
Numerical processing for cognitive tasks

Frédéric Alexandre¹ and
Hervé Frezza-Buet²

¹ LORIA-INRIA, BP 239, F-54506 Vandœuvre.

email: falex@loria.fr

<http://www.loria.fr/equipes/cortex/>

² Supélec, 2, rue Edouard Belin, F-57070 Metz.

email: Herve.Frezza-Buet@supelec.fr

Abstract

Cognitive modeling can be derived from expert knowledge integration or experimental data analysis. This chapter introduces these two main ways of modeling and concentrates on the latter aspect and the related numerical tools together with their possible role. More precisely, the evolutionary approach is presented for optimization and tuning, artificial neural networks for associative reasoning and the stochastic approach for planning. We then conclude on the need for a modular framework, integrating these numerical and other symbolic tools.

Introduction

Artificial Intelligence (AI) can be defined as the set of techniques that aim at implementing in computers models of human cognitive capabilities. The idea to design softwares including typical human cognitive capabilities can be justified for several reasons, depending on the considered domain of application. In the domain of cognitive sciences, of course, the goal can be the study of the human mind (or brain) itself. In this case, no need for other justifications. In other non specific domains like industrial or medical domains, the goal can be to include human factors in the processing loop. It can also correspond to add ergonomic constraints to the software design. In both cases, the point is that human knowledge, expertise or skills are generally present in the process. Accordingly, they must also be integrated in the modeling phases for the sake of plausibility. This is particularly the case with economy where, beyond statistics and other mathematical aspects, human factors play an important role and should be included in any complete analysis.

Now comes the question of the computational paradigm(s) to be used for such cognitive modeling. Two major paradigms - numerical and symbolic

- can be described and are closely linked to the history of AI and even of computer science itself. Indeed, it is interesting to note that the inception of computer science was related to the idea of building a machine emulating an artificial brain through numerical techniques. Related works can correspond to cellular automata by John von Neumann [33], artificial neurons by Warren McCulloch and Walter Pitts [24] and even the first artificial neural network, the perceptron, by Frank Rosenblatt [26]. Later, considering the difficulties to build such connectionist models and due to the development of logic and other formal methods in computer science, AI has explored the symbolic way, through the physical symbol system hypothesis [25], postulating that a system of manipulation of symbols can emulate any general purpose intelligent action. The famous expert systems have been derived from such an approach. Later, other developments including even more structured knowledge representation and manipulation have been proposed in the symbolic paradigm. Similarly, the numerical paradigm has been consolidated with many algorithms, mainly from operation research.

It is widely reported [11, 15, 16, 23, 29] that these two paradigms have complementary strengths and weaknesses. To sum up, the symbolic paradigm is efficient for formal aspects like reasoning, explanation and structured symbol manipulation whereas the numerical paradigm offers such abilities as learning, real world data processing and noise tolerance. Readers are referred to the above mentioned references for more details about these complementary profiles and about the long and difficult debate in the AI community to decide for the best paradigm. As a result, the community was divided into two parts, each centred on one paradigm, until recently when the Neuro-Symbolic Integration (NSI) research area has emerged [29].

It is proposed in [23, 29] that the two AI paradigms address in fact two different aspects of human cognition, namely analytic operations for the symbolic paradigm and synthetic operations for the numerical paradigm. The former operations are concerned with high level, conscious reasoning tasks whereas the latter correspond to low level, unconscious perceptual tasks. From this certainly too oversimplified view, the goal of NSI is to propose ways to combine both paradigms and hence cover the whole spectrum of human cognitive capabilities. In that direction, two major strategies have been proposed. The hybrid strategy [17] is a framework that proposes tools and techniques to combine classical symbolic and numerical modules and make them cooperate. This elementary strategy is particularly suitable for industrial applications (see for example [2]), where numerical databases must be exploited together with a priori known and formalized expertise.

The unified strategy [1] wishes to use only numerical techniques and proposes to extend their properties toward the symbolic analytic side [3]. In its definition itself, the unified strategy is more general and more powerful than the hybrid strategy. It seems also more adapted to such fields as economy, where expertise and knowledge are too often unconscious and hard to formalize precisely and where numerical formalism and adaptive properties are

particularly suitable. Now comes the problem to be able to identify numerical models candidates to cognitive modeling in economy and to see how they can be linked together in a more global framework. The goal of this paper is to present such numerical models and to discuss their possible articulation.

