Pattern Recognition as an Internalized Motor Programme

M. Jändel

***Abstract*—**A new conceptual architecture for low-level neural pattern recognition is presented. The key ideas are that the brain implements support vector machines and that support vectors are represented as memory patterns in competitive queuing memories. A binary classifier is built from two competitive queuing memories holding positive and negative valence training examples respectively. The support vector machine classification function is calculated in synchronized evaluation cycles. The kernel is computed by bi-symmetric feed-forward networks feed by sensory input and by competitive queuing memories traversing the complete sequence of support vectors. Temporary summation generates the output classification. It is speculated that perception apparatus in the brain reuses structures that have evolved for enabling fluent execution of prepared action sequences so that pattern recognition is built on internalized motor programmes.

***Keywords*—**Competitive queuing model, Olfactory system, Pattern recognition, Support vector machine, Thalamus

I. Introduction

THE perception systems of higher vertebrates include trainable pattern recognition functions that work reliably and flexibly in a wide range of contexts. One single exposure is often sufficient for learning a new significant pattern [1]. Pattern recognition skills are quite stable even if not rehearsed and in spite of intervening learning of unrelated classifications. Although the processing speed of biological neural systems is comparatively low, low-level pattern recognition is typically performed within a few hundred milliseconds e.g. the duration of a sniff cycle in the olfactory system [2]. The behavior of natural pattern recognition systems is hence qualitatively different from that of feed-forward artificial neural networks where one-shot learning is difficult and training examples must be reviewed frequently to avoid overwriting of established skills.

Reference [3] conjectures that low-level pattern recognition systems in the brain implements support vector machines (SVM) of a specific type (zero-bias ν-SVM). The core of this scheme is that support vectors are memory patterns in a randomly oscillating associative memory and that classifications are computed by temporal integration in a feed-forward pathway in which inputs are provided by external sensors and by a stream of support vector patterns from the oscillating associative memory. This model supports one-shot learning and that once established skills are stable without rehearsal. An interesting architectural match between neural zero-bias ν-SVM and the olfactory system is described in [3]. Gross oscillatory patterns in the brain are also reproduced by the model. The zero-bias ν-SVM model is moreover proposed as a model for burst behavior in the thalamocortical system [4].

In spite of intriguing correspondences to brain systems, there are several problems with the zero-bias ν-SVM approach.

A) Excluding the bias factor in the SVM classification function reduces generalization performance as discussed in [3]. It is, however, very difficult to find a biologically believable realization of biased ν-SVM using one single support vector memory unit.

B) Temporal integration over randomly sampled support vectors converges rather slowly to the required classification function. This means that only ~10 support vectors can be used under biologically realistic conditions.

C) Given the wide scope of pattern recognition tasks in natural circumstances it appears to be extravagant to build a dedicated pattern recognition machine with stored support vectors for each classification context. It would be more efficient to reuse training examples for many different tasks. The zero-bias ν-SVM model appears, however, to require a specific collection of support vectors for each task.

The present paper suggests a new model that preserves the match to brain architecture and dynamics that was found in [3] and [4] but resolves the three problems that plagues the zero-bias ν-SVM model.

It is found that biased ν-SVM can be implemented using separate banks of memory for support vectors with different valence.

The convergence problem is alleviated by employing competitive queuing memory for storing support vectors. The ability to precisely repeat a time series of patterns is exactly what is needed for optimal convergence of the neural SVM classification calculation. This capability is also needed for skilled motor behavior so it is conjectured that the same basic machinery is used both for perception and for action.

Using different banks of competitive queuing memory for support vectors with different valence also solves the specialization problem since it enables flexible combinations of training examples for different pattern recognition tasks.

M. Jändel is with the Swedish Defence Research Agency, SE-164 90, Stockholm, Sweden (phone: +46 709277264; fax: +46 855503700; e-mail: magnus@jaendel.se).

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

Although the objective of this paper is to launch a new biologically relevant model of low-level pattern recognition in the brain, a high level architectural model will be used and the style of the description will be slanted to engineering rather to biology. The match to brain systems for models of this type is discussed in [3] and [4] and evolutionary aspects are covered in [5]. Here we will focus on elucidating the overall structure of the neural biased ν-SVM model.

The structure of the paper is as follows. A high-level model of competitive queuing memory is defined in section II. Section III describes ν-SVM including a novel training algorithm that is optimized for the purpose of implementation with bi-symmetric competitive queuing memory. The neural implementation is described in section IV with subsections for architecture, classification, one-shot learning of new training examples, learning of optimal weights and bias learning. Section V concludes the paper with discussion of biological feasibility and possible extensions.

II. Competitive Queuing Memory

A basic skill underlying purposeful motor behavior is the ability to express a sequence of actions according to a pre-prepared plan. There are two main approaches to learning and reproducing action sequences: recurrent network architectures [6] [7] [8] and competitive queuing models [9][10][11]. Bullock [12] argues that competitive queuing models are used in the brain as a basic architectural component for producing complex skilled actions. The present paper will speculatively apply competitive queuing to trainable pattern recognition. The main reason for using competitive queuing memory rather than recurrent network models is the one-shot learning capacity of the former approach.

