Procedia Computer Science

Volume 99, 2014, Pages 430–446

BICA 2014. 5th Annual International Conference on Biologically Inspired Cognitive Architectures

A Neuroscience Inspired Gated Learning Action Selection Mechanism

Klaus Raizer[1] and Ricardo R. Gudwin[1]

University of Campinas
School of Electrical and Computer Engineering
Campinas, SP, Brazil
[klaus][gudwin]@dca.fee.unicamp.br

Abstract

This paper presents an algorithm for action selection, in the context of intelligent agents, capable of learning from rewards which are sparse in time. Inspiration for the proposed algorithm was drawn from computational neuroscience models of how the human prefrontal cortex (PFC) works. We have observed that this abstraction provides some advantages, such as the representation of solutions as trees, making it human-readable, and turning the learning process into a combinatorial optimization problem. Results for it solving the 1-2-AX working memory task are presented and discussed. We also argue the pros and cons of the proposed algorithm and, finally, address potential future work.

Keywords: action selection, neural networks, machine learning, reinforcement learning, genetic algorithm

1 Introduction

This paper presents an algorithm for action selection in the context of intelligent agents capable of learning from sparse in time rewards. Inspiration for the proposed algorithm was drawn from computational neuroscience models of how the human prefrontal cortex (PFC) works.

The mathematical and algorithmic study of how the human conscious mind solves the problem of selecting the next action to be taken has produced many interesting results (Reggia, 2013; Baars & Franklin, 2009). By controlling and managing other cognitive processes "executive functions" are those responsible for what is usually considered to be "intelligent behavior". They are a "macroconstruct" (Alvarez & Emory, 2006), in the sense that multiple sub-processes must work in conjunction to solve complex problems. The term "executive function" is therefore used as an umbrella for a wide range of cognitive processes and sub-processes (Chan, Shum, Toulopoulou, & Chen, 2008), with the most prominent being *action selection*, *planning*, *selective attention* and *learning* (Baars & Gage, 2010; Fuster, 2008; Frank & Badre, 2012).

 Selection and peer-review under responsibility of the Scientific Programme Committee of BICA 2014

In their nature, executive functions are mostly future directed and goal oriented, whilst exerting supervisory control over all voluntary activities. They deal with prospective actions and deliberate plans to achieve goals which can be defined by the executive itself. In a sense, this is what the frontal lobe, in particular the prefrontal cortex, does for humans (Baars & Gage, 2010; Chersi, Ferrari, & Fogassi, 2011).

In this work we propose an action selection algorithm inspired by the computational neuroscience model described in the Leabra framework ((O'Reilly, Munakata, Frank, Hazy, & Contributors, 2012), (Hazy, Frank, & O'Reilly, 2007)). With its PBWM (Prefrontal Cortex Basal Ganglia Working Memory) algorithm, Leabra models how the human PFC interacts with basal ganglia in order to learn from rewards separated in time and select the most appropriate action given a particular stimulus. In other words it performs, with the exception of *planning*, all major executive functions.

The PBWM mechanism strives to follow the biological cognitive process as closely as possible. The present work, however, focuses more on the development of a new algorithm, which is also biologically inspired but ultimately designed to be used in the development of intelligent agents. In order to do so, the inner workings of the PBWM mechanism (Hazy et al., 2007) were abstracted in the form of an action selection algorithm, whose behavior towards new stimuli is defined by an optimized tree structure. We have observed that this abstraction provided a number of advantages, such as:

- Representing solutions as trees, instead of neural networks, allows one to see the knowledge it encodes in a direct manner.
- This method turned the learning process into a combinatorial optimization problem. This potentially makes the use of different optimization techniques straightforward.

Details on how we achieved this can be seen in Section 3. The remainder of this paper is organized as follows. Section 2 presents our initial motivations for developing this algorithm. Section 3.1 provides a brief description of how the original PBWM mechanism works, while describing the "1-2-AX" working memory[1] task used as a benchmark for validation. Section 3.2 then describes our proposed gated-learning action selection (GLAS) mechanism. In Section 4 we apply GLAS to learn the 1-2-AX working memory task and present training results given a particular sequence of events. The paper closes with Section 6, where we discuss obtained results. We also argue the pros and cons of the proposed algorithm and, finally, point to potential future work.

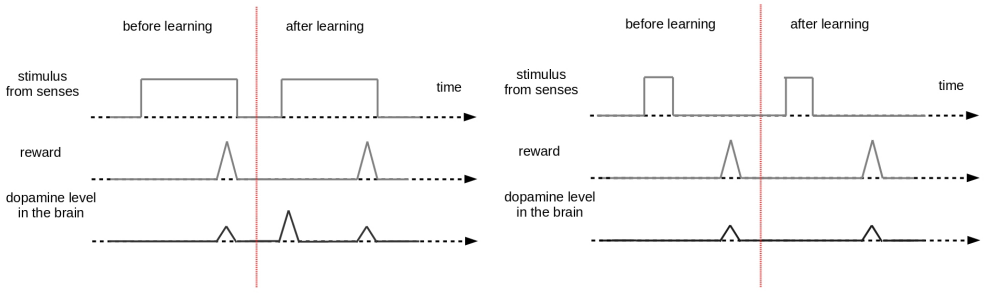
2 Motivation

The core motivation for developing this algorithm is to take advantage of what neuroscience, and more specifically computational neuroscience, has produced that could be useful for the development of artificial intelligent agents.

Specifically, adapting the PBWM mechanism to provide human-readable solutions was motivated by our previous work with behavior networks[2] and action selection (Raizer, Paraense, & Gudwin, 2012; Raizer, Rohmer, Paraense, & Gudwin, 2013).

