) is invalidated.

3. Neural network implementations of SVMs

Any biological realization of an SVM must implement four main processes. The primary task of the SVM is to *perform classifications* that are used by the perception system of the organism. In addition the system must be able to *acquire relevant training examples*, *learn appropriate weights* and, for biased algorithms, *compute the bias factor*. In this section we first describe the key architectural elements and then introduce the four processes starting with the most generic cases.

3.1. Key components

3.1.1. Sensor system and sensory memory

All pattern recognition processes depend on the system's ability to capture data from the external world. Advanced biological organisms have many sensory modalities. In the following we focus on a generic modality that will be described at a high level of abstraction but with explanatory references to the olfactory system.

The sensor system (SS) acquires data from the external world, performs preliminary signal processing and outputs the primary input vector $\mathbf{x}''$. The receptor neurons in the olfactory epithelium together with the olfactory bulb would be the SS of the olfactory system.

It is assumed that the classifications are computed during a time interval T_{eval} that in the following will be called an evaluation cycle. The system includes a time-keeping function that triggers regular cycling of consecutive evaluation cycles. Sniffing cycles in the olfactory system is an example of similar behavior in biological organisms (Macrides, Eichenbaum, & Forbes, 1982).

The primary input vector will often fluctuate wildly e.g. because of the unpredictable density variations of chemical compounds in a plume of scent spreading in the wind. This makes sensory memory (SM) a crucial part of the system (see Fig. 1). The task of sensory memory is to take a snap-shot of the primary input vector at the beginning of each evaluation cycle and hold it unchanged over the evaluation cycle so that downstream systems can analyze a stable input. The input vector $\mathbf{x}$ is the output of sensory memory. Both $\mathbf{x}''$ and $\mathbf{x}$ is in the following modeled as real-valued vectors.

3.1.2. Competitive Queuing Memory

For implementing SVMs as neural networks we need a component for learning, storing and retrieving training examples (in particular support vectors) and the associated weights. Competitive Queuing Memory (CQM) is class of neural network algorithms that is ideal for this purpose. CQM learns and displays time sequences of output vectors. Bullock (2004) argues that the brain uses CQM-like modules for learning and generating the motor action sequences that underlie smooth skilful actions. In the following we will consider a high-level functional CQM model that is appropriate for the purpose of this paper.

A CQM consists of a memory layer and a choice layer. The memory layer stores a set $\{\{\mathbf{x}_1, y_1\}, \{\mathbf{x}_2, y_2\}, \ldots \{\mathbf{x}_m, y_m\}\}$ of patterns where each pattern is a training example. A weight α_i is associated with each pattern $\{\mathbf{x}_i, y_i\}$. The choice layer outputs one single pattern. When the CQM is triggered into action, the choice layer selects the pattern with the largest weight and displays it for a time interval that is proportional to the weight of the pattern. At the end of this interval the selected pattern is disabled and the pattern with the next highest weight is selected for display—again for a time that is proportional to the weight. The CQM will continue to operate in this manner until all patterns have been showed once and the display sequence stops. A new trigger signal resets all disabled patterns so that the time sequence can be traversed again.

Consider a CQM that holds SVM training examples and weights. As the CQM displays the stored sequence each training example

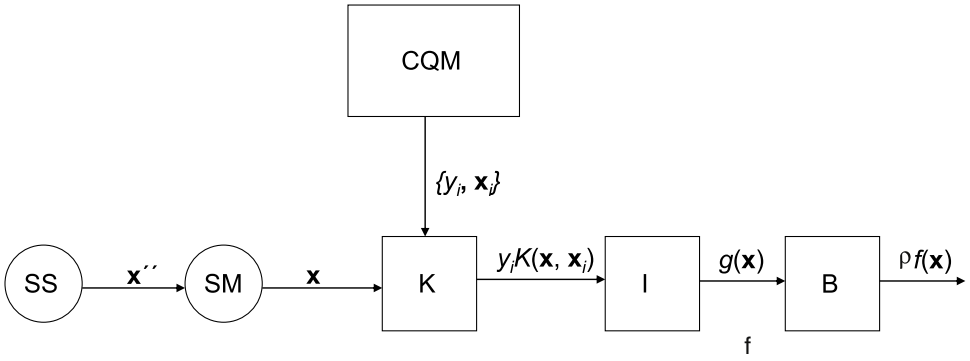

Fig. 1. Generic mechanism for computing SVM classifications in single-CQM neural network architectures. Symbols are explained in Section 3.2.1.

$\{\mathbf{x}_i, y_i\}$ is exhibited for a time period T_i that will be called the endurance time of the training example. The endurance time is proportional to the SVM weight according to,

$$T_i = \rho\alpha_i. \tag{16}$$

The endurance time is hence the physical representation of the SVM weight in the neural system. The total time for showing the stored time sequence of a CQM is the evaluation time $T_{eval} = \sum_{i=1}^{m} T_i$ where m is the number of patterns in the CQM. We further assume that the CQM has an interface for modifying the weights and for adding new patterns.

Although the present model is described at a functional level, detailed ANN implementations of CQM with similar functional features are described by Boardman and Bullock (1991), Bradski, Carpenter, and Grossberg (1994) Bullock and Rhodes (2003), Grossberg (1978), Houghton (1990) and Rhodes and Bullock (2002).

3.2. Single-CQM systems

We will first consider SVM-implementations with one single CQM that holds all the training examples of the classifier. The architecture and operation of the system are described by considering each of the main processes in turn. The processes are Classification, Learning training examples, Learning weights and Learning the bias factor.

In this section we will first introduce the basic operations of the systems by describing a simplified architecture and the processes for classification and learning training examples. These processes apply to all eight types of SVMs and all the different architectures to be discussed in the following.

The process for learning weights and the bias factor differ depending on SVM type and the architectural options. Gradient ascent weight learning rules for the various SVMs will first be discussed in general and then be applied to two different system concepts. Finally bias learning is described. At appropriate points we will discuss the complexity of the various systems from the point of view of biological implementation.

3.2.1. The classification process

Classifications are computed in the same manner in all of the eight SVM variants under consideration (see Fig. 1). The system performs one classification during each evaluation cycle. External inputs are captured by the sensor system (SS) and forwarded to the sensory memory (SM). The SM captures the input vector $\mathbf{x}$ and holds it unchanged for the duration of the evaluation cycle.

At the beginning of each evaluation cycle the CQM starts playing the stored sequence of training examples. The K-unit computes the SVM kernel function and outputs $y_iK(\mathbf{x}, \mathbf{x}_i)$ where the arguments to K are the input vector and the presently displayed training example from the CQM.

The unbiased classification function is calculated by temporal integration in the I-unit according to,

$$\int_{t_0}^{t_0+T_{eval}} y_iK(\mathbf{x}, \mathbf{x}_{i(t)})dt = \rho\sum_{i=1}^{m} y_i\alpha_iK(\mathbf{x}, \mathbf{x}_i) = \rho h(\mathbf{x}) = g(\mathbf{x}) \tag{17}$$

where t_0 is the start of the evaluation cycle. The integral is proportional to the sum in Eq. (1) since each training example is displayed for a time that is proportional to the associated SVM weight. The expression (17) will be referred to so frequently in the following that we give it a special name $g(\mathbf{x})$.

The B-unit adds a constant that is proportional to the bias in Eq. (1) and produces the final output $\rho f(\mathbf{x}) = \rho(h(\mathbf{x}) + b)$. It is inconsequential that the constant ρ from Eq. (16) multiples the output since only the sign of $\rho f(\mathbf{x})$ matters for classifications.

3.2.2. Learning new training examples

The system needs a mechanism for adapting to new environments by learning new training examples and thus extending the inventory of support vectors. For this purpose we assume that a Trainer selects suitable training examples from the input stream and supplies appropriate valence values (see Fig. 2).

