A Comparison of Simple Agents Implemented in Simulated Neurons

Christian Huyck, Carl Evans and Ian Mitchell
Middlesex University - Dept. of Computer Science
London - UK
email: c.huyck, c.evans & i.mitchell@mdx.ac.uk

Abstract

Neuromorphic embodied cell assembly agents that learn are one application being developed for the Human Brain Project (HBP). The HBP is building tools, available for all researchers, for building brain simulations. Existing simulated neural Cell Assembly agents are being translated to the platforms provided by the HBP; these agents run on neuromorphic chips in addition to von Neumann based computers. Whilst translation of the agents to the software technology demanded by the HBP platforms is relatively straightforward, porting to the neuromorphic chips is a non-trivial software engineering task. Versions of the simple agent, CABot1, have been developed in fatiguing leaky integrate and fire neurons, Izhikevich neurons and leaky integrate and fire neurons. These have been developed to run in Java, PyNN, NEST and Neuromorphic hardware. All variants are roughly equivalent. The agents view a picture, implement simple commands, and respond to a context sensitive directive involving the content of the picture. By running variants of these agents on different platforms, and with the different simulated neural models, implicit assumptions in these models can be revealed. Once these Cell Assembly agents have been translated and embodied in a virtual environment, they will be extended to learn more effectively. The use of neural hardware supports the real time simulation of many more neurons, providing a platform for exploration of more complex simulated neural systems.

1. Introduction

The Cell Assembly (CA), initially proposed in 1949 (Hebb, 1949; Huyck and Passmore, 2013), is a key component in brain function linking neural behaviour to psychological behaviour. A CA is a set of neurons that are able to support firing amongst themselves; when activated, the firing CA becomes a short-term memory that persists while many of the neurons continue to fire at a rate much higher than the background base neural firing rate. Most typical concepts humans have, for instance *dog*, *mother*, or *foot*, have a CA that represents them.

CAs are central to a series of embodied agents, the Cell Assembly robots (CABots) (Huyck et al., 2011), implemented entirely in simulated neurons. CABots are interactive agents operating within a virtual environment that explore, plan and learn from sensory information. The agent can be directed by a user with natural language commands.

Three Java versions of the CABot agent were developed. The first version of the agent, CABot1, included a component for natural language processing (NLP) and could process user input for goal setting via a planning component. It also had a vision system which could detect

lines within its environment. The second version, CABot2, made some advances on the earlier version. In particular, it included a verb learning component and had improved vision which could detect edges rather than simply lines. The third version of the agent, CABot3 (see Figure 1), is composed of four major sub-systems, and a simple control system to manage some inter-system behaviour. The major systems, each consisting of hundreds of CAs, are:

- Natural Language Processing (NLP): process user input for goal setting;
- Visual: identification and location of objects;
- Plan: goal-oriented, communication and action; and
- Spatial Cognitive Map: long-term mapping of the environment.

When commanded, the CABot3 agent explores the environment, building a spatial cognitive map from information gathered via its visual field. The planning mechanism is a type of Maes network (Maes, 1989). A Maes net is a spreading activation system with goals, modules, facts, and actions. For example, facts support goals and modules that are related to them.

The virtual environment has four rooms and each room has a different object (either a pyramid or stalactite) within it. Testing the system relies on the agent using the cognitive map to find an object.

The agent is managed by sub-networks composed of simulated neurons, a technical possibility thought too complex in 1988 (Smolensky, 1988). There are some compromises made with regard to biological plausibility, but the agent is composed of, and controlled entirely by, simulated neurons.

Recently, work has begun translating the CABots from their Java based neural simulation and porting them, via PyNN (Davison et al., 2008), to neuromorphic chips (Indiveri et al., 2011). This work has commenced with CABot1, but ultimately the intention is to complete this re-engineering process for all three versions. This is one small component of the Human Brain Project (HBP) (HBP, 2014). Porting the existing CABots to the chips is a non-trivial software engineering task. Once done, the agent will be extended to learn visual objects, learn plans, and will be tested on new cognitive modelling tasks. The HBP is in its early stages, but will include a series of platforms that researchers, including those not affiliated with the project, can use for large scale neural simulations. CABots are one type of simulation.

This paper presents several initial prototypes based on several different types of neurons, on different simulators, on standard hardware and on the SpiNNaker neuromorphic chip. The paper presents a qualitative comparison of these systems. There is a system with the authors' bespoke Fatiguing Leaky Integrate and Fire neural model implemented in Java, and systems using the HBPs typical PyNN middleware. PyNN simulations specify the topology and then run that topology in a standard simulator. There is a PyNN system using Izhikevich (Izhikevich, 2004) neurons in the NEST simulator; a system using Integrate and Fire (Brette and Gerstner, 2005) neurons on NEST; a system using Integrate and Fire neurons on the SpiNNaker emulator; and a system using Integrate and Fire on SpiNNaker chips. These systems have roughly the same functionality, however there are some differences that make software engineering in some systems simpler, at least for this project. This provides evidence for a rough equivalence of these point neural models.

All of these prototype agents have both vision and planning. While vision uses almost half of the neurons in CABot3, for example, planning is much smaller; none the less, this is a solid test of the basic agent technology. Whilst the focus for this paper is the translation and porting

of the CABot1 agent prototype, the first major version of the FLIF Java agent (Huyck, 2008), the longer term aims of the Neurmorphic Embodied Agents that Learn (NEAL) project include an effort to reproduce CABot3 (Huyck et al., 2011) using PyNN. This project aims to execute the code on neuromorphic chips. This will then be extended to perform more tasks in richer environments aiming to provide further understanding of general neural cognitive architectures. This application makes use of several components of the HBP.

2. NEAL, an Application in the Human Brain Project

The authors have been awarded a grant as part of the HBP to build NEAL. The HBP is broken into 13 subprojects, and NEAL is a part of the Applications Subproject, but is also aligned with the Neuromorphic Computing Subproject.