[1]Working memory (WM) is a term, coined by behavioral neuroscience, which describes the cognitive capacity for storing and manipulating novel information for a short period of time (Baars & Gage, 2010). It was initially proposed by Allan Baddeley (Baddeley & Hitch, 1974), and it is believed that the PFC plays a critical role in active maintenance of WM information (Fuster, 2008).

[2]A behavior network is an action selection mechanism initially developed by Pattie Maes (Maes, 1989), which is capable of selecting the most relevant action at the present time, while at the same time deliberating



(a) Dopamine firing propagates backwards towards continuous stimulus

(b) Dopamine firing can not propagate back due to gap between stimulus and reward

Figure 1: Dopamine firing due to reward, given a particular stimulus. Adapted from (Hazy et al., 2007) and (O'Reilly et al., 2007).

As a matter of fact, not only should an agent be able to select the most relevant action at a given time, but it should do so while taking into consideration its future consequences. Traditional reinforcement learning mechanisms, such as variations of SARSA and Q-Learning (Russell & Norvig, 2003), have often been used to solve challenging problems in engineering and computer science (Stone, Sutton, & Kuhlmann, 2005; Mahadevan & Connell, 1992; Riedmiller, Gabel, Hafner, & Lange, 2009; Bagnell & Schneider, 2001; Zico Kolter & Ng, 2011). They lack, however, an ability the mammalian brain excels at: to bridge the gap between actions and late rewards (Bakker, Zhumatiy, Gruener, & Schmidhuber, 2003).

Let us take for instance the task of teaching a dog that taking a bath is a rewarding experience.

In Figures 1 and 2, "stimulus from senses" represents the dog's perception of having a bath. Figure 1a represents the use of a synchronous reward (reward is given while the dog is still perceiving the stimulus) and Figure 1b represents the use of a late reward. We see therefore 4 bath episodes being represented here, and learning is represented by the dotted vertical line. If every time during bath its owner gives the dog a cookie (reward), a burst of dopamine neural firing happens in the dogs' brain. Since the stimulus of taking a bath is still active, the brain manages to correlate this reward with the beginning of the stimulus, linking the perception of taking a bath to being something good.

However, if the owner waits to give its dog a cookie long after each bath is finished, something like what is described in Figure 1b could happen. In this case, there is nothing linking the moment of reward to the appearance of the stimulus.

Dogs, however, are mammals with highly developed prefrontal cortexes. The PFC works, among other things, as a temporary container for storing stimuli representations. Therefore, what really should happen in the previous case, is something like what we see in Figure 2.

In this case, the PFC stimulus representation was held long enough for the brain to make the association. In other words, the represented stimulus, stored in PFC, acted as if it were the perceived stimulus coming from the dog's senses.

Traditional artificial neural networks, such as MLPs (multi layer perceptrons) and recurrent neural networks, are known to be bad at establishing a link between longer time lapses, since

the consequences of those actions. Deliberation is made possible because each behavior has a list of preconditions, which hold propositions necessary for the behavior to become relevant, and lists of consequences. These lists contain human-readable propositions about the world state. For more details we refer to Maes' original paper.

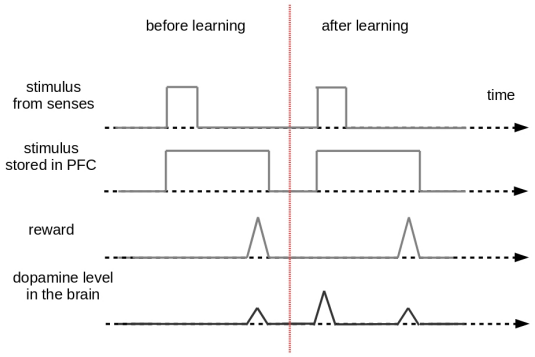


Figure 2: The PFC stores a temporary representation of stimuli. Bridging the gap between reward and stimulus.

backpropagated errors tend to either explode or exponentially decay during training (Pérez-Ortiz, Gers, Eck, & Schmidhuber, 2003).

Computational models successful at solving this kind of problem are usually those based on gating mechanisms. A gating mechanism is responsible for holding stimuli in what is called a "short-term memory". The information held in memory can then affect both action selection at a given instant and the learning process alike. Examples of algorithms using this sort of gating mechanism are the Long-Short Term Memory recurrent neural network (Bakker, 2002), and the aforementioned PBWM mechanism.

The choice of PBWM mechanism as inspiration for GLAS was motivated by the possibility of abstracting its core functionality into an algorithm able to perform a gated action selection learning, while at the same time keeping this learned knowledge in a human-readable form.

3 Methods

Before getting into the algorithms, we must first define a simplified problem to serve as a benchmark.

In order to study working memory in a controlled environment, a number of tasks were devised to demonstrate the demands of our brain's executive system. Examples of such tasks are the AX-CPT (AX continuous performance task, widely studied in humans) and a more complex variant called 1-2-AX task (Hazy et al., 2007).

In the 1-2-AX task, a person sits down on a chair with two buttons, one to the left and another to the right. A sequence of visual stimuli is shown to this person who, in turn, must either press the right button (R) or the left one (L). Possible stimuli are: 1, 2, A, B, X and Y.

At each choice of action, pressing the correct button (according to a hidden set of rules) produces a positive reward, whilst pressing the incorrect button produces a negative reward. The challenge is to find out which button to press, given not only the current stimulus, but also a recent history of stimuli.