We shall use an abstract high-order model for a competitive queuing memory (CQM) as shown in Fig. 1. The CQM consists of a memory layer and a choice layer. The memory layer holds m^* patterns $\mathbf{x}_i$ (i=1, 2, … m^*) that in the following is understood to be real-valued vectors. Each memory pattern has an associated activation level α_i that also will be called the weight of the pattern. Weights are positive real numbers and cannot exceed a maximum value. The CQM is turned on by a reset signal and the choice layer outputs the memory pattern with the highest weight for a time that is proportional to the weight. The selected pattern is then inhibited and the choice layer selects the memory pattern with the next highest weight to be displayed for at time that again is proportional to the weight of the selected pattern. At the end of this period the active pattern is inhibited and the memory pattern with the third highest weight is selected and so on. The CQM will hence transverse the stored memory patterns until all patterns with non-zero weights have been displayed.

A training interface can adjust the value of a weight which means that all the other weights also are modified with a constant amount so that the sum of weights is conserved. It is also possible to inject a new memory pattern into the CQM. The initial weight of new memory pattern is set to zero. The sum of the weights is hence a constant. This means that the total time for displaying the complete sequence of patterns also is constant.

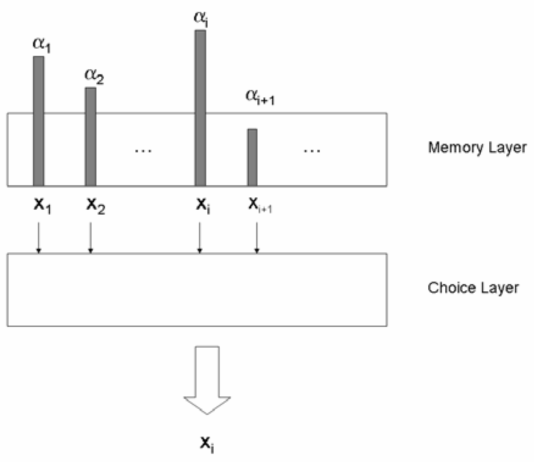

Fig. 1 Competitive queuing memory (CQM). The choice layer selects and outputs the memory pattern $\mathbf{x}_i$ with the highest activation level α_i for at time proportional to the activation level. Selected patterns are inhibited after presentation so that the CQM traverses all memory patterns with non-zero activation level.

Note that some competitive queuing models [9] [13]- [15] redistribute the activity level after each iteration so that the activity of not yet selected patterns increases for each iteration. The present model avoids such redistribution thus keeping the activity level of patterns constant until they are selected, displayed for the allotted time and then quenched until the reset signal restores the original value.

Reference [12] presents considerable evidence that the brain uses competitive queuing for learning and expressing motor action sequences and that this mechanism underlies skilled fluent behavior including drawing, speech and musical expression. Recent electrophysiological support for the competitive queuing model is found in [16]. In the following we will apply the present abstract model to one-shot trainable pattern recognition in the brain leading up to the proposition that nature employs competitive queuing for both perception and action.

III. Support Vector Machine Definition and Solution

We shall consider an algorithm for binary classification in which the valence $y \in \{+1,-1\}$ of a test sample $\mathbf{x}$ is found by generalizing from a training set consisting of examples ($\mathbf{x}_i$, y_i) where i=1,2, … m. Bold letters signify vectors, $\mathbf{x}_i$ is a real-valued vector and y_i is the correct classification.

A support vector machine (SVM) [17] implicitly performs a non-linear projection of input vectors $\mathbf{x}$ to a high-dimensional feature space. The classifier is a hyperplane in feature space that with maximum margin separates examples with positive valence from examples with negative valence. Regular support vectors are training examples that sit right on the margin and thus define the hyperplane. Soft-margin SVMs allow outlier support vectors to violate margins so that the classifier can handle noisy training sets. Model parameters control the balance between accuracy and generalization performance.

The ν-SVM is a soft-margin SVM in which a single parameter $0<\nu<1$ controls the generalization ability [18]. For a set of m training examples ν-SVM is solved by maximizing the dual objective function

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

where K is the positive definite kernel function that defines the implicit projection to feature space. Each training example $(\mathbf{x}_i, y_i)$ is associated with a weight a_i and W is maximized with respect to $\boldsymbol{\alpha}$ under the three constraints

$$0 \le \alpha_i \le \frac{1}{m}, \tag{2}$$

$$\sum_{i=1}^{m}\alpha_i \ge \nu \tag{3}$$

and

$$\sum_{i=1}^{m} y_i\alpha_i = 0\,. \tag{4}$$

Chang and Lin [19] demonstrated that there in general exists an optimal solution in the α-space hyperplane,

$$\sum_{i=1}^{m}\alpha_i = \nu\,, \tag{5}$$

so that the tighter constraint (5) can replace (3).

There are no false optima since equation (1) is quadratic with respect to $\boldsymbol{\alpha}$ and the optimum is sought for in the α-space hyperplane that is defined by the constraints. Once an optimal solution has been found, inputs $\mathbf{x}$ can be classified as having negative valence if $f(\mathbf{x}) < 0$ and positive valence if $f(\mathbf{x}) \ge 0$ using the classification function

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i\alpha_i \,\mathrm{K}(\mathbf{x}_i,\mathbf{x}) + b \tag{6}$$

where b is a bias parameter. Section IV E describes how the optimal value of b is found. Note that the constraint (4) is removed if b is set to zero as in the zero-bias ν-SVM [3]. The classification margin is defined as $M(\mathbf{x}) = y\, f(\mathbf{x})$. For future use we also define the function

$$h(\mathbf{x}) = f(\mathbf{x}) - b = \sum_{i=1}^{m} y_i\alpha_i \,\mathrm{K}(\mathbf{x}_i,\mathbf{x})\,. \tag{7}$$