While NEAL is part of the Applications Subproject, and to some extent the Neuromorphic Computing Subproject, it is also linked to other Subprojects. It takes advantage of the Neurorobotics, High Performance Computing, Brain Simulation and Cognitive Modelling Subprojects.

The HBP is a European Union sponsored project to further understanding of mammalian brains in general, and human brains in particular. As brains are complex and poorly understood, the project aims to build tools and platforms to support scientific exploration of them. The HBP and the American Brain Research through Advancing Innovative Neurotechnologies initiative are complementary projects that hopefully will lead to a significant advance in both understanding of brain function, and improved computer systems.

The HBP platforms include a neuromorphic computing platform, a Neuroinformatics Platform, a Brain Simulation Platform, a High Performance Computing Platform, a Medical Informatics Platform, and a Neurorobotics Platform. These computational platforms will all be developed for scientists in and outside the HBP to use. The Neuroinformatics Platform will contain neuroinformatics data such as ontologies, brain atlases and neural connectivity of both human and mouse brains. The Brain Simulation Platform will use parallelized versions of simulators such as NEST to support large scale simulations of neural systems; this extends the Blue Brain Project (Markram, 2006). While the Brain Simulation Platform focuses on software, the High Performance Computing Platform focuses more on hardware for the same purpose. The Medical Informatics Platform focuses on medical data for problems such as Alzheimer's disease.

While the authors' work will make use of these other platforms, it is most closely involved with the Neuromorphic Computing and Neurorobotics Platforms. The Neurorobotics Platform will develop virtual environments and virtual robots, that are linked to neural simulations. This is quite similar to the games environment used in the authors' earlier CABot work (Huyck et al., 2011).

The Neuromorphic Computing Platform makes use of, and provides access to, two types of neuromorphic chips: SpiNNaker (Furber et al., 2013) and HICANN (Brüderle et al., 2011). These chips are designed to simulate large numbers of neurons rapidly; they are radically different from typical von Neumann architectures (Von Neumann, 1958).

One Subproject not directly linked to a platform is the Cognitive Architectures Subproject. The goal is to select studied cognitive tasks, simulate them in an artificial neural network, and align activity to known brain activation patterns. NEAL will be informed by this Subproject.

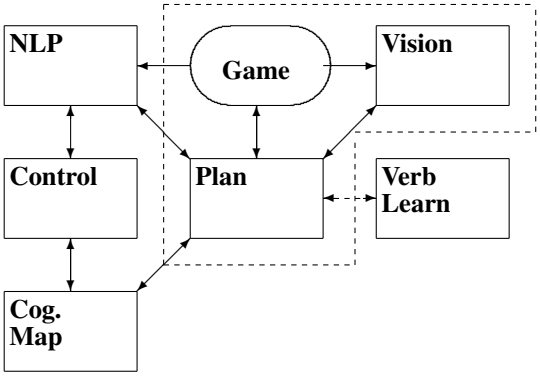

Figure 1: Gross Topology of CABot3. Boxes represent subystems of subnets. The oval represents the environment. The dashed box represents subsystems for CABot1; note that the environment is static for the simulations described in this paper. The vision system is introduced in section 4.2, and the planning system is most fully described in section 4.3.

3. CABot Translation

In earlier work, the authors and colleagues developed an agent, implemented entirely by simulated neurons, in a virtual environment (Huyck et al., 2011). The virtually embodied agent used plans, responded to natural language commands, viewed the environment, and moved around the environment. Figure 1 shows the gross structure of the sub-systems, and how they interconnect. This is sufficient to implement the full agent, though other connectivity could be added.

These CABot agents were implemented entirely in simulated fatiguing leaky integrate and fire (FLIF) neurons, which were implemented in Java, and did all of the agents' processing. The environment was a simple three dimensional virtual space with objects implemented in a shareware games engine, Crystal Space (Crystal Space, 2008). The largest agent had approximately 100,000 neurons with several million synapses, and ran approximately in real time along with the virtual 3D environment on a standard PC.

The first step in the NEAL project is to translate CABot agents from FLIF neurons simulated in Java, to a standard neural class in PyNN. This agent can then be run on the neuromorphic chips. PyNN (see Section 3.1) is a Python package for simulator-independent specification of neuronal network models.

The Java CABot3 system was broken into 36 sub-networks (Huyck et al., 2011) with the systems described in Figure 1 consisting of multiple sub-networks. The sub-networks provide some modularity, aiding in software engineering, and, in a few cases (e.g. primary and secondary visual cortices), link directly to cortical areas.

While the original Java neuron is a model of a biological neuron, albeit a point model, the network's behaviour does not readily translate to PyNN. For instance, the Java CABot models used a 10 ms step, and when a neuron fires, neural activation is propagated to synaptically adjacent neurons in the next time step. PyNN and the underlying neuromorphic chips do not allow this. So, a complete respecification of the neurons and synapses is needed.

3.1. PyNN

PyNN (Davison et al., 2008) is a type of middleware. A neural simulation is described in PyNN. The simulation is then run through PyNN and passed onto a neural simulator such as

NEST (Gewaltig and Diesmann, 2007) or NEURON (Carnevale and Hines, 2006). The neural simulator interacts with the original Python specification to manage interactions.

The HBP takes advantage of PyNN's ability to use multiple simulators for the backend with roughly the same results. While new neural models can be developed, PyNN specifies standard models and several supported neural simulators can run these. Other simulators can be developed for PyNN and, in particular, simulators for the two neuromorphic chips (SpiNNaker and HICANN) have been developed. The PyNN description can run in these and this should emulate the behaviour of the hardware. The PyNN description of the neural topology can also be directly loaded to the neuromorphic hardware. Moreover, this is not limited to neuromorphic hardware but can also be run on high performance computers. There may be differences in the three types of hardware, and indeed in the neural simulators, and these differences need to be understood.