1 General presentation and justification

We have just explained above why we believe that numerical AI models are especially interesting as one wants to model human cognitive abilities related to the field of economy. We now propose to give an overview of several important classes of numerical models and mention accordingly possibly related cognitive abilities. The overview is structured in three parts, respectively corresponding to the evolutionary approach, artificial neural networks and the stochastic approach. Other existing numerical approaches, like statistics for example, will not be described here. Before describing these techniques in the forthcoming sections, we briefly relate them to cognitive abilities.

1.1 The evolutionary approach for optimization and tuning

This point does not really correspond to a cognitive operation per se. It arises from the following observation: All the considered numerical models are in fact classes of models that need to be adapted to a given task. Considering complex tasks, it is also clear that several numerical models need to be integrated and linked one with the other to emulate such tasks. As a consequence, specific algorithms can be envisaged to allow such architectural and parametric tuning, in order to obtain a more optimized system. Such algorithms, inspired from the evolution process, will be presented in Section 2, together with the more general operation research approach (Subsection 2.1).

1.2 Artificial neural networks for associative reasoning

Most basic cognitive operations can be labelled with the term “associative reasoning”. Such low-level cognitive operations are generally perceptual tasks. The cognitive effort is to learn to *associate* perceptual fragments to actions, which are either external actions on the real world or internal actions, corresponding to access to internal representations. It is important to underline here that this learning is carried out through data from the real world and not with a priori models of the world (constructive approach).

From this general definition, several cases can be encountered. First, the goal of the learning can be to extract regularities of the external world, from examples. The task is then to perform generalization, to define new prototypical examples built from a large set of real examples. On the opposite, the goal of the learning can be to memorize particular cases, to store them as

prototypes without modification and to be able to recall them when a similar case occurs. In the following, we will refer to this dichotomy as learning with generalization or learning particular cases. It is interesting to notice that this dichotomy is also mentioned in human memory function and neuronal substrate, respectively as procedural and episodic memory [28].

Secondly, the way of learning can also depend on the criterion that has to be minimized in order to improve learning. The criterion can be explicitly given by the external world, as a measure of error for example. This corresponds to supervised learning. Otherwise, the criterion can be explicitly but internally evaluated, as a degree of confidence or a prediction. This corresponds to semi-supervised learning. Otherwise, no explicit criterion can be given and the goal can be only to find regularities in the external world, without any guideline. In this case, an implicit criterion can be determined and depends on the way defined to compare two fragments of perception (measure of distance for example). This corresponds to unsupervised learning.

In Section 3, we will present numerical models, namely artificial neural networks, able to perform unsupervised learning with generalization (Subsection 3.1), unsupervised learning of particular cases (Subsection 3.2), supervised learning with generalization (Subsection 3.3) and semi-supervised learning with generalization (Section 4).

1.3 The stochastic approach for planning

Of course, beyond these elementary capabilities, it is also important to be able to temporally organize one's behavior, i.e. to decide at any moment which actions are to be triggered, on the basis of the status of the world and of past experience. Such a reasoning, called planning, has been extensively studied in symbolic AI, but, in this case, a model of the world has to be known a priori. It has also been studied in the framework of biologically inspired cerebral modeling [9]. Below, Section 4 will present a stochastic numerical model for that task.

Beyond this important kind of reasoning, temporal processing is more generally encountered in cognitive modeling. Subsection 3.4 summarizes some numerical techniques able to take time into account.

2 The evolution analogy

The evolution process described by the neo-Darwinist conception of genetics is believed to be responsible of the amazing adaptation of biological species to their world. The current understanding of this process leads to the design of computational models to solve many kinds of problems, that are sometimes very far from biology. This generic computational approach is called Genetic Algorithms (GA), and it can be nowadays considered as a particular class of algorithms by itself, with regard to its initial biological inspiration. As GAs are

a particular case of more general operation research algorithms, the common properties of operation research techniques will be discussed before detailing GAs.

2.1 Operation research approach

Operation research is a wide field in Computer Science that is concerned with finding as fast as possible an as good as possible solution to complex problems. Let us take the very classical example of the Traveling Salesman problem. This problem consists in determining an ordered list of the n given towns so that visiting successively the towns in that order leads to the shortest travel. For this problem, the *search space* is the set of all possible lists of n towns, and it has $n!$ elements. The *optimization function* is a scalar value that can be assigned to each element of the search space. In our example, it is the total length of the path that successively visits the towns in one list. A typical operation problem consists in finding among all elements of a search space the one for which the optimization function returns an optimal value. One simple solution is to test all elements in the search space. Nevertheless, in all interesting cases, the search space is very large and it may take centuries to explore it. The purpose of operation research is to find methods (called heuristics), in order to explore subparts of the search space in a reasonable time. Exploring only few elements but in a clever way can allow to find a good element, but, on the other hand it cannot be assured to be the best one. Hence, the philosophy of operation research is that finding a reasonable solution in a reasonable time is better than finding the optimal solution in an infinite time. This is particularly linked to the relation between the size and the complexity of the problem and particularly true for problems where the complexity increases in a non polynomial way as a function of the size of the problem (so-called NP-hard problems).

