STUDIES ON EMERGENCE AND COGNITION
PART 2: HIGH-LEVEL FUNCTIONALITIES

Heikki Hyötyniemi
Helsinki University of Technology
Control Engineering Laboratory
P.O. Box 5400, FIN-02015 HUT, Finland

This paper discusses the challenges one is facing when trying to match the theory of complex systems against the realm of cognitive systems. The first part [7] studied the low-level functions, whereas this part concentrates on showing how the same function structure can also be used to implement more natural-looking functionalities.

1 INTRODUCTION

Typically, complex systems are studied through *simulation:* The nonlinear functions are iterated until (hopefully) something interesting emerges. However, in the nonlinear iteration processes the results soon become analytically intractable. If one would like to truly *understand* a complex system and its properties — like cognition and higher mental functions[1] — this kind of experimental approach is not useful. Nonlinear functions and their behaviors are so diverse that it is difficult to say anything general about them; to have closer understanding of the emergence processes, a special system structure is here selected for closer study:

$$x(k+1) = f_{\text{cut}}(Ax(k) + a). \tag{1}$$

where $f_{\text{cut}}(\cdot)$ is the "generalized cut" function (see [7]). This system structure seems to be a nice compromise between expressional power and mathematical simplicity.

Some researchers say that when studying complex systems, traditional mathematics should be forgotten altogether. The claim here is that, on the contrary, mathematics is probably the only tool to reach the higher level view of the emergent patterns, and the only language for discussing them. There exist different kinds of mathematical tools that can be efficiently applied when tackling with cognitive systems — these possibilities are illustrated in this paper. For example, the following tricks turn out to be useful:

- Extending the search space from logical to real values, solving the problem utilizing the differentiability properties, and interpreting only the results in symbolic terms.
- Transforming a dynamic (causal) poblem into a static pattern in a high-dimensional space and searching for this pattern again using dynamic processes.

[1] Here, speaking of "high level cognitive functions" only means that rather ambitious concepts (like "concept" itself) are being exploited. But — sincerely — it seems that such intuitively rich terminology conveys the most appropriate connotations

But what is perhaps more fundamental, mathematics also offers intuitions to understanding the complex systems issue in a wider perspective. It has been shown (for example, see [6]) that the studied functional structure (1) is *universal,* so that all possible computable functions can be expressed in that framework. This universality has a price — it is the Gödelian undecidability and unanalyzability of the system behavior. However, just as mathematics has shown that there are problems buried in the system structure, it is mathematics that also offers us tools to circumvent the problems.

In concrete terms, it turns out that a good framework for modeling of more advanced cognitive phenomena is *optimization;* that is, only such subclass of the systems of the form (1) will be studied here that are *solutions to some optimization problems.* As will be shown in this paper, cognitively relevant phenomena can often be represented in such a form. The iteration that is carried out is now seen as just the search process that finally (hopefully) converges to the optimum. The chaotic process that is typically regarded as the essence of complexity becomes irrelevant, unveiling the real substance of the complex system. In this way one can have a higher-level, holistic view of the resulting emergent pattern, making it possible to analyze and modify the behavior of the complex system. It is clear that abstract cognitive tasks can often better be understood and expressed declaratively, without the need of taking the actual data manipulation procedures simultaneously into account.

2 OPTIMALITY AND MINIMIZATION

Let us first define an essentially quadratic, general optimality criterion to be minimized:

$$J(x) = \frac{1}{2} \cdot x^T H_1 x + \frac{1}{2} \cdot (y - Cx)^T H_2 (y - Cx) + h_3^T x, \tag{2}$$

matrices H_1 and H_2 being symmetric, typically positive semidefinite, compatible with the vectors x and y. The criterion is formulated in this way to emphasize the role of its components: The first term tries to keep the elements in x low, and the second term tries to keep the difference between y and Cx low (more specific interpretations for these data structures are presented later). This criterion has a unique minimum (assuming it exists). To search for this minimum, the gradient can first be calculated as

$$\frac{dJ(x)}{dx} = \left(H_1 + C^T H_2 C\right) \cdot x + h_3 - C^T H_2 y. \tag{3}$$

The steepest descent method for finding the minimum can be formulated as

$$\begin{aligned} x(k+1) &= x(k) - \mu \cdot \frac{dJ(x)}{dx} \\ &= \left(I - \mu H_1 - \mu C^T H_2 C\right) \cdot x(k) - \mu h_3 + \mu C^T H_2 y, \end{aligned} \tag{4}$$

starting from some $x(0)$ and continuing until the process (hopefully) converges to some $\bar{x}$. Here (and later), it is assumed that I is the identity matrix of compatible dimension. If the step size μ is conservative enough, the minimum will finally be found. Intuitively speaking, if it is assumed that x represents the candidate solution to the problem, the above algorithm gradually turns the solution vector towards the actual optimum. If the search space is limited to the first (hyper)quadrant, so that all variables are positive, the optimization algorithm can be written as

$$x(k+1) = f_{\text{cut}}\left(\left(I - \mu H_1 - \mu C^T H_2 C\right) \cdot x(k) + \left(\mu C^T H_2 y - \mu h_3\right)\right). \tag{5}$$

