

Development of a Large-Scale Integrated Neurocognitive Architecture

Part 2: Design and Architecture

J. Reggia, M. Tagamets, J. Contreras-Vidal, D. Jacobs,
S. Weems, W. Naqvi, R. Winder, T. Chabuk, J. Jung, C. Yang

University of Maryland

October, 2006

TR-CS-4827, UMIACS-TR-2006-43

Abstract:

In Part 1 of this report, we outlined a framework for creating an intelligent agent based upon modeling the large-scale functionality of the human brain. Building on those results, we begin Part 2 by specifying the behavioral requirements of a large-scale neurocognitive architecture. The core of our long-term approach remains focused on creating a network of neuromorphic regions that provide the mechanisms needed to meet these requirements. However, for the short term of the next few years, it is likely that optimal results will be obtained by using a hybrid design that also includes symbolic methods from AI/cognitive science and control processes from the field of artificial life. We accordingly propose a three-tiered architecture that integrates these different methods, and describe an ongoing computational study of a prototype “mini-Roboscout” based on this architecture. We also examine the implications of some non-standard computational methods for developing a neurocognitive agent. This examination included computational experiments assessing the effectiveness of genetic programming as a design tool for recurrent neural networks for sequence processing, and experiments measuring the speed-up obtained for adaptive neural networks when they are executed on a graphical processing unit (GPU) rather than a conventional CPU. We conclude that the implementation of a large-scale neurocognitive architecture is feasible, and outline a roadmap for achieving this goal.

Correspondence:

James A. Reggia
Dept. of Computer Science
A.V. Williams Bldg.
University of Maryland
College Park, MD 20742

Email: reggia@cs.umd.edu
Phone: (301) 406-2686
Fax: (301) 405-6707

Acknowledgement: This work is supported by DARPA BICA Award FA87500520272.

Contents – Part 2

| | |
|---|----|
| I. Introduction | 3 |
| II. Extending the Architecture | 4 |
| A. Behavioral Requirements | |
| B. Need for a Hybrid Architecture | |
| III. Design and Implementation | 7 |
| IV. Pilot Study 1: Mini-Roboscout | 11 |
| V. Implications of Non-Standard Methods | 16 |
| A. High Performance Computing | |
| B. Nanotechnology and Quantum Computing | |
| C. Genetic Programming | |
| VI. Pilot Study 2: GPU Cluster Experiment | 20 |
| VII. Pilot Study 3: Evolution of Recurrent Networks | 24 |
| VIII. Roadmap | 28 |
| IX. Conclusions | 29 |
| Literature Cited | 30 |

I. Introduction

In Part 1 of this report [Reggia et al 2006], we presented a conceptual framework for implementing a large-scale neurocognitive architecture based on modeling the hierarchical and modular organization, dynamics and plasticity of the human brain. Our emphasis was on the neocortex and its interactions with subcortical structures that are most closely related to problem-solving, learning and cognition in general. The central motivation for such an approach is that the human brain is currently the only known entity capable of exhibiting robust general intelligence in the form of integrated problem solving, language processing, planning, creative design, and learning.

The primary features of our framework can be summarized as follows. The core of our neurocognitive architecture is a hierarchical network of nested and iterated modules, inspired by the neurobiological structures listed above. These modules have spatial relationships to one another, unlike with many neural models, and this has significant implications for connectivity, functionality and learning. Functionality in our architecture is provided by the activation dynamics of its modules, occurring simultaneously at multiple levels of the structural hierarchy. In other words, our framework is based on a dynamical systems perspective rather than the primarily logical/symbolic approach used in many mainstream cognitive models in psychology and AI. Cognition is viewed as an emergent property of self-organizing neural processes, not something that is directly “programmed in”. Both the structural architecture and the neurobiologically-inspired functional mechanisms are guided not only by the need for good performance but also by a drive to minimize costs (energy use, connectivity, etc.). In part, cost minimization is based upon the strength and nature of functional interactions between brain regions, and is informed by recent human functional imaging data (fMRI, PET, etc.) and electrophysiological data (EEG, MEG, etc.). Working memory, executive control functions, and sequential behavioral processing are represented in multiple ways in our framework, including competition between neural modules for activation that influences global control of activity (one aspect of attentional mechanisms), sustained patterns of neural activity in cortical regions, and recurrent connectivity between regions that can gate one another’s activity. Perhaps most important, the functions of modules are largely learned, not pre-programmed, so that a module’s functionality is determined in part by its location and connectivity, and in part by a “learning agenda” during which different components of the model learn separately in a prescribed, multi-stage fashion before being integrated and trained further collectively, somewhat as occurs in human brain and childhood cognitive development. This learning is a continuous process, implying among other things that our architecture can reorganize after damage and partially recover via dynamic reallocation of functionality.

In this second report, we begin by specifying the behavioral requirements of a large-scale neurocognitive architecture. The core of our long-term approach remains focused on creating a network of neuromorphic regions as described in the preceding paragraph, and these provide the mechanisms needed to meet the specified requirements. However, for the short term of the next few years, it is likely that optimal results will be obtained by using a hybrid design that also includes symbolic methods from AI/cognitive science and control processes from the field of artificial life. We accordingly propose a three-tiered architecture that integrates these different methods, and describe an ongoing computational study of a prototype “mini-Roboscout” based on this architecture. We also examine the implications of some non-standard computational methods for developing a neurocognitive agent. This examination includes computational experiments assessing

the effectiveness of genetic programming as a design tool for recurrent neural networks for sequence processing, and experiments measuring the speed-up obtained for adaptive neural networks when they are executed on a graphical processing unit (GPU) rather than a conventional CPU. We conclude that the implementation of a large-scale neurocognitive architecture is feasible, and outline a roadmap for achieving this goal.

II. Extending the Architecture

Building on large-scale neurocognitive architecture described above, in this section we first outline the behavioral requirements needed for anticipated Decathlon and Challenge problems, and then discuss the implication of these requirements: that over the short term of the next five years, a hybrid architecture is the most reasonable compromise between innovation and satisfying short term performance requirements.

A. Behavioral Requirements

The Decathlon and Challenge problems are intended to provide tests of a broad range of cognitive functions that are integrated within a single agent. The following is a brief summary outlining many of the important capabilities that an agent will need to address these and related tasks. These capabilities are divided into three categories for convenience and for compatibility with our planned architecture: sensorimotor, cognitive and executive capabilities. While some capabilities are arguably within multiple categories, they are only listed once in the category that seems most directly relevant. An exception is learning, which spans all categories. Finally, the point of some of the Decathlon tasks is that multiple capabilities be used in an integrated fashion, and how these capabilities are integrated is not fully captured in the following.

Sensorimotor

Visual system:

- view-independent object recognition/classification
- identification of landmarks for navigation

Audition:

- identification of sounds (gun shot, explosion, car motor, etc.)
- translate speech signal to phoneme sequence?

Proprioception:

- arm/manipulator state

Movement:

- move to specified location, follow vehicle/individual

Arm control:

- move "hand" to specified object, grasp, execute motor program, etc.

Learning:

- sensorimotor transformations
- feature discovery

Cognitive

Navigation:

- navigate through complex environment, recognize and avoid impassable barriers, search for specified object; move while staying concealed

Map reading:

- route planning

Knowledge base (semantic memory):

- general common sense knowledge
- spatial relations (in, on, etc.)
- human relations (military ranks, prisoner vs. guard, etc.)
- naïve physics
- object properties
- containment and other relationships
- types of buildings, vehicles
- people vs. animals, etc.

Motor control:

- motor programs (procedural memory)

Episodic memory:

- items seen, route followed

Inference:

- rule-based deduction
- cause-effect reasoning
- identification of dangerous situation from evidence (e.g., IED)
- significance of emotional state
- reason by analogy (enemy vs. friend)

Spoken language:

- understand and produce commands, statements and questions
- integrate visual information with language processing to disambiguate commands
- identifying nouns etc. in sentences
- taskable via spoken language

Learning:

- categorization of abstract visual patterns
- acquisition of new object name
- abstract rules for classification given feedback (enemy vs. friend)
- relational correspondences between two regions/spaces (e.g., items in a room)
- control UAV via a keypad or other interface (discovery, H&T)
- self-explanation from observation of another agent doing self-explanation
- routes through a city and their landmarks
- map generation: from route following/exploration, location of objects in room(s)
- episodic learning in general
- map of city and objects observed from exploration/following
- recognition of dangerous settings (e.g., IED)

Executive

Goal-directed behavior:

tasking, priority setting, attention mechanisms

Planning:

route to destination, search for object

Re-planning:

recognition of unexpected situations/failure/novelty, diagnosis of reason
revised plan generation, goal revision

Metacognition:

hide and find game (decisions made based on memory of current situation)
self-explanation (e.g., for route determined from a map)

Social:

distinguish animate vs. inanimate entities and understand the implications

Theory of mind:

recognize implications of incorrect knowledge of another agent
predict intended behavior of another agent

Learning:

imitation learning (e.g., ways to open a tube)
learn an opponent's strategy
structure learning episodes and determine when learning is done

B. Need for a Hybrid Architecture

As outlined in the *Introduction*, in Part 1 of this report we put forth a theoretical framework for developing a large-scale neuromorphic architecture. This architecture takes its inspiration largely from current knowledge of the neurobiological basis of human cognition. Our specific architecture, involving repetitive use of generic neural components and multistage learning, should facilitate highly parallel processing, robustness to damage, and eventual physical realization in fine-grained parallel processing architectures. We believe that this approach, or something very much like it, will ultimately be successful in creating a general-purpose machine intelligence. However, uncertainties in contemporary knowledge about brain functions, and in our understanding of how to capture some aspects of cognition in neural algorithms, raise the question of what the optimal strategy is for achieving such an architecture. This is a critical question if one plans to gauge success by the ability of a developing cognitive architecture to function in naturalistic settings within the short period of a few years.

Our answer to this question is that trying to implement a full-scale, purely neuromorphic architecture immediately and all at once would be extremely difficult and carry a high risk of failure. A much better approach over the short term of the next five years would be to develop a *hybrid architecture* that combines neurobiologically-inspired methods and cognitively-inspired methods within a single unified framework. By “cognitively inspired methods”, we mean more conventional symbolic and numeric methods from cognitive science and AI rather than neural computation methods. In such a hybrid architecture there is a third “dimension of integration” in addition to the behavioral tasks and cognitive mechanisms that we described in the Introduction to Part 1. This *computational methodology dimension* of integration refers to combining the variety of computational methods that are available today for producing various aspects of machine

intelligence. At one end of the spectrum are the cognitively-inspired methods that have dominated cognitive science, AI, and related fields. They are often referred to as “top-down” approaches and include symbolic methods such as first order predicate calculus, production systems, and heuristic search as well-as statistical pattern classification techniques. At the other end of this spectrum are neural computation approaches inspired by the brain that have formed the basis of our theoretical framework throughout Part 1 of this report. Also at this end of the spectrum are biologically-inspired methods developed in the field of artificial life. For example, swarm intelligence methods for movement control are particularly relevant [Rodriguez, 2005]. These “bottom-up” approaches typically start with a distributed representation of information and emphasize learning and self-organization, viewing cognition as a phenomenon that emerges from the dynamics of a neurobiologically-inspired complex system, not something that one explicitly programs in. The view presented here is that integrating these computational methodologies rather than restricting one’s approach to just one class of methodologies is most likely to be productive over the short-term of the next few years.