Immediately after an evaluation cycle is an appropriate time for learning new training examples since the Trainer can use the result of the classification to judge if the input vector that just has been evaluated and still lingers in sensory memory is suitable as a new training example. Consider e.g. an animal that just has evaluated a poisonous speck as edible based on scent alone. Chewing on the morsel reveals a bitter taste thus enabling the Trainer to provide a correcting training example. The method for selecting training examples is not crucial for the system architecture but will be briefly discussed in Section 4.

3.2.3. Learning optimal weights

After learning new training examples, the CQM endurance times are not guaranteed to represent optimal SVM weights. In this section generic learning rules are first explored and then applied to two different system concepts. The learning rules will be expressed in terms of the innate quantities of the system notably the endurance times.

Generic gradient ascent learning

We will explore learning rules of the form,

$$\forall k : T_k \to T_k + \Delta T_k, \tag{18}$$

based on gradient ascent with respect to the objective function W and taking into account the applicable linear constraints. Endurance times are hence incremented according to,

$$\Delta\mathbf{T} = \gamma\rho\tilde{\mathbf{U}} \tag{19}$$

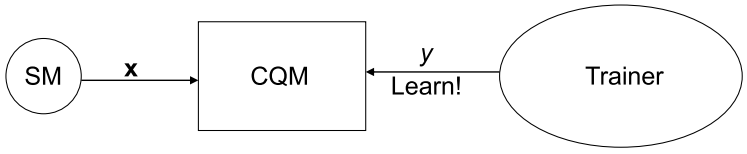

Fig. 2. Generic mechanism for learning new SVM training examples in CQM-based neural network architectures. The CQM receives a stream of input vectors **x** from sensory memory (SM). The Trainer provides the correct valence y and signals when the CQM shall imprint the input vector as a new training example.

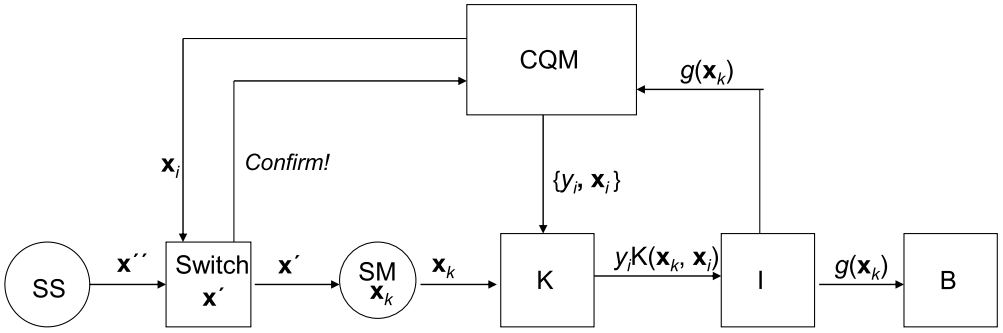

Fig. 3. Weight learning in the single-CQM outer-loop architecture. Symbols have the same meaning as in Fig. 1 or else are explained in Section 3.2.3. The Switch confirms when a pattern from the CQM is captured.

where γ is a dimensionless learning rate and $\tilde{\mathbf{U}}$ is the appropriate gradient. Bold symbols $\tilde{\mathbf{U}}$, $\mathbf{T}$, $\Delta\mathbf{T}$ etc. denote m-dimensional vectors.

The family of SVMs under consideration includes members with zero, one or two linear constraints. The gradient $\tilde{\mathbf{U}}$ takes different forms depending on the linear constraints. If there are no linear constraints we get,

$$\tilde{\mathbf{U}} = \mathbf{U} = grad_{\alpha}(W) = \left(\frac{\partial W}{\partial \alpha_1}, \frac{\partial W}{\partial \alpha_2}, \ldots, \frac{\partial W}{\partial \alpha_m}\right). \tag{20}$$

The constraints define hyperplanes in $\mathbf{T}$-space and the gradient ascent should follow the projection $\tilde{\mathbf{U}}$ of the gradient $\mathbf{U}$ on the constraint hyperplane. For one linear constraint we hence get,

$$\tilde{\mathbf{U}} = \mathbf{U} - (\mathbf{e}_C \cdot \mathbf{U})\mathbf{e}_C, \tag{21}$$

where $\mathbf{e}_C$ is a unit vector that is perpendicular to the constraint hyperplane.

For two linear constraints we write the gradient as,

$$\tilde{\mathbf{U}} = \mathbf{U} - (\mathbf{e}_A \cdot \mathbf{U})\mathbf{e}_A - (\mathbf{e}_B \cdot \mathbf{U})\mathbf{e}_B. \tag{22}$$

The unit vectors $\mathbf{e}_A$ and $\mathbf{e}_B$ define an orthonormal base spanning the subspace of the perpendicular unit vectors that identify the constraint hyperplanes. The unit vectors corresponding to the constraints (3) and (15) are $\mathbf{e}_y = \frac{1}{\sqrt{m}}(y_1, y_2, \ldots, y_m)$ and $\mathbf{e}_1 = \frac{1}{\sqrt{m}}(1, 1, \ldots, 1)$ respectively.

To simplify the notation in Tables 1 and 2 we define $g_k = \rho h(\mathbf{x}_k)$, $\hat{g} = \frac{1}{m}\sum_{k=1}^{m} g_k$ and $\hat{\hat{g}} = \frac{1}{m}\sum_{k=1}^{m} y_k g_k$ where h is defined in Eq. (5). The average valence $\hat{y} = \frac{1}{m}\sum_{k=1}^{m} y_k$ is also used in the tables.

The learning rules for the zero-bias SVMs are summarized in Table 1. No linear constraints applies to the Maximum margin, 1-norm and 2-norm SVMs so the learning rules are obtained from Eqs. (19) and (20). Applying the ν-SVM objective function Eq. (13) and the linear constraint Eqs. (15)–(21) produces the learning rule in Eq. (26). We note that Eq. (26) can be rewritten $\gamma\rho((\frac{1}{m}\sum_{i=1}^{m} M_i) - M_k)$. The learning rule strives therefore to equate the margin of the present example to the average margin. This will succeed for regular support vectors while the weights of trivial examples and outlier support vectors are driven to the extreme values $\alpha = 0$ and $\alpha = 1/m$ respectively. The non-linear constraints of each SVM have been expressed as constraints on the endurance times. Since SVM weights are represented by endurance times Eq. (2) is satisfied by design in all considered architectures.

Table 1
Learning rules for zero-bias SVMs.

SVM type	Learning rule		Non-linear constraints
Maximum margin	$\Delta T_k = \gamma(\rho - y_k g_k)$	(23)	None
1-norm	$\Delta T_k = \gamma(\rho - y_k g_k)$	(24)	$\forall k : T_k \leq T_{\max} = \rho C$
2-norm	$\Delta T_k = \gamma(\rho - y_k g_k - \frac{T_k}{C})$	(25)	None
ν	$\Delta T_k = \gamma(\hat{\hat{g}} - y_k g_k)$	(26)	$\forall k : T_k \leq T_{\max} = \rho/m$

Table 2 gives the learning rules for the biased SVMs where the linear constraint in Eq. (3) applies in all cases. Eqs. (19) and (21) have been employed for calculating the learning rules for the Maximum margin, 1-norm and 2-norm SVMs while Eqs. (19) and (22) together with both of the linear constraints (3) and (15) lead to the learning rule for the ν-SVM.

Outer-loop architecture

The outer-loop architecture (see Fig. 3) is our first example of a complete SVM implementation. We will therefore first introduce concepts that apply to weight learning in all CQM-based architectures. The basic idea is that training examples from the CQM replace external inputs. The system ignores hence external stimuli while learning. A long series of training examples are classified just as if they were external inputs. The CQM endurance times are modified according the learning rules of Section 3.2.3 using feedback from the classification process.