By the end of the experiment, if the subject's memory is working correctly, he should be able to learn the following rules about the 1-2-AX task:

- If the last number I saw was a 1, I must press 'R' when I see an 'X' after an 'A'
- If the last number I saw was a 2, I must press 'R' when I see a 'Y' after a 'B'

- In all other cases, I should press 'L'

An example of applying those rules to a given sequence of stimuli can be seen in Table 1. Notice how the subject should ignore 'Y' at step $i = 5$ by pressing 'L' instead of 'R', and correctly presses 'R' at step $i = 7$.

Table 1: Short example stimuli with correct action choices for the 1-2-AX working memory task.

$\mathbf{i}$	$\mathbf{s_i}$	$\mathbf{a_i}$
0	1	L
1	B	L
2	A	L
3	X	R
4	2	L
5	Y	L
6	B	L
7	Y	R

Performing this task, even when knowing the rules, is not trivial. The subject must keep in mind at all time which number was last seen, while at the same time paying attention to the inner task sequences (AX or BY). Learning those rules in the first place is even harder[3].

We are going to use this 1-2-AX task to help us briefly explain the PBWM mechanism, and then to show how GLAS works while solving the same problem.

3.1 The PBWM Mechanism

As the name suggests, the PBWM mechanism is a model for how working memory operates given the interactions between our prefrontal cortex and basal ganglia.

The basal ganglia are a set of older brain structures known to be responsible, among other things, for action selection and reward based learning (Chakravarthy, Joseph, & Bapi, 2010; Stewart, Choo, & Eliasmith, 2010).

As we saw in Figure 2, the PFC is considered to be responsible for storing a temporary representation of relevant stimuli for action selection. However, how does it know what is relevant and what is not?

According to PBWM, the basal ganglia is the structure responsible for deciding what should be gated into the PFC. It has, therefore, two important roles; deciding what should be stored in the PFC, and which action to select given a certain scenario. In this context, both functions should be learned from experience.

Figure 3 presents a snapshot of Leabra's PBWM neural network structure while solving the 1-2-AX working memory task.

As can be seen in this picture, the PBWM mechanism is based on a group of interconnected neural networks, which play the roles of input, output, hidden neural layers, PFC representations and Basal Ganglia structure. The mechanism behind Leabra's neural network is quite

[3]Parallels to this kind of problem can be found in many real world engineering problems. For instance, in assistive technology, a robotic wheel chair system might need to learn that the patient chooses to go to the kitchen only on Tuesdays and Thursdays, and only if he has a visit, or there is a particular program on TV, respectively. With this information, the wheelchair system could make automated suggestions to the user in order to help him around the house (Raizer et al., 2013). Other potential applications are the development of artificial intelligent agents for video games, traffic control and mobile robotics in general.

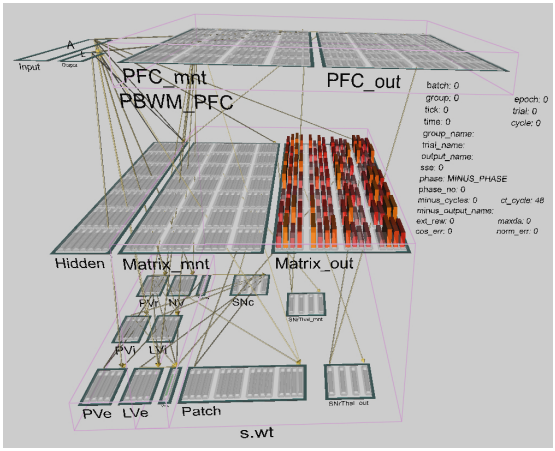


Figure 3: Snapshot of Leabra's PBWM neural network mechanism and connections for the 1-2-AX working memory task (O'Reilly et al., 2012).

complex, and explaining its inner workings is out of the scope of this work. For a complete description of this mechanism, please refer to (Hazy et al., 2007). Figure 4 shows a simplified version (diagrams were simplified for clarity) of how the PBWM mechanism would work with the sequence of events seen in Table 1.

Square boxes represent PFC stripes (Hazy et al., 2007). Each stripe is a section of PFC, responsible for holding a single stimulus representation. The diagram depicts a unit with three stripes, but in theory they could be formed by a much longer chain of stripes. Each stripe has its own "basal ganglia component", represented here by circles. At any given time, a stimulus (the hexagon shape) is presented to the whole unit, and an action (downwards arrow) is produced.

A clear circle means that whatever comes from stimuli can fill, and replace, its neighbour's stripe content. A dark circle means its neighbour stripe is closed. Being closed means two things. First, if empty it will not allow new stimuli to enter it and will, therefore, remain empty. Second, if it is already holding a stimulus it will keep it in, preventing new stimuli from altering its contents.

As can be seen in Figure 4, the following would happen assuming the basal ganglia has already learned what to do:

a) No stimulus available. b_0 keeps s_0 open. s_0 is empty.