There are at least two reasons for starting with a hybrid approach. First, past successful applications of neurobiologically-inspired and cognitively-inspired machine intelligence on focused, limited-scope tasks have largely been complementary. Neurocomputational methods have excelled at *learning* to do “low-level” tasks like pattern classification, autonomous movement control, and associative memory, plus they have demonstrated a robustness to damage/noise and an ability to generalize. In contrast, more traditional symbolic methods of cognitive science and AI have excelled at “high-level” aspects of cognition such as problem-solving, inference, planning and executive control. The point is that if these complementary neurocomputational and symbolic methods can be effectively integrated, the resultant combination would potentially be much more powerful than either methodology alone. The second primary reason for adapting a hybrid approach is that it automatically leads one to a roadmap for achieving the long-term goal of a fully neuromorphic machine intelligence, by proceeding as follows. Initially, implement a hybrid system with both neuromorphic and symbolic methods. Then, as knowledge of brain function improves and neurocomputational technology advances, gradually replace functions captured in the more traditional symbolic components of this architecture with neuromorphic components. At any time in this process, the remaining cognitively-inspired components effectively define the critical research agenda for achieving a full-scale purely neuromorphic architecture: this consists of the aspects of cognition that remain implemented in the symbolic framework because their neuromorphic implementation remains undefined. In the following we extend our theoretical framework to encompass cognitively-oriented symbolic and other methods in a unified setting and present a roadmap for the development of a full-scale architecture within this extended framework.

III. Design and Implementation

Figure 1 provides an overview of our proposed three-tier architecture. The *sensorimotor level* interacts most directly with the environment, the *cognitive level* is the heart of the system, and the *executive level* captures “executive functions” as that term is used in neuropsychology. This three-tier structure is directly inspired by the organization of the human nervous system. The sensorimotor level corresponds not only to subcortical structures, but also to primary sensorimotor neocortex and to neocortical regions dealing with automatic sensorimotor transformations (e.g., Brodmann area 7a) and inter-modal sensory transformations. The cognitive level corresponds to

neocortical regions often referred to as “association cortex”, such as language areas, substantial portions of parieto-temporal cortex, etc. The executive level corresponds to prefrontal cortex, anterior cingulate gyrus, and related regions. Of course, this partitioning of brain regions and functionality is imprecise and ignores some overlapping of functionality (for a review of this issue, see our earlier report [Tinerella et al, 2006]).

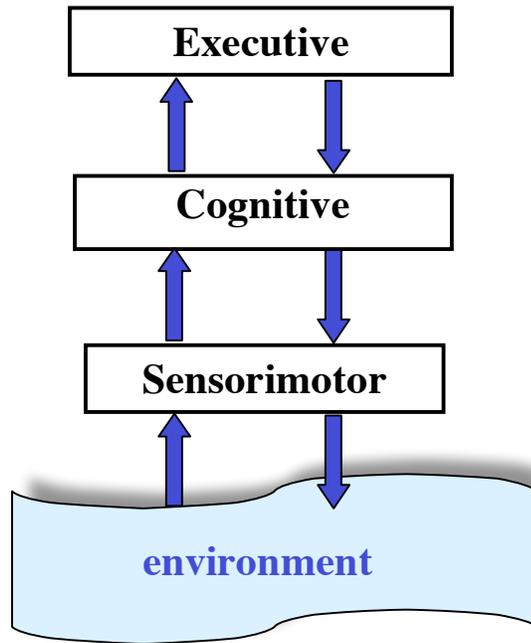


Figure 1: Top level view of the three-tier organization of our proposed large-scale neurocognitive architecture.

Independent of its neurobiological inspiration and correlates, the three tier organization of Figure 1 provides a powerful organizing framework for a cognitive architecture, as follows. The sensorimotor level, which could be implemented and function in isolation of the other levels in an environment, captures functionalities whose execution is largely automatic or “reflexive”. These include basic pattern recognition, sensorimotor coordinate transformations, and elementary actions such as moving through an environment having obstacles or executing an arm control command. This level is also the appropriate place for responses to environmental events that must be responded to very quickly, such as the immediate recognition of dangerous situations and reflexive behavioral actions.

In contrast, the cognitive level encompasses so-called “higher cortical functions” such as language, deduction, cause-effect reasoning, problem solving, motor program selection, etc. Mechanisms at this level, when coupled with the sensorimotor level and even in the absence of the executive level, should make an agent taskable, i.e., able to carry out specific albeit relatively simple goals in response to a command. Episodic memory is also available at this level, permitting an agent, for example, to recall a recently followed route and the objects it observed along the way. While slower than the sensorimotor layer, much of the processing at this cognitive level is still fairly automatic/deliberative, e.g., the recognition of a sequence of phonemes as corresponding to a

specific word. With just the cognitive and sensorimotor levels present, although an agent could follow commands, it would have no ability to judge the appropriateness of its goals, to make elaborate hierarchical plans for achieving those goals, or to understand its own reasoning status.

The third, highest executive level of the architecture carries out “executive functions” by interacting with the cognitive level. It is able to generate new goals and subgoals to be followed by the cognitive level (and thus also the sensorimotor level) based on information about its current situation. This is the most reflective and potentially slowest part of the overall architecture. It is responsible for generating plans (goal sequences), for monitoring execution of those plans, and for generating revised plans (re-planning) when unanticipated events occur. It is also at this executive level that social intelligence appears (theory of mind) and at which inferences can be made about the agent’s own state and reasoning processes (metacognition).

The three-tier architecture we are presenting also provides a fairly natural organizing framework for combining different computational methodologies in a single system. Current neurocomputational and artificial life methods are fairly effective at the sensorimotor level, especially relative to symbolic methods in AI and cognitive psychology. In contrast, symbolic methods currently are more effective at the executive level. In between, at the cognitive level, all of these methods have a role to play. What becomes critical is the need for modularity in the components that form the three levels. Modularity is important for information hiding, including the nature of computations inside of a module, to allow a clear integration of different computational methods in a single, full architecture. Modularity is also critical to a rational roadmap to implementation, as we explain in the Roadmap section later in this report.

Our discussion so far has focused only on how the basic three-tiered architecture of Figure 1 functions in its environment, managing problem-solving and carrying out tasks. We refer to this aspect of the agent as its *basic operation*. An important question in this context is how functions such as memory, learning and attention are to be accommodated within this framework. The answer is that these kinds of functionality span and are essentially orthogonal to the three “horizontal” levels of sensorimotor, cognitive and executive functions, as shown in Figure 2. Their “vertical” nature indicates that, like the agent’s basic operations, their functionality involves all three levels of the core architecture, providing an overall matrix organization.

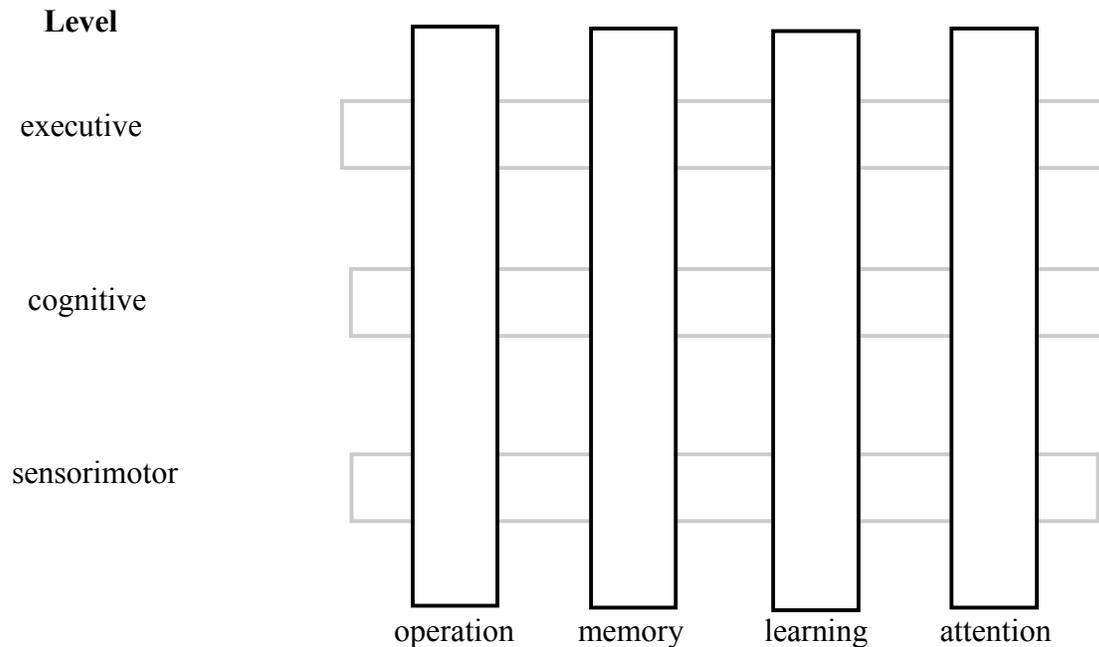


Figure 2: The orthogonal nature of processing mechanisms with respect to the three-tier architecture of Figure 1. The leftmost vertical box corresponds to the basic operation of the trained agent (roughly, this is what is pictured in Figure 1 and described in the preceding text). The remaining vertical boxes indicate that memory, learning and attention all also fit in this same three-tier framework.

Memory functions fit into the same three levels as the basic operation of the agent. For example, at the sensorimotor level weight matrices store inter-modal sensory transformations and simple motor programs, the latter in recurrent networks. The cognitive level encompasses semantic associative knowledge, a lexicon, and episodic plus working memory. The executive level maintains a memory store that includes a current goal/subgoal stack and recollection of the status of specific previously encountered external agents (friend, foe, mentor, competitor, etc.). Likewise, *learning* that alters the contents of memory is distributed across the three functional levels. Many examples are given of level-specific learning mechanisms in Section IIA, and these are assumed to operate in parallel. Finally, attention and control mechanisms operate concurrently at all three levels, an organization that is a natural fit to the three main classes of attentional mechanisms recognized by many cognitive psychologists (reviewed in [Raz & Buhle, 2006]). More specifically, bottom-up *alerting* to external stimuli associated with subcortical brain regions roughly corresponds to our sensorimotor level, bottom-up (exogenous) and top-down (endogenous) *selection/orientation* to specific sensory modality events associated with parietal lobe activation to our cognitive level, and *executive attention* (dealing with conflict identification and metacognition) associated with anterior cingulate cortex activation to our executive level [Raz & Buhle, 2006].