The system has two main modes that can be compared to the waking state and the sleeping state of a living creature. Classification and learning of new training examples are performed in the *waking state* while processing sensor data. Weight and bias optimization is performed in the *sleeping state* in which sensory data is replaced by regurgitated training examples from the CQM. The terms waking state and sleeping state are convenient mnemonic devices but are not intended to imply that the system implements all aspects of wakefulness and sleep in biological organisms. The idea that sleep is related to memory consolidation has, however, been discussed since ancient times (Quintilianus, 95).

To enable switching between the two modes we introduce a new system component—the Switch (see Fig. 3). The CQM copies the stream of training examples to the Switch during each evaluation cycle. In the sleeping state the Switch captures an example

Table 2
Learning rules for biased SVMs.

SVM type	Learning rule		Non-linear constraints
Maximum margin	$\Delta T_k = \gamma(\rho - \rho\hat{y}y_k + y_k(\hat{g} - g_k))$	(27)	None
1-norm	$\Delta T_k = \gamma(\rho - \rho\hat{y}y_k + y_k(\hat{g} - g_k))$	(28)	$\forall k : T_k \leq T_{\max} = \rho C$
2-norm	$\Delta T_k = \gamma(\rho - \rho\hat{y}y_k + y_k(\hat{g} - g_k) - T_k/C)$	(29)	None
ν	$\Delta T_k = \gamma(\hat{\hat{g}} - y_k g_k - \frac{y_k - \hat{y}}{1-\hat{y}^2}(\hat{y}\hat{\hat{g}} - \hat{g}))$	(30)	$\forall k : T_k \leq T_{\max} = \rho/m$

vector $\mathbf{x}'$ according to some policy that ensures that all examples have some probability of being selected. In the following we shall assume that the Switch captures a new example each evaluation cycle at a time given by a uniform probability distribution. The Switch is transparent to the sensor signal $\mathbf{x}''$ in the waking state and opaque in the sleeping state. This makes the system in Fig. 3 consistent with the system displayed in Fig. 1.

The SM works precisely as in the classification process. At the beginning of each evaluation cycle it locks on the vector $\mathbf{x}''$ that is provided by the Switch and outputs the trapped vector for the duration of the evaluation cycle. The vector that is held by the SM is labeled $\mathbf{x}_k$ in Fig. 3 and in the following explanations.

We know turn to the specific features of the outer-loop architecture (see Fig. 3). It is based on the notion that the output $g(\mathbf{x}_k)$ of the classification is feed back to the CQM and used as input to the weight learning process. It is the weight of the example $\mathbf{x}_k$ that is held in sensory memory that is the subject of modification. One endurance time T_k is thus modified at the end of each evaluation cycle.

We need a mechanism for informing the CQM about which example that is due to have its weight updated. In the outer-loop architecture the Switch therefore sends a confirmation signal causing the CQM to activate a pointer to the training example that was selected by the Switch. Note that the CQM must manage two pointers: one marking the weight to be updated after the present evaluation cycle and the other marking the weight to be updated after the next evaluation cycle. This rather complex mechanism makes the outer-loop architecture somewhat less credible for biological modeling.

Zero-bias outer-loop system

Applying the outer-loop architecture to the zero-bias SVM we find that the identical learning rules (Eqs. (23) and (24)) that apply to the Maximum margin and the 1-norm SVMs are readily implementable since g_k is feed back to the CQM and y_k is in CQM memory. Note that the parameter ρ is defined only by the learning rule since there is nothing else in the system that constrains its value.

The ν-SVM learning rule (Eq. (26)) also uses the feedback quantity but the CQM must in addition compute the average of $y_k g_k$ which adds complexity but is quite feasible in a neural network context.

The 2-norm learning rule (Eq. (25)) is somewhat more problematic since it assumes that the CQM can access the endurance time of the example that is subject to update. Since the learning rule is applied at the end of the evaluation cycle this would require that endurance times are stored in a separate register which is possible but rather contrived for a neural network.

The 1-norm SVM and the ν-SVM have non-linear constraints requiring that all endurance times fall below a maximum value. It is quite conceivable that biological neural neurons are depleted by persistent firing so that memory patterns only can be sustained for a maximum time (see Jändel (2010a) for a discussion).

Biased outer-loop system

Applying the outer-loop architecture to biased SVMs we note that all learning rules in Table 2 are seriously complicated by including the constraint (3). The dual linear constraints of the ν-SVM give a particularly complex algebraic form of the learning rule. In addition to the hurdles previously mentioned the CQM must keep track of the average valence $\hat{y}$ and the average of g_k. Since the CQM has access to g_k and all valences it is, however, feasible to form the required aggregates by neural computation.

Stability of linear constraints in the outer-loop architecture

The outer-loop architecture, as described in the preceding sections, works well if there are no linear constraints in the SVM definition. Zero-bias maximum margin, 1-norm and 2-norm SVMs will hence, for a sufficiently small learning rate, find the optimal weights within whatever accuracy that is called for.

The other SVMs have at least one linear constraint. All biased SVMs must fulfill the linear constraint (3) and the ν-SVM must conserve the sum of weights. The linear constraints would be satisfied if endurance times were incremented simultaneously according to $\mathbf{T} \rightarrow \mathbf{T} + \Delta\mathbf{T}$ where $\Delta\mathbf{T}$ is given by a learning rule in Table 1 or Table 2. The outer-loop architecture updates, however, the endurance times one by one and it is obvious that this may violate linear constraints. Starting from initial weights fulfilling Eq. (3) it can be shown that outer-loop learning causes the state to drift away from the α-space hyperplanes given by the linear constraints.

The constraint (3) that applies to all biased SVMs is hence very difficult to accommodate in the present architecture. The Sequential Minimal Optimization (SMO) algorithm handles this problem by selecting a pair of weights for modification and constraining the increments to cancel each other in Eq. (3) (Platt, 1998). Outer-loop systems update, however, weights piecewise and need a different type of patch. Intermediate states in the learning process could be allowed to violate the constraints e.g. by using the relaxed condition $-\mu \leq \sum_{i=1}^{m} y_i\alpha_i \leq \mu$ and successively decrease μ to zero as the computation converges to optimal weights. This and similar approaches are, however, rather contrived as biological models and will not be considered further.

It is easier to handle the ν-SVM constraint (15) without giving up biological realism. The constraint would be enforced by a CQM that distributes a constant amount of activation over the stored patterns. In the outer-loop architecture this would again require the complexity of gradually enforcing a soft constraint $\nu - \mu \leq \sum_{i=1}^{m} \alpha_i \leq \nu + \mu$. The inner-loop architecture offers, as we shall see, a much more natural implementation of zero-bias ν-SVM.

Inner-loop architecture

In the inner-loop architecture learning feedback is provided by the K-unit and weights are updated each time the CQM switches output pattern (see Fig. 4). This removes the need for feedback confirmation from the Switch to the CQM. Pointers to weights that are due for update are also not required. Apart from these simplifications the architecture and operation are similar to the outer-loop system.

During each evaluation cycle, the CQM presents the sequence of training examples one by one. Each example becomes the output of the CQM during one presentation interval. Learning rules are applied at the end of each presentation interval. At a given presentation interval the only pattern that has a special status is the one that is on display. Learning rules can hence apply either to the endurance time of the presently displayed example or indiscriminately to all endurance times. The former individual update is denominated $\delta T^{(ind)}$ and the latter collective update is called $\delta T^{(coll)}$. More complex update schemes could be devised but we strive for simplicity that facilitates biological implementation.