In what follows, the vector h_3 is not explicitly needed and it will always be a zero vector; the iteration becomes

$$x(k+1) = f_{\text{cut}}\left(\left(I - \mu H_1 - \mu C^T H_2 C\right) \cdot x(k) + \mu C^T H_2 \cdot y\right). \tag{6}$$

Whether or not the global optimum will finally be reached is very much dependent of the properties of the matrices H_1 and H_2; problems may arise if they are not strictly positive definite.

Comparing (5) to (1), one can see that they have the same structure. Indeed, one can now select

$$A = I - \mu H_1 - \mu C^T H_2 C \qquad \text{and} \qquad a = \mu C^T H_2 y, \tag{7}$$

and let the process starting from some $x(0)$ converge, to have some "inner image" emerge. The cognitively relevant problem settings just need to be reformulated as quadratic optimization problems, and one is immediately capable of determining a dynamic process from which the solution emerges!

In the following discussions, different kinds of mentally (more or less) plausible functionalities will be presented in the above optimization framework. In addition to offering a higher-level view of the qualitative effects of a complex iteration process, this framework offers also other conceptual benefits: The cost criterion formulation makes it easy to combine different criteria by weighting them appropriately.

3 TOWARDS "COGNITIVE OPTIMIZATION"

When the above optimization approach is applied to cognitive problems in the framework that was discussed in [7], one can notice that the role of the first term in (2) is to keep the elements in the "inner image" $\bar{x}$ low. The second term tries to make the inner and outer images to match, so that there would hold $y = Cx$ for some given chunk matrix C. Matrices H_1 and H_2 can be used to determine the weights of different factors.

The criterion (2) is purely quadratic, and that is why such simple linear gradient formulation is found. However, in [7] it was assumed that the mappings are nonlinear, so that $y = f_{\text{cut}}(Cx)$. For a moment, let us study the properties of this "cut" function. Calculate its gradient when it is applied to an m dimensional vector function:

$$\begin{aligned}\frac{d}{dx}\left(f_{\text{cut}}\left(f(x)\right)\right) &= \frac{df}{dx}(x) \cdot \frac{df_{\text{cut}}}{dx}\left(f(x)\right) \\ &= \begin{pmatrix} \frac{df_1}{dx_1}(x) & \cdots & \frac{df_m}{dx_1}(x) \\ \vdots & \ddots & \vdots \\ \frac{df_1}{dx_{n_x}}(x) & \cdots & \frac{df_m}{dx_{n_x}}(x) \end{pmatrix} \cdot \begin{pmatrix} f_{\text{pos}}\left(f_1(x)\right) & & 0 \\ & \ddots & \\ 0 & & f_{\text{pos}}\left(f_m(x)\right) \end{pmatrix}.\end{aligned}$$

Above, the former part is the standard Jacobian, whereas the latter part stands for the derivative of f_{cut}: This has been expressed using the unit step function

$$f_{\text{pos}}(x) = \begin{cases} 0, & \text{if } x < 0, \text{ and} \\ 1, & \text{if } x > 0. \end{cases} \tag{8}$$

For example, it is easy to show that there holds

$$\frac{d}{dx}\left(f_{\text{cut}}\left(Cx\right)\right) = C^T \cdot \frac{df_{\text{cut}}}{dx}\left(Cx\right) \tag{9}$$

and

$$\frac{d}{dx}\left(y - f_{\text{cut}}\left(Cx\right)\right) = -C^T \cdot \frac{df_{\text{cut}}}{dx}\left(Cx\right). \tag{10}$$

Further, if the cost criterion were defined in a consistent manner, matching with the discussions in [7] as

$$J'(x) = \frac{1}{2} \cdot x^T H_1 x + \frac{1}{2} \cdot \left(y - f_{\text{cut}}(Cx)\right)^T \cdot H_2 \cdot \left(y - f_{\text{cut}}(Cx)\right) \tag{11}$$

its gradient would be

$$\frac{dJ'(x)}{dx} = H_1 x - C^T \cdot \frac{df_{\text{cut}}}{dx}\left(Cx\right) \cdot H_2 \cdot \left(y - f_{\text{cut}}(Cx)\right), \tag{12}$$

and the gradient algorithm for finding the minimum would read

$$x(k+1) = x(k) - \mu H_1 x(k) + \mu C^T \cdot \frac{df_{\text{cut}}}{dx}\left(Cx(k)\right) \cdot H_2 \cdot \left(y - f_{\text{cut}}(Cx(k))\right). \tag{13}$$

However, the resulting dynamic systems in this case cannot be expressed in the assumed simple form (1); this is a major problem in the adopted framework where extreme simplicity was taken as the guiding principle.