So far we have briefly reviewed the basics of the ν-SVM problem. To prepare for the neural implementation some further notation and a special solution method will be introduced. We are aiming for a neural architecture in which training examples with different valence are held in different neural structures. We note therefore that the constraints (4) and (5) can be written

$$\sum_{i=1}^{m}\delta(1,y_i)\alpha_i = \sum_{i=1}^{m}\delta(-1,y_i)\alpha_i = \frac{\nu}{2} \tag{8}$$

where the Kronecker delta function is defined as,

$$\delta(y',y) = \begin{cases} 1, \text{if } y' = y \\ 0, \text{if } y' \ne y \end{cases}. \tag{9}$$

Because of the benign quadratic nature of the ν-SVM problem we can solve it by gradient ascent. The weight vector $\boldsymbol{\alpha}$ is initiated as an arbitrary point in the α-space hyperplane given by the constraints (2), (4) and (5). For simplicity we shall refer to this hyperplane as the constraint hyperplane. We will define a learning rule that keeps the weight vector in the constraint hyperplane while moving it towards the optimum of W. Since the goal is to handle training examples with positive and negative valence separately there will eventually be one learning rule for each of these categories.

The weight vector must move in the direction that is defined by the projection of the gradient of $W(\boldsymbol{\alpha})$ in the constraint hyperplane. The i:th component of $\mathbf{U}=\mathbf{grad}(W)$ is

$$\mathrm{U}_i = \frac{\partial \mathrm{W}}{\partial \alpha_i} = -y_i\sum_{j=1}^{m} y_j\alpha_j \,\mathrm{K}(\mathbf{x}_i,\mathbf{x}_j) = -y_i h(\mathbf{x}_i)\,. \tag{10}$$

To find the projection of $\mathbf{U}$ in the constraint hyperplane we first define the normals $\mathbf{e}_A = \frac{1}{\sqrt{m}}(1,1,...,1)$ and $\mathbf{e}_B = \frac{1}{\sqrt{m}}(y_1,y_2,...,y_m)$ of the hyperplanes (5) and (4) respectively. An orthonormal base $(\mathbf{e}_1,\mathbf{e}_2)$ spanning the sub-space defined by $\mathbf{e}_A$ and $\mathbf{e}_B$ is given by $\mathbf{e}_1 = \mathbf{e}_A$ and $\mathbf{e}_2 = \frac{\mathbf{e}_B - \hat{y}\mathbf{e}_A}{\sqrt{1-\hat{y}^2}}$ where $\hat{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$ is the average valence.

The projection $\tilde{\mathbf{U}}$ of $\mathbf{U}=\mathbf{grad}(W)$ on the constraint hyperplane is,

$$\tilde{\mathbf{U}} = \mathbf{U} - (\mathbf{U}\cdot\mathbf{e}_1)\mathbf{e}_1 - (\mathbf{U}\cdot\mathbf{e}_2)\mathbf{e}_2\,. \tag{11}$$

The i:th component of $\tilde{\mathbf{U}}$ evaluates to

$$\tilde{\mathrm{U}}_i = \mathrm{U}_i - \hat{U} - \frac{y_i - \hat{y}}{1-\hat{y}^2}(\hat{\hat{U}} - \hat{y}\hat{U}) \tag{12}$$

where $\hat{\mathrm{U}}$ is the average value of U_i,

$$\hat{U} = \frac{1}{m}\sum_{i=1}^{m}\mathrm{U}_i\,, \tag{13}$$

and $\hat{\hat{U}}$ is the first moment of U_i with respect to y

$$\hat{\hat{U}} = \frac{1}{m}\sum_{i=1}^{m} y_i\mathrm{U}_i\,. \tag{14}$$

The gradient ascent learning rule for reaching the maximum of (1) is that the weight vector incrementally is updated according to

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \eta\tilde{\mathbf{U}} \tag{15}$$

where η is a sufficiently small learning rate. Since we are aiming for an architecture in which examples with positive and negative valence are handled by different physical units we shall need separate learning rules for positive and negative examples. For an example i with positive valence (12) simplifies to

$$\tilde{\mathrm{U}}_i = \mathrm{U}_i - \hat{U}_+ \tag{16}$$

where the second term averages only over positive valence examples,

$$\hat{U}_+ = \frac{1}{m_+}\sum_{i=1}^{m}\delta(1, y_i)U_i \, . \tag{17}$$

The number of positive valence examples is m_+ and the number of negative valence examples is m_- so that $m_+ + m_- = m$. Correspondingly we get for examples with negative valence

$$\tilde{U}_i = U_i - \hat{U}_- \, , \tag{18}$$

with the second term averaging over negative valence examples,

$$\hat{U}_- = \frac{1}{m_-}\sum_{i=1}^{m}\delta(-1, y_i)U_i \, . \tag{19}$$

Note that (16) and (18), for a fully trained SVM, enforce a partitioning of the training examples into three parts depending on the value of $U_i = -y_i h(\mathbf{x}_i)$.

Trivial examples have $U_i < \hat{U}_*$ so that $\tilde{U}_i < 0$. The index $*$ is here shorthand for + or – according to the valence of the example. The learning rule (15) forces α_i to zero so that trivial examples give no contribution to classifications.