PyNN specifies standard neural and synaptic models. These include exponential integrate and fire models (Brette and Gerstner, 2005), Izhikevich neurons (Izhikevich, 2004), and Hodgkin Huxley neurons (Hodgkin and Huxley, 1952). Several simulators support most of these standard types of neurons.

PyNN also supports current injection as this is often used in neural simulations. Current is injected into biological neurons with electrodes causing them to fire; simulations try to reproduce this behaviour. In the PyNN CABot1 agents, current injection is used to make sensory neurons fire so that the neural system can be driven by visual input from the environment. The goals in the planning system are also set by current injection. The engineering in NEAL is about translating what previously worked in the Java FLIF model to PyNN, and then on to the other neural platforms.

3.2. Translation

With respect to the Java versions of CABot, the aim of the work was to develop neural systems for vision, planning, and natural language parsing. Whilst some aspects of network behaviour do not map automatically to PyNN, as noted earlier, much of the translation from Java to PyNN is a relatively straightforward software engineering task. The original Java version of CABot3 showed that, given enough neurons, anything can be programmed; i.e., neurons are Turing complete (Byrne and Huyck, 2010). However, the Java FLIF models are one example where there is not an implicit structural mapping to PyNN and so new systems have been developed.

The vision systems, both the Java FLIF and PyNN versions, are based on known neural functioning. The retina functions as a set of on-off and off-on centre surround receptive fields (Hubel and Wiesel, 1962). Simulated neurons mimic this behaviour, and are programmed by setting their synaptic weights and connectivity (Huyck et al., 2006). A 3x3 on-centre off-surround neuron has an excitatory synapse from the centre, and inhibitory synapses from the surround. A 3x3 off-centre on-surround has excitatory synapses from the surround, and an inhibitory synapse from the centre. The 6x6 and 9x9 detectors have larger centres and larger surrounds. For the retina, all versions have the same connectivity, though synaptic weights vary.

The simulated primary visual cortex merely gets inputs from these retinal detectors. This includes a series of location specific edge and angle detectors. The edge detectors are orientation specific, and six types of detectors have been developed. There is an up-facing horizontal edge, and a down-facing horizontal edge detector; there are right and left sloping up-facing and down-facing edge detectors. Four angles are detected: a less than angle ($<$), a greater than angle ($>$), an and angle ($\wedge$) and an or angle ($\vee$). Different systems use different retinal detectors with different topologies; these have been developed through a trial and error technique.

The planning system is a neural implementation of a Maes net (Maes, 1989). In the Java CABot systems, facts, modules, goals and actions are all stored as simple CAs in separate sub-nets. The sub-nets are connected so that neural activation spreads, implementing the appropriate plans. The CAs in these sub-nets are simple oscillators. For example, the CA for the *Turn left* goal is a set of neurons that persistently fire once the goal is activated. When the goal is on in the FLIF system, half of the neurons fire in one 10 ms cycle, and these turn the other half on in the next 10 ms cycle, while the initial neurons are off. This allows the goal to persist indefinitely without the neurons accumulating fatigue. This is a poor CA from a psychological perspective, but it enables short term memory that can be explicitly turned on and off. In the PyNN versions of the CABot systems being developed, facts and actions do not persist.

There needs to be input from the environment to the system, and in turn there needs to be output from the agent to the environment. Input is in the form of a binary bitmap of the environment (the agent's camera), and user commands in the form of typed words. Output is provided by the action sub-net of the planning system. If one of the four actions (turn left, turn right, move forward, or move backward) is on, a symbolic command is passed to the game agent. Here, 'on' means that a sufficient number of neurons in the CA for that action are firing. The system translates the subsymbolic firing behaviour of these neurons to the symbolic actions taken by the agent. In the PyNN systems, each action is represented by a single neuron, and 'on' means the neuron fires.

The first working PyNN version of CABot1 (see Section 4.2) consisted of just vision and a simple version of planning. This is able to run using the NEST simulator on a PC. While NEST runs much more slowly on a PC than the original Java code, this still works effectively in simulation, because the action does not need to be real time; it is slow, but verifiably correct. This agent was rewritten with an integrate and fire neural (IF_curr_exp in PyNN) model.

This IF_curr_exp agent supported porting to the SpiNNaker emulator, and SpiNNaker chips. In this case, input is from vision. The goal is explicitly set, since it cannot be set by user language commands, because NLP remains to be implemented.

As noted earlier, a key aim is to develop a working PyNN version of CABot3. Translating the CABot3-specific components will involve language, cognitive mapping, learning verbs, and making it all work together. This work is not about solving new theoretical problems. The Java CABot3 had binding, and cognitive models for parsing, and the Java CABot2 agent had verb learning. These will be reimplemented, but it will make no conceptual breakthrough. Rather they will merely show that the basic idea is flexible across different neural models. It will, however, provide a system that can be extended and modified.

The PyNN versions of the CABot agents have similar connectivity to their Java predecessors, but the synaptic weights are different, and individual neurons behave slightly differently. The PyNN versions also have different numbers of neurons for some functions, and some functions have different numbers of sub-networks; the sub-net concept is somewhat blurred in PyNN, because it is not a predefined class in PyNN, while it is a class in the Java simulations.