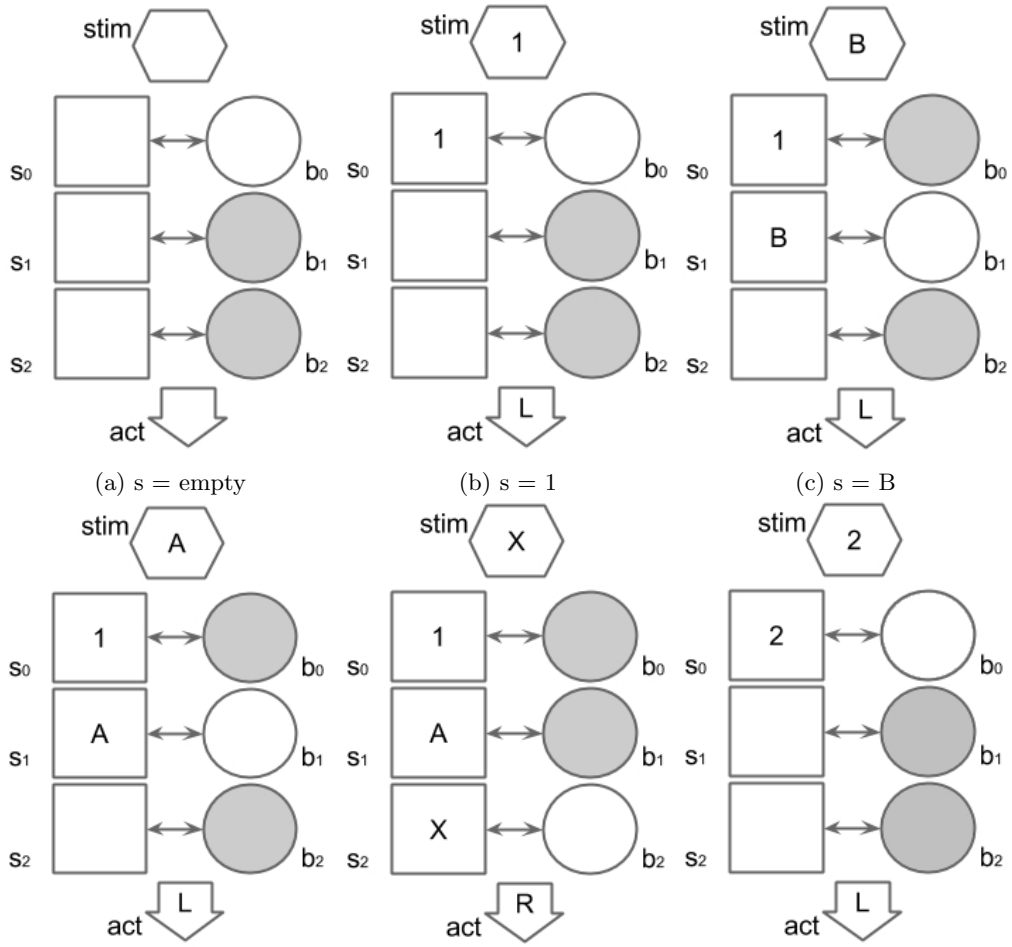
b) Stimulus "1" available. b_0 allows stimulus to get into s_0.

c) b_0 closes gate for s_0. b_1 opens gate for s_1 and allows "B" to get in.

d) b_1 keeps s_1's gate open and stimulus "A" gets in.

e) b_1 closes gate for s_1, keeping "A" inside s_1. b_2 opens gate for s_2, and allows "X" stimulus to get in.

f) b_0 detects a "2" stimulus. It opens s_1's gate for it to get in and, by doing so, resets the unit.



(a) s = empty (b) s = 1 (c) s = B

(d) s = A (e) s = X (f) s = 2

Figure 4: Stimuli gating and action selection for the PBWM mechanism. Simplified from (Hazy et al., 2007)

See that the first stripe holds a special type of stimuli (for the 1-2-AX task the "1" and "2" stimuli), which are capable of resetting the unit. These are called "task stimuli" (or task sequence starters), because they identify the starting stimulus for a particular task sequence.

Up to this point we have assumed that the BG already knows what to do: what to gate in, what to gate out and which actions to select. In order to learn what to do, the PBWM model uses a reinforcement learning mechanism (a variation of temporal differences learning called PVLV - primary value and learned value (O'Reilly et al., 2007)). The net effect is that the algorithm learns working memory tasks based only on experience.

3.2 GLAS - A Gated-Learning Action Selection Mechanism

Knowledge, in the PBWM model, is stored in a group of interconnected neural networks. Because information about task sequences and gating rules are encoded in the BG's neural