To describe operation research, the following analogy is convenient. Consider the search space as a bi-dimensional set, each element being on an horizontal plane. Let us consider the value of the optimization function at each point as the height of this point. This defines hills where function is high and valleys where it is low. The problem is to walk in this landscape to find the lowest place (or the highest, depending if the goal of the optimization is the minimization or the maximization of the function; in any case, the problem is the same). The shape of the landscape depends on the problem to be solved, and is crucial to determine the best way to explore it. For example, if the landscape for a given problem is just a single smooth hill, one just has to take a random point, compute its value with the optimization (say minimization) function and also compute the slope (gradient) at this point. Just make a step down the steepest direction from there, and iterate that computation until being down the hill (null gradient). This method is called *gradient descent* and is used in many algorithms, as in the backpropagation learning algorithm mentioned in Subsection 3.3.

The problem raises as soon as the landscape contains several hills, because the lowest region must be found before moving using a gradient technique, otherwise a local minimum will be found. Here stands one of the central problems in operation research, the *exploration versus exploitation* conflict. When considering one point, one has to choose either to follow the gradient, in order to go down the hill and improve the solution (exploitation), or to choose another point randomly (exploration), allowing to find a possibly better hill. One classical way to do this choice is the *simulated annealing*, inspired from physics of crystals [20, ?]. The exploration process depends on a decreasing parameter, the temperature, that is initially high and favors random exploration and decreases for progressively favoring exploitation. This kind of decreasing parameter may be found in many fields of operation research, and in many techniques presented in this paper.

Last, when hills are numerous or very sharp, solving the problem with operation research techniques may become difficult unless specific properties of the landscape are known. This is actually the case with Genetic Algorithms.

2.2 Genetic Algorithms

A good description of GAs can be found in [12]. The design of a GA consists in first designing a code for each element of the search space. In our traveling salesman example, this code may be directly a permutation of the n towns, representing the order in which these towns are visited. GAs operates on codes in the following way. First randomly choose p codes. This defines a population of p individuals, that represent p corresponding points in the search space. GAs consist in looping the execution of three stages on that population: reproduction, crossing over, and mutation, under the control of a fitness function.

The fitness function

Each individual is given a *fitness value* according to its code (the code of an individual is called *chromosom*). The GA will allow to simulate evolution processes to increase the fitness of individuals in the population. Hence, to solve the problem, the fitness may be related to the optimization function.

In our example, the fitness of an individual can be computed in the following way. From the chromosom (the code) of this individual, i.e. from the list of towns, the path followed by the traveling salesman can be built, which defines one possible way to visit the towns, i.e. one element of the search space. From this representation, we can compute the optimization function, i.e. the length l of this path. The fitness f of the individual may be given here by $f = L - l$, or $f = 1/l$, or any other decreasing function of l . Nevertheless, in other problems, defining a fitness function may be much more critical and difficult to do.

Reproduction

The purpose of reproduction is to determine from population p a new population p' with the same number of individuals, and whose average fitness is better. Each individual of p' is obtained by selecting an individual in p and making a copy of it. The probability for an individual in p to be copied in p' increases with its fitness. Thus, p' contains several copies of the best individuals in p .

Crossing-over

The second stage consists in making random couples of individuals, and applying with probability p_c the crossing-over transformation of the individuals in each couple. Crossing-over consists in exchanging pieces of codes of the two individuals. In our example, the codes are lists of towns, and crossing two individuals can be performed by selecting a rank k randomly, and exchanging tails of the lists from k of both individuals (and find a tricky way to make each town appear once in both lists). This example is pedagogical, see [12] for a real resolution of this problem by GAs. Nevertheless, in other problems, such consistency constraints can be very difficult to obey.

Mutation

The last stage consists in randomly modifying each individual chromosom with probability p_m . In the traveling salesman example, mutation can be a permutation of two random items in the list. The mutation operator ensures that every element in the search space can be theoretically tested.