4. Prototypes

There are engineering results that can be reported from the translation of the simplest agent, CABot1. The goal was to make an agent with a visual system, and simple planning. The earlier FLIF agent (see Section 4.1), significantly informed the process. An initial system was made using Izhikevich neurons and the NEST simulator (see Section 4.2). It was hoped that this would

Neural Model	FLIF	Izhikevich	Integrate and Fire
Java	Section 4.1 Pro: More Functionality Control Of Code Fatigue Con: No Reuse		
Nest		Section 4.2 Con: Inhibition causes spiking	Section 4.3 Pro: Many hardware platforms Con: Slow to simulate
SpiNNaker			Section 4.4 Pro: Fast Con: Need hardware

Table 1: The major pros and cons of the systems described in the paper.

be translated to the SpiNNaker system, but the system had problems with inhibition, and so there was no further development of the agent using the Izhikevich model. A second system was developed using integrate and fire neurons and the NEST simulator (see Section 4.3). This was then translated to the SpiNNaker emulator and chip set (see Section 4.4). Code for all the PyNN models can be found on http://www.cwa.mdx.ac.uk/NEAL/code/cabot1/cabot1.html, and the original Java model for CABot3 can be found at http://www.cwa.mdx.ac.uk/CABot/cabot3/CABot3.html.

4.1. Java FLIF CABot1

The original CABot1 was developed several years ago (Huyck, 2008). It was based on a long-standing fatiguing leaky integrate and fire neural model, originally written in C++ and later translated to Java. This system had a natural language parsing component that was not included in PyNN versions of the CABot1 agents discussed below. It had a vision system and a planning system.

While the original Java system was more sophisticated than the PyNN agents described below, the planning and vision components were roughly equivalent. The vision system had six receptive fields, and recognised lines in three orientations and four angles. It is to be noted that it recognised lines, while the agents described below, and indeed later versions of the Java CABot agents recognised edges. That is to say, the environment within which it operated had wire-frame shapes rather than solid shapes.

The planning system of the original Java CABot1 agent was also more sophisticated than its PyNN successor. Like the later systems it had the primitives (turn left, turn right, move forward and move backward), compound commands (turn left then move forward, and turn right then move forward) and the context sensitive commands (turn toward the pyramid, and turn toward the stalactite). Additionally, this agent had two context sensitive commands that could issue multiple primitive actions (move to the pyramid and stalactite). In this case it would turn toward the item, then move forward, possibly having to recentre the object as it travelled. As the later agents were open-looped systems, they could not do this. That is to say, the PyNN agents could not modify the environment, and then see the modification. This is difficult in PyNN (though see Section 4.5).

All of the CABot1 agents made use of relatively simple cell assemblies. One simple version of this is a well connected set of neurons. While this can persist indefinitely in the PyNN systems, this type of system would eventually "fatigue out" in the Java FLIF system. Fatigue in the individual neurons would accumulate, and eventually, the whole circuit would cease firing. This was solved by making simple oscillators (see Section 3.2).

The biggest advantage of the Java system was that the engineers had complete control of all of the models. If a new behaviour for the neural model was needed, it could be readily introduced. Similarly, access to all of the internal variables was simple, and particular code could be optimised to make the system more efficient.

The toothbrush adage is pertinent here.

> A modeller would rather use another modeller's toothbrush, than their model.

One reason this adage works is that a modeller is in complete control of their model, understands it implicitly and explicitly, and has spent a great deal of time gaining these advantages. Unfortunately, this means that model, and in this case code, reuse becomes very difficult.

The HBP mechanism of standardising middleware, e.g., PyNN, and supported simulators such as NEST and SpiNNaker, enables models to be run on different systems. Also, it means that modellers can focus on the front end (the model), and not have to support the later layers. There is support for importing new neural models (e.g., FLIF) and new learning models into PyNN, SpiNNaker and other simulators.

4.2. Izhikevich NEST CABot1

The first PyNN system used standard Izhikevich neurons (Izhikevich, 2004). The Izhikevich neuron is a reverberate and fire neural model. It has a small number of parameters, and these can be set to account for several types of biological neurons, for example regular spiking and bursty neurons. These run on NEST, and the SpiNNaker emulator, and should run on the SpiNNaker and HICANN chips (though the authors have not tested these). This agent can be found at http://www.cwa.mdx.ac.uk/ NEAL/code/cabot1/versJune17.tar.gz.

As the 10 ms model from the Java implementation will not work, and there are some assumptions in SpiNNaker around a 1 ms model, the PyNN simulations use a 1 ms model. That is, input is considered, and activation spreads in discrete 1 ms cycles. This in itself requires entirely new engineering. So the connections have been entirely respecified, and in many cases the number of neurons have been changed.

With regard to the vision component, all of the on-centre off-surround and off-centre on-surround detectors have been implemented. For example, the 3x3 on-centre off-surround detector has 6 different behaviours. It fires maximally when the centre input is on, and the surrounding eight neurons are off. The firing rate and onset decrease as surrounding inputs are added. With five to eight surrounding inputs on there is no firing. The 3x3, 6x6, and 9x9 detectors have all been implemented.

The primary visual area consisted of six edge detectors and four angle detectors (see Section 3.2). These were position specific, so that there was, for instance, a detector for horizontal edge on the bottom of a solid for each input pixel. Object recognition was done by connections from the primary visual features to neurons for the two objects. They were split into four visual facts, a pyramid or stalactite on the left or right.

Much of the Java FLIF version of the CABots used oscillators to compensate for fatigue. As noted earlier, these are poor CAs from a psychological perspective, because, unlike short term

memory, they persist forever. However, they are easy to work with as programming primitives. One set of, typically five, neurons would fire in one cycle, a second in the next, then the first again and so forth. Once the oscillator was turned on, it persisted until it was turned off. This has been implemented in PyNN, again with Izhikevich neurons. The first few spikes are more rapid, but the oscillator then persists indefinitely with each set, again of five, firing every 10 ms, or every 10 cycles; this is twice as fast in simulated time as the Java FLIF version, which takes two cycles.

The planning subsystem requires facts, goals, modules and output actions based on a Maes Network (Maes, 1989). The facts, goals, and modules were all sets of five neurons forming simple CAs. The action was represented by one neuron, and the action occurs when that neuron fires.