To make it easier to rewrite the cognitive tasks using the optimization formulation, and to circumvent the linearity problem, an extended and more structured view of (2) can be given in the framework of *constrained optimization.* Study the linear/quadratic programming problem

$$\begin{cases} \text{Minimize} & \frac{1}{2} \cdot x^T H_1 x + h_3^T x \\ \text{when} & Cx = y \quad \text{and} \quad x \geq 0. \end{cases} \tag{14}$$

If $H_2 = \eta \cdot I$, and additionally $H_2 \gg H_1$ (somewhat loosely speaking), solving the unconstrained problem (2) with the presented data structures approximately solves (14). The hard limits have been "smoothened"; however, many of the applications that will be discussed later can be formulated so that in optimum the zero penalty is reached exactly, all terms in the cost criterion being zeros. This makes it easy to check the status of convergence, and whether the constraints have been met.

Using *slack variables* it is easy to implement constraints of the form $y - Cx \leq 0$, and also relax the constraint $x \geq 0$: With the cost of higher dimensionality, entry x_i, for example, can be divided in two parts, $x_i = \xi_i^+ - \xi_i^-$, where both variables ξ_i^+ and ξ_i^- are non-negative. The dilemma of $y = f_{\text{cut}}(Cx)$ can also be avoided — solve the optimization problem $y - \zeta = Cx$ for x and ζ, applying no cost for ζ.

From the point of view of real mental processes, there are some points about the above optimization approach that deserve closer analysis. Assuming that the data structures like C in (1) are known, complicated matrix functions (matrix transpositions and multiplications) are needed to find the formulas for iterating x. Physiologically, how could such manipulations be explained in a credible way?

It needs to be noted that the negative gradient direction is not the only option, it is just the (locally) optimal one; the formulation (4) can be relaxed. Assume that one defines the update law as $x(k+1) = x(k) + \Delta(x(k))$, so that the matrix $\Delta(x)$ indicates the adaptation direction. Assuming smoothness of the optimality criterion, it can be

noticed that when such a step is taken (if it is short enough) the cost criterion decreases only if the projection of the update direction onto the gradient vector is negative. That is, according to the Lyapunovian stability theory, the minimum will finally be reached if there always holds

$$\Delta^T(x) \cdot \frac{dJ(x)}{dx} < 0. \tag{15}$$

What comes to neural plausibility of the presented optimization scheme, one more comment is in place: It can be claimed that, indeed, in some cases it can be implemented even if only the Hebbian – Anti-Hebbian adaptation capability of the neuronal machinery is assumed. Looking at (6), and comparing it to the results in [7], it is evident that *if* there holds $H_1 = \mathbf{0}$ and $H_2 = I$, the knowledge representations coded in C can readily be implemented. Note that the matrix C^TC that needs to be calculated is a sum of the outer products of the individual rows in C; interpreting these rows as "virtual data vectors" that are input in the Hebbian – Anti-Hebbian system, the "virtual covariance matrix" that will be constructed will be $1/n_y \cdot C^TC$. This determines A, and when B is selected simply as C, all data structures that are needed in (6) are readily available (for details, see [7]). Explicit matrix multiplications can be avoided altogether.

From the pragmatic point of view, it seems that the optimality criteria constructed in the ways described later sometimes result in rather badly conditioned optimization problems and one could perhaps think of other iterative optimization schemes instead of the straightforward gradient descent method. However, it seems that the more efficient methods cannot easily be expressed in the form (1).

4 EXPERIMENTS

As an example of how to utilize the above formulation, study the determination of *maximum;* that is, find the location of the largest-valued entry in the vector x. It turns out that this problem is solved when one selects

$$H_1 = \mathbf{1} - I, \quad H_2 = 1, \quad C = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}, \quad \text{and} \quad y = 1. \tag{16}$$

Above, $\mathbf{1}$ stands for a matrix consisting solely of ones; this means the H_1 is a matrix having zeros on the diagonal and ones elsewhere[2]. If there is only one non-zero entry in x, this term causes no penalty; the role of H_2, C, and y is to keep the total sum of entries in x have numeric value 1 (so that there is only one constraint equation). The linear weighting h_3 is not needed so that it is a zero vector (even though this could be used to explicitly emphasize the numeric values). Finally, if a zero-cost solution is found in iteration, there must be a unique 1 in $\bar{x}$ — if the update step length μ is not too long, the search process assures that it is located where the maximum element of $x(0)$ was located. If there are exactly equal entries in $x(0)$, the process may get stuck in a saddle point.