2.3 Discussion on GAs

It has been demonstrated that a GA with no crossing-over ($p_c = 0$) and a slowly decreasing mutation probability p_m is a generalized simulated annealing algorithm [6]. The new point with GAs is the crossing-over stage. Crossing-over can be viewed as a specific way to explore the search space, exploring a point obtained by recombining previously explored ones. Thus, GAs are dedicated to problems that can be solved by combination of smaller parts. The traveling salesman problem is of that kind. An optimal way to visit a small subset of the towns can be used as a basis for building a solution. Such small good subparts of solution, that must be visible and exchangeable at the level of code and crossing-over, are called *building blocks* and are computed in parallel by the population of individuals managed by the GA [12].

GAs can be applied as well on symbolic problems as on numerical ones. It can also be applied indirectly on programs that allow to build the solution of the problem. This latter method is called Genetic Programming, and is closer to biology (DNA can be roughly viewed as a program interpreted by

ribosomes). It has been successfully applied in evolving symbolic trees, from which an incremental process builds neural networks that control the behavior of an artificial ant [14, 21].

GAs are powerful methods to explore possible solutions of complex problems, requiring that this problem can be solved by recombinations of good subparts of solutions. Finding an optimal symbolic program is a problem of that kind. Therefore, Genetic Programming may allow to apply GAs to large classes of problems where a solution can be expressed as the result of a program, whereas solutions themselves have not directly the required recombination property. Nevertheless, as many operation research techniques, GAs may require tuning parameters empirically, finding a suitable code and an assorted crossing-over operator, that may lead to very ad hoc and not reusable methods to find a solution for the problem.

3 Artificial neural networks

Before presenting several Artificial Neural Network (ANN) models, some features shared by all the models are presented. ANNs can be defined as networks of simple interconnected units, performing numerical computations from their inputs to evaluate their output that will be integrated by other units. Three features allow to define an ANN. The functioning rules define the computation performed by units, the learning rules define the way internal parameters (and especially the weights of the connections) are adapted during learning phases and the architecture defines the way units are connected one with the others. Choosing various rules and architectures allows to define several kinds of ANNs and accordingly several properties will emerge from their functioning.

3.1 Distribution mapping

A large field in neural networks techniques (and related non connectionist statistics) is referred to as *unsupervised learning*, meaning that no supervisor tells the system what it has to learn. What can you learn from an observed phenomenon in such cases? You can first learn it by heart, storing explicitly each occurrence of the phenomenon. This will be presented in Subsection 3.2. Secondly, you can try to evaluate the probability distribution of the examples and for example simply compute the mean of the phenomenon, avoiding huge storage but losing many elements of information. In both extreme cases, very poor information concerning the phenomenon is extracted, particularly if the distribution is not homogeneous. Extracting relevant information in the framework of unsupervised distribution mapping consists in finding a method between the two extreme cases previously mentioned, in order to map the distribution of the phenomenon with the appropriate resolution.

Components of neural unsupervised learning

For illustration, let us assume that the examples are humans, defined by their size and weight only. The two extreme cases presented above share the fact that what is stored has the same nature as one occurrence of the phenomenon. Learning by heart consists in storing all humans, and the mean value is still a human (that may actually not exist), with a weight and a size. Hence, the components of unsupervised neural learning are a set of possible occurrences (called *prototypes*) of the phenomenon (a set of humans each having a weight and a size). In the distribution mapping framework, several prototypes, defined by neurons, are used and are adapted to best match the probability distribution of the phenomenon.

Learning mechanisms

Learning with a set of neurons, with prototypes, consists in presenting one occurrence of the phenomenon (one particular human), comparing it to the prototypes of the neurons (which is possible because prototypes and the phenomenon have the same nature), and modifying some prototypes according to the presented occurrence. This occurrence is called an *example*. The techniques differ one from the others according to the way prototypes are modified consecutively to the presentation of the example.

All distribution mapping unsupervised learning algorithms roughly share the following stages. First, an occurrence of the phenomenon is presented to the network of neurons. Then, the neurons compute a competition, where neurons whose prototypes are close to the examples are favored. After the competition, the best matching neurons and few others learn, i.e. they make their prototype closer to the presented example.

Kohonen's Self Organizing Maps [22]

A Kohonen's Self Organizing Maps (SOM) is a set of neurons having lateral connections with the shape of a rectangular grid (this is the usual case, but it is not restrictive). When an example is presented, the neuron whose prototype is the closest to the example learns, i.e. makes its prototype be closer to the example, but also neurons that are close to it on the grid, whatever the fitting of their prototypes to the example. This mechanism ensures the conservation of the topology of the phenomenon, because prototypes of neighboring neurons of the grid correspond to close instances of the phenomenon.