All four primitives, both compound commands, and both context sensitive commands work; the context sensitive commands work with the shapes on either the left or right. For example, each primitive has a goal CA, a module CA and an action neuron. The goal CA is stimulated by injected current and represents a command issued by the user, which will eventually be provided by the language subsystem. When the goal CA is active, it stimulates the appropriate module CA. When the module CA is active, the appropriate action neuron is activated. This occurs in a feed-forward manner via excitatory connections between each CA. When the action neuron fires, the goal is fulfilled and there is a feedback process via inhibitory connections; this turns the goal and module CAs off. By injecting current in 100 ms cycles all of the commands can be activated, and they work with the shapes provided.

When the authors started development, they did not properly understand that Izhekivich neurons are not integrate and fire neurons, but resonate and fire neurons. This means that if a neuron receives too much inhibition, it would fire. A great deal of effort was made while developing this agent to parameter fit the topology so that inhibition did not cause firing. As subsequent engineering of the later CABot3 version would make more extensive use of inhibition, it seemed wise to try another neural model.

4.3. IF NEST CABot1

Another standard model in PyNN that is supported by the NEST simulator is the IF_curr_exp model (Brette and Gerstner, 2005), a leaky integrate and fire model. This is more similar to the original FLIF model than the Izhikevich neuron that was used in the Java system, but still significantly different. Like the Izhikevich model, a 1 ms time step was used.

Also like the Izhikevich model, the synapse was appreciably more sophisticated than the synapse from the Java FLIF model. In the FLIF model, if a neuron fired, activation equal to the synaptic weight was transmitted to the post-synaptic neuron in the next cycle. In the Izhikevich neuron and the IF_curr_exp model, the current continued over several cycles. This does make more sense with a 1 ms model.

The default values for neural parameters were largely used, though the clamped input was varied for different tasks. The code can be found on http://www.cwa.mdx.ac.uk/NEAL/code/cabot1/cabot1Nov17.tar.gz. Note that this code can also be run on SpiNNaker (see Section 4.4).

All of the same commands used with the Izhikevich agent can be run with the IF_curr_exp model. Vision works slightly better, and works on all four included figures. One of the pleasing aspects of this system is that the 3x3 off centre receptive fields perfectly outline the figures.

As expected, the inhibition problem of the Izhikevich model ceased to be an issue with the IF_curr_exp model. It was possible to inhibit neurons with very large negative values without firing.

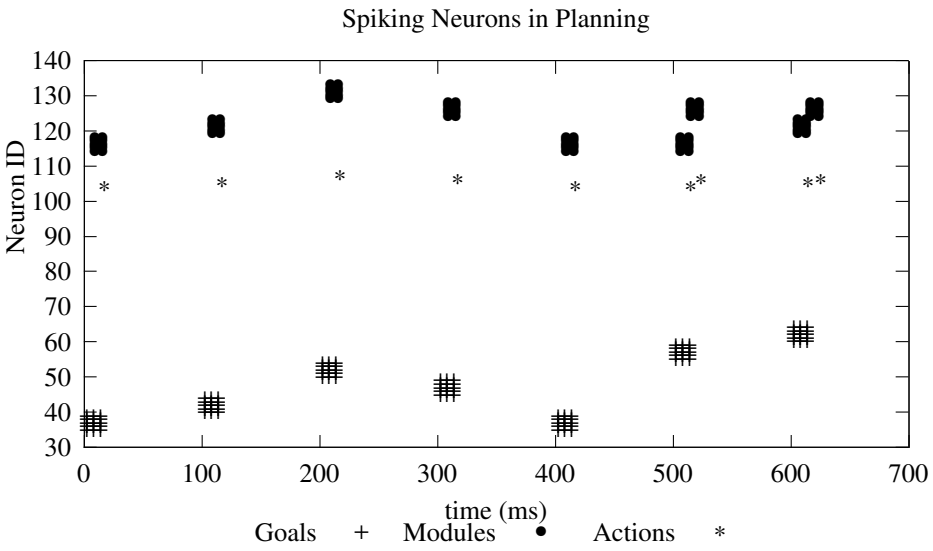

Figure 2: Planning: Maes network displaying Spiking Neurons run over 700 m/s. The commands where issued in intervals of 100 m/s. The top, middle and bottom show Spiking Neurons for the Module, Actions and Goals, respectively. Notice that Actions have one Neuron.

Figure 2 shows spiking neurons over a 700ms simulation run time. Each neuron has an identification, with 30–65 to Goals, 100–104 to Actions and 110–135 to Modules. Other neurons shown here but not spiking are allocated to other functions represented by Goals or Facts that have not been used in this simulation. The population size is 5 for Goals and Modules, and 1 for Action. Goal neurons are fired by a basic command, these then fire the module neurons, which fire the action neurons. When action neurons fire, inhibitions is sent back to connected Goals and Modules, which suppresses further activation. For the simulation the command sequence is shown in each row in Table2. As shown in the table, the spikes occur for the Goal 2.452 ms. (column 4) after the initial current injection. The time difference between Goals and Modules to spike for non-compound statements is 6.801 ms. (column 5). Finally, the time difference between simple commands being set and action is on average 17.306 ms. (column 6). When the action neuron fires, inhibitory signals are sent back to associated Goals and Modules, which suppresses neurons in the population from further spiking. This shows the sequence of Maes events via neural firing times.

4.4. IF SpiNNaker Simulator and Chip Set CABot1

Apart from some minor differences, the code from the PyNN NEST model runs on the SpiNNaker chip. The APIs do vary somewhat, so a global configuration parameter is used to switch between specific simulators in the Python files. The variable is the simulator name, and the API specific differences are accounted for in this manner.