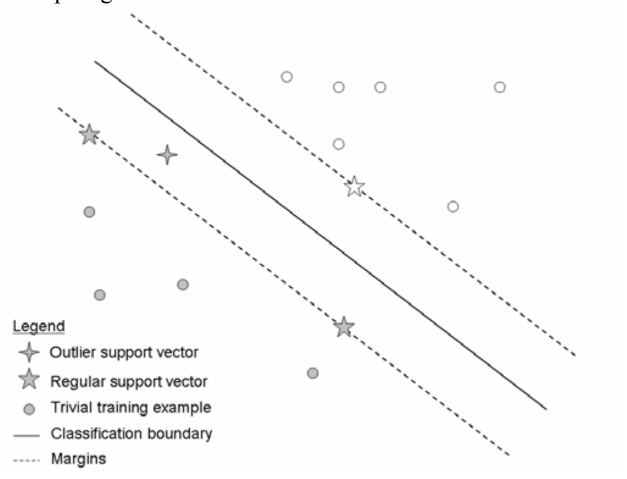

Fig. 2 Training example types in a soft margin support vector machine. Closed symbols are positive valence training examples and open symbols are negative valence training examples. A fully trained soft margin SVM divides the training examples in three groups: regular support vectors, outlier support vectors and trivial examples.

Soft margin classifiers allow some training examples to violate margins. Such outlier support vectors have $U_i > \hat{U}_*$ and hence $\tilde{U}_i > 0$ so that the learning rule pushes the corresponding weight to the maximum value $\alpha_i = 1/m$.

The remaining group of examples forms the regular support vectors with weights falling in the intermediate range $0 < \alpha_i < 1/m$. Such stable intermediate weights require that $\tilde{U}_i = 0$ and hence that $U_i = -y_i h(\mathbf{x}_i)$ is equal for all regular support vectors belonging to the same valence group. In section III E we will use this result in the bias computation.

IV. Neural Implementation

A. Architecture

Fig. 3 shows the full architecture of the proposed neural implementation of a ν-SVM.

Sensors capture signals from the external world and provide a continuously updated sensory vector $\mathbf{x}''(t)$ where t is time.

The Switch selects between sensory input and internal signals carrying recalled training examples depending on if the system is in the classification mode or the training mode. It provides the input vector $\mathbf{x}'(t)$ to the Trap.

The Trap is a piece of sensory memory that periodically captures a snapshot of the input vector $\mathbf{x}'(t)$ and holds the trapped copy $\mathbf{x}$ as a stable input for the downstream system.

The core of the classification engine is bi-symmetric. Each bi-symmetric part includes a competitive queuing memory (CQM) for storing and recalling training examples and a unit for computing the SVM kernel. Training examples are typically support vectors as explained in section IV D. The upper part of the system in Fig. 3 handles training examples with negative valence whereas the lower part handles training examples with positive valence. The meaning of positive and negative from the point of view of the organism depends on the context. In a food search situation positive valence could e.g. mean edible and negative valence mean not edible.

Signals from the bi-symmetric parts come together in the Integrator where $\sum_{i=1}^{m} y_i\alpha_i \, K(\mathbf{x}_i, \mathbf{x})$ is computed. The Bias unit adds the factor b and outputs the classification function.

The cyclic operation of the system is synchronized by a clock signal that marks the beginning of each evaluation or training cycle. The system performs four main functions:

1) *Classification in which it evaluates sensory input*
2) *Surprise learning in which it learns new training examples*
3) *Importance learning in which it learns the optimal SVM weights*
4) *Bias learning in which it learns the optimal bias factor*

The detailed operation and functions of the system and its parts will be explained by describing each process in turn.

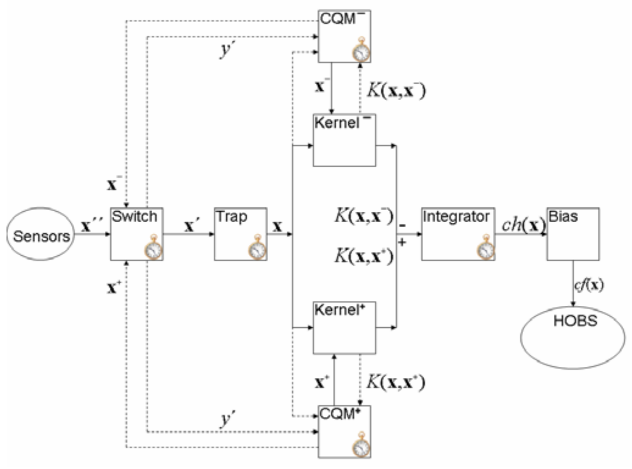

Fig. 3 Architecture of a neural ν-SVM system. Boxes are system parts. Ovals are surrounding systems. Sensors provide raw sensory input. Higher-order brain systems (HOBS) receive classifications and control the processes of the system. Dashed lines indicate connections used only for training the system according to the Surprise learning and the Importance learning processes. Solid lines are connections used for the Classification process and on occasion also for the learning processes. The clock symbol marks units that receive the synchronization signal indicating the start of an evaluation cycle.

B. Classification

The sensors provide a continuously updated sensory vector $\mathbf{x}''(t)$ where t is the current time. The Switch is, in the classification mode, transparent for the sensory vector so the Trap receives $\mathbf{x}'(t) = \mathbf{x}''(t)$. The system follows a cyclic process in which each cycle is an evaluation period during which a classification is performed. The length of the evaluation period is T_{eval}. Cued by a clock signal indicating the start of a cycle at time t_0, the Trap captures a snapshot $\mathbf{x} = \mathbf{x}'(t_0)$ of the sensory input and holds this sample unchanged for a full cycle. The next snapshot $\mathbf{x} = \mathbf{x}'(t_0 + T_{eval})$ is captured at the beginning of the next evaluation period. This process is repeated incessantly.

At the start of an evaluation cycle the CQM$^+$ is reset by the clock signal and starts outputting a series of training examples with positive valence. As explained in section IV D these examples are, for a fully trained system, the support vectors with positive valence. The stored examples in the CQM compete for the opportunity to be selected as described in section II.