Fritzke incremental Networks [10]

The problem (and sometimes the advantage) of SOMs is that the topology of the network is arbitrarily defined, and is not related to the inner topology

of the phenomenon. The network has to force its topology to map the topology of the phenomenon distribution. Fritzke defined algorithms that involve initially very few neurons, and make the number of neurons increase until the existing neurons represent the distribution of the phenomenon correctly. Some of these techniques (growing grids) are resizeable Kohonen grids, allowing to find the best size of the grid to map the phenomenon. Some other techniques, the Growing Neural Gas, build topology of the network according to the distribution of the phenomenon.

Discussion on distribution mapping

The purpose of distribution mapping unsupervised learning mechanisms is to represent a phenomenon with only few well chosen prototypes, related one to the others consistently with the topology of the phenomenon. Topology preservation is not provided by usual statistical methods like k-means. After learning, one can label the prototypes with classes and use the network for classification, considering prototypes instead of the phenomenon for compression, or use it for many other kinds of analysis. We would also like to underline here the fact that grid shaped SOMs are a model of the cortical circuitry, where neurons are connected in a surface and perform some competition and self organization. The ability of unsupervised learning to represent in a “clever” way a complex phenomenon with relatively few prototypes seems fundamental in human cognition.

3.2 Memorizing examples

Examples can be memorized (learning by heart) with ANNs among which the Hopfield model [18] is the most famous. This model defines an entirely connected (or recurrent) ANN, each neuron receiving an elementary piece of information from the example to memorize. Connections between neurons are adaptative and compute the correlation between neuron activities. Presenting a set of examples will make the ANN memorize all the corresponding configurations. Later, starting from any initial configuration, the network will converge toward the closest memorized configuration, i.e. example. It can be noticed that this ANN has also been adapted to perform optimization tasks [32].

3.3 Supervised learning

Supervised learning can allow for pattern matching and function approximation. In both cases, the input (a pattern or attributes of a function) is presented to an input layer of neurons. The output (the corresponding class of the pattern or the value(s) of the function) is computed on an output layer of neurons and compared to the known desired output, thus yielding the evaluation of an error. This error is the basis on which weights are adapted. If the

input and output layers are directly connected, the ANN is a perceptron [26] and very simple learning rules, like the Widrow-Hoff rule [36], can be used, but only simple linear functions can be learned. Adding intermediate hidden layers of neurons allows to approximate, with any accuracy, any complex function. The so-called multilayered perceptron can be trained with a more complex learning algorithm, the backpropagation learning algorithm [7].

Multilayered perceptrons are today the most famous ANNs. They have been applied to a wide class of problems in many application domains. Various adaptations have also been proposed to improve the model and to extend its properties (see for example Subsection 3.4 for adaptation to temporal processing).

3.4 Temporal computation

A neuron in the brain is a chemical system, behaving with its own dynamics, as every physical systems. The formal neuron by McCulloch and Pitts [24], that underlies many neurons in usual ANNs, models the neuron without focusing on this temporal aspect. So usual neural techniques don't fit well for the management of information for which time is crucial, even if recent approaches like the modeling of spiking neurons aim at dealing with such problems.

Computing time with MacCulloch and Pitts formal neuron

One way to deal with time with a neural paradigm is to use the formal neuron, even if it doesn't take time into account. Two main strategies have then been employed. The first one is to consider a time window for observation of the process to analyze, and compute as if time was a supplementary spatial component. The idea is to work with chronograms as if they were snapshots of a fully spatial event. This has been successfully done for speech processing with the TDNN network [34]. The second strategy consists in using a perceptron, with additional connexions used as delay lines, allowing to copy past activities of neurons and hence to include them in the computation. These networks are referred to as recurrent neural networks, among which the most famous are the Elman [8] and the Jordan [19] models.

Modeling time properties of real neurons

Some authors, and especially Grossberg [13], have investigated the design of neural models that include the dynamics of the biological neuron. This leads to models where the neuron is driven by a set of time differential equations, representing neuro-transmitter regeneration and release, or many other metabolic properties. We have discussed in detail such models and their role in artificial cognitive processes in [9], and we refer to that paper for references. The main idea of such techniques is that activities (input signals usually) are

not reliably received by the neuron, because of a transfer function. The result is that when a signal occurs, the corresponding activity takes time to raise. Symmetrically, it also takes time to reset when the signal shuts down. Thus, the neuron perceives time extended *traces* of the phenomenon it is fed with. Learning consists then in correlating traces, with similar algorithms to the ones that are used in non temporal neural networks for correlation of instantaneous signals. The shape of the trace determines the temporal regularities that can be detected, and is therefore a crucial point in such techniques.