Initially, the system was modified to work on the SpiNNaker emulator. Unfortunately, the current SpiNNaker emulator does not have the capacity for simulating many neurons; difficulties occurred around 10,000 neurons. If visual input was a 50x50 input, and the receptive fields and

1.Time	2.Command	3.Abbr.	4.Goal	5.Module	6. Action
0	Turn Right	TR	2.452	9.253	17.306 (TR)
100	Turn Left	TL	102.452	109.253	117.306 (TL)
200	Backward	BW	202.452	209.253	217.306 (BW)
300	Forward	FW	302.452	309.253	317.306 (FW)
400	Turn Right	TR	402.452	409.253	417.306 (TR)
500	Move Right	MR	502.508	506.444	515.017 (TR)
Compound Statement				515.330	523.880 (FW)
600	Move Left	ML	602.452	606.111	614.548 (TL)
Compound Statement				616.901	625.567 (FW)

Table 2: Commands and associated spike times of first neuron in population of Goals, Modules and Actions in the Maes Net. Time is in milli-seconds (ms) and rounded to 3 d.p. Results match with Figure 2, where neuron fires multiple times only the first neuron firing time is recorded, with the exception of compound statements that result in two actions, e.g. in the final row command 'Move Left' results in two actions: 'TL + FW'. R

primary visual feature detectors were this size, capacity was quickly exceeded. It was possible to work with smaller inputs (e.g. 10x10), and to work with a subset of the components.

Once a small SpiNNaker board, with four SpiNNaker chips was acquired, the system was ported. Aside from problems with large numbers of neurons, everything that worked on the emulator worked on the board. The board also provided a much larger capacity.

One of the difficulties faced was that each PyNN population required a different processor on the SpiNNaker board. Each SpiNNaker chip has 16 processors. This meant that the populations had to be arranged differently; for example, initially, there was a population for each fact; this was changed so that all the facts were in one population.

Similarly, the default number of IF_curr_exp neurons on a processor was 100. This could be modified, and in the full system, 300 were allocated to each processor, and in later systems 1000. Speed did not seem to be a problem, but it was an open-loop system.

On the chip, all of the commands were successful. Though it is not entirely transparent, it is relatively straightforward to make a system that runs in both NEST and on the SpiNNaker board.

4.5. Beyond CABot1

The second version of CABot, CABot2 has also been re-engineered using PyNN. At present, this only runs on the SpiNNaker board, but includes Natural Language Parsing and closed-loop behaviour.

Perhaps the best feature of the original Java CABot3 was its natural language parsing system. This is a cognitive model of parsing implemented in neurons (Huyck, 2009). Context free parsing requires variable binding of some sort, and in this model, binding was done by short-term potentiation. Unfortunately, the standard synaptic models in PyNN do not have an appropriate model; the weights in the Tsodykys-Markram synapses (Tsodyks et al., 1998) do not decay when the neurons are not firing.

Since context-free parsing is not a simple option, a regular parser was developed. This accounts for all of the commands that a user can issue. The results of the commands are then used to set the goals.

Another significant extension from CABot1 is that the agent developed in PyNN is a closed-loop system. That is to say, the agent exists within an environment, and when the agent moves, the environment changes. In particular, when the agent moves, the visual input changes.

This requires interaction between the virtual environment and the neural simulator. PyNN provides a mechanism for the simulator to 'call-back' a function in Python. However, this is not cleanly transferred to SpiNNaker. In SpiNNaker a spike server is needed that sends spikes to particular SpiNNaker neurons, and reads the values from other particular neurons. In both CABot2 and CABot3 (Java versions) the input neurons are the words for parsing, the visual input neurons, and a "bump" sensor for when the agent runs into something. The bump sensor is hooked to a fact in the Maes net. The output neurons are the four action neurons.

In CABot1, the environment was an image. The original FLIF CABot3 was linked to a Crystal Space environment (Crystal Space, 2008); unfortunately, this environment will not run in the Linux operating system needed for SpiNNaker. Eventually, the HBP will provide a virtual environment, but this is not ready. Consequently, a Python virtual environment has been developed. For CABot2 it merely has an object.

Additionally, a controller subsystem is needed. This enables the system to parse, then set the goal, then return to parsing when the goal has been satisfied. Parsing restarts after the goal has been completed, and the new command from the user is available.

CABot3 requires a more sophisticated environment, extra visual features, an improved planning system, and simple cognitive mapping. The most complex command in the original Java CABot3 was "Explore." The environment consisted of four rooms connected by corridors. Each room had a unique object, a striped or barred pyramid or stalactite. The agent had to go through all four rooms, and remember the object in each room. The user could then request the agent to go to (for example) the room before the striped pyramid. This showed that the simple cognitive map had been learned.

This required an extra visual feature. In the original agent this was a pattern on the object, either barred or striped. In the new PyNN version it is anticipated that this will be colour; it is easier. The agent also needs to find the corridor to the next room and move through it. This requires recognition of a third shape (a corridor), and will make use of bumps to get through the corridor.

This also requires a more complex planning mechanism. The agent needs to execute repeated actions that are context sensitive. In the original CABot3, the agent had to execute hundreds of actions in response to an explore command.

Finally, a learning mechanisms is required for cognitive map building. Standard PyNN long-term potentiation rules should be sufficient.

Once a PyNN version of the CABot3 agent is completed, it will be extended. One of the tasks the Java agents completed was to learn verb associations; this was done using reinforcement learning (Belavkin and Huyck, 2010), though this was implemented in neurons using Hebbian learning rules to modify synaptic weights.

Binding in parsing also needs to be implemented. PyNN, NEST and SpiNNaker all enable specification of neural models and synapse models. The short-term potentiation rule will need to be reimplemented in PyNN and SpiNNaker. Once this is done, a psychologically plausible parser can be reimplemented on the chip.

Finally, an agent needs to work in HICANN in addition to SpiNNaker. Hopefully at this stage, the HICANN simulator and hardware will function seamlessly with the PyNN code, but it may not because HICANN may have made different assumptions.