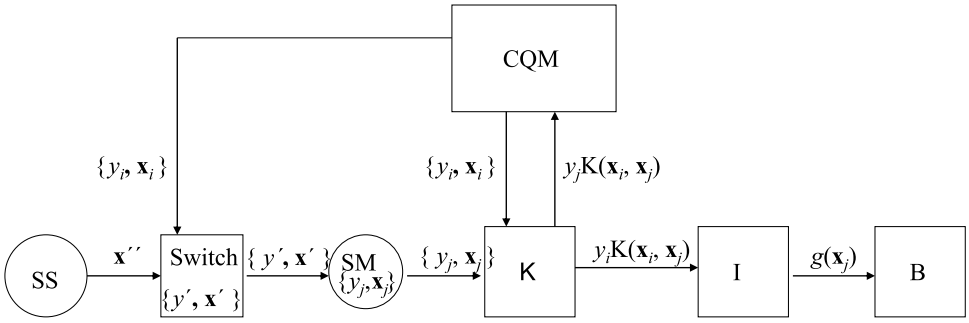

Fig. 4. Weight learning in the single-CQM inner-loop architecture. Symbols have the same meaning as in Fig. 3.

Table 3
Learning rules for zero-bias SVMs in the inner-loop architecture. The index j refers to the training example that is held in sensory memory while the index i indicates the presently displayed training example in the CQM. $\delta T^{(coll)} = 0$ if not defined explicitly.

SVM type	Learning rule		Non-linear constraints
Maximum margin	$\delta T^{(ind)} = \gamma(\rho - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))$	(33)	None
1-norm	$\delta T^{(ind)} = \gamma(\rho - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))$	(34)	$\forall k : T_k \leq T_{\max} = \rho C$
2-norm	$\delta T^{(ind)} = \gamma(\rho - \frac{T_i}{C} - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))$	(35)	None
ν	$\delta T^{(ind)} = -\gamma T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$	(36)	$\forall k : T_k \leq T_{\max} = \rho/m$
	$\delta T^{(coll)} = \frac{1}{m} \gamma T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$	(37)	

During an evaluation cycle each training example is subject to the individual update once and the collective update m times. The average increment for training example k during an evaluation cycle is hence

$$\langle \Delta T_k \rangle = \sum_{j=1}^{m} p_j \sum_{i=1}^{m} (\delta T^{(ind)} \delta_{ik} + \delta T^{(coll)}), \tag{31}$$

where p_j is the probability of having the example $(\mathbf{x}_j, y_j)$ in sensory memory. In the following we apply the convention that the example held in sensory memory has index j and that the example presented by the CQM has index i. The increments $\delta T^{(ind)}$ and $\delta T^{(coll)}$ may depend on i and j. In (31) we apply the probability distribution,

$$p_j = \frac{T_j}{T_{eval}}, \tag{32}$$

where $T_{eval} = \sum_{i=1}^{m} T_i$. This distribution would arise if the Switch captures examples from the CQM with a uniform random distribution over time. To learn optimal weights, $\delta T^{(ind)}$ and $\delta T^{(coll)}$ should be selected so that $\langle \Delta T_k \rangle$ is proportional to the learning rules of Tables 1 and 2.

Zero-bias inner-loop architecture

Table 3 provides learning rules for inner-loop zero-bias SVMs. Note that only information that is available to the CQM according to Fig. 4 is used in the table and that the otherwise free parameter ρ is defined by the learning rules.

It is straightforward to demonstrate that Table 3 and Eq. (32) applied to Eq. (31) produces the learning rules of Table 1. For the maximum margin case we get e.g.

$$\begin{aligned}\langle \Delta T_k \rangle &= \sum_{j=1}^{m} p_j (\gamma(\rho - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))) \\ &= \gamma\rho \left(1 - y_k \sum_{j=1}^{m} y_j \alpha_j K(\mathbf{x}_k, \mathbf{x}_j)\right) = \gamma(\rho - y_k g_k).\end{aligned}$$

Inserting Eqs. (36) and (37) in Eq. (31) reproduces the ν-SVM learning rule (26) of Table 1,

$$\begin{aligned}\langle \Delta T_k \rangle &= \gamma T_{tot} \sum_{j=1}^{m} p_j \left(- y_k y_j K(\mathbf{x}_k, \mathbf{x}_j) \right. \\ &\quad \left. + \frac{1}{m} \sum_{i=1}^{m} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right) = \gamma(\hat{\hat{g}} - y_k g_k).\end{aligned}$$

Note that the linear constraint (15) now is precisely satisfied in all ν-SVM iterations. The sum of weights is conserved since any increment, according to Eq. (36) is exactly compensated by opposite sign increments distributed over all examples according to Eq. (37).

Biased inner-loop architecture

Table 4 provides learning rules for inner-loop biased SVMs. Again we note that adding another linear constraint generates more complex learning rules. The system now has to keep track of the aggregated quantities $\hat{g}$, $\hat{\hat{g}}$ and $\hat{y}$. This increases the complexity of the system but is otherwise quite feasible in a bio-realistic neural network. Valences are available internally in the CQM and the other aggregates can be estimated by averaging over the feedback input.

As in the previous section it is easy to show that the inner-loop learning rules and Eq. (32) applied to Eq. (31) produce the biased SVM learning rules of Table 2. Note that the rules of Tables 3 and 4 are not unique. Constants can e.g. be included in either of $\delta T^{(ind)}$ and $\delta T^{(coll)}$.

Stability of linear constraints in the inner-loop architecture

The conclusion about the stability of linear constraints in the inner-loop architecture is similar to the outer-loop case. The zero-bias maximum margin, 1-norm and 2-norm SVMs have no linear constraints and can thus be realized according to Fig. 4 without stability concerns.

Zero-bias ν-SVM offer a pleasant surprise since the inner-loop learning rules turn out to conserve the linear constraint locally in each iteration. Stable zero-bias SVMs are hence supported by the inner-loop architecture. This is one of the reasons why zero-bias ν-SVM (using oscillating associative memory) has been chosen as a prime candidate for biological modeling (Jändel, 2009, 2010a, 2011).

Biased inner-loop SVMs share the same stability concerns as biased outer-loop SVMs since updating endurance times sequentially means that the solution drifts away from the constraint hyperplane.

3.2.4. Learning bias

The B-module is needed only for biased SVMs where it works in the same way both for outer-loop and inner-loop systems. It has a classification mode in the waking state and a learning mode in the sleeping state. In the classification mode it will just output the

Table 4
Weight learning rules for biased SVMs in the inner-loop architecture. The index j refers to the training example that is held in sensory memory while the index i indicates the presently displayed training example in the CQM. $\delta T^{(coll)} = 0$ if not defined explicitly.

SVM type	Learning rule		Non-linear constraints
Maximum margin	$\delta T^{(ind)} = \gamma(\rho - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + y_i(\hat{g} - \rho\hat{y}))$	(38)	None
1-norm	$\delta T^{(ind)} = \gamma(\rho - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + y_i(\hat{g} - \rho\hat{y}))$	(39)	$\forall k : T_k \leq T_{\max} = \rho C$
2-norm	$\delta T^{(ind)} = \gamma(\rho - \frac{T_i}{C} - T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + y_i(\hat{g} - \rho\hat{y}))$	(40)	None
ν	$\delta T^{(ind)} = -\gamma(T_{eval} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + y_i(\frac{\hat{y}\hat{g} - \hat{g}}{1-\hat{y}^2}))$	(41)	$\forall k : T_k \leq T_{\max} = \rho/m$
	$\delta T^{(coll)} = \frac{\gamma y_j T_{eval} K(\mathbf{x}_i, \mathbf{x}_j)}{m(1-\hat{y}^2)}(y_i - \hat{y})$	(42)	

input incremented by the constant ρb. In the learning mode it must find the correct value of ρb.

Algorithms for computing the bias factor are provided in Section 2. Maximum margin, 1-norm and ν-SVM share the same algorithm,

$$\rho b = -\frac{1}{2}(\tilde{g}_+ + \tilde{g}_-), \tag{43}$$

where $\tilde{g}_* = \rho\tilde{h}_*$, $\tilde{h}_*$ is given by Eq. (7) and the symbol $*$ refers either to $+$ or $-$. The main learning task of the B-unit is hence to identify the regular support vectors and compute the aggregated quantities of Eq. (43).