Discussion on temporal neural networks

The three main techniques mentioned here have complementary properties. The first one, the spatialization of time, allows to use improved spatial techniques, but the analysis through a temporal window may not lead to a correct analysis of some process dynamics. The second technique, adding delays in perceptrons, is suitable for computation of sequences where the states t and $t + 1$ are clearly defined, which is not the case for some applications, like servo-control loops for example. The last approach, the modeling of dynamics of neurons, seems closer to biology, and allows (with a suitable time discretization), to deal with on-line continuous time computation. The drawback is that computation with such neurons is not clearly understood for large networks.

4 A stochastic behavioral approach: Reinforcement Learning

It is explained in [30] how an adapted perceptron (using traces, as explained just above) can model pavlovian conditioning. As far as cognition is concerned, instrumental conditioning must also be considered, but it is clear that, in this latter case, supervised learning is no more valid. In this domain, the famous Skinner experiments [27] can be viewed in a computer science framework as a decision problem, that raised mathematical and computational difficulties. Basically, the experiment consists in putting a rat in a cage where a lever stands. Pressing the lever enables the rat to get a pellet of food. While initially acting randomly in the cage, the rat sometimes presses the lever and gets food. The Skinner experiments show that the rat is able to learn, after several random trials, that the lever allows to get food. Even simple models of this experiment raise the problem that each action the rat can perform to get reward is not simultaneous with the food delivery, except pressing the lever. For example, turning toward the lever, walking to it, are never directly rewarded. One computational way to deal with decision problems for which reward is delayed is the resolution of Markovian Decision Processes (MDP).

4.1 Markovian Decision Processes

The purpose of MDP is to model the learning of successive choices of action to reach a rewarding state, i.e to model a motivated adaptive behavior. A good description of MDP resolution can be found in [31], the main ideas are discussed here. A MDP consists of a finite set \mathcal{S} of states, a finite set \mathcal{A} of actions, a transition function \mathcal{T} that returns the probability $\mathcal{T}(s, a, s')$ to reach state s' from s when performing action a , and a reward function \mathcal{R} that returns the reward value $\mathcal{R}(s, a)$ got by performing action a in state s . These elements describe the rules of the world. A behavior consists in performing a succession of actions, then visiting successive states. Solving a MDP allows to find the succession of actions for which the total amount of rewards received along the behavior is optimal. The problem is then to find a function π that associates to each state s the action $\pi(s)$ to perform this optimal behavior. Such a function is called a policy and the purpose of the resolution of a MDP is to find a policy π^* that maximizes the cumulated reward obtained by visiting states according to it.

MDP resolution

Even if some gradient descent based resolutions on parametrized family of policies exist [4], the most usual way to solve a MDP is to associate to each state s a value $\mathcal{V}^\pi(s)$ being the expected cumulated reward obtained by using policy π to choose actions from state s during an infinite number of steps. Cumulated reward expectation from state s is actually the way to take delayed reward into account because it can be non null even if s is not directly rewarded. A policy consisting in staying in a weakly rewarded state s^- forever since it is visited will lead to infinite values $\mathcal{V}^\pi(s)$ for every s that lead to s^- by following the policy. The same would be obtained for policies that stay on a strongly rewarded state s^+ . Thus, this basic definition of $\mathcal{V}^\pi(s)$ is not strong enough to compare policies. The idea is then to consider that whatever the rewarding function \mathcal{R} , reward has a probability $1 - \gamma$ to extinguish forever, with γ close to 1. That means that policies that visit more quickly best rewarding states will lead to higher finite $\mathcal{V}^\pi(s)$ values. Equation 1 gives the expected cumulated reward $\mathcal{V}^\pi(s)$ from every s , given a policy π .

$$\mathcal{V}^\pi(s) = \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \mathcal{V}^\pi(s') + (1 - \gamma) \times 0 \quad (1)$$

Equation 1 means that from s , and following π forever, two contributions can be expected as cumulated reward $\mathcal{V}^\pi(s)$. The first one is the instantaneous reward $\mathcal{R}(s, \pi(s))$. The second one has a probability $(1 - \gamma)$ to be 0, if reward stops forever. If not (with probability γ), this second contribution is the expected cumulated reward from the state s' that is reached, i.e $\mathcal{V}^\pi(s')$. As transitions are stochastic, the expected cumulated reward at next state is

estimated by the sum of all $\mathcal{V}^\pi(s')$, weighted by the probability $\mathcal{T}(s, \pi(s), s')$ to actually reach s' from s when performing action $\pi(s)$.