This engineering and proposed engineering is interesting because different assumptions are implicit to each underlying simulator, and indeed to the initial Java CABots. The engineering reveals these assumptions, leading to a better understanding of large scale neurodynamics, and a better understanding of how to program these neural systems.

5. Comparison

With regard to the various versions of the agents described above, there are significant differences. Some key differences relate to the translation of FLIF neurons written in Java to new systems written in Python using PyNN, the various neural models employed and the various underlying simulators on which they were tested. In addition, there are also differences between the PyNN systems themselves.

The main advantage of the Java FLIF version was largely in the control of the underlying system. The neural model could be modified, the learning rules could be modified, and the correlation between real time and simulated time could be modified. The neural model was a fatiguing leaky integrate and fire model, but there were a range of mechanisms that could be used for fatigue. In the original system this mechanism was stable, but later the fatigue model was changed so that unstimulated neurons hypo-fatigued and fired leading to spontaneous firing.

None of the CABot1 systems learned in the sense that the system could not return to its initial state. However, short-term potentiation was used in the Java CABot1 for variable binding in parsing. This mechanism was not available in the PyNN simulations. Control of the rule gave a great deal of flexibility for binding.

In the Java model, control of timing was very important. This model assumed a 10 ms cycle, while the PyNN model did not function well with this time step. Firstly, this enabled the model to ignore behaviours such as refractory periods and synaptic delay. Secondly, the updates were performed less frequently. Accordingly, the system was able to simulate more neurons in real time. While the PyNN system can run with 10 ms time cycles, it still accounts for those behaviours that are ignored by the Java system. Additionally, the SpiNNaker hardware actually assumes a 1 ms model, and so has difficulty at these slower speeds.

Another advantage of the FLIF model was fatigue itself. This supported CAs being triggered on their own. The down-side to this was that oscillators were needed when CAs needed to persist indefinitely.

A key advantage of using the PyNN system is that it can be run on a variety of platforms. This eventually should enable others to reuse this code in similar systems.

The differences between the PyNN systems were relatively minor. The major difference between the Izhikevich model and the IF_curr_exp model was inhibition. The Izhikevich model, being a reverberate and fire model, fired rapidly when heavily inhibited.

The differences between the NEST and SpiNNaker model are quite minor. There is a difference in calling the API. Another difference is that the SpiNNaker system has a limit on the number of neurons simulated. This is somewhat flexible, but there is an absolute limit of 2048 neurons per processor. The NEST system does not have this limit, and merely runs more slowly with more neurons.

The overarching point of this comparison is that, despite some differences, all of the systems are, in fact, quite similar. That is to say, despite differences in language (Java vs. Python), middleware (no middleware vs. PyNN), neural model (FLIF vs. Izhekivich vs. LIF), and simulator (Java vs. NEST vs. SpiNNaker), the systems have very similar functionality. Moreover, though not shown here, the systems can be combined as all communicate via spikes.

A note about SpiNNaker systems. This paper describes the use of a 64 processor SpiNNaker board, but a 100,000 processor system exists, and a 1,000,000 processor system is planned. This would give a modeller acess to real time simuation of a billion spiking neurons.

6. Future work

While the engineering work of the CABot translation is useful and interesting, the second phase of the project will lead to advancements in the theory of computing with neurons. The CABot project has shown that neural systems are Turing complete, but the real benefit of neural systems over standard symbolic computation is that they can learn. The CABot agents have some simple learning, but running agents on neuromorphic chips will support larger numbers of neurons behaving in real time. These will be able to learn in a psychologically and neurally plausible manner. The system will learn a range of visual categories, and learn them as more psychologically plausible CAs. The system will also learn new plans.

Since developing the Java CABots, the authors have focused on learning. By modifying the FLIF model so that neurons spontaneously fire when hypo-fatigued, a system that learns categories has been developed (Huyck and Mitchell, 2014). Biological neurons spike without input, and hypo-fatigue is one mechanism that could drive this behaviour. By integrating this with earlier visual category learning (Jamshed and Huyck, 2010), making use of more neurons, and further exploring dynamics, a wide range of visual categories from the environment can be learned.

To take advantage of this work, the FLIF models will be translated to PyNN, SpiNNaker and perhaps HICANN and NEST. This will support the use of these old systems, but also the use of explored system wide learning mechanisms.

One flaw with existing CA simulations, is that the CAs do not persist for psychologically realistic durations. Ignited CAs are CAs with enough neurons firing to support persistent firing throughout the CA. They are the neural implementation of psychological short-term memories, and the duration of the persistence depends on many factors. For example, they persist longer when more activated, and they can be reactivated. Some unpublished work with small-world topologies has led to more realistic persistence, and this may be a solution or part of the solution for more psychologically realistic CAs. Another avenue of exploration is to use short term potentiation to change persistence.

A first step at learning new plans will be to improve existing plans and plan elements via reinforcement learning. This is similar to the verb learning done in the Java CABot2 that makes use of reinforcement learning. A more significant advance may be made by using improved CAs to learn new plans. Here instead of oscillators, more sophisticated CAs will be used giving the system the ability to use hierarchical CAs. This will support the development of new planning CAs, which can then be combined to make new plans.

New cognitive models will also be developed. A simple model for learning categories based on the experiments by Shepard et al. (1961) will link neural and cognitive category learning. The system will also be given a Wisconsin Card Sorting Task (Monchi et al., 2001), and should perform this task reproducing human results. This will work through the virtual environment so the system will have to actually view the cards.

The Neurorobotics Platform simulator should replace the Python virtual environment. The sooner this happens the better, as it should be better supported by the Neuromorphic Platforms. Finally, it is hoped that the system will be ported to the High Performance Computing Platform. Hopefully, the PyNN implementation and Neurorobotics Platform will make this straightforward.