Once the weight learning process has converged only support vectors will be displayed for classification in the SM. Endurance times of trivial examples vanish at the optimum. Support vectors can, however, be regular or outliers. The B-unit performs therefore a clustering operation with respect to the inputs $g(\mathbf{x}_j)$. This will reveal two main clusters marking the values of $\tilde{g}_+$ and $\tilde{g}_-$ with outliers scattered around the centers. There is no need for identifying the valence of the clusters since Eq. (43) requires only the sum of cluster centers. We note that neural networks readily perform clustering (Kohonen, 2001) and that the method is insensitive to filtering out some regular support vectors and to including some trivial examples. Weight learning must therefore not converge completely before the method can be applied.

Finally, for the 2-norm case the bias is computed according to,

$$\rho b = -\frac{1}{2}\left(\tilde{g}_+ + \tilde{g}_- + \frac{\tilde{T}_+ - \tilde{T}_-}{C}\right) \tag{44}$$

where $\tilde{T}_+$ and $\tilde{T}_-$ are the average endurance times of positive valence and negative valence support vectors respectively. The aggregates $\hat{g}_+$ and $\hat{g}_-$ are obtained with clustering as before but computing $\tilde{T}_+$ and $\tilde{T}_-$ requires knowledge of the endurance time of the training example that presently is held in the SM. Any system that provides that information to the B-unit appears to be quite awkward as a neural network architecture.

3.2.5. Conclusions on single-CQM systems

Table 5 summarizes advantages and disadvantages of the single-CQM SVMs that have been discussed so far. We note that only some of the zero-bias solutions are stable and thus credible for biological modeling.

3.3. Bisymmetric systems

Biased SVMs have a superior generalization ability but hitherto we have found no implementations that appear promising for modeling pattern recognition in the brain. The problem is that there is no innate mechanism for enforcing the linear constraint of Eq. (3). All biased single-CQM models require complex and implausible patches for imposing the constraint. This is in sharp contrast to how the linear constraint (15) is built into the structure of the inner-loop ν-SVM resulting in a simple learning rule and a credible biological model. We would like to find an architecture where Eq. (3) likewise is integrated seamlessly.

Table 5
Advantages and disadvantages of single-CQM implementations of SVM algorithms.

SVM type	Comment
Zero-bias: reduced generalization power	
Maximum margin	Outer- and inner-loop architecture stable Brittle for noisy data. Weights can increase without limit
1-norm	Outer- and inner-loop architecture stable
2-norm	Outer- and inner-loop architecture stable Requires endurance time register in the CQM
ν	Outer-loop architecture not stable. Complex learning rule Inner-loop architecture stable. Simple learning rule
Biased: full generalization power	
Maximum margin	Outer- and inner-loop architecture not stable Brittle for noisy data. Weights can increase without limit Bias learning simple
1-norm	Outer- and inner-loop architecture not stable Bias learning simple
2-norm	Outer- and inner-loop architecture not stable Requires endurance time register in CQM Bias learning complex
ν	Outer- and inner-loop architecture not stable. Complex learning rules Bias learning simple

As a prelude we reconsider the form of the linear constraints using the definition

$$T^*_{tot} = \sum_{i=1}^{m} \delta_*(y_i) T_i, \tag{45}$$

where the symbol $*$ takes the values $+$ or $-$. Expressing the constraint in terms of endurance times Eq. (3) can now be written,

$$T^+_{tot} = T^-_{tot}, \tag{46}$$

where T^+_{tot} and T^-_{tot} are the sum of endurance times for positive and negative valence examples respectively. Similarly the linear constraints (3) and (15) together is expressed as,

$$T^+_{tot} = T^-_{tot} = \frac{\rho\nu}{2}. \tag{47}$$

A structure that inherently includes the linear constraints should have separate CQMs for examples with positive and negative valence respectively. Hence we require that CQM^+ holds all the positive valence examples and CQM^- holds all the negative valence examples. It is further required that both CQMs display the complete inventory of examples in one evaluation cycle so that $T^+_{tot} = T^-_{tot} = T_{eval}$. According to Eq. (47) this system automatically fulfils both of the linear constraints. Bisymmetry is ubiquitous in nature so this concept is not unnatural from a biological point of view.

It is easy to show that the bisymmetric architecture is of no advantage in most of the cases that we have considered. It increases

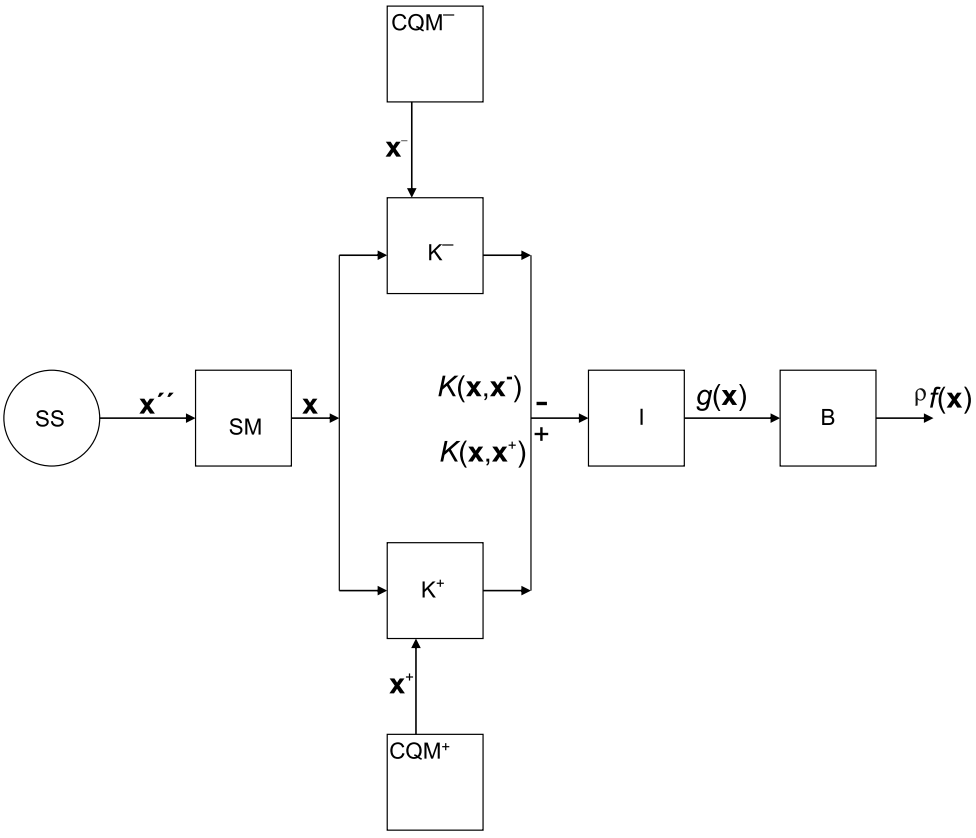

Fig. 5. Generic mechanism for computing SVM classifications in bisymmetric CQM-based neural network architectures. Symbols have the same meaning as in Fig. 1 or are else explained in Section 3.3.1.

architectural complexity but gives no benefit to zero-bias systems where Eq. (3) does not apply and where Eq. (15) anyway is embedded in the ν-SVM inner-loop architecture. Requiring that T^+_{tot} and T^-_{tot} are constant adds a new constraint to biased maximum margin, 1-norm and 2-norm SVMs so the bisymmetric architecture is not a valid implementation of these algorithms. Bisymmetry is hence only applicable to the biased ν-SVM. This section presents therefore the details of a bisymmetric biased ν-SVM by describing the same three processes as in Section 3.2. Bias learning in the bisymmetric architecture is conducted just as described in Section 3.2.4.