Given the architecture outlined above, what remains to fill in are the detailed components and their interrelations. The specific components that are present at each level are implicitly defined by the sensorimotor, cognitive and executive requirements already listed in Section IIA. Rather than rehashing these components and their details, we turn next instead to the implementation of a prototype system based upon this architecture, using it both to illustrate the details involved and to begin a critical evaluation of the concepts upon which it is based.

IV. Pilot Study 1: Mini-RoboScout

We are currently implementing a prototype agent, referred to here as “mini-RoboScout”, to assess the feasibility of the neuromorphic framework outlined in Part 1, to examine the issues that arise in combining different computational mechanisms, to further evaluate the use of a developmental approach, and to assess the adequacy of the three-tiered architecture described in the preceding section. The central goal of this pilot study is to determine the implications of and barriers to this approach to creating an intelligent agent that is based upon integrating a variety of behavioral functions and computational mechanisms within a single framework. Our intent is to create a “skeletal system” that includes many of the needed components, focusing on the key issue of integrating these components in a coherent fashion, but that does not incorporate components that are as powerful as would be needed in a real environment. We are ultimately interested in an agent that captures many of the abilities of a child, and thus do not focus on a large initial knowledge base. We keep the environment and input/output to the system relatively simple so that we can focus on the primary issue of integrating those components and not the important but low-level details that will eventually need to be addressed. Thus, for example, language input to our prototype agent consists of a sequence of phonemes rather than of an unprocessed acoustic signal, and the visual input has the various objects scattered around the environment color-coded to make their identification and separation from the environment much easier. Further, some aspects of learning are done off-line as a practical step. Ultimately more powerful auditory and visual processing methods and more online learning methods will replace those currently used. However, as long as we use a modular architecture as planned, such improvements should be viable and allow for the incremental and progressive enhancement of the system’s performance. In summary, mini-RoboScout is a partial exploratory implementation of the ultimate target system that is based upon simplified components and in part upon offline learning procedures.

The basic scenario envisioned for mini-RoboScout is that of a grounded, embodied agent that interacts with simulated environments. Thus our pilot study involves implementation of both a practice environment as well as the prototype agent. At each discrete step of simulated time, mini-RoboScout receives an input image indicating what it can see from its current location, and also perhaps a sequence of auditory phonemes representing a heard spoken sentence. It then generates a movement and possibly a sequence of motor phonemes representing its spoken output. The state of the environment is then updated and the cycle begins again.

We first consider the environment, which represents an urban area surrounded by fields and a lake. Figure 3 below shows an aerial view of the simulated setting, with a central “city” composed of roads and buildings. The prototype agent we study does not have access to this map: it is shown here solely to illustrate the environment’s organization. Various objects are scattered around the environment, such as people and vehicles (not illustrated in this figure). At present, life is kept simple by making these objects non-mobile.

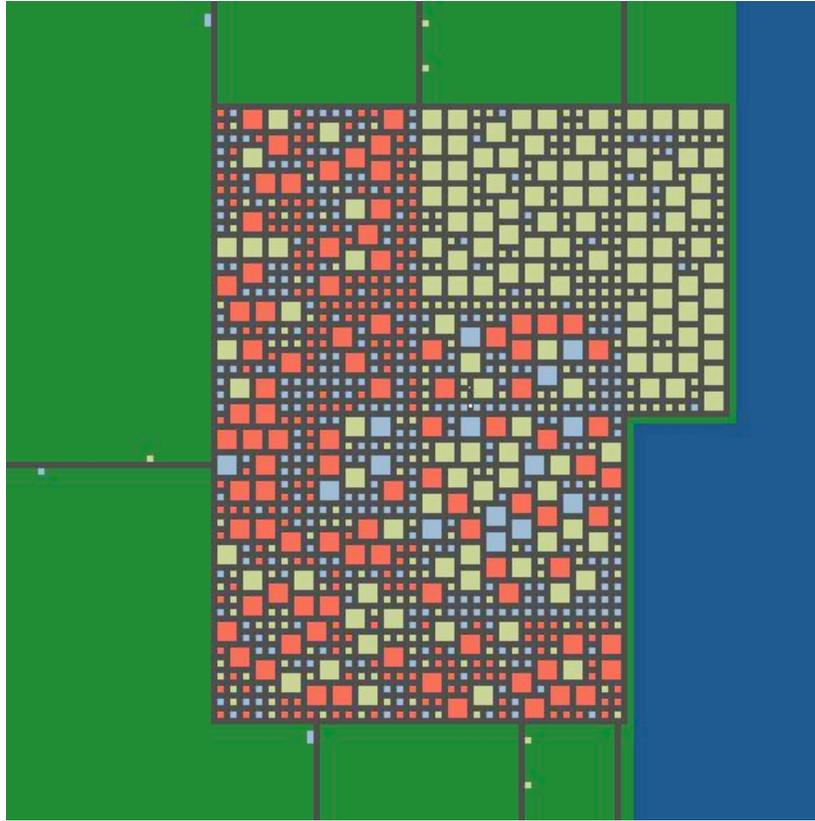


Figure 3: An “aerial view” of the simulated environment that represents an urban area in the central regions consisting of buildings (multicolored) and roads (black). Surrounding regions represent open fields (green) with scattered roads and small buildings, and a body of water (blue) on the right.

At each time step, the agent (mini-RoboScout) receives an updated image representing what it sees at its current location with its current orientation. The environment generates this image based on its current state (i.e., the agent’s location and orientation, the map of the environment, and a database of existing objects). Figure 4 shows a single snapshot of the agent’s current input image. The image is a 512 x 512 pixel array where each array entry is an RGB coded triplet representing the pixel’s color. In this example the agent is in the urban area looking down a road. A stylized person (left near the front) and vehicle (down the road on the right) are represented by distinctive icons. As indicated earlier, these objects are currently color-coded to facilitate their identification so that the agent does not need to deal with the difficult issue of separating objects of interest from background. If the agent elects to focus on one of the agents in the environment, the environment automatically produces a corresponding sketch of the object that is used as input to the agent’s visual system. Figure 5 shows the sketches that correspond to the two objects in Figure 4.

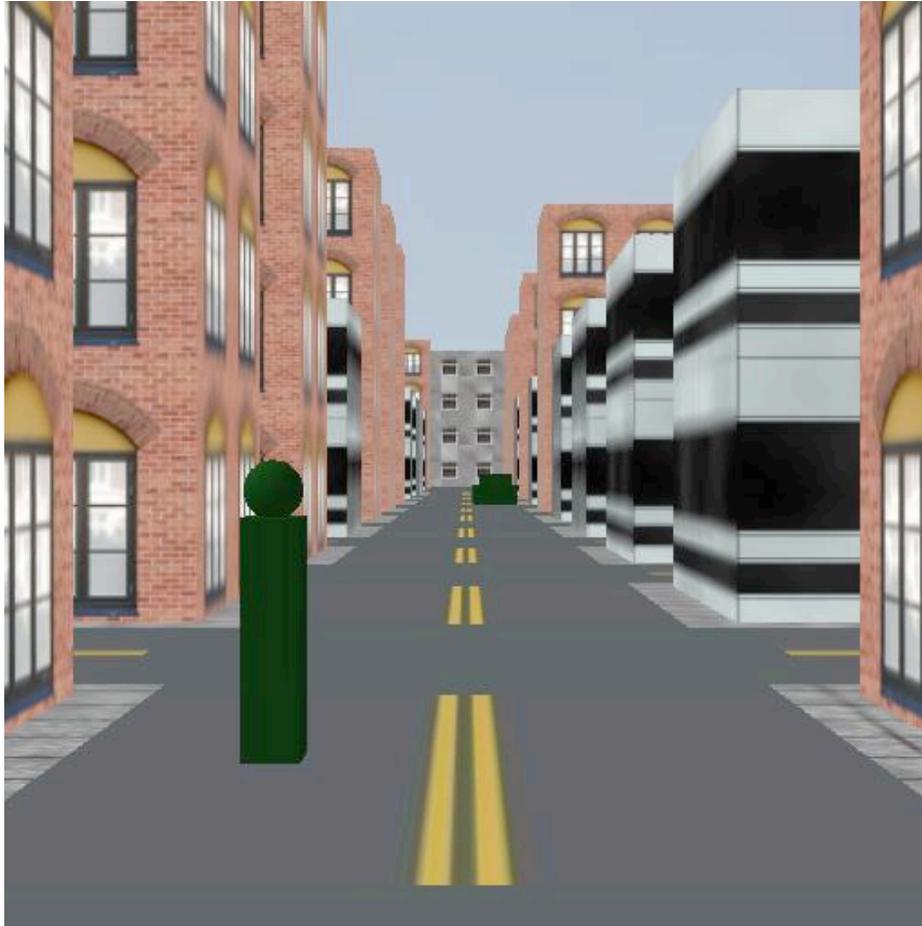


Figure 4: An example snapshot of what the agent sees at a single step of a simulation. The agent is currently looking down a street somewhere in the urban region. This image contains two idealized objects, a person (left near the front) and a vehicle (further down the right side of the road) that are indicated by dark green icons. Visual input to the agent consists of a temporal sequence of such images, each determined by its position and location.

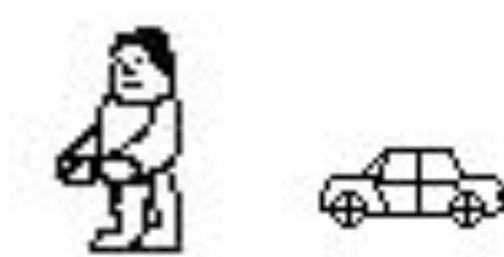


Figure 5: Simple sketches provided by the environment as input to the prototype agent's visual system if it focuses on one of the two objects in Figure 4. A person viewing these sketches would typically classify them as a person (perhaps carrying an object) and a car.

The other input to the agent during a time step is a sequence of auditory phonemes representing a spoken sentence that it hears. Each phoneme is encoded as a set of distinctive features, just as with the WLG model described in Part 1 of this report. On many time steps, no auditory input is received. Using the given visual and auditory input at a time step, the agent must *learn* to identify the objects present, interpret the spoken utterance, adjust its goals, determine its next incremental movement, and generate any appropriate spoken utterance.

The agent's controlling software is implemented as a three-tiered system as illustrated in Figure 1 in the preceding Section. We next consider how the components of each level are currently implemented. Figure 6 below summarizes the mostly neurocomputational components in the agent's sensorimotor level. After learning, the sensorimotor level in isolation (i.e., in the absence of a cognitive or executive level), when set in the state *MoveForward*, is capable of wandering the city, autonomously avoiding barriers to movement and noting objects that it encounters. This level of the system includes swarm intelligence methods for guiding movement control that we have found effective in past multi-agent systems [Winder 2004].

