USING SYSTEMIONS TO MODEL EMERGENCE IN LEARNING ENVIRONMENTS

GOUARDERES Guy
Université de Pau - IUT de Bayonne, 3 Av. Jean Darrigrand
64000 BAYONNE - FRANCE
Tel: (33) 0559528996, Fax: (33) 0559528989
E-mail: gouarde@larrun.univ-pau.fr

Keywords: Agent architecture, Learning Environments, Knowledge Mining, Weak Determinism, Genetic Algorithms.

Abstract:
Recent trends in multi-agent ITS can be split in a movement away from the traditional ITS architecture consisting of modules (i.e., the expert, student, and instructional modules) and a movement towards looking at the process (i.e., planning, monitoring, and diagnosing). The strong idea as a core assumption for this second approach is that the term "cognitive agent" can be described as an agent that learns in the same way as people learns. So, focus is put both on learning protocols and mutant processes within a new paradigm for cognitive agents: the Systemion (Systemic Daemon). Systemions are designed as agents that powerfully increase their knowledge by learning from other and agents that assume their survival by joining two unique properties of the living systems: replication and evolution. Life cycle in systemions is self-controlled by two concurrent mechanisms - first, a reproduction system, continuously modified by a learning algorithm, is used to fertilize the cloning of a "child" agent into a given lineage; - second, selective genetic algorithms act as a mutant processes to create new fathers of an improved lineage.

1 Introduction

Computer networks to support human learning is a field in major change which has to cope with recent approaches of distributed artificial intelligence (DAI) techniques and new concepts to learning environments (i.e. intelligent tutoring systems (ITS)). Facing with the two key advantages - modularity and openness - that these systems afford, agents are a powerful tool for making modular tutoring systems where components can be more easily added or replaced and smoothly integrated [Wooldridge, 95].
Till the two last years, traditional ITS development was mainly based on inference engine power and monolitical knowledge-base paradigms. In fact, many of studies on the development of ITS have considered the ITS as an individual tool where different sources of knowledge (teacher, student, media,...) are predefined components (i.e., the expert, student, and instructional modules). Thus, it is very difficult for pedagogues and teachers to imagine all strategies to adapt to every student.

Nowadays, an ITS must be viewed as a system that focuses on the process (i.e., planning, monitoring, and diagnosing) and which evolves by acquiring knowledge during the time. In fact, in ITS, the acquisition of knowledge is not limited to the design step but must be developed during its using, according to not only its own processes but the environment too [Frasson & al., 96, 98]. This duality in the design is very useful to exploit the full potential of these multi-agent architectures, both at engineering and theoretical levels. It is one of the reasons, the multi-agent architecture paradigm is increasingly becoming popular with the ITS designers. To fulfill these goals the paper suggests a new architectural scheme for intelligent context-sensitive agents based on determinism, rational actor and emergence. That enforces the strong idea that the term "cognitive agent" can be described as an agent that learns in the same way as people learn [Vidal & al., 96].
So, focus is put both on learning protocols and mutant processes within a new paradigm for cognitive agents: the Systemion (Systemic Daemon). Systemions provides suitable hybrid support for synchronous versus asynchronous, generic dialogues among multiple autonomous agents [MEITA, 97] via a robust representation of a cognitive model involving users "in-the-loop" at the basis of the assessment of the cognitive process itself [Gouardères & al., 98].
In this perspective, Systemions are designed as agents that powerfully increase their knowledge by learning from others and agents that assume their survival by joining two unique properties of the living systems: replication and evolution. Replication allows a systemion to create locally, or from a distance, a clone of itself. Evolution from generation to generation occurs by systemions adapting themselves to context changes in their local environment.
In order to adapt their evolution, systemions integrate a life cycle mechanism which self-modify their behavior. Life cycle in systemions is self-controlled by, - first, a reproduction system, continuously modified by a learning algorithm that runs on the complete set of rules at a current state of each systemion, to fertilize the cloning of a "child" agent preserving a basic set of rules ("genotypes"); - second, if a previously learnt sub-set system of rules is frequently used with a better performance than the inherited one, this is used to modify the basic set of genotypes as a mutant process to create a new lineage of systemions.

2 Traditional ITS versus evolving ITS

In a previous work [Frasson et al., 96] we have shown how an ITS architecture can be designed with *intelligent agents*. Various definitions and classifications which have been given to intelligent agents [Wooldridge, 95]. Depending of the degree of sophistication, an intelligent agent can possess the following properties: reactivity, autonomy, collaborative behavior, knowledge level, temporal continuity, personality, adaptability, mobility, instructability. A first classification formulated by Gilbert et al., [1995] differentiated intelligent agents according to three parameters: *mobility*, *agency (or autonomy)* and *intelligence*. Mobility is the degree to which agents travel through a network. Agency is the degree of autonomy (in terms of interactions) and authority given

to an agent (at a minimum an agent can work asynchronously). Intelligence is the degree in which we can gradually deploy preferences, reasoning, planning and learning. At the limit of these last parameters, an agent can learn and adapt to its environment both in terms of objectives and resources available (At a first stage, this three fundamental properties can be studied as multi-agent components).
However, even if the design of the previous properties with construction blocks of knowledge is very important for educational purposes, it is still insufficient for ITS requirements. Indeed, we need to have agents which model human behavior in **learning situations**. This *cognitive* aspect of an agent relies upon its capability to learn and discover new facts or improve its knowledge for a better use. (At a second stage, this aspect can be studied as a generic learning process which can be achieved and assessed by a variety of methods (genetics algorithms, machine learning,.) [Gouardères et al., 1998].

2.1 Multi-agent ITS components

In fact, the three main components of an ITS (the *student model*, the *knowledge model*, and the *pedagogical model*) can be built in the form of intelligent agents. However, the evolution of intelligent tutoring systems toward the use of multiple learning strategies calls on reusable components in a multi-agent architecture. Thereby, we have first designed an ITS where several agents assume different pedagogical roles such as a *co-learner*, a *learning companion*, ...; consequently, we have called them Actors [Frasson et al, 96].

2.1.1 ACTORS

An actor is an intelligent agent which is reactive, instructable, adaptive and cognitive. The actor architecture includes four modules (Perception, Action, Control and Cognition) distributed in three layers (Reactive, Control and Cognitive). It is similar (but extended) to the Touring Machines [Ferguson, 92], consisting in perception and action subsystems that interface with the environment of the agent and three control layers (reactive, planning, modeling).
ITS improvement by actors has progressively highlighted two fundamental characteristics: (1) learning in ITS is a constructive process involving several partners, (2) to improve it, various learning strategies can be used such as one-on-one tutoring, learning with a co-learner, learning by teaching, learning by disturbing [Frasson et al., 96].
However, the success of a pure multi-agent architecture based tutoring system depends on a learners' motivation and self-discipline. Most of the ITS inherits a 'context gap' at the points of divergence between the purpose of the tasks performed within an ITS and the purpose of the predicted solutions expected by the pedagogue. The difficulty is to know the level of understanding of the learner in order to give him/her adapted assistance and how to represent this required knowledge.
Related to this first difficulty, another point of view refers to a second problem of agents'adaptation in a society of learning agents: the 'dropping gap' (i.e., the rate of renunciation due to the lack of motivation and help). To weaken the inherent negative

effects (i.e. the "gaps"), we have achieved a project (LANCA[1]) using individual or collective learning agents to model a convivial and collaborative learning session from the merging of Internet and ITS technology using the Web as a virtual classroom.
Thereby, this method for weakening the "dropping gap" inevitably introduces the 'context gap' restraint jointly with the shared initiative problem between the learner and the system. Then, in a third stage of experiments, we intend to identify aspects of a user's behavior that can be monitored and learned when evaluating several achieved prototypes of cognitive agents working together with the learner "in-the-loop". Consequently, the knowledge acquisition process in LANCA requires an embedded reviser agent (SERAC), to control the evolving knowledge and behavior of each pedagogical agents involved in a learning session [Millet & al., 98].