3.3.1. The classification process

Fig. 5 demonstrates how classifications are computed in a slight simplification of the full bisymmetric architecture. The sensory memory captures a snap-shot of the external input and holds it stable for the duration of an evaluation cycle. The CQM units play synchronously the sequence of stored examples. The stream of positive valence training examples $\mathbf{x}^+$ from CQM$^+$ merges with the input vector in the K^+-unit that outputs the kernel function $K(\mathbf{x}, \mathbf{x}^+)$. The CQM$^-$ and the K^--unit mirror this behavior to produce $K(\mathbf{x}, \mathbf{x}^-)$. There is no need to signal the valence of the examples explicitly since valence is implicit in the source. The I-unit computes,

$$\int_{t_0}^{t_0+T_{eval}} (K(\mathbf{x}, \mathbf{x}^+) - K(\mathbf{x}, \mathbf{x}^-))dt = \rho \sum_{i=1}^{m} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) = g(\mathbf{x}), \quad (48)$$

and the B-unit adds ρb to produce the final output which is proportional to the classification function (Eq. (1)) of a biased support vector machine.

3.3.2. Learning new training examples

Learning new examples works just as described in Section 3.2.2. The only difference being that The Trainer keeps track of the valence of new training examples and adds them to the appropriate CQM.

3.3.3. Learning weights

The gradient ascent learning rules of Section 3.2.3 apply also to the bisymmetric case but should be expressed in quantities that can be computed locally in the bisymmetric architecture. For that purpose we define $\hat{g}_* = \frac{1}{m_*} \sum_{i=1}^{m} \delta_*(y_i) g_i$ and $\hat{\hat{g}}_* = \frac{1}{m_*} \sum_{i=1}^{m} \delta_*(y_i) y_i g_i$ where $g_k = \rho h(\mathbf{x}_k)$, $*$ can take the values $+$ or $-$ as before and m_+ and m_- denote the number of positive and negative valence examples respectively. It is straightforward to express all the learning rules of Tables 1 and 2 using the new aggregates. We focus, however, on the biased ν-SVM. Writing Eq. (30) as two separate rules applying to the CQM$^+$ and the CQM$^-$ respectively we get,

$$\Delta T^*_k = \gamma(\hat{\hat{g}}_* - y_k g_k) \quad (49)$$

where ΔT^+_k applies to the CQM$^+$ and ΔT^-_k applies to the CQM$^-$. The non-linear constraint $\forall k : T_k \leq T_{\max} = \rho/m$ holds for all endurance times. The quite complex expression in Eq. (30) simplifies hence in the bisymmetric architecture to local learning rules of the same form as the corresponding zero-bias learning rule of Eq. (26). Such simplifications do not emerge for any of the other SVM types.

Outer-loop architecture

The overall operation of the bisymmetric system is very similar to the single-CQM case (see Fig. 6). Weight optimization is performed in the "sleeping state" where internally generated training examples replace external inputs. The operation of the Switch in

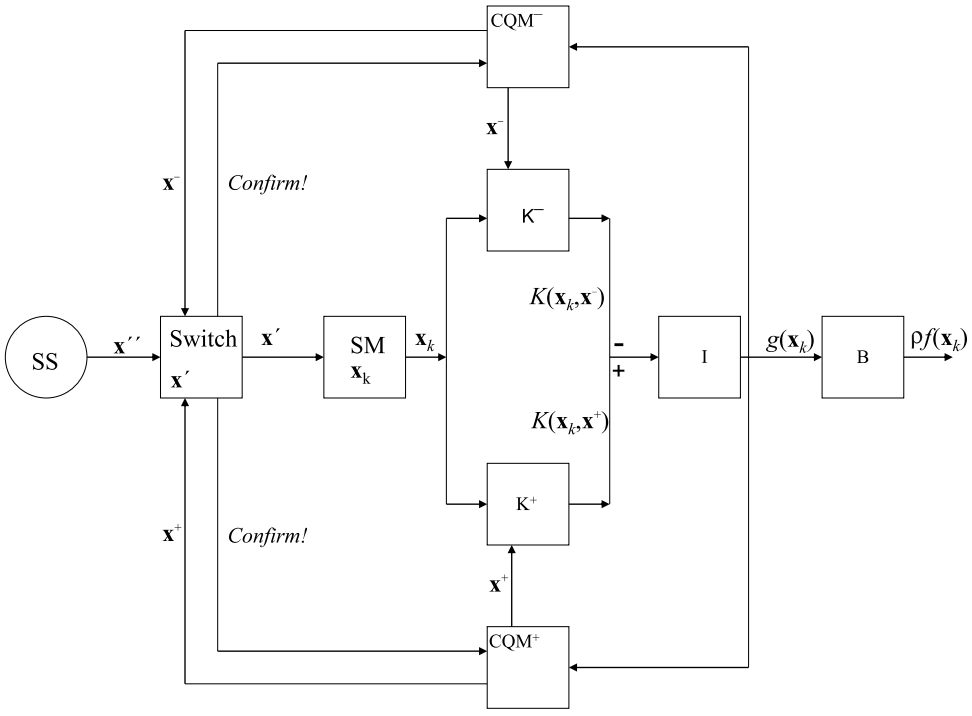

Fig. 6. Weight learning in the bisymmetric outer-loop architecture. Symbols have the same meaning as in Fig. 5 or are else explained in Section 3.3.3. The switch confirms to the source CQM when a pattern is captured.

the sleeping state is slightly different compared to the single-CQM case. Both CQM units send a stream of positive valence $\mathbf{x}^+$ and negative valence $\mathbf{x}^-$ examples respectively to the Switch. The Switch selects one example vector $\mathbf{x}'$ each evaluation cycle. The probability for selecting an example is proportional to the endurance time of the example. The Switch confirms its choice to the source of the selected example so that the parent CQM can mark the chosen example for weight update in the next evaluation cycle. The SM locks on the output of the Switch at the beginning of each evaluation cycle and holds the trapped example $\mathbf{x}_k$ stable for the duration of the cycle.

Each CQM operates just as a single CQM in the corresponding zero-bias outer-loop case. The output $g(\mathbf{x}_k)$ of the integrator is feed back to both of the CQMs and is used to update the endurance time T_k of the selected example according to Eq. (49). Just one endurance time in one CQM is hence modified at the end of each evaluation cycle. Note that the aggregates $\hat{\hat{g}}_+$ and $\hat{\hat{g}}_-$ respectively are just the average of the active feedback signal for each CQM (with the appropriate sign).

Biased ν-SVM weight learning is greatly simplified in the bisymmetric outer-loop case compared to the corresponding single-CQM architecture. The linear constraints are, however still violated as updates are applied sequentially and the stability issues of the single-CQM systems apply in equal measure.

Inner-loop architecture

The bisymmetric inner-loop architecture (see Fig. 7) is quite similar to the outer-loop case but weight learning feedback from the K-units connects directly to the associated CQM (Jändel, 2010b). The confirmation signal from the Switch (see Fig. 6) is not needed but the system must carry information about the valence of the example that is held in sensory memory.

Inner-loop learning works otherwise as described in Section 3.2.3 with each CQM in the duo mirroring the operation of the single CQM in Section 3.2.3. Both CQMs work hence independently each updating the endurance time of the presently displayed example at the end of each presentation period. Two endurance times are thus modified, according to Eq. (31), for each of the m presentation periods in each evaluation cycle.

Both CQMs apply a slightly modified version of Eqs. (36)–(37),

$$\delta T^{(ind)} = -\gamma T_{eval} y^* y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{50}$$

$$\delta T^{(coll)} = \frac{1}{m_*} \gamma T_{eval} y^* y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{51}$$

where y^* is $+1$ or -1 for CQM$^+$ and CQM$^-$ respectively while $y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is the feedback signal from the K-unit and the nonlinear constraint $\forall k : T_k \leq T_{\max} = \rho/m$ is applied. Again we note that the biased bisymmetric system locally works as the corresponding zero-bias single-CQM system. The match between Eqs. (50)–(51) and Eq. (49) is demonstrated using the same method as in Section 3.2.3.