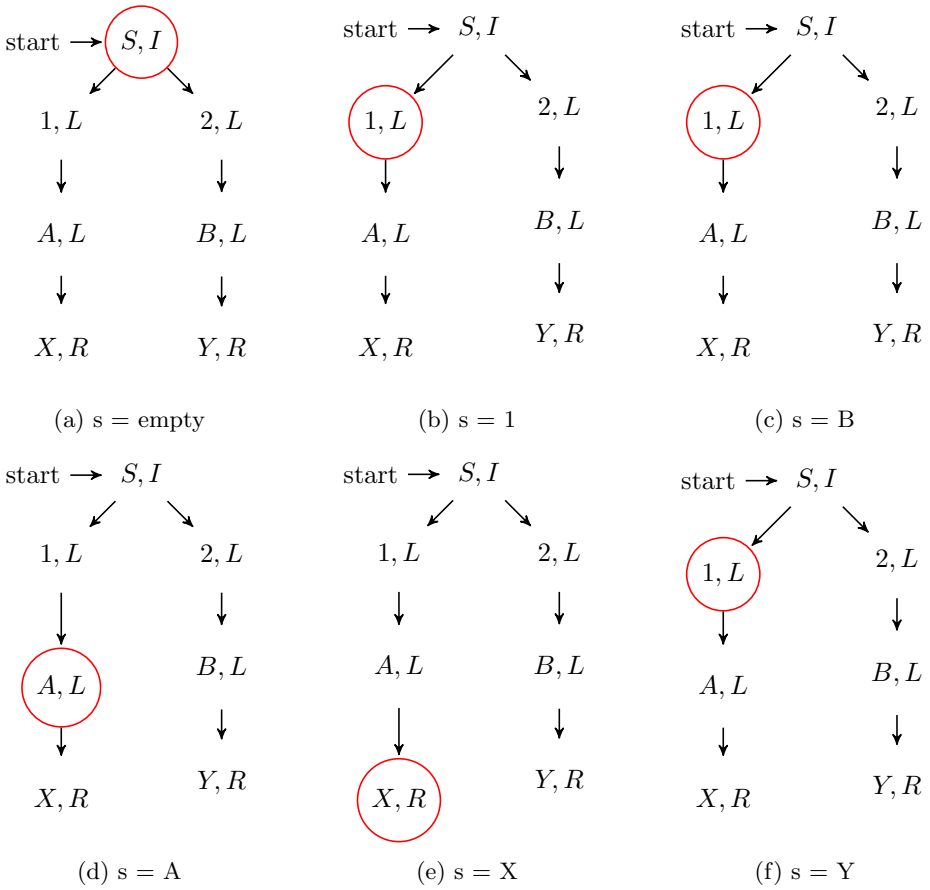


Figure 5: GLAS solving the 1-2-AX task. Red circle shows current node.

networks weights, it is hard for a human user to learn what the network knows.

In order to avoid this we proposed GLAS to represent knowledge in the form of a tree, with each node holding a stimulus and an action. An example for the 1-2-AX task can be seen in Figure 5. The tree starts at the root node, and we can see two distinct task sequences coming from it; 1-A-X and the 2-B-Y.

3.2.1 Action Selection

The algorithm itself starts at the root node. The current node is denoted in Figure 5 by a red circle surrounding a particular node. When initially presented to a sequence of stimuli, the algorithm should ignore it until it sees one of the task sequence starters, in this case either "1" or "2". If it identifies a task sequence starter in its input, it should move from the root node to the one related to the aforementioned input stimulus. From here on, it can only move to a child node or to another task sequence starter node (with the later taking precedence, representing the unit being reset), and only if some of them contains the currently seen stimulus. Once it reaches the last node in a task sequence (its ending node), it goes back to its respective task sequence starter. Notice that the mechanisms shown in Figure 4 and 5 produce the same action

selection at each step.

Pseudo-code for this algorithm can be seen in Algorithm 1.

Algorithm 1 Action Selection

```
1: function runAS(s)
2:     i ← 0(current stimulus)
3:     c ← 0(current node)
4:     for i = 0 to s length - 1 do
5:         if c is root then
6:             if s(i) is at start node then
7:                 c ← next viable node
8:         else if c is sequence start or intermediate then
9:             if s(i) is at start node then
10:                c ← viable start node
11:            else
12:                if s(i) is at child node then
13:                    c ← next child node
14:        else if c is sequence end then
15:            if s(i) is at start node then
16:                c ← viable start node
17:            else
18:                c ← this sequence's first node
19:        actions(i) = known_actions(c)
20:    return actions
```

For general cases, this action selection algorithm starts at root node in the solution tree, and at the first stimulus in the given sequence of events. The algorithm keeps track of which node it is and, as new stimuli are presented to the algorithm, it behaves differently depending on what kind of node it is at any given moment. If it is at the root node, it can only go to one of the starting nodes if one of them holds the current stimulus. If at a starter or intermediate node, it can only go either to a child node or a starting node holding the current stimulus. If the current node ends a sequence two things can happen: it can go to a starting node if there is a starting node holding the current stimulus, but if this is not the case it jumps back to the node which started the current sequence. Every time a new stimulus is presented an action is selected based on which node the algorithm is currently at.

3.2.2 Encoding

In the previous section we explained that GLAS represents knowledge in the form of a tree, with each node holding a stimulus and an action. We must, therefore, encode information for three different components: the tree structure, the stimulus at each node and the action at each node.

Encoding tree structure. We encode the "tree structure" as an array of integers, with position i in the array determining node i's parent in the tree. For instance, let us take the "tree structure" seen in Figure 6. In this example the tree has 8 nodes, and n1 is the root node. Nodes n2 and n3 are "task sequence starters". Nodes n4 and n6 are "intermediate nodes", and nodes n5, n7 and n8 are called "task sequence ending" nodes.

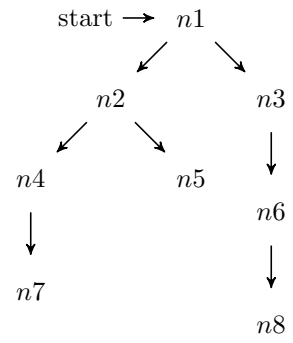


Figure 6: Example of a tree with 8 nodes.

We can represent this "tree structure" as the following vector of integers:

$$\begin{array}{rl} \text{tree structure} = & [\ 0\ 1\ 1\ 2\ 2\ 3\ 4\ 6\ ] \\ i & \quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \end{array}$$

Position $i = 1$ informs that node 1 has 0 as parent, which means node 1 is the root node. Position $i = 2$ informs that node 2 has node 1 as parent. Position $i = 3$ informs that node 3 also has node 1 as parent, which means that both nodes 2 and 3 are sequence starters. Following the same logic for the remaining elements of this array allows us to encode the same information we see in Figure 6.

Encoding stimuli. Stimulus at each node is also represented as a vector of integers. A possible example for the tree presented in Figure 6 is as follows.

$$\begin{array}{rl} \text{stimuli} = & [\ 0\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ ] \\ i & \quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \end{array}$$

In this example, the 0 at position $i = 1$ means node 1 is associated stimulus 0. The 1 in position $i = 2$ means node 2 is associated with stimulus 1, and so forth.