2.1.2 LANCA as a testbed for cognitive Agents

LANCA©: Learning Architecture Based on Networked Cognitive Agents relates to a system and a method for assisting the learner involved in distance learning situation on the Internet using several types of intelligent Agents. Today, we use it as a test bench to evaluate and assess several models of cognitive agent working together to model emergence in virtual learning environments.
Using LANCA, no two learners learn from the same presentation in the same way or as well. No two moderators or instructors have to use the same medium in the same way or as well. No two lessons are suited for delivery by the same style or through the same medium.

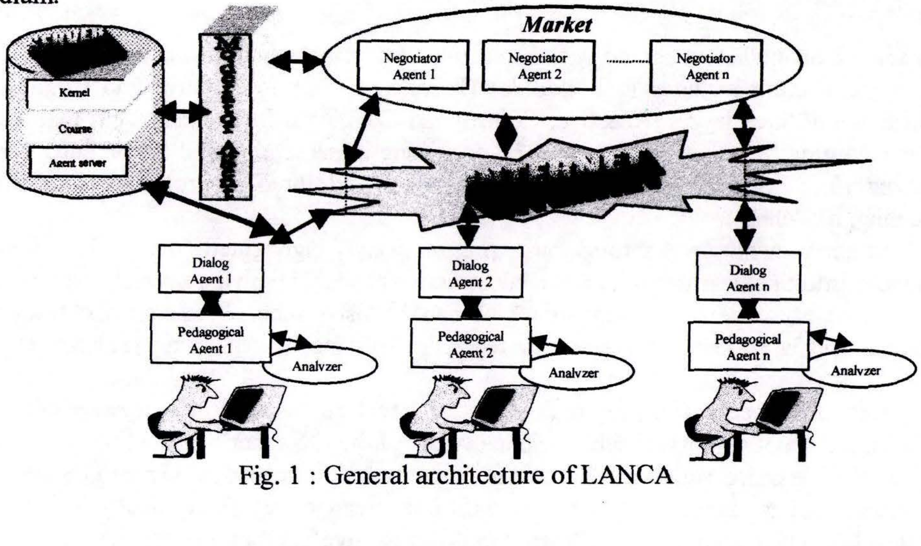


Fig. 1 : General architecture of LANCA

[1] LANCA : Learning Architecture Based on Networked Cognitive Agents. supported by AT&T, is underway at the University of Montreal and involves three countries (Canada, France, Mexico), [LANCA©, 98].

The assessment process is based on the trace of various agents which act collaboratively to support learning at different steps. A first agent (Pedagogical Agent), close to the learner, is able to detect his difficulties using a learner model and provides local explanations when needed or on request. A second agent (Dialog) provides access to other explanations or learners available on the Web, in synchronous or asynchronous mode. Taking into consideration all possible helps, a third agent (Negotiating) is in charge of finding and selling explanations according to a market of requests and helps. A fourth Agent (moderator) is in charge of determining which explanation was finally useful to serve as a permanent source of explanation (chunks as pieces of reasoning) for future learners.

2.1.3 Tracking & Analyzing the learning process between agents.

First experiments have focused on decisions that have to do, for evaluating that a given agent (human or artificial) is just in time, bringing up (or not) useful or decisive helps. The critical question that remains is how the agent moderator goes about calculating adequacy of agent interventions and evaluating profitability of the prescript helps.

According to these previous requirements, we can identify five attributes which benefit the learning process supplied by a multi-agent architecture, as

(i) mixing one-to-one, one-to-many and many-to-many activities in a productive deal,

(ii) place independence (distance problem),

(iii) time independence (synchronization problem),

(iv) multimode-based communication (text, speech, audio, video) and

(v) computer mediated interaction.

Now, let us introduce the complete apparatus to track emergence of useful criteria to model Systemions.

2.2 Knowledge re-engineering with a "reviser" agent

This method relies on Reviser Agent that permits knowledge re-engineering from a multi-criteria evaluation (computer, ergonomic...) where the revision mechanism has to be independent from the evaluation agents that transmit the knowledge. The problem is to extract knowledge on how to analyze and explain the discrepancies between “near-miss” incorrect responses of a student and the system’s knowledge of the probably most useful “correct” line of reasoning. In fact, each part tries both to explain its own reasoning to the other and to explain to itself the other’s reasoning.

The selected approach to detect conflicts in system’s knowledge, reasoning and helps, is to track the learning process with agent based simulation for both the case and domain studied and actors interaction (learner or user, respectively pedagogue or designer). The basic dialogue between the previous agents (user and designer facing the revision system) must be split into a three layered architecture according to different agent tasks (fig. 2) :

- first level, pedagogical agents (learner, tutor): to play simulated learning scenarios,
- second level, evaluator agents (ergonomist, didactician, psychologist, pedagogue...): to trace their reasoning. They are source of conflicts.
- third level, cognitive and learning agent (reviser, instructable): to detect and solve conflicts and improve new helps.

On the following example, the revision system operating includes six steps (see fig. 2):

(1) the use ergonomist agent evaluate the interaction in real time and detects errors due to interface manipulation by the user (user point of view).

(2) when an error is detected, ergonomist agents send the reviser agent a real-time message. The sent message includes the name of the detected error and the solution proposed. These agents that combine user and technical viewpoints are going to be a source of conflict. For example, when the "interface" agent finds a large lift on a screen, it can send a message with a view to suppress the lift and replace it by "turn-pages" whereas the "use" agent may propose to abbreviate the text so as to reduce the lift.

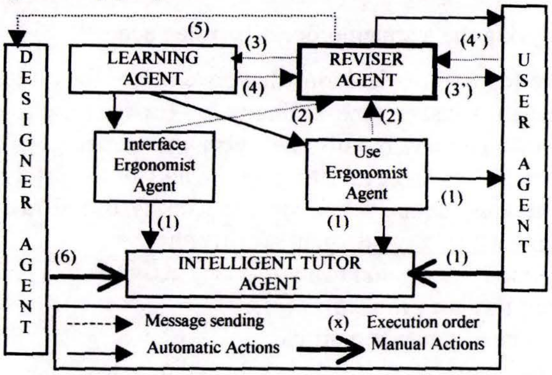


Figure 2 : Operating the knowledge revision system

(3) (4) (3') The reviser agent stores and controls all messages received from ergonomist agents. After the manipulation stage by the user agent and in case of conflict between messages, three possible solutions are available :

1 - it can solve the conflict itself with its current knowledge,

2 - it needs external knowledge (4) and then asks for the help of the learning agent (3)

3 - if it does not success, it then asks the user agent for further information (3').