3.3.4. Conclusions on the bisymmetric architecture

The bisymmetric ν-SVM fulfils our goal of expressing both of the linear boundary conditions as innate constraints of the architecture. Each CQM conserves the local sum of endurance times. The constraint (15) is therefore fulfilled. The sum of positive valence endurance times equals the sum of negative valence endurance times thus satisfying the linear constraint (3). In the inner-loop bisymmetric ν-SVM the constraints are applied in each iteration of the learning rules. The system is therefore stable with respect to the linear constraints and employs also simple learning rules that readily could be implemented in the brain.

4. Discussion and conclusions

For the ultimate purpose of modeling pattern recognition with one-shot learning in living creatures we have investigated CQM-based neural network architectures for eight different SVM algorithms. Eight different architectural options have been considered for each algorithm. Most of the 64 investigated configurations are, however, deemed to be unsuitable. A major problem is that many

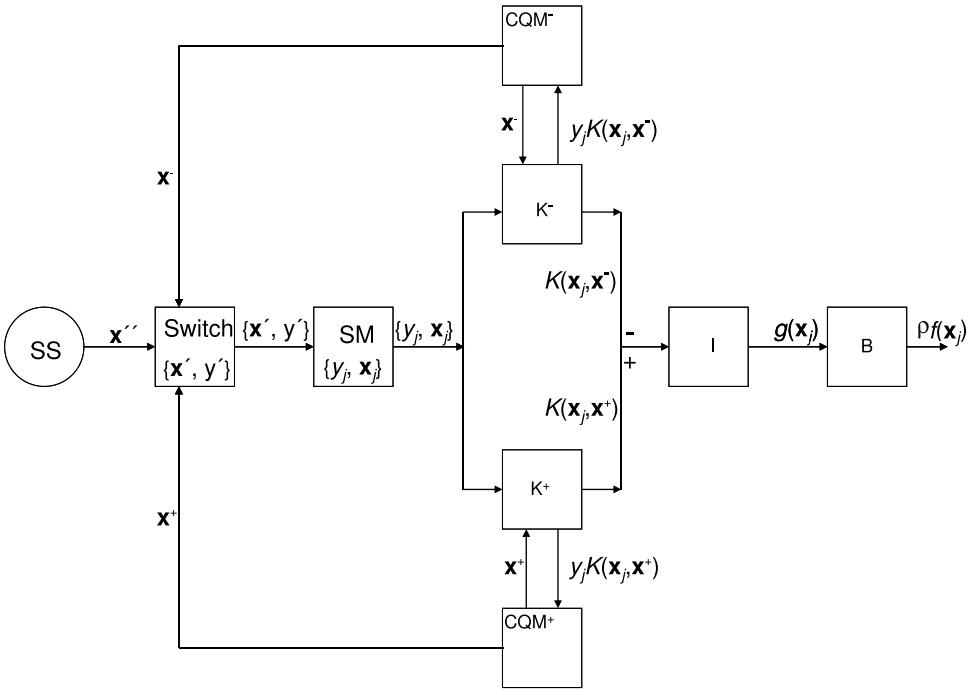

Fig. 7. Weight learning in the bisymmetric inner-loop architecture. Symbols have the same meaning as in Fig. 6.

configurations cannot handle linear constraints in the SVM algorithms. Complex auxiliary machinery with poor biological credibility would be required for implementing the constraints. For the single-CQM architecture we note in Table 5 that all biased solutions are unstable with respect to the linear constraint (3). Only zero-bias models with reduced generalization ability remain. The maximum margin algorithms are excluded since they cannot handle the noisy inputs that are expected in a natural context. Zero-bias 1-norm SVMs lack a linear constraint so both the outer- and inner-loop variants are feasible. Table 5 further notes that 2-norm learning rules (35) and (40) require a separate register of endurance times that appears to be hard to implement in neural networks. The outer-loop zero-bias ν-SVM cannot handle the linear constraint (15) but the inner-loop ν-SVM learning rules (35) and (37) enforce Eq. (15) in each iteration. Bisymmetric models cause only additional complexity in the zero-bias case but provide a unique opportunity to build a simple and elegant biased model. This is, however, feasible only for the inner-loop ν-SVM where architectural constraints and particularly benign learning rules (50) and (51) seamlessly implement both linear constraints. Hence only four promising options for biological modeling emerge:

– Single-CQM zero-bias inner-loop ν-SVM
– Single-CQM zero-bias outer-loop 1-norm SVM
– Single-CQM zero-bias inner-loop 1-norm SVM
– Bisymmetric biased inner-loop ν-SVM.

Note that the four preferred neural network architectures that we have found are not competitive as computer implementations of SVM and are only proposed for the purpose of biological modeling. With regard to the biological feasibility of the four favored models we will argue that they are built from components that realistically could be implemented in living tissue. Sensor systems and sensory memory are well known features of the brain. The routing function in the Switch can be understood as inhibition/activation of alternative neural pathways. Similar routing functions are e.g. found in the thalamic relay matrix (Sherman & Guillery, 2006).

Bullock (2004) reviews substantial evidence that the brain employs CQM for learning fluent action sequences. The four models that we have selected are characterized by particularly simple learning rules (Eqs. (24), (34), (36), (37), (50), (51)) that credibly could be realized in a biological CQM. The kernel module implements a multivariate function and it is known that feed-forward three-layer neural networks can implement any continuous multivariate function with arbitrary accuracy (Cybenko, 1989). SVM Kernels must be positive definite which poses an additional requirement on the biological substrate but even kernels that violate positive definiteness may work well in practice (Ong, Mary, Canu, & Smola, 2004). Temporal integration is approximately realized in living neurons by temporal summation (Johnston & Wu, 1995). The clustering operation that is central to the bias computation can be performed by neural networks in the form of self-organizing maps (Kohonen, 2001).

The zero-bias inner-loop 1-norm SVM is very similar to corresponding ν-SVM—a constant term is the only difference in the learning rules (compare Eqs. (34)–(36)). This slight difference causes, however, significantly different behavior. The duration of an evaluation cycle is constant in the ν-SVM but the 1-norm evaluation time differs depending on the current selection of support vectors. It is, however, advantageous for an organism that the evaluation time is kept within a narrow range so that pattern recognition latency is predictable for higher-levels systems. Rodent olfactory sniffing cycles are e.g. in phase with the limbic θ rhythm (Macrides et al., 1982). Hence ν-SVM was selected as a prime candidate for biological modeling (Jändel, 2010a).

The architectures that we have analyzed do not depend on any particular strategy for selecting training examples from the stream of sensory data. Any organism using SVM-based pattern recognition would, however, need to manage the cache of stored training examples carefully to avoid performance degradations caused by an ever increasing load of data. Jändel (2010a) suggest that only misclassified examples should be captured into the CQM and trivial examples (with zero weight) after a suitable period of grace should be deleted. This simple policy ensures that only good support vector candidates are selected for storage and that eventually only support vectors remain in memory.

A zero-bias inner-loop ν-SVM using randomly oscillating associative memory rather than CQM has been discussed extensively in Jändel (2010a). This approach supports one-shot learning and

models qualitatively important aspects of the structure and dynamics of the olfactory system. Bursts in the thalamocortical system have speculatively been explained, within the framework of the ν-SVM model, as a signature of a thalamic pattern recognition mode (Jändel, 2009). Replacing associative memory with CQM preserves all functional features of the model and also the match to the anatomy and dynamics of the olfactory system. Jändel (2010a) can therefore be read as an in-depth discussion of the biological significance of single-CQM zero-bias inner-loop ν-SVMs.