Encoding actions. Finally, action at each node is also represented as a vector of integers. A possible example for the same tree presented in Figure 6 is as follows.

$$\begin{array}{rl} \text{actions} = & [\ 0\ 2\ 2\ 1\ 3\ 4\ 2\ 1\ ] \\ i & \quad 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \end{array}$$

In this example, the 0 at position $i = 1$ means node 1 is associated with action 0. The 2 in position $i = 2$ means node 2 is associated with action 2, and so forth.

We call "solution tree", an array of integers with all three components in a row. For instance, the "solution tree" for the the previous example is the following vector of integers:
solution tree = [0 1 1 2 2 3 4 6 0 1 1 2 2 2 3 3 0 2 2 1 3 4 2 1]

In other words, the "solution tree" is encoded as an array of integers with three parts; tree structure, stimuli and actions. The final solution is a $3 * N$ sized vector, with N being the number of nodes: "solution tree" = ["tree structure" "stimuli" "actions"].

Table 2: Codification for the 1-2-AX benchmark task

For stimuli:

stim	U	1	2	A	B	X	Y
int	0	1	2	3	4	5	6

For actions:

act	I	L	R
int	0	1	2

Representing the solution tree for the 1-2-AX task. For the 1-2-AX benchmark task, we have used the following codification with "U" representing an unknown stimulus and "I" an ignore action, as seen in Table 2.

As an example, a solution tree for the 1-2-AX task is as follows.

- tree structure = [0 1 1 2 3 4 5]
- stimuli = [0 1 2 3 4 5 6]
- actions = [0 1 1 1 1 2 2]

solution tree = [0 1 1 2 3 4 5 0 1 2 3 4 5 6 0 1 1 1 1 2 2]

3.2.3 Learning

It is important to notice that the action selection algorithm itself is fixed. The only thing that changes is how it deals with stimuli (what to gate in, gate out and which action to select), which is defined by the solution tree.

Therefore, the learning process in GLAS consists on finding the solution tree which maximizes reward given a sequence of events. Each event is composed by a current stimulus, a selected action and a reward.

By using the same coding presented in Table 2, a possible sequence of events is shown in Table 3. For each event "i", there is a stimulus $\mathbf{s_i}$, a selected action $\mathbf{a_i}$ and a respective reward $\mathbf{r_i}$. Given rewards can be of values "1", when it chooses the correct button for non-ending cases, and "-1" for when it pushes the wrong one. Rewards are "10", when it chooses the correct button for sequence ending cases, and "-10" when it makes the wrong choice.

Then, we search for a tree maximizing the total accumulated reward by the algorithm as it is presented to the given sequence[4]

In this work, we used a traditional genetic algorithm (GA) to explore the space of possible solutions. We named the most important parameters as:

α : population size

μ : number of parents

λ : number of offspring

n_g : number of generations

The GA was implemented using Opt4J framework (Lukasiewycz, Glaß, Reimann, & Teich, 2011) for meta-heuristic optimization. Chromosome codification was defined as explained in Section 3.2.2, and the GA's operands and fitness function were implemented as follows.

[4]In other words, GLAS is currently an offline learning algorithm. A fixed sequence of events must be presented to it in order for it to learn a new solution. Much like how an MLP (Multi Layer Perceptron) neural network learns in batch mode.

Table 3: Example of sequence of events used to learn solutions for the 1-2-AX working memory task. Rewards r_i are given by the environment.

i	$\mathbf{s_i}$	$\mathbf{a_i}$	$\mathbf{r_i}$
0	1	1	1
1	4	1	1
2	3	1	1
3	5	2	10
4	2	1	1
5	3	1	1
6	4	1	1
7	6	2	10
8	1	2	-1
9	4	1	1
10	3	1	1
11	5	1	-10
12	2	1	1
13	3	0	-1
14	4	1	1

i	$\mathbf{s_i}$	$\mathbf{a_i}$	$\mathbf{r_i}$
15	6	2	10
16	2	1	1
17	4	1	1
18	3	1	1
19	5	1	1
20	1	1	1
21	5	1	1
22	4	1	1
23	6	1	1
24	2	1	1
25	3	1	1
26	3	1	1
27	6	2	-10

Operands. Mutation is slightly different for each part of the chromosome. For actions, it is a random variation between 0 and the number of known actions available (including the ignore case). For stimuli it is the same, but in the interval ranging from 0 to the number of known stimuli (including the unknown case). For structure, each position is only allowed to vary in a range that produces a valid tree. We used the default crossover.

Fitness. Each individual in the population goes through the same sequence of events, using the knowledge in its solution tree to choose the best action at each time step. If it chooses the same action in the current event, it accumulates its reward (which could be positive or negative). If it chooses a different action, it cannot judge the merit of the action it has chosen and, therefore, ignores the reward at the present event and moves on. The total final fitness is calculated as follows.

Given:

ϕ : total fitness

N : number of nodes

ρ : accumulated reward

n_e : number of events in the sequence

n_i : number of intermediate nodes in the history of nodes

ι : reward to avoid staying in intermediate nodes[5]

[5]Adding this reward helped the algorithm escape some local maxima by encouraging individuals going through whole task sequences more often.

In order to calculate ι, we first present the sequence to a given individual, and record its history of nodes. At each time step, it should be at a particular node from the solution tree. Whenever the current node is an intermediate node, we add "1" to n_i.

Equation 1 shows how to calculate the reward to those individuals who avoid staying too long in intermediate nodes.

$$\iota = 1 - \frac{n_i}{n_e} \tag{1}$$

Finally, Equation 2, shows the final fitness for a particular individual.

$$\phi = \rho + \iota - N \tag{2}$$

There is a penalty for larger trees (larger N), which acts as an incentive for more parsimonious solutions.