(4') The user agent gives its viewpoint on the question asked.

(5) The reviser agent reviews including the solutions proposed and return a diagnosis.

(6) The designer agent (a human agent) is going to use the diagnosis of the reviser agent to correct the intelligent tutor in off-line time; it is the re-engineering of the IT.

The learning agent works in parallel with the other agents. It observes the interaction between the user agent and of all the other agents so as to deduce some new rules. These rules will then be able to be transmitted to the reviser agent that increases thus its knowledge base (for example, the reviser agent will be able to solve more conflicts). This agent operates from a genetic algorithm that generates new rules by existing rule combinations.

In this case, an agent (reviser) can model others by starting off with assumptions of their default behavior and then approximating the actual behaviors of others in the group by learning deviations from the default behavior. We are interested in endowing autonomous agents with the same capability so that they can adapt individual behavioral rules to enhance performance.

To sum-up, the research questions that we address in the three projects are the following:
- How does an agent model others agents with its own skills and competencies?
- When does an agent learn skills or abilities?
- When and how does an agent use learning skills to interact with others?

3 SYSTEMION, Agents which learn and mutate

In § 2 we have discussed different experiments to implement cognitive agents as agents that learn (like Actors or LANCA). In the following, we propose a new paradigm for agents that mutate using learning mechanism : the Systemions. Such agents must include determinist and rational mechanisms to support their cognitive and emergent behavior in learning situation.

3.1 Principles of learning mechanisms

3.1.1 Determinism

Determinism assumes that introduction of new technology in organization, has predictable unidirectional consequences on human behavior. Experimental issues (§ 2.1.2), suggest us to adopt a "soft" definition of determinism: combining technology advances with issues emerging from social interaction in a softer approach. The production of emergent phenomena is taken to be a characteristic of the self-organization.

3.1.2 Rationality

Considering a learning environment as a conceptual high-level structure we can supply the system knowledge with goals, and available actions, and the ability of predicting its own behavior based on the principle of rationality. The individual perspective on rational actors was sketched in the described model of actors (§ 2.1.1) as an approach which accommodates such learning, modeling and analysis.

3.1.3 Emergence

Combining determinism and emergence tends to focus on the concept of an emerging phenomenon viewed through a collective learning environment for agents. The following does not really focus on a mechanistic simulation of emergence by rational agents but instead, how to model social learning via a new type of agent.

3.2 Systemion architecture

A Systemion (contraction for 'systemic daemon'), is a simple and experimental software agent model integrating two unique properties of the living systems: replication and evolution. Replication allows a systemion to create locally or from a distance a clone of itself. Evolution from generation to generation occurs by systemions adapting themselves to context changes; each systemion integrate a mechanism which self-modify its behavior.

These important systemic properties distinguish them from classic systems, they are open to the external world, exchange items with the environment, integrate both internal structuring and non structuring elements (interactions of these different elements neither killing nor stopping the entity). In the same way, a systemion has an internal architecture whose some elements can dynamically subtracted, added, modified or inhibited on time.

3.2.1 Internal architecture

Inherited from the Actors paradigm, the general architecture of systemions is distributed in three layers where genotypes inheritance, adaptability and context contingencies are respectively similar to the Reactive, Control and Cognitive layers. The structure is composed of two subsystems: a functional and a behavioral subsystem.

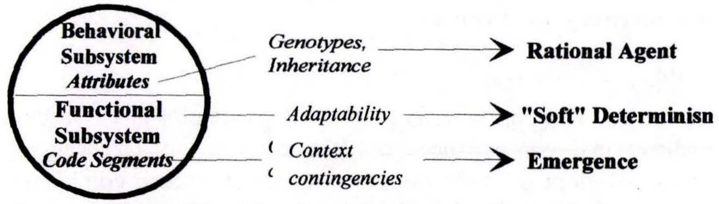


Fig 3 : Three level architecture for Systemions

The functional sub-system (Fig. 4) implements what is relative to the achievement of the function assigned to a Systemion. This function can change during the Systemion lifecycle and constitutes the most flexible part of the Systemion. The basic element of the functional sub-system is the code segment.

The behavioral sub-system implements independent characteristics of the current function assigned to a systemion, such as the ability of replication, the mobility or the maintenance of the systemion identity. The basic element of the behavioral subsystem is the Attribute.

3.2.2 Definition of attributes

From the previous experiments described in § 2 (Actors, LANCA & SERAC), we have selected the following criteria to model cognitive agents:

1- **Autonomy**: autonomous agents interact by messages (delegation mechanism),
2- **Delegation**: a specified task or sub-task is delegated to an dedicated agent,
3- **Context**: environment in which the agent can operate (implicit or bounded domain),
4- **Predictability** (determinism) : its behavior is based on principle of rationality,
5- **Complexity**: agents can't learn by reactivity only (as deliberative agent),
6- **Specificity**: it is specifically tailored for a given task or mission.

The next additive properties are specific to Systemions :

7- **Evolution**,
8- Learning (with **emergence**),
9- **Mutation** / adaptation in the sense of artificial life.

Thereby, supplied with an evolution mechanism, Systemions are sometimes :
- functionally well-determined agents (i.e., fulfilled with #1 to #6 attributes)
- functionally undetermined agents (i.e., soft determinism assumed by #7 to #9 attributes).

3.2.3 Learning

The learning mechanism is an inherent characteristic of a Systemion; it is implemented with a genetic algorithm as "Anytime Learning". A model possibly identified at a given moment, may be no longer accurate in a while since the main characteristics of the environment and of the agent may change. So Systemions, faced to unexpected context have no more alternative than migrate (mobility) into a new site or adapt itself (mutation).

3.3 SYSTEMION "Life Cycle"

Life-cycle in systemions is supported by a reproduction system. Father's lineage of systemions has a variable but limited life time. It disappears when a derived Child's lineage is able to assume all current father's tasks with more efficiency.

3.3.1 - Basic operations

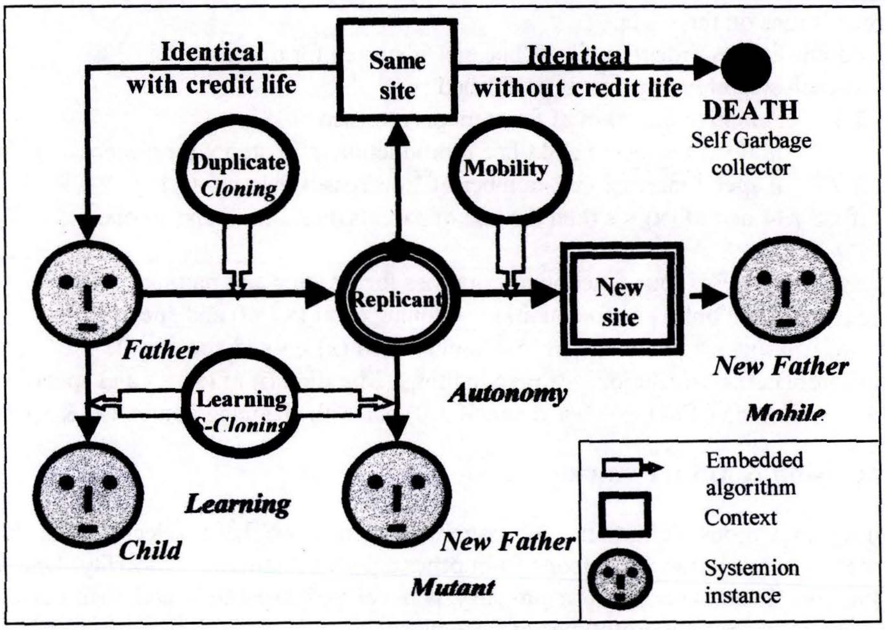


Fig. 4 : Complete "life Cycle" of Systemions

3.3.2 - Main features:

Mobility, allows a systemion to migrate to others sites, *Replication and mutation* produce new generations of systemions (using genetic algorithm or/and copying code segments). Informational elements in systemion instances describe how knowledge is symbolized / formalized (i.e. as First Class Function in Scheme - FCF).
When the reproduction system runs, each systemion can check in time whether the new adopted behavior matches or not the assigned goal, in order to increase the efficiency by specifying more and more its own competency by learning.