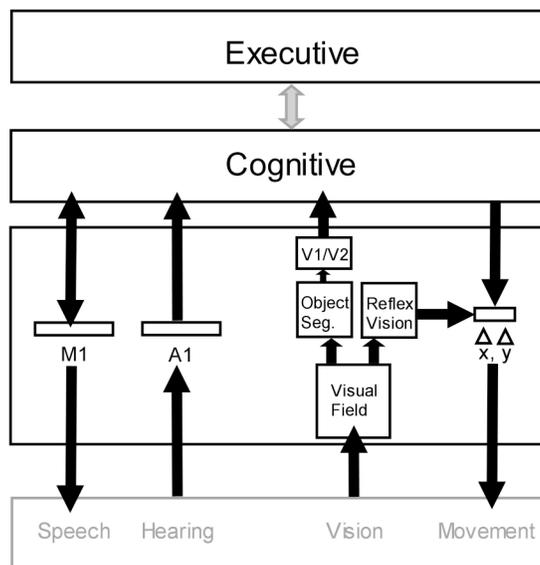


Figure 6: Details of the prototype agent's sensorimotor system. Images flow through two parallel pathways: one in which they are segmented into the individual objects in the environment, and a second one that guides the automatic avoidance of obstacles such as a building, water, or an object when it is too close. These two visual paths roughly correspond to the "what" and "where" pathways in the brain, respectively. The agent also controls incremental movement through the environment (Δx , Δy) and sequences of spoken phonemes via M1.

Figure 7 shows the components in mini-RoboScout's cognitive level. It is at this level that learning to classify object images and their significance is analyzed. Also, after learning, any input temporal sequence of phonemes undergoes a mapping process into its meaning. At present these auditory phoneme sequences consist primarily of simple commands, e.g., the phoneme sequence equivalent of "Go to the market district", where the prototype agent has been given the location of selected regions of the city a priori as rough coordinate boundaries. This level also

generates spoken sentences when appropriate, and exerts top-down influences on the sensorimotor level's movement control to guide the agent to appropriate target locations.

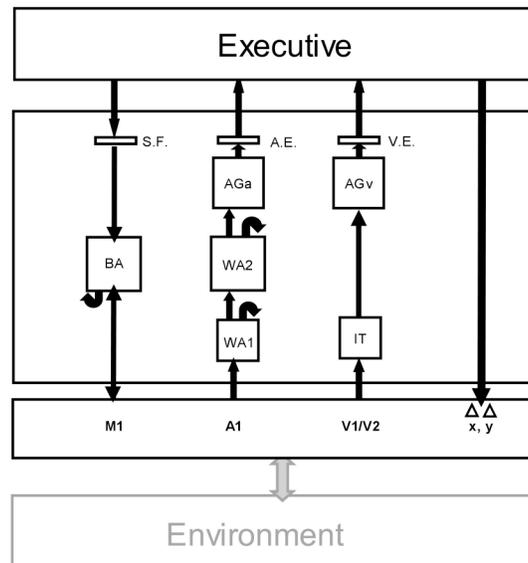


Figure 7: Details of the prototype agent's cognitive level. Object images are processed through a sequence of neural networks here (IT, AGv). Sequences of auditory phonemes are also processed via a sequence of recurrent neural networks (WA1, WA2, etc.), generating a representation of the input sentence's meaning. Both visual and auditory information are passed to the executive control to influence agent decisions and goal selection. In the opposite direction, phonemes are also produced from a set of concepts the agent has decided to state, and these phonemes are sent to the sensorimotor system to be spoken. S.F., A.E., and V.E. stand for speech features, audio encoding, and visual encoding, respectively.

Finally, Figure 8 illustrates the components in mini-RoboScout's executive level. These components currently consist of two functions implemented as symbol-processing modules. A command memory stores recently received commands. The second component, a production system of control rules, examines the current situation and generates and prioritizes the prototype agent's goals.

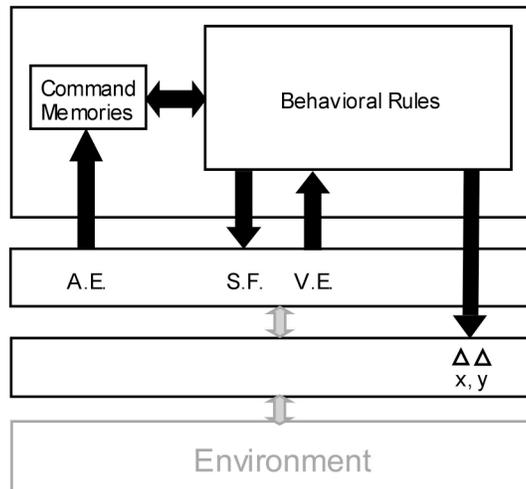


Figure 8: Details of the agent's executive control. The command memories are a list of all the auditory stimuli passed into the executive control via the cognitive network that have not become obsolete yet. Depending on these memories, along with current visual stimuli and obstacles present in the reflex vision system, the behavioral rules determine both the movement of the agent, any spoken output that the agent would make, and also which command memories are obsolete or accomplished and may be removed.

At the time of this writing, the environment simulator is implemented but still undergoing some changes. The sensorimotor level is complete, but is undergoing refinements to improve its learning and performance abilities. Both the cognitive and executive levels are partially implemented and integrated into the system, but need expansion to meet the functionality described above, and this expansion is currently in progress.

V. Implications of Non-Standard Methods

The highly parallel nature of neural computations, and the potential parallel implementation of symbolic and other cognitive/AI algorithms, raise the issue of how the processing in our neurocognitive architecture can take advantage of parallel processing in non-standard computer architectures. Given that the computational costs in cognitive systems can be quite large, there is substantial motivation for examining this issue. In the following, we first consider high performance computing systems that are currently available, and then we examine some longer term possibilities such as nanotechnology, quantum computing, and the use of evolutionary computation as a design aid.

A. High Performance Computing

The brain employs massive parallelism to allow it to perform complex calculations in real time. Artificial systems of sufficient complexity also face significant challenges in producing real time performance, and these systems could potentially benefit from using massive parallelism too. For this reason, we consider here the possible use of parallel computation in our

hybrid reasoning system. We focus on performing neural net computations on parallel hardware. These neural computations map more naturally into parallel systems than the more inherently sequential processes in the non-neural components of our architecture.

Different types of neural networks present different challenges to parallel processing. Most importantly, the type of interconnectivity in a network can influence the amount of communication required between different processors. This can vary significantly between feed-forward and recursive neural nets. Communication between neural units is typically one of the chief bottlenecks in parallel processing. None-the-less, significant speedups have been reported by using parallel hardware to implement multi-layer perceptrons ([Long and Gupta 05; Pethick et al.; Seiffert 02]. For example, [Long and Gupta 05] report experiments on both a 160 node Beowulf cluster, and on a system containing 500 Intel Itanium processors. They report that they are able to maintain constant run times as the number of neural units and the number of processors both scale linearly.

The asynchronous nature of processing in spiking neural networks may create additional challenges to parallelization. [Grassman et al, 02] and [Mouraud et al. 06] model this using event driven simulations. This requires more complex control and synchronization than would be needed for a more uniform network.

While a variety of hardware platforms have been considered for parallel implementations of neural networks, including FPGAs [Zhu and Sutton 03], [Seiffert 02] argues that typically clusters, such as a Beowulf cluster, are the most practical choice due to their wide availability and good performance/price ratio. In fact, one of our goals is to explore parallel implementations of neural networks for clusters that use efficient, off-the-shelf hardware. For example, Graphics Processing Units (GPUs) offer exciting potential for high performance computing because their use in 3D consumer games is driving a dramatic increase in their performance/price ratio. Some estimates describe a 2.8-fold annual growth rate in processing power, compared to a Moore's law rate of 1.7 per year. Because of this opportunity, general purpose programming tools are springing up for GPUs, including linear algebra libraries and high level programming languages.

In part due to interest in using neural nets to control game characters, there have already been implementations of neural nets for GPUs. [Bernhard and Keriven 05] describe an implementation of a neural net with spiking neurons on a GPU. This includes a general purpose spiking neuron simulator. They obtain significant speedups of up to a factor of twenty compared to a comparable system running on a CPU. [Oh and Jung 04] obtain comparable speedups in their implementation of multilayer perceptrons on a GPU. At the same time, implementation on a GPU is more difficult, because operations must be mapped onto vertex and pixel shading operations, which the GPU can then perform in parallel using its 24 pipelines. However, the growing availability of general-purpose libraries such as [BrookGPU] should ease this process in the future.

To our knowledge, neural network systems have not yet been implemented on a cluster of GPUs. This offers the potential for tremendous speedups, but also raises a number of challenges. These include finding effective ways to partition the computation in a scalable way across a large number of GPUs. The modularity of our brain inspired architecture should make this process somewhat easier, however. Some preliminary steps are summarized in Section VI, which

describes a pilot study we undertook as part of this work to evaluate neural network simulations run on GPU processors.

B. Nanotechnology and Quantum Computing

Nanotechnology refers to the fabrication processes and device structures used to build transistors or circuit elements that are smaller than roughly 100 nanometers in size. For example, the insulating gate oxide in the state-of-the-art metal oxide semiconductor field-effect transistors (MOSFETs) consists of only five atomic layers, which add up to only 1.2 nanometers in thickness [Tyagi et al, 2005; Jan et al, 2005]. The gate length has reached 35nm in high-volume microprocessor manufacturing on 300mm wafers. Downsizing has been successful in increasing the density and enhancing performance of integrated circuits. With new fabrication methods, strained silicon, low- and high-k dielectric, surround-gate structure, etc., industry continues to develop new downscaling recipes and to pack more transistors on a chip. However, even before the laws of physics set a clear, hard limit, heat dissipation imposes a practical size limit. There are two main applications of nanoscale MOSFETs: CPU and memory, and the most advanced chips contain more than one billion transistors. The high-speed operation of the CPU inevitably results in high power consumption. Today, conventional power dissipation technology sets a limit at about 100Watts [Ravi et al 2005]. Because of this heat dissipation problem, Intel officially abandoned their effort to boost up clock frequency to 4GHz, and is beginning to look into multiple-processor and parallel computing as an engineering solution.

While the IC industry continues on the path of downscaling for commercial products, researchers are also looking for solutions. The ultimate downsizing of transistors is expected to be to only nanometers, and operating principles will face a fundamental change. As opposed to classical diffusive transport, quantum phenomena, including single electron charge, size quantization, electron wave-like interference, and ballistic transport, are expected to dominate transistor characteristics. New classes of materials, such as semiconductor nanowires, carbon nanotubes, and even molecules are being considered and investigated. The current thinking is that these alternatives could work in conjunction with conventional CMOS circuits and that a hybrid chip can, for example, deliver both high speed computing and high volume memory. Note that these potentially lower-power alternatives must still obey the laws of physics, including the same thermal dissipation issue. We will potentially be able to utilize the remarkable new properties of these quantum-based transistors in computing. The main anticipated difficulty is that quantum-based transistors operate without dissipation, for otherwise phonon emission (energy loss) destroys the quantum state of electrons. Although appealing for high-speed, low-power switching, it remains to find ways to cascade many stages in series for practical system applications.