7. Conclusion

The main aim of NEAL is to build an agent, embodied in a virtual environment, based solely on simulated neurons running on neuromorphic chips. This will make use of many resources provided by the Human Brain Project, whose main goal is to build tools for building brain simulations.

The first phase of this project involves porting the existing CABot agents from a Java based neural simulation to PyNN, and then onto neuromorphic chips. This will require re-engineering of the neural network, but should be relatively straightforward. Special care is needed for interactions between the simulated agent and the virtual environment. Good initial progress has been made toward this goal.

The second phase of the project will involve extending the agent, so that it can better learn CAs. These CAs will be more psychologically plausible and will make the agent more flexible. CAs will be learned for visual categories so that a CA will persistently fire when an item of a particular category is presented. New plans will be learned. New neuro-cognitive models will be developed.

This is another step along a path using a unified approach to develop embodied agents. All computation will be done in neurons; communication between neurons will be via spikes and different neural subsystems will communicate by this mechanism. The system will constantly learn. As part of the HBP it will become a staging post for other work. Other researchers can use this system as a starting point. Existing subsystems can be modified and new subsystems added. These can then be used to perform multiple tasks, including cognitive models, and embodied agents that learn their environment and communicate with a human via natural language.

Acknowledgements:
This work was supported by the Human Brain Project Grant 604102, Neuromorphic Embodied Agents that Learn.

References

Belavkin, R., Huyck, C., 2010. Conflict resolution and learning probability matching in a neural cell-assembly architecture. Cognitive Systems Research 12, 93–101.

Brette, R., Gerstner, W., 2005. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. J. Neurophysiol. 94, 3637–3642.

Brüderle, D., Petrovici, M., Vogginger, B., Ehrlich, M., Pfeil, T., Millner, S., Grübl, A., Wendt, K., Müller, E., Schwartz, M., et al., 2011. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. Biological cybernetics 104 (4-5), 263–296.

Byrne, E., Huyck, C., 2010. Processing with cell assemblies. Neurocomputing 74, 76–83.

Carnevale, N., Hines, M., 2006. The NEURON Book. Cambridge University Press.

Crystal Space, 2008. *http : //www.crystalspace3d.org/main/main_page*.

Davison, A., Brüderle, D., Eppler, J., Muller, E., Pecevski, D., Perrinet, L., Yqer, P., 2008. PyNN: a common interface for neuronal network simulators. Frontiers in neuroinformatics 2.

Furber, S., Lester, D., Plana, L., Garside, J., Painkras, E., Temple, S., Brown, A., 2013. Overview of the spinnaker system architecture. IEEE Transactions on Computers 62 (12), 2454–2467.

Gewaltig, M., Diesmann, M., 2007. Nest (neural simulation tool). Scholarpedia 2 (4), 1430.

HBP, 2015. The human brain project.
URL `https://www.humanbrainproject.eu/`

Hebb, D., 1949. The Organization of Behavior. John Wiley and Sons.

Hodgkin, A., Huxley, A., 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. Journal of Physiology 117, 500–544.

Hubel, D., Wiesel, T., 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. Journal of Physiology 160, 106–154.

Huyck, C., 2008. Cabot1: a videogame agent implemented in fLIF neurons. In: Proceedings of 2008 7th IEEE international Conference on Cybernetic Intelligent Systems. pp. 115–120.

Huyck, C., 2009. A psycholinguistic model of natural language parsing implemented in simulated neurons. Cognitive Neurodynamics 3(4), 316–330.

Huyck, C., Belavkin, R., Jamshed, F., Nadh, K., Passmore, P., Byrne, E., D.Diaper, 2011. CABot3: A simulated neural games agent. In: 7th Intl Workshop on Neural-Symbolic Learning and Reasoning, NeSYS'11. pp. 500–544.

Huyck, C., Diaper, D., Belavkin, R., Kenny, I., 2006. Vision in an agent based on fatiguing leaky integrate and fire neurons. In: Proceedings of the Fifth International Conference on Cybernetic Intelligent Systems. pp. 131–136.

Huyck, C., Mitchell, I., 2014. Post and pre-compensatory Hebbian learning for categorisation. Computational Neurodynamics 8:4, 299–311.

Huyck, C., Passmore, P., 2013. A review of cell assemblies. Biological Cybernetics 107:3, 263–288.

Indiveri, G., Linares-Barranco, B., Hamilton, T., Schaik, A. V., Etienne-Cummings, R., Delbruck, T., Liu, S., Dudek, P., Häfliger, P., Renaud, S., et al., 2011. Neuromorphic silicon neuron circuits. Frontiers in neuroscience 5.

Izhikevich, E., 2004. Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks 15:5, 1063–1070.

Jamshed, F., Huyck, C., 2010. Grounding symbols: Learning 2-d shapes using cell assemblies that emerge from fLIF neurons. In: Proceedings of the AISB.

Maes, P., 1989. How to do the right thing. Connection Science 1:3, 291–323.

Markram, H., 2006. The blue brain project. Nature Reviews Neuroscience 7, 153–160.

Monchi, O., Petrides, M., Petre, V., Worsley, K., Dagher, A., 2001. Wisconsin card sorting revisited: Distinct neural circuits participating in different stages of the task identified by event-related functional magnetic resonance imaging. Journal of Neuroscience 21(19), 7733–7741.

Shepard, R., Hovland, C., Jenkins, H., 1961. Learninng and memorization of classifications. Psychological Monographs 75:517, 1–42.

Smolensky, P., 1988. On the proper treatment of connectionism. Behavioral and Brain Sciences 11:1, 1–22.

Tsodyks, M., Pawelzik, K., Markram, H., 1998. Neural networks with dynamic synapses. Neural Computation 110, 821–35.

Von Neumann, J., 1958. The computer and the brain. Yale University Press.