In this simplified proposal, the reproduction process is monitored and updated just using two parameters: replication frequency and learning rate. Moreover, the complete learning system is more sophisticated; the algorithm insures that the best lines of children are selected, by evolving a population of systemions. This requires the evaluation of a lot of genotypes, each one representing a given level of competency to be acquired by a systemion; thus, each inferred competency level should be continuously assessed (for example, by implicit evaluation in Scheme of the threshold of each affected variable).

3.3.3 - General algorithm

In the general algorithm for learning the two parameters for self-supervised learning are :
- Adaptive Threshold (A.T.) fixed to **s,** estimated by the **af** function.
- Learning Rate, **specif**: classify specificity for each frame of selector according to a goal.

The process runs on three steps :

```
1-    Parent-line reproduction for a "lineage" is started for n steps.
2-    at each step crossing over is performed :
      2.1 -  Adaptive Function af for step x is returned
             if af (x) ≥ s then a child-line reproduction is  at step x for n' steps.
      2.2 -  if specif increase (i.e. number of * decrease) then n<–n+1.
3-    if x ≥ n+1 and af (x) < s then the line of parents dies...........and so one.....
```

In fact, systemions uses four functional properties for learning and muting :
Cloning (replication only) => precondition : Cloning (if **af** (x) = 0 and **specif** = 0)
Supervised Cloning => precondition : S-Cloning (if **af** (x) ≥ **s** and **specif** = 0)
Mutation (replication+evolution) => precondition : Mutation (if **af** (x) ≥ **s** and **specif** > 0)
Mobility => Cloning (if **af** (x) = 0 and **specif** = 0) but with changing context(**af** & **specif**)

4 Conclusion & future work

According to various definitions of cognitive agents, we have selected four basic properties to characterize Systemions from other agents : Autonomy, Mobility, Learning and Mutation. Each class of basic property is developed separately and then currently integrated into successive prototypes of systemions.
Clearly, mutation seems for us the most significant capacity and an important effort has been done to design and improve this embedded mechanism in Systemion. Achievement of other functionality mostly calls on packages improved from two public systems Aglets [Meita, 97] and Agent-Tcl [Gray, 96]. Furthermore, to allow more autonomy in distributed applications, Systemions support real-time generation of direct run-time code (interpretable) on a local or distant machine.
As a first result, many of Systemion's unique features, such as security, are not yet fully implemented. But, it is clear to us that the basic engineering problems raised by Systemion's design have already been solved, and we have an exact appraisal of how

many are still under investigation. As the two agent systems we have used and developed before, -Actors, LANCA -, Systemion seems to hold the most promise for the future. Systemions have been applied on a concrete case (Agro-Systems [Millet, 98]), with different, and more or less efficient learning algorithms and with software environments and we prepare currently two tests in real world experiments, one in the area of management of health and the other in aeronautic operations and maintenance.
All in all, we feel that this work will result in a new and technical and theoretical framework based on Systemions to study and develop ITS environments (conceptual methods and models) on the three levels: reactive (rational), adaptive (determinist) and cognitive (emergent).

References:

Ferguson, I. A., (1992) Touring Machines: An Architecture for Dynamic, Rational, Mobile Agents. PhD Thesis, Clare Hall, University of Cambridge

Frasson C., Mengelle T., Aïmeur E, Gouardères G., (1996) "An Actor-based Architecture for Intelligent Tutoring Systems", Third International Conference ITS'96, Montreal, Lecture Notes in Computer Science, Heidelberg Springer Verlag

Frasson C., Martin L., Gouardères G., Aïmeur E, (1998) "LANCA : a distance Learning Architecture based on Networked Cognitive Agents"- 4° International Conference on Intelligent Tutoring Systems -ITS'98- ACM/AFCET/SIGART/SIGCUE/IEEE - San Antonio. USA (to appear).

Gilbert, D., Aparicio, M., Atkinson, B., et al.,. (1995) "Intelligent Agent Strategy". Technical report, Research Triangle Park, IBM Corporation.

Gouardères G., Canut M.F., Sanchis E., (1998), "From Mutant to Learning Agents. Different Agents to Model Learning", A symposium at the 14th European Meeting on Cybernetics and Systems Research - EMCSR'98, Vienna, Austria.

Gouardères G., Frasson C., (1998) "On effectiveness of distance learning using LANCA" 4° International Conference Intelligent Tutoring Systems-ITS'98- Workshop Pedagogical Agents -ACM/AFCET/SIGART/SIGCUE/IEEE- San Antonio. USA

Gray R.S. (1996) "Agent-Tcl: a secure and mobile-agent system", Proceedings of the Fourth Annual Tcl/Tk Workshop, California, http://www.cs.dartmouth.edu /~agent/

MEITA - Mobile Agent Computing (1997) A White Paper- Mitsubishi Electric ITA - http://www.meitca.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html

Millet S., Gouardères G., (1998) "SERAC : a multi-agent system for the evaluation and revision of knowledge in ITS" 4° International Conference Intelligent Tutoring Systems -ITS'98 - Workshop Pedagogical Agents - ACM/AFCET/SIGART/SIGCUE/IEEE - San Antonio. USA.

Terano T., (1997) "Genetic Algorithm Based Feature Selection in a Muti-agent System for Questionnaire Data Analysis", APORS'97, Fourth Conference of the Association of Asian Pacific, Melbourne.

Vidal J.M., Durfee H.M., (1997) "Agents Learning about Agents: A Framework and Analysis", Articial Intelligence Laboratory, University of Michigan.

Wooldridge M., Jennings N., (1995) "Intelligent Agents : Theory and Practice", The Knowledge Engineering Review, Vol 10, N° 2, pp 115-152

Life-long learning in incremental neural networks

Fred Henrik Hamker[1]
Technische Universität Ilmenau, Neuroinformatik, D-98684 Ilmenau, Germany
http://cortex.informatik.tu-ilmenau.de/~fred
e-mail: fred@informatik.tu-ilmenau.de

Abstract
This approach presents a possible solution to the stability-plasticity dilemma in incremental neural networks with a local insertion criterion. The main advantages are i) the capability of life-long learning, i.e., learning throughout the entire lifetime of a neural network, ii) stability in a stationary environment and iii) plasticity in a non-stationary environment, but only if the current knowlege does not fit the need of the task.
Thus, the network structures its internal representation not like a copy of the environment but in order to fulfill the current task.