Each example $\mathbf{x}_i$ is presented for an endurance time T_i that is proportional to the corresponding SVM weight,

$$T_i = c\alpha_i\,, \tag{20}$$

where the factor $c = 2T_{eval}/\nu$ is identical for both sides of the bi-symmetric system since $T_{eval} = \sum_{i=1}^{m}\delta(1, y_i)T_i = \sum_{i=1}^{m}\delta(-1, y_i)T_i$ (see also (8)). The endurance time is the physical representation of SVM weight in the neural system. Note that the upper limit according to the constraint (2) means that there must be a corresponding maximal endurance time

$$T_{\max} = c/m\,. \tag{21}$$

The Kernel$^+$ unit receives both the trapped sensory input $\mathbf{x}$ and the present output $\mathbf{x}^+$ from CQM$^+$. It computes the kernel function K($\mathbf{x}$, $\mathbf{x}^+$) and forwards the result to the Integrator. The Kernel units are feed-forward networks and require no synchronization.

The CQM$^-$ and the Kernel$^-$ mirror the operation of the positive valence side. CQM$^-$ is triggered by the same clock signal as CQM$^+$ and cycles through the stored negative valence training examples. The present output $\mathbf{x}^-$ of the CQM$^-$ and the sensory vector $\mathbf{x}$ are inputs to the Kernel$^-$ unit that computes K($\mathbf{x}$, $\mathbf{x}^-$).

The integrator calculates over an evaluation cycle

$$ch(\mathbf{x}) = \int_{t_0}^{t_0+T_{trap}} (\mathrm{K}^+(\mathbf{x}^+, \mathbf{x}) - \mathrm{K}^-(\mathbf{x}^-, \mathbf{x}))dt \tag{22}$$

where h is given by (7). Note that the contribution from the positive valence side to (22) is excitatory whereas the contribution from the negative valence side is inhibitory. Since each of the CQM units cycles through the support vectors and since each support vector is presented for a time $T_i = c\alpha_i$ we conclude that the integral in (22) is proportional to $\sum_{i=1}^{m} y_i\alpha_i \,\mathrm{K}(\mathbf{x}_i, \mathbf{x})$. The Integrator computes the core part of the classification function.

The Bias unit receives $ch(\mathbf{x})$ from the Integrator adds cb and outputs a signal that is proportional to the SVM classification function

$$c(h(\mathbf{x}) + b) = cf(\mathbf{x})\,. \tag{23}$$

Note that sgn($cf(\mathbf{x})$)=sgn($f(\mathbf{x})$) since c is a positive constant.

This sub-section has demonstrated how the classification process works assuming that the system is fully trained so that the CQM units contain support vectors with optimal weights and the Bias unit knows the correct value of b. The following sections will focus on the training processes that achieve this.

C. Surprise Learning

The Surprise learning process collects relevant training examples from the environment. Ideally it should load only the support vectors into the CQM since trivial examples do not contribute to classifications.

Consider a creature in an environment with several different types of food and poison with distinctive scents. The olfactory system of the life form is organized as the system shown in Fig. 3 and is used for guiding the organism to find food and avoid poison. The system should not store a memory of every encounter with food or poison in the CQM units. It would quickly fill up memory space with trivial examples that contribute little to pattern recognition performance. The more proficient the animal becomes in finding its normal food, the more will trivial examples dominate in the flow of odors. It needs some filtering scheme for only imprinting odors that are good support vector candidates.

A viable strategy is to only register surprises where a surprise is defined as a misclassified input. Such inputs are likely to be close to the classification boundary and are hence good support vector candidates. Note that regular support vectors are borderline cases that sit right on the margin (see Fig. 2). Infant organisms would be poor at recognizing food and starts with a series of dangerous experiment where various objects are eaten. Poison will taste bad and make the life form sick thus constituting a surprise with negative valence. Occasionally the animal classifies edible stuff as poison and watches how more proficient mates devours the food. That will be a surprise with positive valence. Note that higher-order systems assist the SVM classifier in inferring the valence of surprises. Accumulating surprises provides the creature with an increasingly relevant set of support vector candidates. A major advantage with this learning mode is that relevant experiences are loaded into the CQM mode immediately thus providing a mechanism for one-shot learning.

A new training example, that just has been acquired by surprise learning, is put into the CQM with some initial endurance time. This time could in a complex life form conceivably be proportional to the emotional intensity of the event. In the following we will simply assume that the initial endurance time is zero so that the system is stable when acquiring new training examples.

D. Importance Learning

A surprise learning episode leaves the system in a sub-optimal state in which the SVM weight vector is detached from the optimum as defined by the solution of the ν-SVM problem according to section III. To reinstate optimal pattern recognition performance, the system should apply the gradient ascent procedure in which the weights repeatedly are updated according to the learning rule defined by (15) - (19). In this sub-section we will discuss SVM weights while remembering that the endurance times represent the weights according to the proportionality relation (20).

There are several possible algorithms for implementing ν-SVM gradient ascent in the present architecture, one of which will be presented here. The following general aspects are shared by workable methods.

I) Inputs from the external world are replaced by training examples evoked from the CQM units. The Switch shuts out the sensory input and locks on inputs from one of the CQMs. The organism is hence incapacitated during this process. A reasonable assumption is that importance learning is performed in sleep. Replacing sensory input with memories bears some similarity to dreaming. Philosophers of the mind have for a very long time suspected that dreaming is related to off-line learning and memory consolidation [20]. We should, however, be careful to draw this analogy too far since there are many competing theories of dreaming and low-level perceptual learning in sleep might not be associated with dreaming.