Knowing \mathcal{T} and \mathcal{R} , finding the $\mathcal{V}^\pi(s)$ for each s is the resolution of $|\mathcal{S}|$ equations with $|\mathcal{S}|$ variables, which can classically be done by Gauss algorithm. For a given policy π_i , solving equation 1 allows to evaluate its ability to lead to rewarded behaviors. From this policy π_i , using the computed $\mathcal{V}^\pi(s)$, it is possible to build a policy π_{i+1} that is better than π_i , defining π_{i+1} with equation 2 (see [31] for justifications).

$$\pi_{i+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left\{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \mathcal{V}^{\pi_i}(s') \right\} \quad (2)$$

The policy π_{i+1} is said to be a greedy policy according to values \mathcal{V}^{π_i} . That means that it is better from s to go to a state s' for which π_i will lead to the best cumulated reward than to go to $\pi_i(s)$. The first stage, the computation of \mathcal{V}^{π_i} , is called *policy evaluation* (equation 1), and the second stage, the computation of a better policy, is called *policy improvement*. Alternating evaluation and improvement of policies allows to build a policy series $\{\pi_i\}_{i \geq 0}$ that has been demonstrated to converge toward optimal policy π^* , that is the greedy policy, defined by the Bellman's equation 3.

$$\mathcal{V}^{\pi^*}(s) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \mathcal{V}^{\pi^*}(s') \right\} \quad (3)$$

The present resolution of MDPs is computationally heavy, because each step requires the resolution of a large linear system. This algorithm has been improved by mixing evaluation and improvement stages (see policy iteration algorithm in [31]). The resolution has also been adapted to the cases where transition and reward functions are not known, which is more realistic. This latter algorithm, the Q-Learning algorithm by Watkins [35], allows to compute expected cumulated rewards of the optimal policy π^* while visiting states according to a “dummy” policy (it can be a random walk). This models the situation of the rat in a Skinner cage, and allows to deal with selection of action inferred from interacting with an unknown world where reward is delayed.

4.2 Discussion on MDPs

Computing decision functions through the resolution of MDP is a general way to solve decision processes, but it suffers from three main restrictions. First, most of the algorithms are grounded on discrete and finite state space, which is not the case for many real systems. Second, addressing real problems with such techniques leads to combinatory explosion of the state space, whose size is often prohibitive. Third, interaction of the agent with the world is synchronous: you get at time $t + 1$ the result (new state) of action at time t . For real agents like robots, this assumption cannot be done. Nevertheless,

these algorithms for decision and behavior are grounded on mathematics that provide convergence proofs, which is rarely the case for numerical approaches of such cognitive functions. Moreover, even if actually poorly efficient for an actual behavior of a non-simulated robot, MDPs are not restricted to the navigation of an agent: the concept of states and actions can be applied to many kinds of information processing problems.

Conclusion

The main goal of this paper was to present numerical approaches, among the most frequently used in AI today. We have also tried to relate these approaches to cognitive abilities and hence to explain how they can be integrated in cognitive modeling. The usefulness of these numerical models can also be discussed in relation to the different strategies of NSI presented in the Introduction. Concerning the hybrid strategy, two points can be evoked.

On one hand, implementing a complex system including several numerical models can obviously be seen as a hybrid architecture, in that sense that the goal is to design the cooperation and the communication of several modules. As a conclusion, researches [29, 17, 2] that have been done to formalize such a framework of software design could be used and adapted to that end.

On the other hand, if we consider that the hybrid strategy was developed to integrate symbolic and numerical components, it can be thought that symbolic components could also be added to such systems. This particularly means that expertise and a priori knowledge from the field of economy could be integrated through symbolic AI tools and linked together with the numerical models.

Concerning the unified strategy, it must be noticed that one goal of this strategy is to adapt classical numerical models to draw them toward the symbolic side. This can for example correspond to knowledge extraction or assimilation in such numerical models. At the moment, only few numerical models have been studied and adapted in that direction [3]. This work could be pursued. An interesting way to address this point is particularly to try to relate such models to classical mathematical or statistical models [5] where operations are generally explicit and formalized, which is interesting for explanation and knowledge extraction purpose.

As one wishes to go in the unified way, a fruitful way is also to try to tightly couple considered models. Indeed, an important difference between the unified and the hybrid strategy is the degree of coupling which is rather loose for hybrid models (only exchange of parameters between models) and can be tighter for unified models (exchange of complex structure) [17]. This can particularly lead to the definition of new models, as the adaptation and the tight integration of several classical models. Such is for example the case with the design of a framework of biologically inspired neuronal mechanisms (see for example [9]). It has the interesting properties to be directly inspired

from the human cognition neuronal substratum and to be elaborated with a common neuronal formalism, which is interesting to obtain a tight coupling.