Keywords: Life-long learning, stability-plasticity dilemma, incremental neural networks, Growing Neural Gas, Dynamic Cell Structures

1 Introduction

Learning is one of the main issues of artificial neural network design. It describes a mechanism by which a system obtains a representation of its environment. Recent research addresses the topic of on-line learning, incremental learning and life-long learning, which all discuss the same problem but emphasize different aspects. The necessity for on-line learning, in which the couplings of the network are updated after the presentation of each example, arises if not all training patterns are available all the time (Freeman and Saad, 1997; Heskes and Kappen, 1993). Most publications referring to on-line learning focus on the role of the learning rule and the convergence of the learning process, but stop learning when a performance criterion is reached. For systems, like robots, which are faced with patterns during their entire lifetime, studying on-line learning in contexts such as a changing environment (Heskes and Kappen, 1993) encounters the problems of stability and plasticity. Incremental learning addresses the ability of repeatedly training a network with new data, without destroying the old prototype pattern. Life-long learning, or also called continuous learning, emphasizes learning throughout the entire life-time and has to cope with changing environments and overlapping decision areas. It is not sufficient to only follow a non-stationary input distribution like (Fritzke,

1 Since September 1998 he joined the medical data analysis project MEDAN at the J.W.Goethe-Universität Frankfurt (email: hamker@cs.uni-frankfurt.de).

1997), life-long learning has to solve the stability-plasticity dilemma, which demands the adaption to new patterns and the preservation of old patterns.
Networks with a local or distributed representation of knowledge appear to be good candidates for life-long learning scenarios. One type of a local representation of knowledge utilized in recent literature of on-line learning are RBF's (Freeman and Saad, 1997) or similar networks (Gaussier and Zrehen, 1994). Nevertheless, they have a fixed number of nodes which has to be determined by the designer.
Incremental networks have the advantage that the number of nodes is also a result of learning by doing. The most important question in life-long learning incremental networks concerns the rule of insertion. ART networks, like FAM (Carpenter et al., 1992) insert new nodes based on a similarity measure. Other families of incremental networks use an error measure to insert new nodes. They can be subdivided into local error based insertion rules like Growing Cell Structures (GCS) (Fritzke, 1994), Growing Neural Gas (GNG) (Fritzke, 1995), Dynamic Cell Structures (DCS) (Bruske and Sommer, 1995) and global error based insertion rules like Cascade-Correlation (Fahlman and Lebiere, 1989). Inserting new nodes solely depending on the similarity of the input pattern leads to a purely sensor-based representation, which does not reflect the requirements of further processing stages. In contrast, an error-based insertion adapts the representation depending on the task and therefore leads to a task-based representation (Hamker and Gross, 1997). Compared to a global insertion criterion, a local criterion has the important advantage that insertion can be controlled locally. Summarizing, incremental networks with a local error based insertion rule are optimal candidates for life-long learning – but only if the insertion of new nodes can be managed properly.

2 General approach

On the one hand, incremental networks are not allowed to grow permanently. On the other hand, growing is an important feature to decrease the error of the net for the task and to adapt to changing environments. According to Grossberg (Grossberg, 1988) a switching-off of plasticity is a problem in nonstationary environments. But for the type of incremental networks with a local error based insertion rule, like GCS, GNG and DCS, an error-based learning of the insertion parameters is proposed to dynamically and locally control the stability and plasticity in the network. For this reason, each node not only owns an averaged longterm error counter, it is also equipped with an insertion threshold and an averaged longterm error counter at the moment of the last insertion (insertion error). The learning of the insertion parameter can be explained by an insertion evaluation cycle (Figure 1). By adaptation of an insertion threshold based on the evaluation of previous insertions, the network learns locally when it is useful to insert further nodes or to stop insertion.
The definition of the error counters as averaged error counters similar to (Ahrns et al., 1995) leads to an error measure that is independent of the input probability density in contrast to the error measure in (Fritzke, 1994; Fritzke, 1995; Bruske and Sommer, 1995). It has the advantage that the error is independent of the input probability density, which is important for life-long learning.

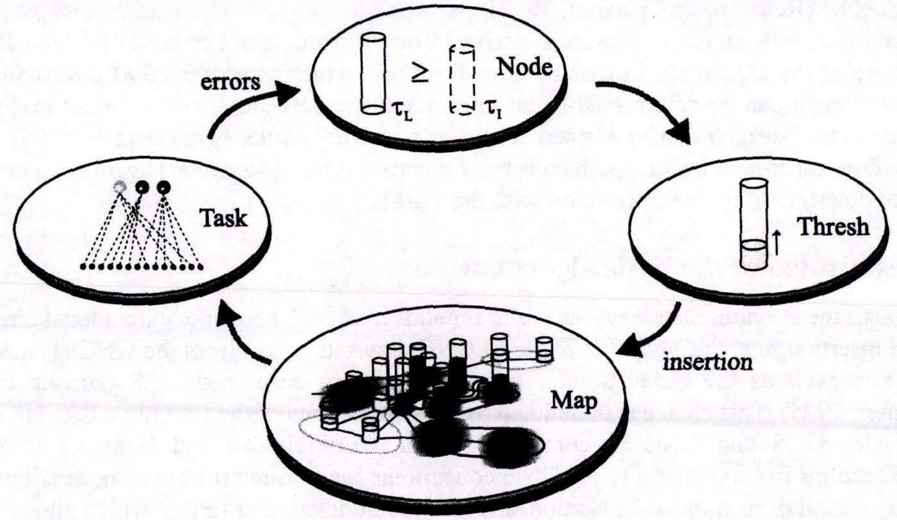


Fig. 1. Insertion evaluation cycle. The average long time error τ_L of the task is compared to the error at the moment of the last insertion τ_I. If this error is greater or equal, the insertion was not successful and the insertion threshold τ_h is increased. If the threshold reaches the average long time error, a further insertion at that location is not possible. To permit exploration in the future, the threshold can be decreased with a large time constant.

Another aspect concerns the adaptivity of the nodes. In (Ahrns et al., 1995), an error-modulated Kohonen type learning rule was used to achieve a uniform approximation error independent of the input probability density. Here, the modulation depends on the ratio of the average long time error and the average short time error and aims at reducing fluctuations when the input probability does not change any more. This means a node learns more, if the input probability changes and new errors occur.
Furthermore, a deletion criterion is introduced to remove redundant nodes. Candidates for deletion are located nearby in the input space and are responsible for similar outputs. For tasks with real-time demands the deletion criterion allows to restrict the number of nodes to an upper bound: By a simultaneous insertion of a new node and the deletion of the "worst" node, the nodes of the network are optimally fitted for static as well as for changing environments.
Interestingly, learning and insertion in the Life-long Learning Growing Neural Gas (LLGNG) shows similarities to the reward-based control of the plasticity of activated Hebbian synapses in biology. While the reward is usually delayed, the ratio of the average long time error and the average short time error reflects very well changes of the expected error i.e., the average long time error. Similarly, the insertion of new nodes depends on the difference between the predicted error, i.e., the insertion threshold and the actual error, i.e. the average long time error. The basic rule of learning behind learning and insertion in the LLGNG is that 'organisms only learn when events violate their expectations', previously

assumed by (Rescorla and Wagner, 1972).
The method, how an error measure is derived from the task, is not essential for the basic structure of the algorithm. In case of error-feedback, which is addressed as an example here, the error can be determined by an inter-module supervision or an external teacher and the output weights can be adapted according to the delta rule, as in (Fritzke, 1994). In case of reinforcement learning, which is most interesting for autonomous agents, the error can be determined by the interaction with the world.