Using CQM rather than associative memory solves a serious problem that was discussed in Jändel (2010a). Temporal summation in a given evaluation cycle converges rather slowly if support vectors are presented randomly by an oscillating associative memory. This means that the maximum number of support vectors in a given pattern recognition module is about ten which is an uncomfortably small number given the complexity of pattern recognition in nature. A CQM presents all support vectors in sequence during the evaluation cycle which guarantees precise computation of the classification function. This removes the convergence problem and allows for applying 50–100 support vectors in realistic biological models.

It is intriguing that the same neural structure that is thought to be vital for learning action sequences (Bullock, 2004) also can be the foundation of trainable pattern recognition. Speculatively it is thus proposed that advanced pattern recognition with one-shot learning first evolved as an adaption of motor systems. As bisymmetry is common in organisms it is conceivable that bisymmetric motor modules provided a rather direct evolutionary path to the bisymmetric biased inner-loop ν-SVM (Jändel, 2011).

This paper does not provide any numerical experiments because theoretical arguments in Section 3 demonstrate that the four biologically relevant architectures fully implement the associated SVM algorithms as defined in Section 2. The classification function (1) is precisely implemented, the bias parameter is correctly computed and SVM weights are learnt according to gradient ascent with respect to the proper objective function under the appropriate boundary conditions. Theoretical reasoning has likewise demonstrated that the majority of the studied architectures are not credible biological models for support vector machines. The relevant numerical experiments should study biologically realistic implementations of the four preferred models where the quintessential components that are assumed in Section 3.1 are replaced by systems of simulated bio-realistic neurons. In future research we would like to explore whether the promising architectural models that we have found will provide sufficiently good approximations of support vector machines in detailed and biologically realistic neural network simulations. A key issue in such experiments is to understand how much the resulting classifiers can deviate from ideal support vector machine algorithms and still be sufficiently useful for low-level perception.

Acknowledgment

This work was supported by the Swedish Foundation for Strategic Research.

References

Anguita, D., & Boni, A. (2002). Improved neural network for SVM learning. *IEEE Transactions on Neural Networks*, *13*, 1243–1244.
Anguita, D., & Boni, A. (2003). Neural network learning for analog VLSI implementations of support vector machines: a survey. *Neurocomputing*, *55*, 265–283.
Anguita, D., Ridella, S., & Rovetta, S. (1998). Circuital implementation of support vector machines. *Electronics Letters*, *34*, 1596–1597.
Boardman, I., & Bullock, D. (1991). A neural network model of serial order recall from short-term memory. In *Proceedings of the international joint conference on neural networks. II* (pp. 879–884).
Bradski, G., Carpenter, G. A., & Grossberg, S. (1994). STORE working memory networks for storage and recall of arbitrary temporal sequences. *Biological Cybernetics*, *71*, 469–480.
Bullock, D. (2004). Adaptive neural models of queuing and timing in fluent Action. *Trends in Cognitive Sciences*, *8*, 426–433.
Bullock, D., & Rhodes, B. (2003). Competitive queuing for serial planning and performance. In M. Arbib (Ed.), *Handbook of brain theory and neural networks* (pp. 241–244). MIT Press.
Chang, C-C, & Lin, C-J. (2001). Training ν-support vector classifiers: theory and algorithms. *Neural Computation*, *13*, 2119–2147.
Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based methods*. Cambridge: Cambridge University Press.
Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*, 303–314.
Galán, R.F., Sachse, S., Galizia, C.G., & Herz, A.V.M. (2003). Odor driven calcium dynamics in the antennal lobe of honeybee: a hypothesis about the olfactory code. In *Proceedings of the 29th Göttingen neurobiology conference (poster)*.
Galán, R. F., Sachse, S., Galizia, C. G., & Herz, A. V. M. (2004). Driven attractor dynamics in the antennal lobe allow for simple and rapid olfactory pattern classification. *Neural Computation*, *16*, 999–1012.
Grossberg, S. (1978). A theory of human memory: self-organization and performance of sensory-motor codes, maps, and plans. In R. Rosen, & F. Snell (Eds.), *Progress in theoretical biology, Vol. 5* (pp. 233–374). Academic Press.
Guthrie, E. R. (1952). *The psychology of learning* (Revised edition). Massachusetts: Harper Bros.
Houghton, G. (1990). The problem of serial order: a neural network model of sequence learning and recall. In R. Dale, et al. (Eds.), *Current research in natural language generation* (pp. 287–319). Academic Press.
Jändel, M. (2009). Thalamic bursts mediate pattern recognition. In *Proceedings of the 4th international IEEE EMBS conference on neural engineering* (pp. 562–565).
Jändel, M. (2010a). A neural support vector machine. *Neural Networks*, *23*, 607–613.
Jändel, M. (2010b). Pattern recognition as an internalized motor programme. In *Proceedings of international conference on neural networks* (pp. 828–836).
Jändel, M. (2011). Natural evolution of neural support vector machines. In C. Hernández, R. Sanz, J. Gómez-Ramirez, L. S. Smith, A. Hussain, A. Chella, & I. Aleksander (Eds.), *From brains to systems* (pp. 103–208). Springer Neurosciences.
Johnston, D., & Wu, S. M.-S. (1995). *Foundations of cellular neurophysiology*. Cambridge MA: MIT Press.
Kohonen, T. (2001). *Self-organizing maps* (3rd ed.). Springer.
Liu, H., & Liu, D. (2009). A neural network approach for least squares support vector machines learning. In *Joint 48th IEEE conference on decision and control* (pp. 7297–7302).
Macrides, F., Eichenbaum, H. B., & Forbes, W. B. (1982). Temporal relationship between sniffing and the limbic θ rhythm during odor discrimination reversal learning. *Journal of Neuroscience*, *2*, 1705–1717.
Noble, W. S. (2004). Support vector applications in computational biology. In B. Schölkopf, K. Tsuda, & J. Vert (Eds.), *Kernel methods in computational biology* (pp. 71–92). MIT Press.
Ong, C. S., Mary, X., Canu, S., & Smola, A. J. (2004). Learning with non-positive kernels. In *Proceedings of the 21st international conference on machine learning* (pp. 81–89). ACM Press.
Perfetti, R., & Ricci, E. (2006). Analog neural network for support vector machine learning. *IEEE Transactions on Neural Networks*, *17*, 1085–1091.
Platt, J. C. (1998). *Sequential minimal optimization: a fast algorithm for training support vector machines, Technical Report MSR-TR-98-14*. Microsoft.
Quintilianus, M. F. (95). *Book XI*. Institutio Oratoria. English translation in The Orators Education. Vol. 5. Loeb classical library, books 11-12.
Rhodes, B., & Bullock, D. (2002). *Neural dynamics of learning and performance of fixed sequences: latency pattern reorganizations and the N-STREAMS model. Boston University Technical Report CAS/CNS-02-007*.
Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels*. Cambridge MA: MIT Press.
Schölkopf, B., Smola, A. J., Williamson, R. C., & Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation*, *12*, 1207–1245.
Sherman, S. M., & Guillery, R. W. (2006). *Exploring the thalamus and its role in cortical function* (2nd ed.). Cambridge, MA: MIT Press.
Tan, Y., Xia, Y., & Wang, J. (2000). Neural network realization of support vector methods for pattern classification. In *Proceedings of the IEEE-INNS-ENNS international joint conference on neural networks, Vol. 6* (pp. 411–416).
Teyke, T. (1995). Food-attraction conditioning in the snail, Helix Pomatia. *Journal of Computational Physiology, A*, *177*, 409–414.
Vapnik, V. (1998). *Statistical learning theory*. Wiley-Interscience.
Viéville, T., & Crahay, S. (2004). Using a Hebbian learning rule for multi-class SVM classifiers. *Journal of Computational Neuroscience*, *17*, 271–287.
Xia, S., & Wang, J. (2004). A one layer recurrent neural network for support vector machine learning. *IEEE Transactions on Systems Man and Cybernetics B*, *34*, 1261–1269.
Yang, Y., He, Q., & Hu, X. (2012). A compact neural network for training support vector machines. *Neurocomputing*, *86*, 193–198.