The use of nanotechnology in neuroscience is an emerging research area, with current work examining technologies that can interact with neurons and glial cells at the molecular level, advanced imaging and manipulation of neurons using functionalized quantum dots, and approaches to supporting functional neural regeneration following nervous system trauma (reviewed in [Silva 2006]). Much less has been done with nanoelectronics for artificial neurocomputational systems. In spite of the continual advancement of nanotechnology, forming interconnects even with 3D integration is still the primary obstacle in circuit implementation, and the interconnect requirement will ultimately limit the number of neurons and the functionality of conceivable neuron-chips. There is one apparent advantage to developing such new nanotechnologies around silicon: there is already an existing infrastructure.

On the other hand, quantum computing is a totally different approach [Nielsen & Chuang 2000]. Unlike a classical bit, the qubit (quantum bit) is a superposition of the two states $|0\rangle$ and $|1\rangle$. Quantum mechanical operations, such as superposition of two eigenstates, entanglement of two qubits, unitary transformation, logic gates, etc., are designed to perform operations on these qubits. Though quantum computing is still in its infancy, extensive experiments have been carried out and demonstrated the validity of the quantum computer concept. However, a full-blown quantum computer is many years away. The recent development and funding of quantum hardware implementations is actually driven by proposed applications in factorization, teleportation, encryption, and sorting algorithms. New algorithms in computing and information processing are still being investigated. At this moment, there is strong competition in developing a semiconductor-based qubit, preferably in silicon. A single electron confined in a semiconductor quantum dot, under the influence of a magnetic field, forms a two-level system that is considered ideal as a qubit. The practical difficulty is to engineer such a quantum dot and perform experiments to manipulate and measure the electron spin. Multiple operations, estimated to be of the order of 10^6 , must be done before the electron spin loses its spin phase coherence. Any breakthrough in constructing a qubit in any material system that can satisfy several basic requirements, i.e., long coherence times and up-scalability, will have a large impact on the computing community. Inspired by parallelism, which is an aspect of quantum computing by default, applications in artificial neural networks have been proposed in the areas of classification, associative memory, image processing, and pattern recognition. As in the case of developing a classical computer, feasibility has to be determined by the characteristics of the physical properties of the qubit and quantum logic gates. One fundamental problem to be addressed first is that quantum mechanics is linear, but neural networks generally involve nonlinear effects. The discrepancy might be solved by allowing a qubit to interact with its environment and to be subject to a time-varying external perturbation. This is a research field in its infancy. The parallelism nature of quantum computing should be fully exploited for a clear understanding to the potential impact in artificial neural networks.

C. Genetic Programming

Evolutionary computation refers to a set of general-purpose search algorithms inspired by natural selection and evolution [DeJong 2006]. These algorithms use a population of individuals that represent potential solutions for a given problem. During each generation, the environment (via a fitness function) indicates which individuals/solutions are more fit than others, and the next generation of the population is produced by selecting the most fit individuals and modifying them via mutation, recombination, and/or other genetic operators. In this way, starting from an initial randomly-generated population that usually represents poor solutions to a problem, progressively better solutions are identified over time.

There are a variety of different approaches to evolutionary computation, including genetic algorithms, evolutionary programming and evolutionary strategies. Each approach, in its canonical form, has its own representation scheme and genetic operators, as well as different philosophies/details to the simulated evolutionary process. For example, considering just the representation of the genetic encoding, genetic algorithms use binary strings, evolutionary programming uses finite state machines, and evolution strategies use real-valued vectors. The approach we focus on here is called *genetic programming*. Genetic programming (GP) literally

refers to the evolution of computer programs, but is also often taken to mean evolution of data structures represented as trees [Banzhof et al, 1998]. Mutations typically are formulated as replacement of a randomly-selected subtree with a new, randomly-generated subtree; crossover is implemented as the swapping of subtrees between two individuals. As with genetic algorithms, most workers in GP take crossover to be the primary genetic operator.

During the last several years, there has been increasing use of GP, as well as other evolutionary computation methods, as a creativity tool in design problems. For example, GP has been used to evolve patentable electronic circuits, antenna configurations, novel pilot combat maneuvers, music, and robotic mechanisms. For example, our own group has used both genetic algorithms [Lohn & Reggia 1997] as well as GP [Pan & Reggia 2006] to evolve rules that produce self-replicating structures in cellular automata environments. Two key points come out of all of this design-oriented GP work. First, GP tends to discover solutions to problems that are creative in the sense that they are quite different than what human designers produce. Sometimes the solutions are substantially better than past human solutions. Second, the use of GP can be viewed as a type of machine learning. In essence, GP involves a fitness-guided search through the space of potential designs for a problem, learning which designs are most effective as it goes.

Given the recent progress in using GP as a design aid, a natural question is whether GP might be adopted to create/discover novel aspects of a neurocognitive agent. We focus on the neural components in the following, but the applicability may actually be broader than just that. Neuroevolutionary methods have been used to create a substantial range of interesting neural network designs (e.g., see [Yao 1999] for examples). Two aspects of this past work are most relevant to biologically-inspired cognitive architectures. First, *developmental representations* of genetic material have been devised that specify how to “grow” a neural network (the phenotype) rather than directly encoding its structure. Examples include graph generation grammars [Kitano 1994] and cellular encodings [Gruau 1996]. In addition to incorporating a model of the biological process of neurodevelopment, developmental representations let one represent individuals as a tree (the genome) while evolving general-graph structures as neural networks. Second, growing attention has been given to producing neural systems having *modular architectures*. Work in this area has been inspired in part by recognition that biological nervous systems are highly modular and hierarchical. For example, the vertebrate cerebral cortex is composed of cortical columns (small modules) that are in turn components of functional regions (large modules) that collectively form the cerebral cortex.

Combining modular design with developmental encodings, and integrating GP evolution of neural network architectures with more conventional neural network learning via synaptic weight changes, seem especially promising avenues to explore. In Section VII below, we describe a pilot study evolving recurrent neural network modules to examine this hypothesis.

VI. Pilot Study 2: GPU Cluster Experiment

In an attempt to explore the computational power of GPUs, we developed a program written in C and Cg to simulate training a feed-forward neural network with a single hidden layer using error back-propagation. As noted earlier, performing computation on a GPU is challenging because operations must be mapped onto vertex and pixel shading operations. In order to pass information to the GPU, textures are created in which the relevant data is stored. Vertex and

fragment programs are defined such that, when rendering takes place and the programs are executed, the desired computation is performed.

Consider, for example, calculating the weighted sum of I inputs to a hidden layer of size H , for all training examples from a training set of size T . In order to perform this computation using a GPU, two textures were defined. One texture was created that contained all of the input data for every training example. The dimension of this training-set-texture was $T \times I$. A second texture was created to store the connection weights from every input neuron to every hidden neuron. This connection-weight-texture was of dimensions $H \times I$. In order to compute the weighted sum of all of the inputs to the hidden layer, a rectangle with a width of H and a height of T was rendered. Texture coordinates were assigned to the rectangle, such that the lower left corner had texture coordinates of $(0,0)$ while the upper right corner of the rectangle had texture coordinates of (H,T) . After rendering was performed, a pixel in the i^{th} column and j^{th} row of this rectangle contained the weighted sum of the inputs to the i^{th} hidden neuron, for the j^{th} training set.

When rendering takes place, the fragment program being used is executed once for every pixel in screen space. An orthogonal projection and an appropriately sized viewport were used in order to ensure a one-to-one mapping between screen coordinates, texture coordinates, and geometry coordinates. A fragment program was written that received the two textures and the texture coordinates as parameters. Each time the fragment program was executed, it calculated the weighted sum of inputs for a particular hidden neuron for a particular training example. The fragment program was able to identify which hidden neuron and training set it was to perform the computation for by the texture coordinates that it received as parameters, and thus pulled the relevant information from the training-set-texture and connection-weight-texture. The fragment program calculated the weighted sum and used the resulting value to set the red value of the rendered pixel. Similar strategies were used to calculate the sigmoid function values, to calculate values for successive layers of the network, and to perform error back-propagation.

Limitations on the length of the fragment program made it necessary to implement a multiple-pass rendering approach. For example, for a neural network with 128 input neurons, the weighted sum of all 128 input neurons could not be calculated in one execution of a fragment program. Therefore, an approach was taken wherein the weighted sum of the first 64 inputs was calculated, and then in a second rendering pass, the next 64 weighted sums were calculated and added to the previous result. This approach allowed us to perform the computation for neural networks of arbitrary size.

Rather than rendering to the screen, our implementation used an extension to OpenGL called framebuffer objects (FBO) to render to off-screen buffers. The use of FBOs is critical as it improves performance for a number of reasons. FBOs offer 32 bits of precision for floating point numbers, which is considerably higher than the 8 bits of precision offered by rendering to the screen. Additionally, using FBOs allows for rendering directly to a texture. This is crucial, as it means that throughout the computation process data may remain on the GPU and does not need to be passed between the CPU and GPU, which is a very inefficient and slow process.

In order to evaluate the speed of our GPU implementation, a CPU-only version of a neural network, written in C, was also developed as a control. We applied these two different implementations to an image classification problem in order to compare performance. The

training set for this problem consisted of 13 sketch images from an urban warfare setting. For each of these 64 by 64 pixel images, there were 13 possible observations to be made. Each image had a distinct combination of which observations were to be made.

A neural network with 4096 input neurons, 64 hidden neurons, and 13 output neurons was used to solve this classification problem. The GPU and CPU-only implementations were trained for 600 iterations. During each of these iterations, all training sets were evaluated and error back-propagation was performed. For each implementation, two such runs were performed. During each run the error of the networks was recorded, while in the other run the time it took to perform the iterations was recorded.