3 Description of the basic algorithm

Although the previous statements are valid for all incremental networks with a local error-based insertion rule, the algorithm of the LLGNG draws its origin from the GNG (Fritzke, 1995) as well as the rather similar but independently developed DCS (Bruske and Sommer, 1995). This choice is underlined by the good results of the GNG in a benchmark on FAM, GCS and GNG in comparison to a MLP (Heinke and Hamker, 1998). Modifications in comparison to the GNG concern the local counters of each node (Figure 2), the control of learning and insertion, and an explicit deletion criterion, which allows to steer the density of the nodes considering their output-weight similarity. The network consists of two layers. The input determines the representation layer, which is followed by an output or task layer. The representation layer is described by a graph **G**, in which the set of neighbors N_i of a node i is defined by all nodes who share an edge with the node i.

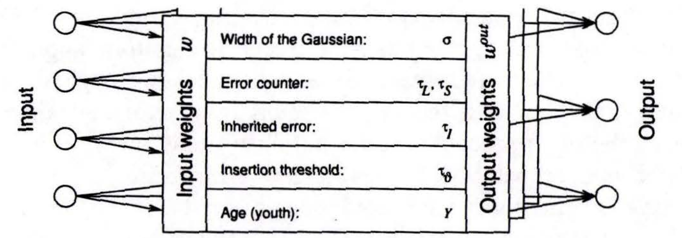

Input
Input weights w
Width of the Gaussian: σ
Error counter: τ_L, τ_S
Inherited error: τ_I
Insertion threshold: τ_ϑ
Age (youth): Y
Output weights w^{out}
Output

Fig. 2. Node of the life-long GNG. Besides the width of the Gaussian each nodes owns a longtern error counter τ_L, a shortterm error counter τ_S, the inherited error at the moment of insertion τ_I, an insertion threshold τ_h, and the youth of the node Y, which decreases exponentially with the time constant T_Y from one to zero when the node was best matching. Despite the inherited error, which remains fixed until the node is selected for insertion again, the error counters are defined as moving averages with their individual time constant.

Adaptation of the representation layer

- For all nodes i, calculate the Euclidian distance d_i of the input $\boldsymbol{x}$ to the weight vector $\boldsymbol{w}_i$ and locate the best matching unit b and the second best s (equal to (Fritzke, 1995)):

$$d_b = \min_{i \in G} (d_i) \; ; \qquad d_s = \min_{i \in G, i \neq b} (d_i) \; ; \qquad d_i = \|\boldsymbol{x} - \boldsymbol{w}_i\| \qquad \forall i \in \boldsymbol{G}$$

- Calculate the activation of all nodes y_i with a Gaussian function (Fritzke, 1994)):

$$y_i = e^{-\frac{\|x-w_i\|^2}{\sigma_i^2}} \quad ; \quad \sigma_i = \frac{1}{\|N_i\|}\sum_{j\in N_i}\|w_i - w_j\| \qquad \forall i \in G$$

- Determine the quality measure for learning B^L of the best node b and its neighbors $c\in N_b$:

$$B^L_{(b/c)} = \frac{\tau_{S(b/c)} + 1}{\tau_{L(b/c)} + 1} \qquad \forall c \in N_b$$

- Determine the input learning rate η^i of the best node and its neighbors from the quality measure B^L, the youth Y, the learning rate of the winner η_b and the neighbors η_n and the input adaptation threshold ϑ^i_L:

$$\eta^i_{(b/c)} = \begin{cases} 0 & \textit{if} \quad \alpha^i_{(b/c)} < 0 \\ \eta_{(b/n)} & \textit{if} \quad \alpha^i_{(b/c)} > 1 \\ \alpha^i_{(b/c)} \cdot \eta_{(b/n)} & \textit{else} \end{cases} \qquad \alpha^i_{(b/c)} = \frac{B^L_{(b/c)}}{1+\vartheta^i_L} + Y_{(b/c)} - 1 \qquad \forall c \in N_b$$

and allow a minimal learning rate of the input weights determined by ϑ_M:

$$\eta^{i'}_{(b/c)} = \max(\eta_{M(b/c)}, \eta^i_{(b/c)}) \quad ; \qquad \eta_{M(b/c)} = \eta^i_{(b/c)} \cdot (1 - y_{(b/c)}) \cdot \vartheta_M \qquad \forall c \in N_b$$

- Increase matching for b and its neighbors $c\in N_b$ (similar to (Fritzke, 1995)):

$$\Delta w_b = \eta^{i'}_b (x - w_b)$$
$$\Delta w_c = \eta^{i'}_n (x - w_c) \qquad \forall c \in N_b$$

Insertion and deletion of nodes in the representation layer

After $\lambda \cdot n_N$ steps:

- Determine the quality measure for insertion B^I considering the insertion tolerance ϑ_{ins}:

$$B^I_i = \tau_{Li} - \tau_{\vartheta i} \cdot (1 + \vartheta_{ins}) \qquad \forall i \in G$$

- Find node q and its neighbor f for insertion, if the following criterion is fulfilled:

$$0 < K_{ins,q} = \max_{i \in G}(K_{ins,i}) \quad ; \quad B^I_f = \max_{i \in N_q}(B^I_i) \quad ; \quad K_{ins,i} = B^I_i - Y_i \qquad \forall i \in G$$

If q and f exist:

- ▸ Delete the edge between q and f, insert a new node r, and connect r with q and f. The weights w_r, w_r^{out} and the counters τ_{Sr}, τ_{Lr}, $\tau_{\vartheta r}$ and τ_{Ir} are determined by the arithmetical average of the weights and error counters of q and f.
- ▸ If

$$\tau_{Li} > \tau_{Ii} \cdot (1 - \vartheta_{ins}) \qquad \forall i \in \{q, f, r\}$$

the last insertion was not successful. Thus, adapt the moving insertion threshold:

$$\tau_{\vartheta i} := \tau_{\vartheta i} + \eta_\vartheta \cdot (\tau_{Li} - \tau_{\vartheta i} \cdot (1 - \vartheta_{ins})) \qquad \forall i \in \{k \mid \tau_{Lk} > \tau_{Ik} \cdot (1 - \vartheta_{ins}); q, f, r\}$$