Note that the search for the best matching unit, or "winner", is the most complicated task when implementing the Kohonen self-organizing map algorithm [9]; this means that

[2]This means that H_1 is *not* positive definite! However, as the entries in x remain strictly positive, x^TH_1x remains always non-negative. This is yet another (mathematical) motivation for using f_{cut} in the formulations: It takes care of conditioning otherwise pathological optimization problems and stabilizes the corresponding optimization processes

the above iteration can be utilized to implement the SOM in an emergent fashion[3]. Some additional control structures are needed around this kernel to carry out the higher-level iterations through the training data set; such iteration structures can be implemented as presented in [8].

Another neural network structure that can readily be emulated when the iteration for finding the largest vector element is available is *Hopfield net:* This network structure can store and associatively recall patterns (see [2]). Assume that there are (positive-valued) pattern vectors C_1 to C_N, where the number of patterns is lower than the dimension of the vectors, $N \leq n$. Further, assume that the patterns are normalized to equal length and collected as columns in the matrix C. Given $\tilde{y}$, a noisy version of some of the pattern vectors, the original pattern can be restored using associative recall as follows:

1. Calculate the correlations against all patterns, $x(0) = f_{\text{cut}}(C^T \cdot \tilde{y})$.
2. Apply the iteration having the data structures as in (16), searching for the maximum element in $x(0)$; the unique "1" in the converged state vector $\bar{x}$ reveals the appropriate pattern index.
3. Restore the pattern as $\hat{y} = f_{\text{cut}}(C \cdot \bar{x})$.

5 MATCHING OF FEATURES

Assume that an observation y, a high-dimensional real-valued vector has been received, and one would like to decompose it into features (chunks); further, assume that the feature prototypes C_i are packed into the matrix C. These features (in this context being high-dimensional, real-valued vectors) can be determined explicitly (for example, using some specialized definition formalisms [5]), or they can be extracted from data (for example, using the *GGHA algorithm* [4], or using *independent component analysis ICA*); these issues are not concentrated on here.

Assume that the vector x contains the contributions of each of the features in the observation vector. Our goal is to determine this vector x so that the observation would be explained as well as possible, $y = Cx$. Typically, when the dimension of x is lower than that of y, no exact match can be found. The general solution to this problem is *pseudoinverse* $x = C^{\dagger}y$, in special cases (if the basis C is orthonormal, that is, if $C^TC = I$) reducing to $x = C^Ty$. However, physically motivated features seldom are orthogonal: The features correlate with each other, and more complicated approaches are needed.

Luckily, the pseudoinverse solution can be characterized also as an optimization problem: The pseudoinverse minimizes the length of the "unexplained" residue remaining after the optimal fitting of the vectors y and Cx. This reconstruction error is defined as $e = y - Cx$, and the square of its length is

$$e^Te = (y - Cx)^T (y - Cx) . \tag{17}$$

Minimizing this can be achieved when one selects the matrices in (4) or (5) as $H_1 = \mathbf{0}$ and $H_2 = I$. The weighting matrices can also be defined in other ways: For example,

[3]Simple implementation of the "emergent self-organizing map", implemented in `Matlab`, together with other examples explained in this paper, is available through Internet in the address `http://www.saato.hut.fi/hyotyniemi/publications/02_step_1/`

letting H_2 be only semidefinite, some constraints (or their combinations) are not weighted in the criterion. This means that *associative search* (or smoothing, etc.) can also be implemented. In some cases, if the features are almost parallel, it may be reasonable to put some weight also on robustness; this can be achieved if the variable size is also weighted, so that $H_1 = \alpha \cdot I$ for some scalar alpha. In fact, this regularization results in an approach that is known as *ridge regression* in multivariate theory. Indeed, this approach deserves to be studied a bit closer.

Assume that the features determine the distribution of individuals with different attribute values (weights of features) around the category center (now assuming this center to be located in the origin). Further assume that this distribution is multivariate normal, so that the observations are scattered around the center forming a Gaussian (high-dimensional) bell curve (this normality assumption should be well motivated, taking into account the *central limit theorem*). This distribution is characterized uniquely by its mean value vector (now zero) and the covariance matrix Σ. Using Bayesian theory, the maximum *a posteriori* probability for x is

$$\begin{aligned} p(x|y) &= \frac{1}{p(y)} \cdot p(x) \cdot p(y|x) \\ &= c \cdot \exp\left(-\frac{1}{2} x^T \Sigma_x^{-1} x\right) \cdot \exp\left(-\frac{1}{2}(y - Cx)^T \Sigma_e^{-1} (y - Cx)\right), \end{aligned}$$

where c is some constant. Natural logarithm of this gives the *log-likelihood function*

$$\ln p(x|y) = c' - \frac{1}{2} x^T \Sigma_x^{-1} x - \frac{1}{2}(y - Cx)^T \Sigma_e^{-1} (y - Cx). \tag{18}$$

Comparing this to (2), one can see that maximization of the log-likelihood can be carried out by the presented gradient algorithm minimization when one selects $H_1 = \Sigma_x^{-1}$ and $H_2 = \Sigma_e^{-1}$. This means that *a priori* information about the attribute variation and error distribution can be utilized. The cost is generally not zero; however, there are intuitive interpretations:

1. The first non-constant term in (18) is closely related to the *Hotellings T^2 statistic* that measures how well the observation can be explained *within* the model; it is also closely related to the *Mahalanobis distance.*

2. The last term is closely related to the *lack of fit Q* that measures how the observation cannot be explained as seen from *outside* the model.