II) The Trap and the feed-forward classification process work precisely as in the waking state but the resulting classification is disconnected from the action system. If e.g. a positive valence training example of food scent is recalled and put into the Trap, this will not cause actual feeding behavior. This is analogous to sleep paralysis in REM sleep.

III) The learning rule applied to either of the CQM units redistributes the endurance time (SVM weight) over the stored patterns but keeps the sum of endurance times in the CQM constant. This is a property of the competitive queuing model and is also consistent with the constraints (4) and (5) as expressed in (8). The learning rules (16) and (18), where the U_i term is balanced by the average of the same term for all memory patterns in the CQM, respect the conservation of the sum of weights in the CQM.

The details of the Importance learning algorithm will now be explained. The Switch receives the presently displayed training examples $\mathbf{x}^+$ and $\mathbf{x}^-$ from the CQM$^+$ and the CQM$^-$ respectively. At a uniformly distributed random time in the evaluation cycle, the Switch locks on one of these input vectors. The positive and negative valence input is selected with equal probability. The chosen training example is forwarded to the Trap and the valence of the selection is sent to both of the CQM units. The result is that the Switch is opaque for sensory input and outputs training examples $\mathbf{x}_j$ randomly selected according a distribution $p_i = \kappa\alpha_i$ where $\kappa=1/\nu$.

The Trap operates just as in the waking state. At the beginning of each evaluation cycle it locks on the output of the Switch. The net result is that the Trap for the duration of each evaluation cycle outputs a training example $\mathbf{x}_j$ randomly selected from the distribution $p_i = \kappa\alpha_i$.

Each of the CQM units applies the same learning rules once for each selected memory pattern:

$$\alpha_i \leftarrow \alpha_i - \eta y^* y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{24}$$

$$\forall k \;\; \alpha_k \leftarrow \alpha_k + \frac{\eta}{m^*} y^* y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{25}$$

where $*$ means either + or -. Note that $\mathbf{x}_j$ is the example held by the Trap for the duration of the evaluation cycle whereas $\mathbf{x}_i$ is the currently displayed example of the CQM. The CQM receives the valence y_j of the trapped example from the Switch and stores it for the duration of the evaluation cycle (see Fig. 3). The learning rule (24) is applied only to the presently displayed example and the learning rule (25) is applied to all training examples stored by the CQM. The training of the bi-symmetric CQMs is hence disconnected, the only link being that they both use the same input example $\mathbf{x}_j$.

We shall now demonstrate that the learning rules (24-25) implements ν-SVM gradient ascent according to (16) and (18) respectively. For this purpose we consider the average change $\Delta\alpha_s$ of the weight of a specific training example with index s during one evaluation interval. The averaging is performed with respect to the probability distribution of examples in the Trap.

The learning rule (24) modifies the selected weight once per evaluation interval with the average result

$$\Delta_1\alpha_s = -\eta y^* \sum_{j=1}^{m} y_j p_j K(\mathbf{x}_s, \mathbf{x}_j) = \kappa\eta U_s \qquad (26)$$

where U_s is given by (10).

The learning rule (25) modifies the selected weight m^* times per evaluation interval with a result that averages to

$$\Delta_2\alpha_s = \eta \frac{1}{m^*}\sum_{k=1}^{m}\delta(y^*, y_k) y^* \sum_{j=1}^{m} y_j p_j K(\mathbf{x}_k, \mathbf{x}_j) = \\ -\eta\kappa \frac{1}{m^*}\sum_{k=1}^{m}\delta(y^*, y_k) U_k = -\kappa\eta \hat{U}_* \qquad (27)$$

where $\hat{U}_*$ is given by (17) for the positive valence CQM and by (19) for the negative valence CQM. The Kronecker delta in (27) is defined according to (9).

Combining the contributions from learning rules (24) and (25) we get $\Delta_1\alpha_s + \Delta_2\alpha_s = \Delta\alpha_s = \kappa\eta(U_s - \hat{U}_*)$ showing that the learning rules implement ν-SVM gradient ascent according to (16) and (18). This means that weights of trivial examples vanish asymptotically so that the effective optimized training example population consists only of support vectors.

E. Bias Learning

In section IV B we assumed that the Bias unit adds the correct bias b to the output $h(\mathbf{x})$ of the Integrator thus finalizing the computation of the classification function $f(\mathbf{x}) = h(\mathbf{x}) + b$. The Bias unit must, however, learn the proper value of b. This is done in the sleeping state in which the Integrator for each evaluation cycle produces an output $ch(\mathbf{x}_j)$ where $\mathbf{x}_j$ is a training example selected randomly according to the $p_j = \kappa\alpha_j$ distribution.

The algorithm for computing the bias is quite simple in the special case where all training examples are regular support vectors. The bias unit could just compute the average value $<ch>=-b$ of the input $ch(\mathbf{x}_j)$ and output $cf(\mathbf{x}) = ch(\mathbf{x}) - <ch>$. The reason for this is that all regular support vectors with positive valence have the same value h^+ of $h(\mathbf{x})$ whereas all regular support vectors with negative valence have the same value h^- of $h(\mathbf{x})$ (see section III). The constraint (8) ensures that positive valence support vectors and negative valence support vectors carry the same statistical weight so that $< ch >= c\frac{1}{2}(h^+ + h^-) = c\frac{1}{2}((f(\mathbf{x}^+) - b) + (f(\mathbf{x}^-) - b)) = -cb$ where $\mathbf{x}^+$ and $\mathbf{x}^-$ are any positive and negative valence support vector respectively and $f(\mathbf{x}^+) = -f(\mathbf{x}^-)$ because all regular support vectors have the same margin.