▸ Determine the new inherited error τ_I of q and f:

$$\tau_{Ii} = \tau_{Li} \qquad \forall i \in \{q,f,r\}$$

- Check the deletion criteria considering a minimal age ϑ_{delY} and find node d, whose criterion K is lower than the deletion threshold ϑ_{del}:

$$\vartheta_{del} > K_{del,d} = \min_{i \in G}(K_{del,i}) \ \wedge \ \|N_d\| \geq 2 \ \wedge \ Y_d < \vartheta_{delY}$$

with

$$K_{del,i} = \frac{\overline{\Delta w_i}}{\bar{l}} \cdot \overline{\Delta w_i^{out}} \qquad \forall i \in G$$

the local similarity of the input weights:

$$\overline{\Delta w_i} = \frac{1}{\|N_i\|}\sum_{j \in N_i} \|w_i - w_j\|$$

the average similarity of the input weights:

$$\bar{l} = \frac{1}{n_N}\sum_{j=1}^{n_N} \overline{\Delta w_j}$$

and the local similarity of the output weights:

$$\overline{\Delta w_i^{out}} = \frac{1}{\|N_i\|}\sum_{j \in N_i} \|w_i^{out} - w_j^{out}\|$$

Adaptation of the output layer

- In case of the error-driven example discussed here, determine the squared error.

$$\mathbf{E}_{task}(x) := \mathbf{E}_{\substack{squared\\error}}(x) = \|\zeta - o\|^2$$

- Determine the local output learning rates from the quality measure B^L, the youth Y, the output adaptation rate η_o and the output adaptation threshold $\vartheta^o{}_L$:

$$\eta_i^o = \begin{cases} 0 & if \quad \alpha_i^o < 0 \\ \eta_o & if \quad \alpha_i^o > 1 \\ \alpha_i^o \cdot \eta_o & else \end{cases} \qquad \alpha_i^o = \frac{B_i^L}{1 + \vartheta_L^o} + Y_i - 1 \qquad \forall i \in G$$

- Adapt the weights of the nodes j of the output layer:

$$\Delta w_{ji} = \eta_i^o(\zeta_j - o_j)y_i \quad ; \qquad \forall j \in \{1 \dots m\}, \ \forall i \in G$$

Adaptation of the counters and edges of nodes in the representation layer

- Adapt the long time error counter τ_L and the short time error counter τ_S for the winner b with the time constant T and the error of the task:

$$\tau_{(L/S)b} := e^{-\frac{1}{T_{(L/S)}}} \cdot \tau_{(L/S)b} + \left(1 - e^{-\frac{1}{T_{(L/S)}}}\right) \cdot E_{task}(x)$$

- Decrease the youth Y of the best node b:

$$Y_b := e^{-\frac{1}{T_Y}} \cdot Y_b$$

- Compared to (Hamker and Gross, 1997) an advanced criterion for the decrease of the insertion threshold τ_ϑ is presented. It takes the changes of the errors into account and reduces the insertion threshold only, if the distribution of the data changes:

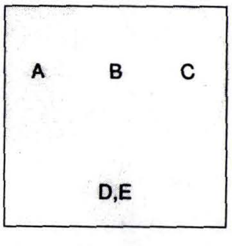
$$\tau_{\vartheta b} := \left(1 - \Lambda(\alpha_b)\right) \cdot e^{-\frac{1}{T_\vartheta}} \cdot \tau_{\vartheta b} \qquad if \quad \Lambda(\alpha_b) > 0$$

$$\alpha_b = \frac{1 + |B_b^L - 1|}{1 + \vartheta_L^i} - 1 \; ; \; \Lambda(x) = \begin{cases} 0 & if \quad x < 0 \\ 1 & if \quad x > 1 \\ x & else \end{cases}$$

- Adapt the edges as follows (equal to (Fritzke, 1995)):
  - Increase all edges emanating from b by one.
  - Set the age of the edge between b and s to zero. If no edge between b and s exists, create a new one.
  - Remove all edges older than ϑ_{age}.
  - Remove all nodes without an edge.

4 Results

For a demonstration of the above ideas, we previously performed simulations of life-long error-feedback learning on an open data set containing overlaps but without changes in the environment. Results presented in (Hamker, Gross, 1997) showed that the network stabilizes and although due to overlapping classes a permanent error occurs, no further insertion takes place. Furthermore, it was shown that in case of a changing environment, the network structure remains adaptive to insert new nodes and to change the weights.
Here, we will focus on the internal dynamics of the algorithm in a changing environment. Mathematically speaking, a changing environment corresponds to a time-dependent input probability (Heskes and Kappen, 1993). For illustration purposes the 2D artificial data set in Figure 3 is chosen.
Figure 4 shows the behavior of the algorithm. In the first 20000 steps the input contains two awfully overlapping classes which cause a high error (b). Nevertheless after 20000 steps, the algorithm has learned by increasing its insertion threshold (c) that a further insertion does not improve the squared error and stabilizes, as can be seen in (d), and the amount of nodes. Now the environment changes, new errors occur and the algorithm tries to minimize them by changing its weights and inserting new nodes. Although the environment gets much easier, there is still an unsolvable overlapping between the ellipse and the line that would cause a further insertion of nodes. By increasing the insertion



	Class	Region	Environment (probability)			
			1	2	3	4
A	1	Rectangle	1	1	0	1
B	1	Line	1	1	1	0
C	2	Ellipse	0	1	1	1
D	3	Circle	1	0	0	0
E	2	Circle	1	1	1	0

Fig. 3. Changing environment based on five areas (A-E). The environment changes from 1-6 after every 20000 steps. The two regions D and E are completely overlapped and the class 1 of the line has an overlap with class 2 of the ellipse. The used parameters are $\eta_b = 0.1$; $\eta_n = 0.01$; $\eta_o = 0.15$; $\eta_\vartheta = 0.5$; $T_S = 20$; $T_L = T_Y = T_\vartheta = 100$; $\lambda = 10$; $\vartheta_{age} = 50$; $\vartheta^i_L = 0.05$; $\vartheta^o_L = -0.05$; $\vartheta_{ins} = 0.1$; $\vartheta_{del} = 0.05$; $\vartheta_{delY} = 0.01$.

threshold (c) of the relevant nodes, the algorithm learns to stop insertion in the overlapping areas. At least after 40000 steps it has adopted to the environment that no further learning is needed (d). If the probability changes in some regions to zero, like in the environment from 40000 to 60000 steps, those remaining nodes, often called "dead nodes", play a major role in life-long learning. They are in no way "dead nodes", instead they preserve the knowledge of previous situations for future decisions. If the old prototype patterns were removed, the knowledge would be lost and the same, already learned situations will again cause errors. Due to a further insertion at the overlapping, still a bit learning takes place (d). In the environment from 60000 to 80000 steps, most of the neurons remain at their positions. Since the environment shows no overlappings the error dereases to zero.

Sumarizing, the algorithm is able to cope with all life-long learning scenarios, like overlaps, never seen inputs and temporarily not appearing inputs.

5 Conclusion

A life-long learning incremental neural network was presented to coordinate insertion and learning. On an abstract level, it demonstrates a biologically feasible selective modification of plasticity induced by a "global teacher" signal.

The experiments show that the network can learn to stop insertion in regions where the error can not be decreased. Furthermore, in changing environments the network remains stable for old prototype patterns and adaptive for new or different patterns. The neural network neither freezes by any decaying parameters nor switches between different learning modes, instead it is able to learn continuously by evaluating its own insertions.

The results obtained indicate a good performance and are a promising step towards life-long learning in neural networks. A performance evaluation on real data shows (Hamker, 1998).