To conclude, if the set of features C_i is given, applying the above adaptation one can see the appropriate reconstruction of the observation vector y, or the "scores" of C_i, emerge in the iteration process. If the features are not too much overlapping, rather high values of step size μ can be tolerated here. In the special case where the feature basis is orthonormal, the feature matching process can be carried out in one step with no iteration at all ($H_1 = \mathbf{0}$, $H_2 = I$, and $\mu = 1$).

Note that in (17) all features are assumed to be freely scalable; if one of the features, say c_0, has a special role determining the cluster center, always having the weight 1, it has to be taken care of separately. The features need to be rearranged so that the matrix C only contains the scalable features, and c_0 is separately subtracted from y; (17) is transformed into

$$e^T e = (y - c_0 - Cx)^T (y - c_0 - Cx). \tag{19}$$

6 LEARNING BY EXAMPLES

Learning that is based on examples, or being capable of seeing the underlying pattern without being explicitly told how, is a typical pattern recognition task that is routinely carried out by using, say, feedforward neural networks. What would make this look more "human-like" would be the capability of finding a *simple* explanation. Technically speaking, this simplicity can be defined compactly: *Use the minimum number of other constructs.* How this task can be carried out — and how an illusion of "intelligence" emerges thereof — was originally presented in [3]; here it is presented how the solution can be obtained using (1) without explicitly applying linear programming or SIMPLEX algorithm.

Assume that the matrix C contains concept definitions. Each row in C represents one concept ("concept" being used here in a rather technical sense), so that when the inner product between it and the "mental image" x is calculated, either 0 or 1 results, depending whether the concept applies in that case or not. Parallel "concept analyses" can be carried out by calculating Cx (or $f_{\text{cut}}(Cx)$). Typically, if the concepts are interdependent, the final analysis is found through iteration (1). External input, or observation, can be integrated in the analysis as presented in [7], so that concepts can be based directly on measurements coming from outside or on other concepts (in any case, finally reducing to functions of external inputs).

Now assume that a *new concept* is to be determined based on the earlier, pre-determined concepts, so that there are *examples* of the new concept, coded in vectors $y(t)$. These observed patterns $y(t)$ are processed so that the corresponding final states $\bar{x}(t)$ are found using the iterations presented earlier. Further, assume that some of these patterns are *positive,* so that they represent the new concept to be learned, and some of them are *negative,* so that the concept explicitly is *not* there (note that negative examples are just as crucial as positive ones when learning the new concept). This means that always when a positive pattern (denoted p) is presented, its inner product with the new (unknown) concept vector C_{new} must be at least 1, so that $C_{\text{new}}^T \bar{x}_{\text{p}} \geq 1$, and when a negative pattern (denoted n) is presented the corresponding inner product must be at most 0, so that $C_{\text{new}}^T \bar{x}_{\text{n}} \leq 0$. Collecting the positive pattern vectors in $\bar{X}_{\text{p}}$ and negative ones in $\bar{X}_{\text{n}}$, these can be written in matrix form as $C_{\text{new}}^T \bar{X}_{\text{p}} \geq \mathbf{1}^T$ and $C_{\text{new}}^T \bar{X}_{\text{n}} \leq \mathbf{0}^T$, where matrices $\mathbf{1}$ and $\mathbf{0}$ are compatible vectors consisting exclusively of 1's and 0's, respectively.

Now there are many fixed $\bar{x}'s$ in the matrices $\bar{X}_{\text{p}}$ and $\bar{X}_{\text{n}}$, and just one vector in C_{new} to be determined; the easiest way to proceed for determining C_{new} using the iteration (1) is to *invert* the roles of the data structures for a moment — note that one can write $C_{\text{new}}^T \bar{X} = (\bar{X}^T C_{\text{new}})^T$. The constraints determining the new concept can be expressed in the familiar form $Cx = y$ as follows:

$$\left(\begin{array}{c|c|c|c} \bar{X}_{\text{p}}^T & -\bar{X}_{\text{p}}^T & -I & \mathbf{0} \\ \hline \bar{X}_{\text{n}}^T & -\bar{X}_{\text{n}}^T & \mathbf{0} & I \end{array}\right) \cdot \left(\begin{array}{c} C_{\text{new}}^+ \\ \hline C_{\text{new}}^- \\ \hline \xi^+ \\ \hline \xi^- \end{array}\right) = \left(\begin{array}{c} \mathbf{1} \\ \hline \mathbf{0} \end{array}\right), \tag{20}$$

that is, the new y contains the correct (binary) classifications for all examples, $\bar{X}$ consists of the "conceptual deconstruction" of the examples, using prior, lower-level concepts, and C_{new} contains the relevances of these lower-level concepts when appropriately explaining the new one. Because the references to prior concepts can be positive or negative, C_{new} is

divided in positive and negative parts, and because it does not matter how much above 1 the positive classifications are, or how much below 0 the negative ones are, the slack vectors ξ^+ and ξ^- are included in the model.

When determining the other data structures in order to run the optimization (5), one can note that H_1 only needs to emphasize variables in C^+_{new} and C^-_{new}, the slack variables in ξ^+ and ξ^- having zero weight. How the weighting of these variables is implemented, is a question of design, and the results may be different if there are various ways of reaching minimum: For example, if only the diagonal is non-zero, alternative solutions are equally demonstrated in solutions, whereas if the non-diagonal entries have considerable weight, one of the solution with minimum number of distinct concepts is selected. It is also possible to put more weight on the lower-level concepts and specially on the direct observations, so that higher-level explanations would be preferred. If the examples have equal relevance, it is natural to choose $H_2 = I$.

When the definition for the new concept has been found, it can be included among the other concepts by augmenting the dimension ot the system matrix:

$$C \quad \leftarrow \quad \left(\begin{array}{c|c} C & \mathbf{0} \\ \hline (C^+_{\text{new}} - C^-_{\text{new}})^T & 0 \end{array} \right) \tag{21}$$

Finding a classification between examples and non-examples is very much like it is with *support vector machines:* Increasing the dimension of the problem description may make classes linearly separable, so that the classification problem becomes easily solvable. The key point here is to increase the number of concepts gradually, so that the new concepts can be distinguished using the available concepts in a hierarchic fashion using the prior concepts. Iterative learning is another thing: After the preliminary concept prototypes have been determined, higher level concepts can be recirculated and utilized to find more sophisticated and efficient concept descriptions; in such cases, the matrix C does not remain triangular with zero diagonal. The cost of having efficient (expert-like) representations is that the originally sequential (declarative) reasoning always lasting only a finite time becomes a potentially very time-consuming pattern matching process (see previous section).

7 BACKWARD INFERENCE

Perhaps the most convincing example concerning the power of the proposed framework is concentrated on next. The claim here is that (a subset of) logic programming problems can be expressed and solved in the quadratic programming framework. There are restrictions, coming from the fact that logic programming formalisms have the power of the Turing machines whereas quadratic programming is more restricted. In concrete terms, the restrictions are reflected as the limited size of the Herbrand universe: The set of constants has to be limited to some predefined level. First we will study proposition logic, and extend the discussions to predicate logic.

The modern logic programming formalisms like Prolog are based on the *Aristotelian syllogism,* meaning that a set of inference steps can be truncated:

$$\begin{array}{ll} & \mathrm{A} \rightarrow \mathrm{B} \\ \wedge & \mathrm{B} \rightarrow \mathrm{C} \\ \hline \Rightarrow & \mathrm{A} \rightarrow \mathrm{C}. \end{array}$$

Using the properties of the logic connectives, it can be shown that the logical content remains unchanged if the above reasoning is written in the following form:

$$\begin{array}{ll} & \neg \mathrm{A} \vee \mathrm{B} \\ \wedge & \neg \mathrm{B} \vee \mathrm{C} \\ \hline \Rightarrow & \neg \mathrm{A} \vee \mathrm{C}. \end{array}$$

What is the most interesting issue about the above formulation is that the somewhat uneasy feeling of "causality" has been ripped off, the original information being expressed in a "non-directional" form. Typically, the rules in a rule base are of a more complex form, so that, for example, $\mathrm{A}_1 \wedge \cdots \wedge \mathrm{A}_n \rightarrow \mathrm{B}$; however, using *de Morgan rules,* they can still be simplified: One has $\neg(\mathrm{A}_1 \wedge \cdots \wedge \mathrm{A}_n) \vee \mathrm{B}$, and further, $\neg \mathrm{A}_1 \vee \cdots \vee \neg \mathrm{A}_n \vee \mathrm{B}$.

Using the *resolution principle,* reasoning in the Prolog systems goes as follows. Assume that one wants to know whether some logical clause can be deduced, given a consistent knowledge base. One first inserts the *negated goal clause* among the other clauses, and starts applying elimination steps of the type shown above. If it turns out that the negated expression contradicts the other rules, that is, an "empty clause" can be deduced, the goal is reached; then it must be so that the non-negated clause must be deducible.