In the general case, the training examples include outliers as well as regular support vectors. We assume, however, that the regular support vectors are the majority of the training example population. The Bias unit performs a clustering computation with respect to the value of the input $ch(\mathbf{x}_j)$. Two clusters centered on the values ch^+ and ch^- are found. The clusters are surrounded by scattered points corresponding to outlier support vectors. The Bias unit computes the average value $< ch >= c\frac{1}{2}(h^+ + h^-) = -cb$ of the centers of the clusters and forms the output $cf(\mathbf{x})$ by subtracting $<ch>$ from the input.

V. Discussion

Simulations show that the online learning strategy of storing misclassifications as new training examples works reliably for a wide range of problem types and SVM kernels. Dropping trivial examples from the training set is also a robust and proven strategy [21]. These methods are suboptimal since using fewer training examples can only worsen classifications but they save plenty of memory space. In the model we have employed radical memory saving strategies. Nature may be more or less generous with memory space.

The present system has the advantage of keeping the time for generating a classification constant in spite of continuous learning. Inserting more training examples in the CQMs means that endurance times eventually rescale but the time for performing pattern recognition remains constant. A predictable time for generating classifications is obviously useful for facilitating timely action and for building higher order pattern recognition systems.

Handling negative numbers gracefully is always a problem when applying mathematical models with positive and negative numbers to brain systems. Neurons have excitatory and inhibitory links and but these mechanisms lack the neat symmetry of the mathematical concepts. Using separate memories for training examples with different valence provides a partial solution to this dilemma since the storage location implicitly defines the valence of the examples. The Integrator must, however, still deal with summation of positive and negative terms by means of excitation and inhibition.

A definite advantage of bi-symmetric competitive queuing ν-SVM compared to the single-memory zero-bias ν-SVM is that the population of training examples can be combined in many different pattern recognition contexts. A CQM can store one set of training examples and several different time sequences. Each time sequence is represented by a set of weights α, α', α'' … A given training example could be a regular support vector in one set, a trivial example in a second set and an outlier in a third set. Many different classifiers can be realized using a restricted set of CQMs and training examples by flexible pair wise combinations of CQMs and appropriate selections of time sequences. This modularization in reusable collections of training examples could also support higher-order systems that control and configure lower-order classifiers [22].

The neural biased ν-SVM model in section IV is intended as a model of the brain but is described as a high-level architecture without explicit relation to brain structures. Mappings of the neural zero-bias SVM to biological neural systems are provided in [3] and [4] and apply largely also to the present model. We shall here briefly review how the biased ν-SVM compares to brain components, structures and processes focusing on the olfactory system as an example. It should be noted that the brain, according to the present hypothesis, implements many different instances of neural support vector machines each instance tailored for a specific pattern recognition task.

The *Sensors* module in Fig. 3 corresponds to the olfactory bulb where input from the primary chemical sensors are compiled and stabilized.

The *Switch* forwards one of several inputs according to the state of the system. This function is clearly implementable in the brain. An example of a similar routing function is found in the thalamic relay matrix [23]. The *Trap* is an abstraction of sensory memory which is characteristic of all senses [24]. The *Switch* and the *Trap* are, as argued in [3], found in the anterior olfactory cortex that is known to receive signals both from the olfactory bulb and broad retrograde connections from the piriform cortex allowing odor memories from the piriform cortex to be copied to the anterior olfactory cortex [25].

Biologically realistic neural models of *competitive queuing memory* in the context of motor systems are discussed in [12]. It is a speculation of the present paper that CQM is employed in perception. In the olfactory context, CQMs could be located to the anterior piriform cortex that is known to resemble an associative memory [25].

The *Kernel* units implement non-linear multi-input functions and are hence readily built from neural hardware. Feed-forward artificial neural networks with at least one hidden layer can approximate any continuous multivariate function with arbitrary accuracy [26]. SVM kernels should be positive definite but even kernels that violate this requirement may work well in practice [27]. Good pattern recognition performance can, for many applications, be achieved with a wide variety of kernels [28]. The kernel functions applied by the Kernel$^+$ and Kernel$^-$ nodes should be identical. Such identical matching of brain structures might be difficult to achieve because of the changeable nature of brain tissue. This is a serious challenge for the present model and good reason for also considering the single-memory model in [3]. The neural basis for the *Integrator* function is temporal summation [29]. The *Bias* unit performs effectively an unsupervised clustering operation which is a quite feasibly function in neural networks. For olfaction it appears that the feed forward pathway consisting of the Kernels, the Integrator and the Bias unit should be found in the posterior piriform cortex that structurally mainly is a feed-forward network [25].

The present model includes a fast cycle in the CQMs and a slow evaluation cycle. This is not unlike the oscillatory pattern of the brain with high frequency cortical oscillations combined with slower rhythms of the sensory systems. The olfactory system exhibits e.g. a sniffing cycle [2], where each cycle means that input scents are evaluated, combined with faster oscillations in the piriform cortex [30]. Brain rhythms in relation to support vector machine models are further discussed in [3] and [4].

Note that there is a rich literature on computational models of the olfactory system with many established alternatives to the present theory. References to competing models are found in [4].

The present model shows an intriguing resemblance to the competitive queuing account of motor action. Skilled motor action is according to [12] characterized by:

- Motor programmes are memories of significant motor sequences that are stored in CQMs.
- Higher-order systems initiate appropriate motor programmes as needed.
- Output motor control signals are produced by blending internal programmes with sensory inputs.
- Failure triggers discontinuous modifications of motor programmes.
- Timing is optimized by repetitious training.