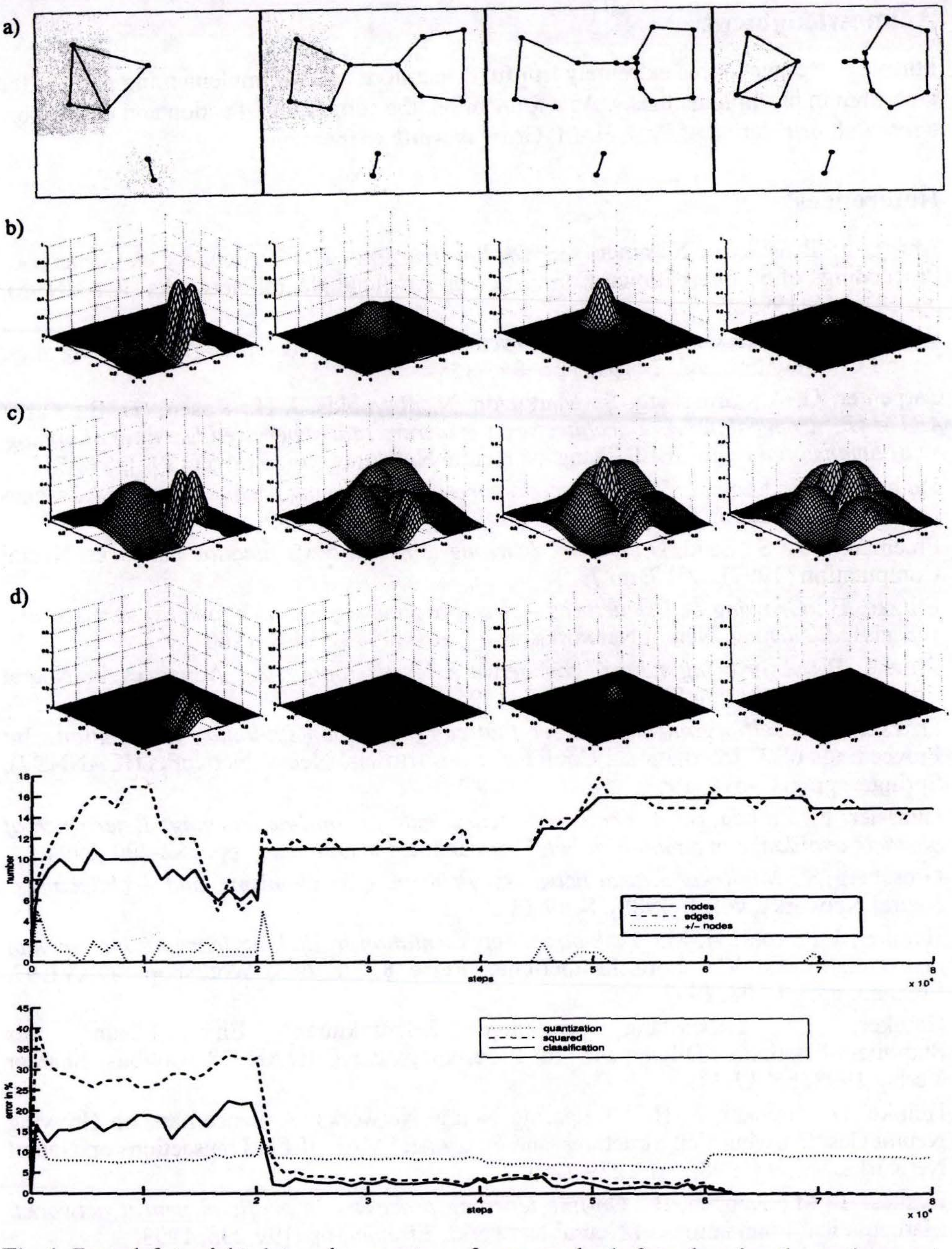


Fig 4. From left to right: internal parameters of every node before changing the environment (20000 steps). From the top to the bottom the: **a)** input weights, **b)** longterm error τ_L **c)** insertion threshold τ_θ **d)** learning parameter α^i, the amount of nodes, and the errors are shown.

Acknowledgment

I thank T. Vesper for his extremely fruitful discussions and for implementing parts of the algorithm in his diploma thesis. As a foundation, the combining of action and perception, a research orientation of Prof. H.-M. Gross is worth to mention.

References

Ahrns, I.; Bruske, J.; Sommer, G.: *On-line learning with Dynamic Cell Structures*. Proceedings of 5th International Conference on Artificial Neural Networks (ICANN'95), pp. 141-146, 1995.

Bruske, J.; Sommer, G.: *Dynamic cell structure learns perfectly topology preserving map*. Neural Computation, vol. 7 (1995) pp. 845-865.

Carpenter, G. A.; Grossberg, S.; Markuzon, N.; Reynolds, J. H.; Rosen, D. B.: *Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps*. IEEE Trans. on Neural Networks, vol. 3 no 5 (1992), 698-713.

Fahlman, S. E.; Lebiere, C.: *The cascade-correlation learning architecture*. In: Advances in Neural Information Processing Systems 2, pp. 524-532, 1989.

Freeman, J. A. S.; Saad, D.: *On-line learning in radial basis function networks*. Neural Computation (1997), vol 9, no 7.

Fritzke, B.: *Growing cell structures – A self-organizing network for unsupervised and supervised learning*. Neural Networks, vol. 7 no 9 (1994), 1441-1460.

Fritzke, B.: *A growing neural gas network learns topologies*. Advances in Neural Information Processing Systems, vol. 7 (1995).

Fritzke, B.: *A self-organizing network that can follow non-stationary distributions*. In: Proceedings of 7th International Conference on Artificial Neural Networks (ICANN'97), Springer, pp. 613-618, 1997.

Gaussier, P.; Zrehen, S.: *A topological neural map for on-line learning: Emergence of obstacle avoidance in a mobile robot*. From animals to animats 3, pp. 282-290, 1994.

Grossberg, S.: *Nonlinear neural networks: Principles, Mechanisms, and Architectures*. Neural Networks, vol. 1 (1988), S. 17-61.

Hamker, F.; Gross, H.-M.: *Task-based representation in lifelong learning incremental neural networks*. VDI Fortschrittberichte, Reihe 8, Nr. 663, Workshop SOAVE'97, Ilmenau, pp. 99-108, 1997.

Hamker, F.: Lebenslang lernfähige Zellstrukturen: Eine Lösung des Stabilitäts-Plastizitäts-Dilemmas? In: Proceedings der CoWAN '98, Cottbus: Sharker Verlag 1998, pp. 17-37.

Heinke, D.; Hamker, F. H.: Comparing Neural Networks: A Benchmark on Growing Neural Gas, Growing Cell Structures, and Fuzzy ARTMAP. IEEE Transactions on Neural Networks, vol. 9 (1998), pp. 1279-1291.

Heskes, T. M.; Kappen, B.: *On-line learning processes in artificial neural networks*. Mathematical Foundations of Neural Networks. Elsevier, pp. 199-233, 1993.

Rescorla, R. A.; Wagner, A. R.: *A theory of Pavlovian conditioning; variations in the effectiveness of reinforcement and nonreinforcement*. Classical Conditioning 2, Current Theory and Research. A. H. Black and W. H. Prokasy (Eds.), New York: Appleton-Century-Crofts, pp. 64-99, 1972.