To implement this kind of reasoning in a mathematical form, we can first notice that the logical values of syllogisms can now easily be operated on — the table below is consistent with the original deduction, showing only the logical values:

$$\begin{array}{lrrr} & \mathrm{A}: & \mathrm{B}: & \mathrm{C}: \\ & -1 & 1 & \\ + & & -1 & 1 \\ \hline = & -1 & & 1 \end{array}$$

where 1 stands for "true" and -1 (now) for "false", value 0 meaning "not used". It seems that logic expressions and manipulations with them can be coded as linear algebra — this is elaborated on below.

Assume that the logical values of all propositions are collected in a vector p. Starting from some truth value distribution in $\neg p(0)$ (negation added because the inversion of the goal clause), if adding vectors representing the logic expressions results in zero vector, the mathematical correspondence of a contradiction has been reached! Assuming that the "rule vectors" are collected as rows in the matrix C and the weights, or "relevance values", or "activities", of the rules are collected in the matrix x, the effect of the combined rules can be expressed as Cx, so that after reasoning there should hold $\neg p(0) + Cx = \mathbf{0}$, or written in another way, $Cx = -\neg p(0) = p(0) = y$. And, looking again at (2), it turns out that the reasoning can readily be implemented in the form (1)!

There are some complications, though. In the numeric formulation, when rule vectors refer to variables, the truths are being "exhausted" as they are utilized — this is not the case when operating with true logical values. To circumvent this, x can contain numeric values other than -1, 0, or 1, and, correspondingly, the truth values other than -1, 0, or 1 are possible in p. The non-binary values simply have to be interpreted so that any positive value stands for "true", and negative values stand for "false"; if the rule base is non-cyclic, and if there is originally only one "1" in y (this can be achieved by an extra expression in the rule base, if necessary), no problems should emerge — it is clear, at least, that *clauses that are logically undeducible cannot be deduced in this framework either.* Another point that deserves attention is the logical structure: If a

rule vector is multiplied by a *negative* factor, in that case the logical contents of the rule changes altogether, and its validity is lost. That is why, the values in x must always remain non-negative — luckily, this positivity assumption is automatically fulfilled when implementing reasoning in the proposed framework.

When implementing *predicate calculus,* so that expressions can contain variables, the situation is much more complicated, and complete congruence between the formalisms cannot be reached. First, because of the "closed world" assumption, the infinite Herbrand universe, the search space, has to be truncated — all literals and their Skolem functions need to be enumerated and listed. What is more, each literal can be used only in one role: If there are non-unary predicates, for example, like "Parent(x,y)", the variables x and y having the same domain, the set of individuals has to be *duplicated,* so that x and y can be genuinely distinct. Of course, speaking of grandparents (as "Parent(x,y) $\wedge$ Parent(y,z)", the set of appropriate literals has to be *tripled.* This kind of duplication also applies to the predicates operating on the literals.

Because it is these variable bindings that are usually of the foremost interest when applying logic programming, there is need to include the literals also in the search space that only contained the weights for the rules in the above propositinal case. No doubt an example is needed here. Assume that the knowledge base consists of the following expressions:

P(a)
$\neg$P(b)
P(x) $\rightarrow$ Q(x)

There is only one "x" here, so that the atomic literals a and b need not be duplicated. The rules need to be rewritten:

a	Literal a
b	Literal b
a $\rightarrow$ P	Expression 1
b $\rightarrow -$P	Expression 2
P $\rightarrow$ Q	Expression 3

Every literal and every expression needs the "activity variable" of its own. Collected together, one has the variable vector

$$x = \begin{pmatrix} x_{\mathrm{a}} & x_{\mathrm{b}} & x_1 & x_2 & x_3 \end{pmatrix}^T .$$

The rule base can be written in a table form as

	a	b	P	Q
x_{a}	1			
x_{b}		1		
x_1	-1		1	
x_2		-1	-1	
x_3			-1	1

so that the variables x_i are the weighting factors for multiplying the rule vectors on the corresponding line. Now, assume that we want to know "When would Q be true?" This means that the objective is to find values for x_i so that the sum in column Q would be 1,