Pseudo-code for the learning component of GLAS can be see in Algorithm 2.

Algorithm 2 Learning from sequence

```
1: function LEARN(events sequence, N)
2:     population ← α random valid solutions with N nodes
3:     for g = 1 to n_g do
4:         fitness ← getFitness(population, events)
5:         population ← selectBestFit(population, fitness)
6:         population ← applyOperands(population)
7:     fitness ← getFitness(population, events)
8:     solution ← individual with best fitness
9:     return solution
```

In *getFitness()*, each individual in the population is passed to *runAS()*, producing a sequence of chosen actions. This list of chosen actions is then used to calculate the individual's fitness based on Equations 1 and 2.

We must stress, however, that the learning component illustrated in Algorithm 2 could be carried out by other optimization algorithms. Therefore, this pseudo-code should be taken as reference for how we solved the problem of finding an appropriate solution, and not as a set of necessary rules that must be followed.

4 Results

In order to train a GLAS instance with the sequence of events given in Table 3, we used the following parameters (other parameters are the default ones): $\alpha = 100$, $\mu = 2$, $\lambda = 2$, $n_g = 1000$.

We ran the learning algorithm for different numbers of nodes, ranging from N=2 to N=9. The result, in fitness, for the best solution found with each number of nodes can be seen in Figure 7.

As the number of nodes grows from 2 to 7, there is an increase in the resulting fitness for the best found solution. After that. if the algorithm tries to add more nodes to the solution, fitness starts to decrease accordingly. This is an indication that the best overall found solution, given this sequence of events, is the one with N=7.

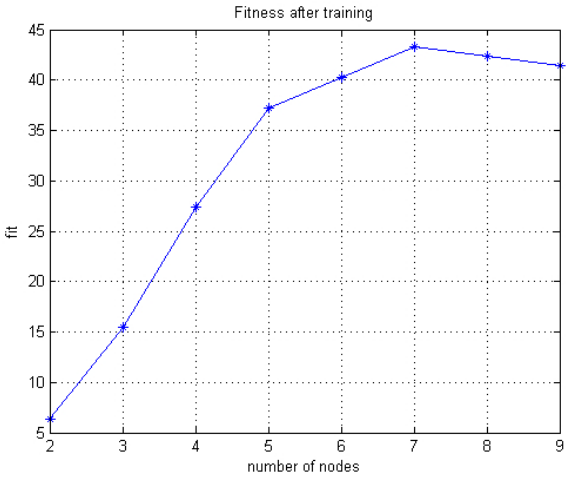


Figure 7: GLAS learning the 1-2-AX working memory task for different numbers of nodes.

In the following figures we can see examples of solutions found by the GA. Each node is denoted by a gray circle with its number written at its lower center. Inside each circle and to the left, we see the stimulus that activates the given node, while to the right we can see the action it performs. Figure 8a shows the best solution found for N=6. Figures 8b and 8c show two possible solutions for N=7. Notice how fitness is slightly higher for the solution described in Figure 8b. At last, Figure 8d shows the best solution found for N=8, which is very similar in functionality to the best one for N=7, but has a penalty for having more nodes in comparison.

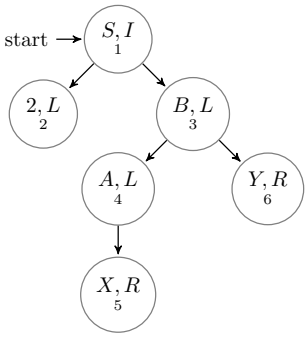
5 Discussion

This GLAS algorithm was inspired by a computational neuroscience model of how the PFC and BG interact in order to learn action selection from stimuli and late rewards. Abstracting this mechanism as an action selection algorithm and a knowledge tree working together provided the following benefits:

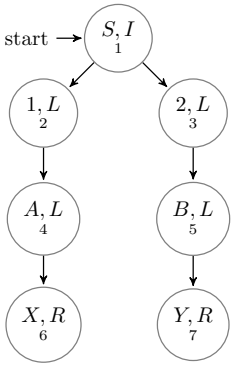
- What the algorithm has learned is now in human readable form.
- As a combinatorial task, it can now be tackled by a number of optimization techniques specialized in solving this class of problems.
- The use of GA could ease its application to multi-objective problems.
- New heuristics could be specifically designed to modify the fitness function in order to improve training for different applications.

We are particularly interested in further investigating how different heuristics would influence the algorithm's capacity to correlate stimuli far apart in time. Some of the drawbacks we identify in this approach are:

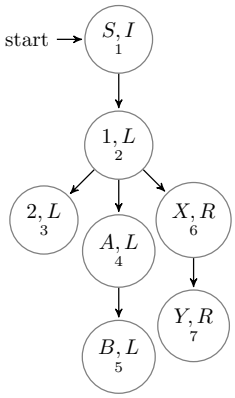
- No guarantee to find the best possible solution.



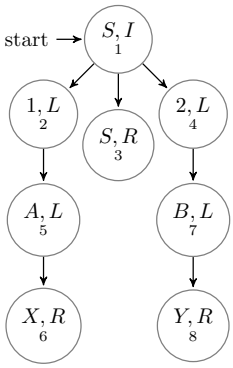
(a) Best found solution for N=6 (Fitness = 40.2857).



(b) Best found solution for N=7 (Fitness = 43.2500).



(c) Alternative solution for N=7 (Fitness = 43.1250).