Network Error

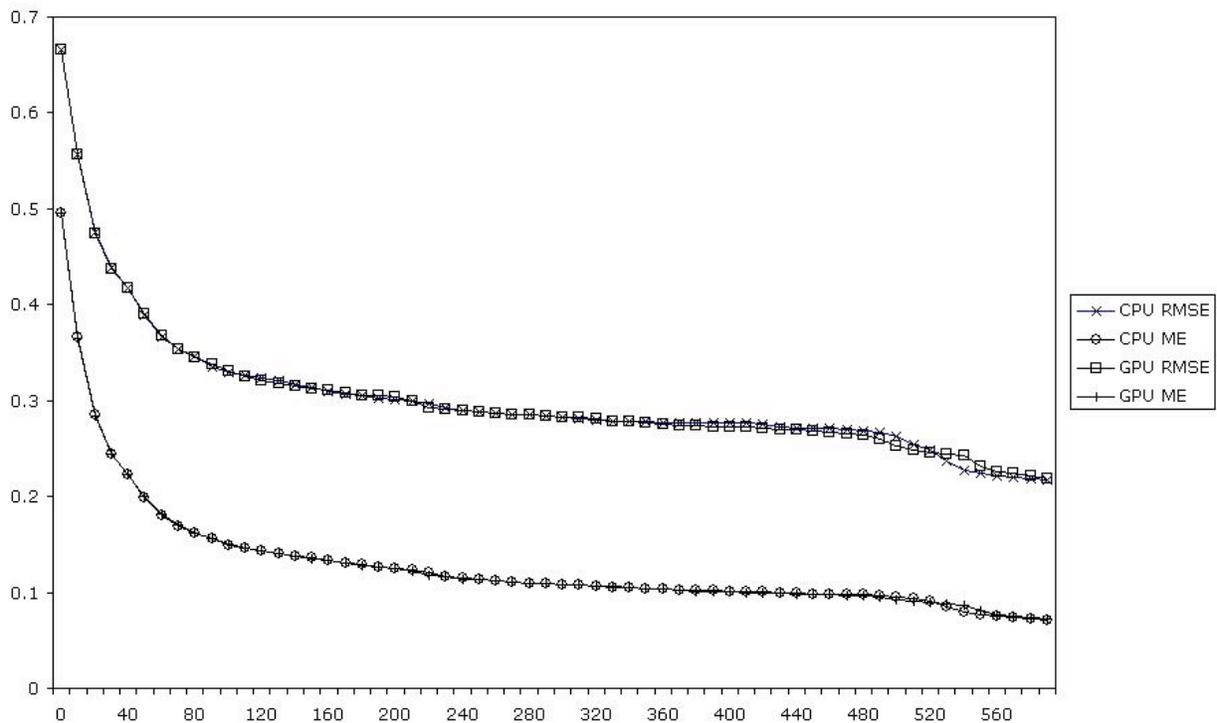


Figure 9: Measures of error (vertical axis) over iterations (horizontal axis) in the GPU and CPU implementations of a neural network.

Figure 9 shows the measures of error in the two simulated networks over 600 iterations. The errors of the two networks are almost identical. This is important as it shows that the two identical networks are being simulated almost exactly the same on the two different implementations. The small discrepancies in network error late in the simulations may be due to the accumulative effect of small differences in rounding error between the two implementations.

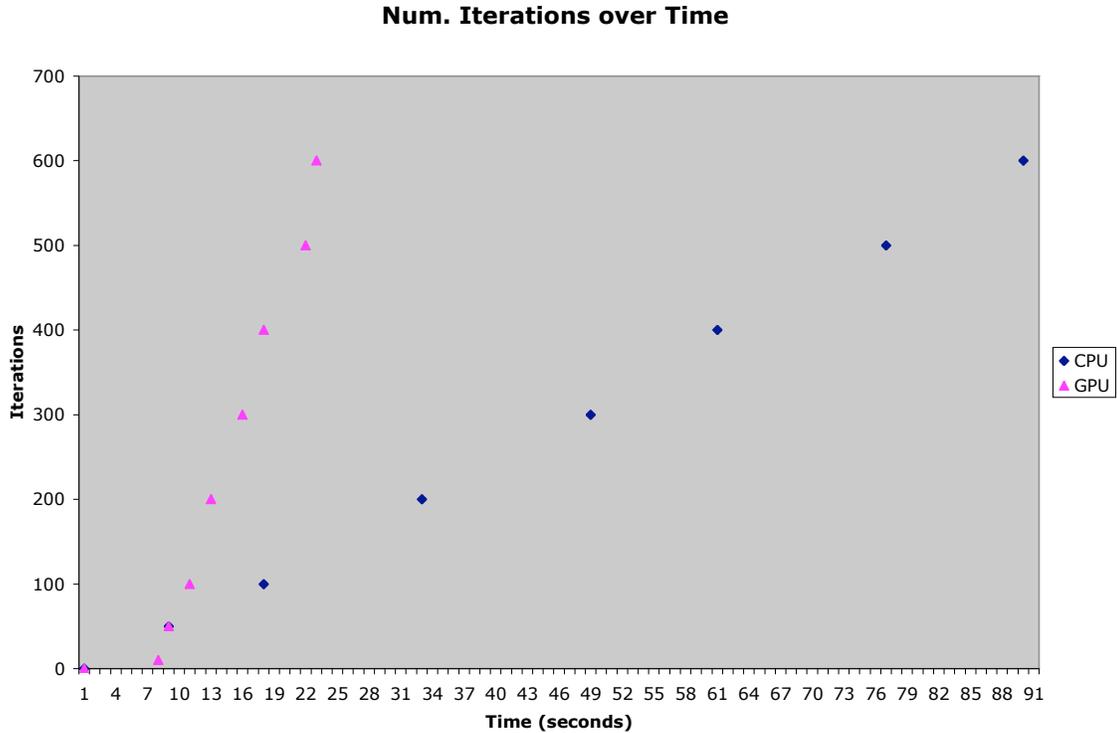


Figure 10: Iterations performed over time by GPU and CPU implementations of neural networks.

Figure 10 displays how much time was required by each implementation to perform the 600 iterations. The GPU implementation started fairly slowly, likely due to one-time start up costs such as creating and binding textures, but quickly surpassed the CPU implementation. In the first 8 seconds, the GPU implementation performed the same number of iterations as the CPU implementation. As more time passed, the GPU implementation significantly outperformed the CPU implementation. While it took the CPU implementation 89 seconds to perform 600 iterations, the GPU implementation took only 22 seconds.

While promising, these results only reflect the performance of the two implementations on one particular set of training data and when simulating one particular network. Additional experiments were performed in order to evaluate the performance of the two implementations on varying random training data. In each of the experiments, the size of the input layer was fixed at 4096, the size of the output layer was fixed at 32, and the size of the sets of training examples (randomly generated) was fixed at 32. The size of the hidden layer was varied between 64, 128, 256, 512, and 1024.

Table 1 below shows the performance of the two implementations during each of these simulations. The GPU clearly outperformed the CPU implementation in every trial. Furthermore, the difference in performance appears to scale in proportion to the size of the hidden layer.

Table 1: Time to perform 100 iterations by CPU and GPU implementations

| Hidden Layer Size | CPU Time (seconds) | GPU Time (seconds) |
|-------------------|--------------------|--------------------|
| 64 | 51 | 10 |
| 128 | 120 | 13 |
| 256 | 213 | 21 |
| 512 | 452 | 35 |
| 1024 | 967 | 62 |

On average, the GPU implementation simulated the neural network 10.6 times faster than the CPU implementation. This is below the improvement by a factor of 20 that was reported by [Oh and Jung 04]. However, there are some improvements that may be made to the current implementation that have the potential to drastically improve performance. The current implementation fails to take advantage of the data types supported by the GPU's specialized hardware. Specifically, using all four channels of textures and the float4 data type, which packs four floating-point numbers into one data member and may be used to perform dot and cross products very quickly, offers the potential for drastic speed improvements when computing weighted sums. These improvements have the potential to push the performance of our implementation well past the speedup by a factor of 20 benchmark.

VII. Pilot Study 3: Evolution of Recurrent Networks

We undertook a pilot study using genetic programming (GP) as a design tool to assist with creating a recurrent neural network for sequence processing. Our goal was to critically assess the potential of this approach to help optimize the components used in a large scale neurocognitive architecture. In performing this "computational experiment", we used a general purpose software environment for evolving neural network architectures that is under development at the University of Maryland [Jung & Reggia, 2006]. Figure 11 illustrates this system, which emphasizes the integration of evolutionary processes with developmental and learning processes, plus supports the creation of modular networks. To use this system, one specifies a class of neural network architectures that are to be considered. In other words, the space of all neural networks is too large to search, so one instead indicates the class of architectures to be considered by the evolutionary process. This is done using a high-level descriptive language to indicate the sets of modules (layers), allowable inter-module pathways, and other aspects of a neural architecture that may potentially be included as part of an architecture (the input description file at the upper left of Figure 11). Following an initialization step in which a random population of genotypes is created within the search space, the evolutionary process then involves a repeated cycle of three stages: development, learning, and genetic operations. The development stage literally grows each neural network (phenotype) from its high-level description; the learning stage then lets weight changes occur during a learning process based on data relevant to the specific task at hand; and finally, the fitness of each network is assessed followed by fitness-guided non-deterministic selection of parents from the environment and mutation, crossover and reproduction (producing the next generation). Fitness criteria may reflect not only network performance on the task at hand (e.g., mean squared error in pattern classification), but also measures of the network's properties (e.g., total number of nodes/connections).

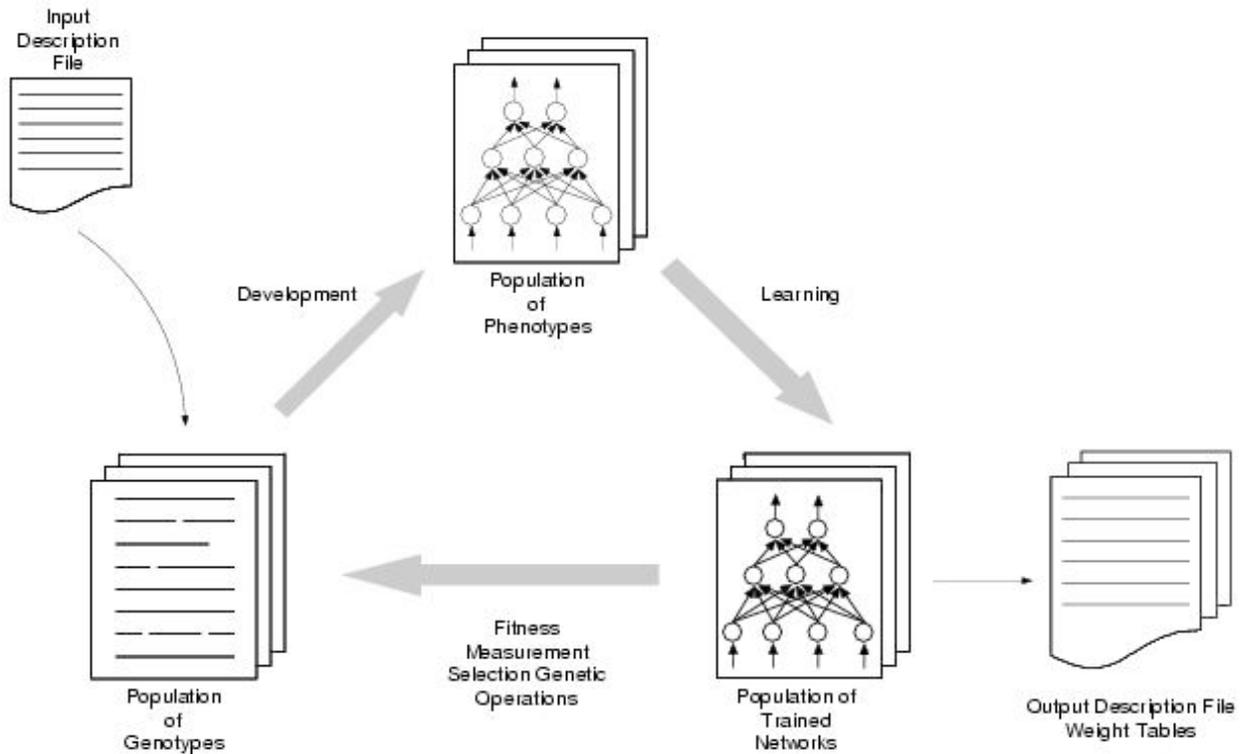


Figure 11: The iterative three-step development, learning and evolution procedure used in our system. The input description file (upper left) is a human-written specification of the class of neural networks to be evolved (the space of neural architectures to be searched) by the evolutionary process; the output description file (lower right) is a human-readable specification of the best specific networks obtained.