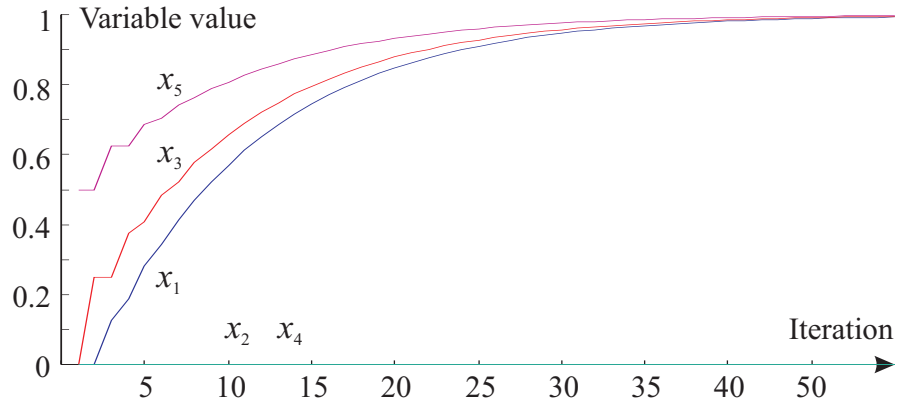

Figure 1: "Backward inference" implemented (see text)

and in all other columns the sum would be 0. To write the matrices in (5) appropriately, one can recognize that C can be constructed from the above table by simply transposing it, and, similarly, y represents the transposed goal:

$$C = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \tag{22}$$

The weighting matrices in the optimality criterion can be selected as

$$H_1 = \left(\begin{array}{cc|ccc} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right) \quad \text{and} \quad H_2 = I. \tag{23}$$

The interpretation of H_2 is clear; H_1 deserves closer inspection. Almost all entries in the matrix are zeros, because no matter how many times the rules are applied, there is no cost for them. The upper left corner, on the other hand, is of the form $\mathbf{1} - I$, meaning that if various literals of the same domain are simultaneously active, it decreases the validity of the solution; if there is only one active literal, there is no cost Assuming that there existed various sets of literals with different domains (with or without duplication), so that these literals would not be mutually exclusive, there should exist various such independent *blocks* of the form $\mathbf{1} - I$ in H_1. If the goal is achievable, zero cost can be reached. The results are shown in Fig. 1, having step size $\mu = 0.5$: It seems that variable #1 fulfills the condition, meaning that "a makes Q true (trough P)".

It is typical for this kind of problems that there are various minima for the criterion (2) — and optimization often becomes specially tedious because the process typically first finds the saddle points with zero derivative! This is the case, for example, if solving the question "When will Q be true?" in the following situation having two equally valid solutions:

P(a)
P(b)
P(x) $\rightarrow$ Q(x)

How to fix this problem? Setting explicit preferences, or selecting different non-zero values in the H_i matrices, should work, giving a unique solution and ignoring the other possibilities. The non-zero matrix elements can also be automatically randomized. At least for simple knowledge bases, the above scheme seems to work quite nicely.

It is interesting to note the transformations along the methodology development: The originally causal, in an obscure way time-bound reasoning process was first formulated in a static form — and after that, the static pattern matching problem was changed back into a dynamic process for solving it. It seems that dynamic thinking is truly useful when modeling complex phenomena; this observation is parallel with the view of cognition as consisting of *dynamic processes* with internal states. If the behavioristic view of direct mapping from input (observed environment) to output (observed behavior) is to be abandoned, one cannot ignore the role of *dynamic systems theory* in cognitive science.

8 CONCLUSIONS

In this paper, a framework was studied where different kinds of functionalities emerge from cumulating correlation calculations. According to the experiments, it can be claimed that this "sieve structure" [6] is the *simplest possible* general framework for implementing any algorithm, pattern matching task, logic inference, etc. And if this approach is the simplest, then, according to *Occam's razor,* there must also be some *truth* in it?

REFERENCES

[1] Chang, C.-L. and Lee, R.: *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[2] Haykin, S.: *Neural Networks. A Comprehensive Foundation.* Macmillan College Publishing, New York, 1994.

[3] Hyötyniemi, H.: *Correlations — Building Blocks of Intelligence?* In "History and Dimensions of Intelligence", FAIS Publications, Helsinki, 1995, pp. 199–226.

[4] Hyötyniemi, H.: *Constructing Non-Orthogonal Feature bases.* Proceedings of the International Conference on Neural Networks (ICNN'96), June 3–6, 1996, Washington DC, pp. 1759–1764.

[5] Hyötyniemi, H.: *Explorations in "Naturalistic Formalisms".* Finnish Artificial Intelligence Days STeP 2000, August 28–30, 2000, Vol. 3, pp. 123–131.

[6] Hyötyniemi, H.: *Complex Systems — Searching for Gold.* Arpakannus 2/2002, special issue on Complex Systems, pp. 29–34.

[7] Hyötyniemi, H.: *Studies on Emergence and Cognition — Part 1: Low-Level Functions.* Finnish Artificial Intelligence Conference (STeP'02), December 16–17, 2002, Oulu, Finland.

[8] Hyötyniemi, H.: *Towards Perception Hierarchies.* Finnish Artificial Intelligence Conference (STeP'02), December 16–17, 2002, Oulu, Finland.

[9] Kohonen, T.: *Self-Organizing Maps.* Springer–Verlag, Heidelberg, 1995.