References

1. F. Alexandre. Connectionist Cognitive Processing for Invariant Pattern Recognition. In *International Conference on Pattern Recognition (ICPR'96)*, Vienna, August 1996.
2. F. Alexandre. Tools and experiments for hybrid neuro-symbolic processing. In *Ninth International Conference on Tools with Artificial Intelligence (ICTAI'97)*, pages pp. 338–345. IEEE Computer Society, November 1997.
3. F. Alexandre and J.-F. Remm. Knowledge extraction from neural networks for signal interpretation. In *European Symposium on Artificial Neural Networks (ESANN'97)*, pages pp. 115–120. D facta, April 1997.
4. J. Baxter and P. L. Bartlett. Reinforcement learning in pomdp's via direct gradient ascent. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.
5. L. Bougrain and F. Alexandre. Unsupervised Connectionist Algorithms for Clustering an environmental data set : a comparison. *Neurocomputing*, 1-3(28):177–189, October 1999.
6. R. Cerf. *Une théorie asymptotique des algorithmes génétiques*. PhD thesis, Université Montpellier 2, 1994. in French.
7. Y. Chauvin and D. E. Rumelhart, editors. *Backpropagation: theory, architectures, and applications*. Lawrence Erlbaum Associates, 1995.
8. J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
9. H. Frezza-Buet, N. Rougier, and F. Alexandre. A cerebral framework for the integration of biologically inspired temporal mechanisms for sequence processing. In *Neural, Symbolic and Reinforcement Methods for Sequence Learning*. Springer, 2000.
10. B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.
11. S. I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2), February 1988.
12. D. E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Springer-Verlag, 1989.
13. S. Grossberg. Some normal and abnormal behavioral syndromes due to transmitter gating of opponent processes. *Biological Psychiatry*, 19(7):1075–1117, 1984.
14. F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, 1995.
15. M. Gutknecht. The postmodern mind: hybrid models of cognition. *Connection Science*, 1992.
16. J. Hawthorne. On the compatibility of connectionist and classical models. *Philosophical Psychology*, 1989.
17. M. Hilario, C. Pellegrini, and F. Alexandre. Modular Integration of Connectionist and Symbolic Processing in Knowledge-Based Systems. In *Proceedings International Symposium on Integrating Knowledge and Neural Heuristics*, Pensacola Beach (Florida, USA), May 1994.

18. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences, USA*, pages 2554–2558, 1982.
19. M. I. Jordan. Attractor dynamics and parallélism in a connectionist séquential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Erlbaum, 1986.
20. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
21. J. Kodjabachian and J.-A. Meyer. Evolution and development of control architectures in animats. *Robotics and Autonomous Systems journal*, 16:161–182, 1995.
22. T. Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer-Verlag, 1989.
23. Y. Lallement, M. Hilario, and F. Alexandre. Neurosymbolic Integration : Cognitive Grounds and Computational Strategies. In M. DeGlas and Z. Pawlak, editors, *Proceedings World Conference on the Fundamentals of Artificial Intelligence*, Paris, July 1995.
24. W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943.
25. A. Newell and H.A. Simon. Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 1976.
26. F. Rosenblatt. A comparison of several perceptron models. In G.T. Jacobi M.C. Yovits and G.D. Goldstein, editors, *Self-Organizing Systems*, pages 463–484. Spartan Books, Washington, 1962.
27. B.P. Skinner. *The Behavior of Organisms*. 1938.
28. L.R. Squire. Memory and the hippocampus: A synthesis from findings with rats, monkeys, and humans. *Psychological Review*, 99(2):195–231, April 1992.
29. R. Sun and F. Alexandre, editors. *Connectionist - Symbolic Interpretation; from Unified to Hybrid Approaches*. Lawrence Erlbaum Associates, 1997.
30. R. Sutton and A. Barto. Toward a modern theory of adaptive neural networks: expectation and prediction. In *Psychol. Rev.*, volume 88, pages 135–170, 1981.
31. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction*. MIT Press, 1998.
32. G. Tagliarini, J. Christ, and E. Page. Optimization using neural networks. *IEEE Trans. on Computers*, 40:1347–58, 1991.
33. J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton Univ., 1956.
34. A. Waibel, T. Hanazawa, G. Hinton and K. Shikano, and K. Lang. Phoneme recognition using time delay neural networks. In *IEEE Transactions on Acoustics Speech and Signal Processing*, volume 37, 1989.
35. C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.
36. B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1969 IRE WESON Convention Record, New-York*, pages 96–104, 1960.