The specific task we considered in this context is the problem of learning a temporal sequence of phoneme outputs that correspond to a given fixed input word pattern. For example, given the word *apple* as a fixed input pattern of five written letters A P P L E, the neural network should learn to generate the phoneme sequence /ae/, /p/, and /l/ followed by an end-of-word break in the output. A set of 230 variable length (2 to 6 phonemes) words were used for training. The space of neural networks to be considered by the evolutionary process is shown in Figure 12. The fixed part of the structures is a feed-forward, three-layer network consisting of input, hidden and output layers as depicted on the right side of the illustration. The parts of the architecture to be evolved included the additional hidden/delay layers and their connectivity as indicated on the left side of the illustration and labeled with question marks. Feedback comes from the core hidden and output layers on the right, but where that feedback goes, how many hidden delay layers are used, etc. in the feedback process are evolved. Fitness of networks was based on two cost measures: root mean squared error (RMSE) to assess performance, and the total sum of absolute values of network weights following learning to penalize larger networks. A multi-objective evolutionary algorithm (SPEA) was used.

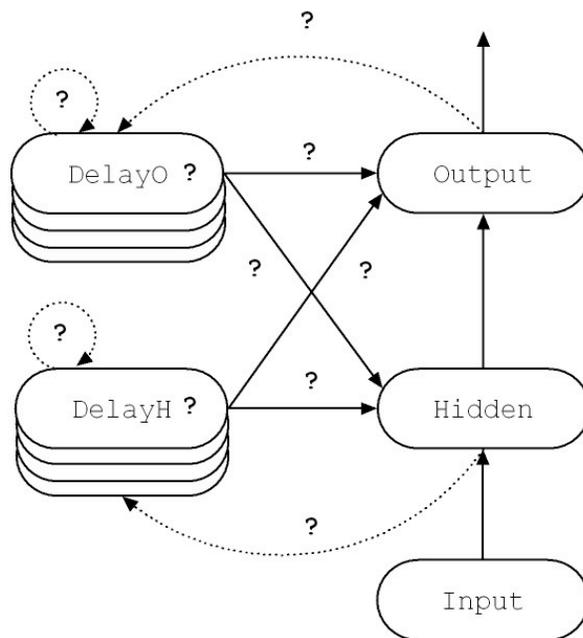


Figure 12: A schematic illustration of the space of neural network architectures that are to be searched in obtaining a good architecture for generating temporal sequences of phonemes representing the correct pronunciations of 230 words. Dotted lines designate non-trainable, one-to-one feedback connections; solid lines indicate weighted, fully connected pathways trained by error backpropagation during each generation of the evolutionary process prior to fitness assessment.

We ran a total of 100 evolutionary processes, each time having a population size of 25 networks and involving 50 generations. The initial population of neural network architectures and their initial weights were determined randomly (and thus differed) in each run. Only mutation was used as a genetic operator. The results are shown in Figure 13, averaged over the same architectures (i.e., each point in the figure represents a network having a specific number of hidden and output delays, and a specific layer-to-layer connectivity). The two fitness criteria are on the axes: network RMSE on the vertical axis, and sum of network weight magnitudes on the horizontal axis. Following the Pareto optimal front (solid line) downwards from the upper left, one has initially very simple networks with only one or two hidden delay layers. As networks get additional hidden delay layers and connectivity, one gets better performance. In other words, moving rightwards along the solid line gives progressively more accurate but more complex networks. Figure 14 shows two examples of networks evolved in this fashion. The network labeled Ho_2 has two evolved delay layers receiving feedback from its fixed hidden layer on the right, with both sending their activity to the output layer directly. The network labeled Hh_1Oo_1 has two evolved delay hidden layers sending/receiving connections from the fixed hidden and output layers in network's center. The point is that even though these two networks have the same numbers of hidden delay layers and the same number of pathways (arrows), they have markedly difference performance measurements (as can be seen in Figure 13), with the network Hh_1Oo_1 being able to learn to generate correct phoneme sequences qualitatively better than the network Ho_2 can do. These and other similar insights were not at all apparent prior to doing the evolutionary runs, and to our knowledge have not been demonstrated in past neural network studies.

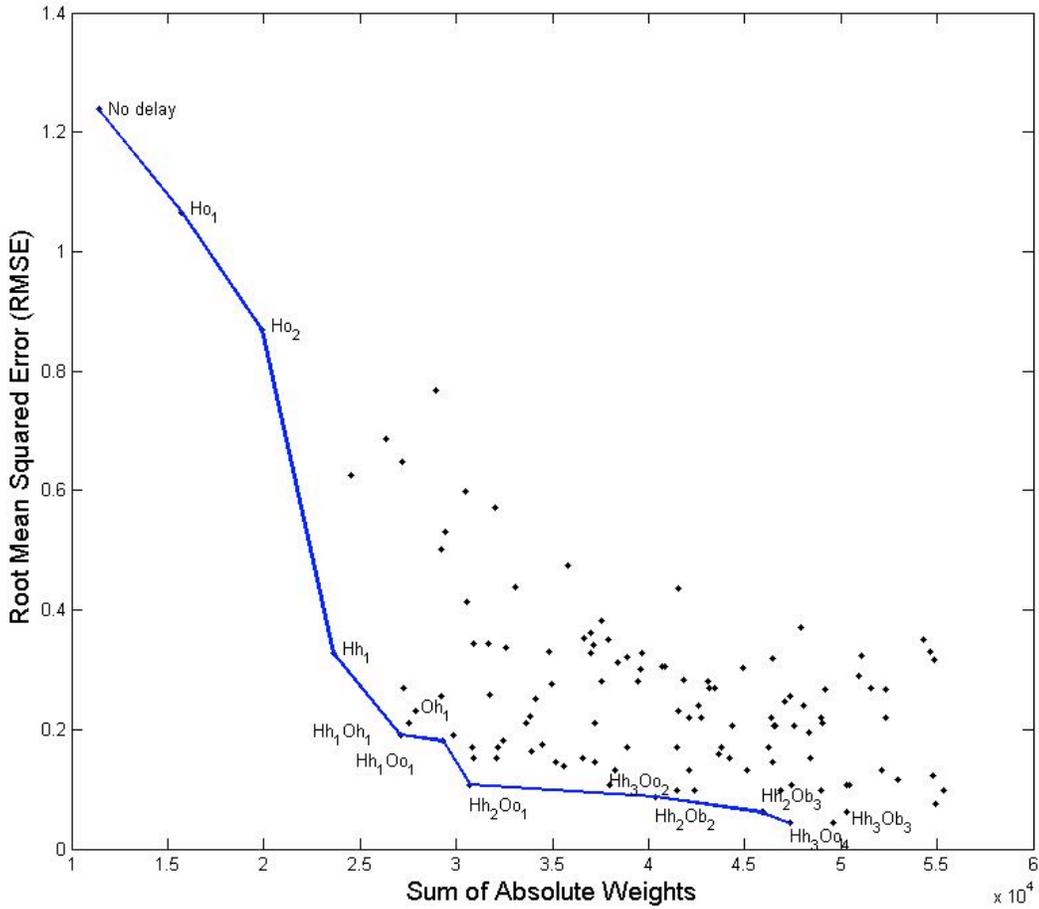


Figure 13: The performance/weights results for networks from all final generations of 100 runs of the evolutionary process are shown. Each plotted point represents one network architecture's values averaged over all evolutionary runs. Points on the solid line represent the Pareto optimal set.

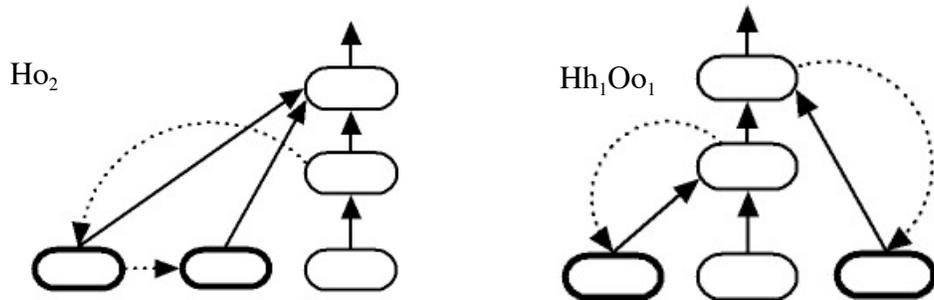


Figure 14: Examples of evolved architectures. The evolved layers are shown as bold ovals. Dotted lines are one-to-one feedback connections, solid arrows are trainable fully-connected pathways.

The results from this study demonstrate the ability of evolutionary processes based on GP to discover parsimonious but effective network architectures for specific given tasks. We conclude from this exercise that GP may have a significant role to play in designing or optimizing components of a cognitive architecture based upon neuromorphic principles

VIII. Roadmap

How should the development of a large-scale, integrated neurocognitive architecture be organized over a five year period? The detailed answer to this question depends in part on the requirements of a continued BICA program. However, a broad outline of an answer to this question can be proposed, at least tentatively, at this point. Our answer is that there should be three aspects to implementation and evaluation of such a system as illustrated in Figure 15 below. First, within the first two years a *full skeletal system* should be implemented (bottom horizontal line in Figure 15). By this we mean that all of the components needed for the full architecture should be in place (“full”), but that none of these components will necessarily be optimal. This is a somewhat non-standard approach that can be viewed as an extension and completion of the prototype “mini-Roboscout” system described earlier in this report. The philosophy underlying this approach is that the integration of the components of the core architecture needs to be achieved first as it is the critical step upon which ultimate success will depend. The full skeletal system should be able to function in Decathlon and Challenge tests, but would be significantly limited on some of the tasks by the non-optimal nature of its components.