Trainable pattern recognition is according to the present model characterized by:

- Support vectors are memories of significant sensory inputs. Patter recognition is based on programmes of support vectors.
- Higher-order systems initiate appropriate support vector programmes as needed.
- Output classifications are created by blending internal programmes with sensory inputs.
- Failure triggers discontinuous modifications of support vector programmes.
- Programme timing is optimized by internal repetition.

The similarity between skilled motor action and trained pattern recognition can speculatively be accounted for by evolution reusing and adapting the same baseline neural machinery for action and perception.

References

[1] T. Teyke, "Food-attraction conditioning in the snail, Helix Pomatia.," *J. Comp. Physiol*, vol. A 177, pp. 409–414, 1995.

[2] F. Macrides, H. B. Eichenbaum, and W. B. Forbes, "Temporal relationship between sniffing and the limbic θ-rhythm during odor discrimination reversal learning," *J. Neurosci.,* vol. 2, pp. 1705–1717, 1982.

[3] M. Jändel, "A neural support vector machine," *Neural Networks*, vol. 23, pp. 607–613, 2010.

[4] M. Jändel, "Thalamic bursts mediate pattern recognition," in *Proc. 4th International IEEE EMBS Conf. on Neural Engineering*, 2009, pp. 562–565.

[5] M. Jändel, "Evolutionary path to biological kernel machines," in *Proc. of Brain Inspired Cognitive Systems*, 2010, to be published.

[6] J. Elman, "Language processing," in *The Handbook Of Brain Theory And Neural Networks*, M. Arbib, Ed. MIT Press, 1995, pp. 508–512.

[7] P. F. Dominey, "Influences of temporal organization on sequence learning and transfer," *J. Exp. Psychol. Learn. Mem. Cogn.,* vol. 24, pp. 234–248, 1998.

[8] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[9] S. Grossberg, "A theory of human memory: Self-organization and performance of sensory-motor codes, maps, and plans," in *Progress in Theoretical Biology*, vol. 5, R. Rosen, and F. Snell, Eds. Academic Press, 1978, pp. 233–374.

[10] G. Houghton, "The problem of serial order: A neural network

model of sequence learning and recall," in *Current Research In Natural Language Generation*, R. Dale et al, Eds. Academic Press, 1990, pp. 287–319.
[11] D. Bullock, and B. Rhodes, "Competitive queuing for serial planning and performance," in *Handbook of Brain Theory And Neural Networks*, M. Arbib, Ed. MIT Press, 2003, pp. 241–244.
[12] D. Bullock, "Adaptive neural models of queuing and timing in fluent action," *Trends in Cognitive Sciences*, vol. 8, no. 9, pp. 426–433, 2004.
[13] I. Boardman, and D. Bullock, "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, 1991.
[14] G. Bradski, G. A. Carpenter, and S. Grossberg, "STORE working memory networks for storage and recall of arbitrary temporal sequences," *Biol. Cybern.*, vol. 71, pp. 469–480, 1994.
[15] B. Rhodes, and D. Bullock, "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, 2002.
[16] B. B. Averbeck, M. V. Chafee, D. A. Crowe, and A. P. Georgopoulos, "Parallel processing of serial movements in prefrontal cortex," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 99, pp. 13172–13177, 2002.
[17] N. Cristianini, and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based methods*. Cambridge: Cambridge University Press, 2000.
[18] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, pp. 1207–1245, 2000.
[19] C-C. Chang, and C-J. Lin, "Training ν-support vector classifiers: theory and algorithms," *Neural Computation*, vol. 13, pp. 2119–2147, 2001.
[20] M. F. Quintilianus, *Institutio Oratoria*, Book XI, 95 (English translation in The Orators Education, vol. 5, books 11-12, Loeb classical library).
[21] B. Schölkopf, and A. J. Smola, *Learning with kernels*. Cambridge MA: MIT Press, 2002.
[22] H. J. Caulfield, and K. Heidary, "Exploring margin setting for good generalization in multiple class discrimination," *Pattern Recognition*, vol. 38, pp. 1225–1238, 2005.
[23] S. M. Sherman, and R. W. Guillery, *Exploring the thalamus and its role in cortical function,* 2nd ed., Cambridge, MA: MIT Press, 2006.
[24] A. D. Baddeley, *Essentials of human memory*. New York: Psychology Press, 1999.
[25] L. B. Haberly, "*Parallel-distributed processing in olfactory cortex: new insights from morphological and physiological analysis of neuronal circuitry*," *Chem. Senses*, vol. 26, pp. 551–576, 2001.
[26] G. Cybenko, "Approximations by superpositions of a sigmoidal function," *Math. of Control, Signals and Syst*., vol. 2, pp. 303–314, 1989.
[27] C. S. Ong, X. Mary, S. Canu, and A.J. Smola, "Learning with non-positive kernels," in *Proc. of the 21st International Conference on Machine Learning*, 2004, pp. 81–89.
[28] B. Schölkopf, C. Burges, and V. Vapnik, "Extracting support data for a given task," in *Proc. First Annual Conference on Knowledge Discovery & Data Mining*, 1995, pp. 252–257.
[29] D. Johnston, and S.M-S. Wu, *Foundations of cellular neurophysiology*. Cambridge MA: MIT Press, 1995.
[30] L. B. Haberly, "Olfactory cortex," in *The synaptic organization of the brain*, 4th ed., G. M. Shepherd, Ed., Oxford: Oxford University Press, 1998, pp. 377–416.