Once this full skeletal system is functioning effectively, the second aspect of the implementation process would be the gradual replacement of the initial components of the architecture with progressively more powerful components, using the best technological solutions available in each case. The need to be able to swap in improved components like this is one of the reasons for requiring a highly modular design. As shown in Figure 15 (middle three horizontal lines in the chart), we believe that there is a natural ordering to this component enhancement that progresses from sensorimotor to cognitive to executive functions. However, based on the requirements of a continued BICA program and experiences with the skeletal system, it is likely that this sequential process would be more concurrent than illustrated here, for example with cognitive and executive components being upgraded in part earlier than indicated.

Finally, the third aspect of the implementation process (top horizontal line in Figure 15) is a concurrent *research process* that addresses fundamental research issues that will be faced in producing a full, integrated architecture. A key example from our perspective is how those operations originally implemented using symbolic methods (e.g., hierarchical partial planner at the executive level) can progressively be replaced by neurocomputational mechanisms). Another issue to be addressed is the optimization of selected components/subsystems using genetic programming methods. The results from these concurrent research efforts will directly feed into and influence the full system implementation.

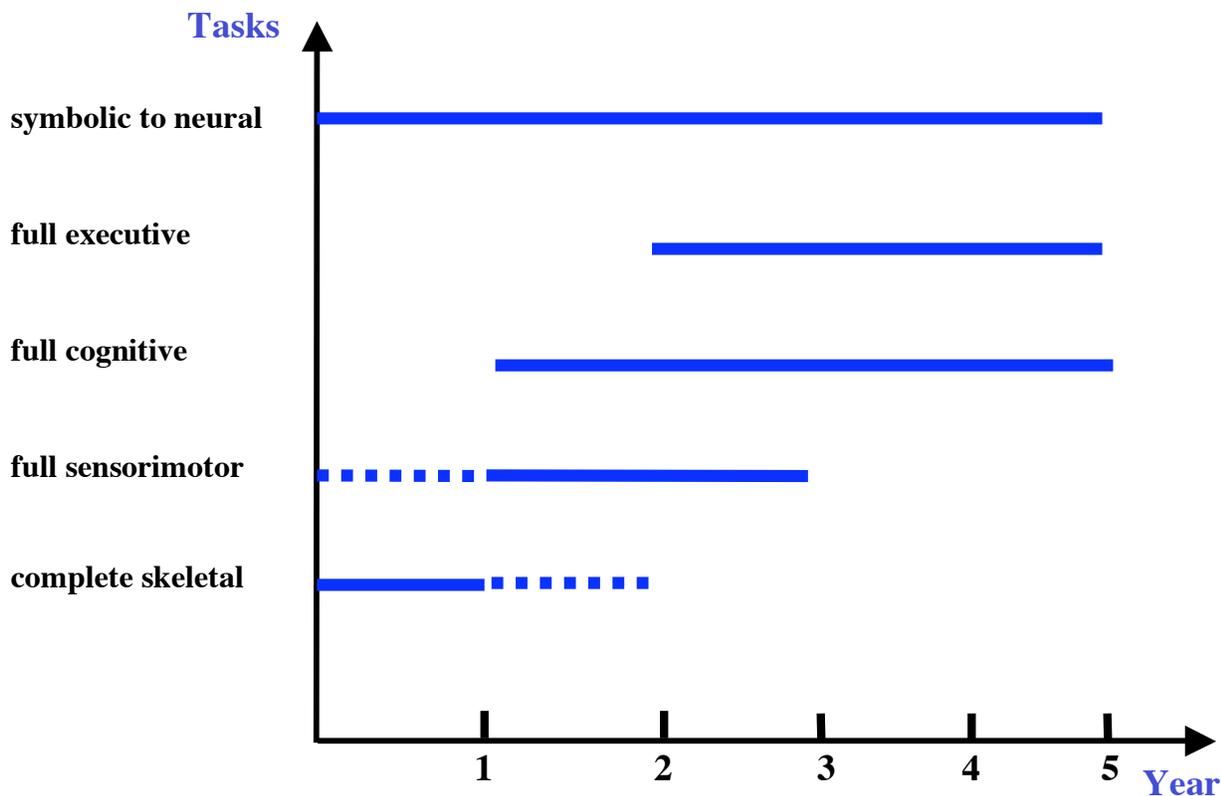


Figure 15: Anticipated schedule of steps towards the implementation and evaluation of a large-scale integrated neurocognitive architecture. The horizontal lines are not intended to be absolute, but only to indicate time periods of most focused activity.

IX. Conclusions

We have outlined both a conceptual framework (in Part 1) and a top-level design for an integrated cognitive architecture, and tested several of the ideas that are involved through some computational experiments. Our principal conclusions from this work are as follows.

1. A large-scale, integrated neurocognitive architecture is feasible. By this we mean that knowledge in neuroscience and advances in computational power make it plausible that a general purpose machine intelligence can be developed that is directly based on the hierarchical and modular organization, dynamics and plasticity of the human brain. Part 1 of this report outlined the different brain modules and functionality that need to be captured in such an architecture.
2. While such a neuromorphic architecture is the ultimate target, our currently incomplete knowledge about the neurobiological basis of cognition suggest that the optimal approach for the short term of the next few years should focus on a hybrid system that combines both neurocomputational and other “bottom-up” methods with symbolic and other “top-down” methods from AI and cognitive science. Such a hybrid approach is most likely to be reasonably successful when assessed critically by performance evaluations, and would be a natural spring-board for a long term effort over decades to produce a fully neuromorphic system.

3. Concurrently with development of a hybrid cognitive architecture for the short term, basic research efforts should be made to precisely define and to remove the remaining barriers to implementing a fully neuromorphic system.
4. The organization of the human nervous system suggests that a three-tiered system like that we have specified (in Part 2), consisting of sensorimotor, cognitive and executive levels, is a very useful approach to organizing implementation of the hybrid architecture. Other key biologically-inspired concepts include the use of developmental principles to guide the staged creation of the system and the use of a highly modular design.
5. Two non-standard computational ideas are likely to make substantial contributions to implementation of a neurocognitive architecture. First, over the short term, the use of a coarse-grained, high-performance computer cluster is probably the most cost effective approach to providing the needed computing power. Second, the use of evolutionary computation methods such as genetic programming as a design tool is likely to suggest efficient and novel neural network designs for use in the cognitive architecture. Nanotechnology and DNA computing offer additional promise for the long term.

Literature Cited in Part 2

Banzhaf W, Nordin P, Keller R & Francone F. *Genetic Programming*, Morgan Kaufmann, 1998.

Bernhard F and R. Keriven. Spiking Neurons on GPUs. Research Report 05-15, CERTIS-ENPC, Ecole Nationale des Ponts et Chaussees, 2005.

BrookGPU is available at: <http://graphics.stanford.edu/projects/brookgpu/index.html>.

DeJong K. *Evolutionary Computation*, MIT Press, 2006.

Grassman, C., Schoenauer, T., & Wolff, C. Penn Neurocomputers; Event Driven and Parallel Architectures. *ESANN '02, European Symposium on Artificial Neural Networks*, 2002, 331-336.

Gruau F, Whitley D & Pyeatt L. A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks, in *Proc. First Ann. Conf. on Genetic Programming*, 1996, 81-89.

Jan C, Bai, P, Jacob S, et al, A 65nm Ultra Low Power Logic Platform Technology using Uni-axial Strained Silicon Transistors, *International Electron Devices Meeting, Technical Digest*, Dec. 5, 2005, page 60.

Jung J. & Reggia J. Evolutionary Design of Neural Network Architectures Using a Descriptive Encoding Language, *IEEE Trans. On Evolutionary Computation*, 2006, in press.

Kitano H. Neurogenetic Learning, *Physica D*, 75, 1994, 225-238.

Lohn J & Reggia J. Automatic Discovery of Self-Replicating Structures in Cellular Automata, *IEEE Trans. On Evolutionary Comp.*, 1, 1997, 165-178.

Long, L. and Gupta, A., Scalable Massively Parallel Artificial Neural Networks, *AIAA Paper No. 2005-7168*, AIAA InfoTech@Aerospce Conference, Sept., 2005, Wash., D.C.

Mouraud, A., Puzenat, D., and Paugam-Moisy, H. DAMNED: A Distributed and Multithreaded Neural Event Driven Simulation Framework. *IASTED, PDCN 2006 International Conference on Parallel and Distributed Computing and Networks*.

Nielsen M & Chuang I. *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

Oh K. and K. Jung. GPU Implementation of Neural Networks. *Pattern Recognition*, Vol. 37, No. 6, pp. 1311-1314, 2004.

Pan Z & Reggia J. Artificial Evolution of Arbitrary Self-Replicating Structures, *Journal of Cellular Automata*, 2, 2006, 105-123.

Pethick, M., Liddle, M., Werstein, P., and Huang, Z., Parallelization of a backpropagation neural network on a cluster computer. *Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*. Marina Del Rey, California (2003) pp. 574-582

Ravi S. Prasher et al., Nano and Micro-Technology-Based Next-generation Package-level Cooling Solutions, *Intel Technology Journal*, 9 (4), Nov. 9, 2005, 285.

Raz A & Buhle J. Typologies of Attentional Networks, *Nature Rev. Neurosci*, 7, 2006, 367-379.

Reggia J, Tagamets M, Contreras-Vidal J, Weems S, Jacobs D, Winder R, & Chabuk T. Development of a Large-Scale Integrated Neurocognitive Architecture, Part 1: Conceptual Framework, CS-TR-4814/UMIACS-TR-2006-33, June 2006.

Rodriguez A. & Reggia J. Collective Movement Teams for Cooperative Problem Solving, *Integrated Computer-Aided Manufacturing*, 12, 2005, 217-235.

Seiffert, U. Artificial Neural Networks on Massively Parallel Computer Hardware. *European Symposium on Artificial Neural Networks*, pp. 319-330, 2002.

Silva G. Neuroscience Nanotechnology, *Nature Reviews Neuroscience*, 7, 2006, 65-74.

Tinnerella M, Tagamets M, Weems S, Contreras-Vidal J & Reggia J. A Behavior-to-Brain Map, CS-TR-4803/UMIACS-TR-2006-24, May 2006.

Tyagi S, Auth C, Bai P, et al, An Advanced Low Power, High Performance, Strained Channel 65nm Technology, *International Electron Devices Meeting, Tech. Digest*, Dec. 5 2005, p. 245.

Winder R & Reggia J. Using Distributed Partial Memories to Improve Self-Organizing Collective Movements, *IEEE Transactions On systems, Man and Cybernetics (B)*, 34, 2004, 1697-1707.

Yao X. Evolving Artificial Neural Networks, *Proc. IEEE*, 87, 1999, 1423-1447.

Zhu, Jihan and Sutton, Peter. FPGA Implementation of Neural Networks - A Survey of a Decade of Progress. In Cheung, Peter Y. K. and Constantinides, George A. and de Sousa, Jose T., Eds. *13th International Conference on Field-Programmable Logic and Applications (FPL 2003)*, 2003, pages 1062-1066.