(d) Best found solution for N=8 (Fitness = 42.3333).

Figure 8: Results for the learning algorithm with different number of nodes.

- Curse of dimensionality. A larger number of stimuli and actions can make this approach unfeasible depending on how big the space to be explored is.

Both drawbacks are common when dealing with combinatorial optimization problems, and should always be taken into consideration whenever choosing GLAS, or similar mechanisms, to solve a specific problem.

6 Conclusions

In this paper we introduced and described a novel gated learning action selection algorithm named GLAS. Its mechanism was inspired by a computational neuroscience model of how the PFC and BG interact in order to learn action selection from stimuli and rewards separated in time.

We are currently working in an online mode of learning, which we believe is more interesting in the context of intelligent agents. Our next step is to validate GLAS with real world problems, such as controlling intelligent agents in games and mobile robotics. Then, we intend to explore

other optimization tools - such as Particle Swarm Optimization, Simulated Annealing and others - in order to investigate whether any of those algorithms might produce a better result. We also intend to address the aforementioned drawbacks, and investigate some of the algorithms properties such as convergence and potential for generalization.

Acknowledgment

We would like to thank CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), and FAPESP (Fundação de amparo a pesquisa do Estado de São Paulo) for the financial support.

References

Alvarez, J. A., & Emory, E. (2006). Executive function and the frontal lobes: A meta-analytic review. *Neuropsychology Review*, *16*(1), 17–42.

Baars, B., & Franklin, S. (2009). Consciousness is computational: the lida model of global workspace theory. *International Journal of Machine Consciousness*, *01*(0101), 23. doi: 10.1142/S1793843009000050

Baars, B., & Gage, N. (2010). *Cognition, brain, and consciousness: Introduction to cognitive neuroscience* (2nd ed., Vol. 1). Academic Press.

Baddeley, A. D., & Hitch, G. (1974). Working memory. In G. H. Bower (Ed.), *Psychology of learning and motivation* (Vol. 8, p. 47 - 89). Academic Press. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0079742108604521` doi: http://dx.doi.org/10.1016/S0079-7421(08)60452-1

Bagnell, J., & Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and automation, 2001. proceedings 2001 icra. ieee international conference on* (Vol. 2, p. 1615-1620 vol.2). doi: 10.1109/ROBOT.2001.932842

Bakker, B. (2002). Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, *14*, 1475–1482.

Bakker, B., Zhumatiy, V., Gruener, G., & Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *Intelligent robots and systems, 2003.(iros 2003). proceedings. 2003 ieee/rsj international conference on* (Vol. 1, pp. 430–435).

Chakravarthy, V., Joseph, D., & Bapi, R. (2010). What do the basal ganglia do? a modeling perspective. *Biological Cybernetics*, *103*(3), 237-253. Retrieved from `http://dx.doi.org/10.1007/s00422-010-0401-y` doi: 10.1007/s00422-010-0401-y

Chan, R. C., Shum, D., Toulopoulou, T., & Chen, E. Y. (2008). Assessment of executive functions: Review of instruments and identification of critical issues. *Archives of Clinical Neuropsychology*, *23*(2), 201 - 216. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0887617707001928` doi: http://dx.doi.org/10.1016/j.acn.2007.08.010

Chersi, F., Ferrari, P., & Fogassi, L. (2011). Neuronal chains for actions in the parietal lobe: a computational model. *PloS one*, *6*(11), e27652.

Frank, M., & Badre, D. (2012). Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis. *Cerebral cortex*, *22*(3), 509–526.

Fuster, J. (2008). *The prefrontal cortex* (4th ed., Vol. 1). Academic Press.

Hazy, T., Frank, M., & O'Reilly, R. (2007). Towards an executive without a homunculus: computational models of the prefrontal cortex/basal ganglia system. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *362*(1485), 1601–1613.

Lukasiewycz, M., Glaß, M., Reimann, F., & Teich, J. (2011). Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proceedings of the genetic and evolutionary computing conference (gecco 2011)* (pp. 1723–1730). Dublin, Ireland.

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

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, *55*(23), 311 - 365. Retrieved from http://www.sciencedirect.com/science/article/pii/0004370292900586 doi: http://dx.doi.org/10.1016/0004-3702(92)90058-6

O'Reilly, R. C., Frank, M. J., Hazy, T. E., & Watz, B. (2007). Pvlv: the primary value and learned value pavlovian learning algorithm. *Behavioral neuroscience*, *121*(1), 31.

O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., & Contributors. (2012). *Computational cognitive neuroscience*. Wiki Book, 1st Edition. Retrieved from http://ccnbook.colorado.edu

Pérez-Ortiz, J. A., Gers, F. A., Eck, D., & Schmidhuber, J. (2003). Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, *16*(2), 241–250.

Raizer, K., Paraense, A. L. O., & Gudwin, R. R. (2012). A cognitive architecture with incremental levels of machine consciousness inspired by cognitive neuroscience. *International Journal of Machine Consciousness*, *04*(02), 335-352. Retrieved from http://www.worldscientific.com/doi/abs/10.1142/S1793843012400197 doi: 10.1142/S1793843012400197

Raizer, K., Rohmer, E., Paraense, A., & Gudwin, R. (2013, April). Effects of behavior network as a suggestion system to assist bci users. In *Computational intelligence in rehabilitation and assistive technologies (cirat), 2013 ieee symposium on* (p. 40-47). doi: 10.1109/CIRAT.2013.6613821

Reggia, J. A. (2013). The rise of machine consciousness: Studying consciousness with computational models. *Neural Networks*, *44*(0), 112 - 131. Retrieved from http://www.sciencedirect.com/science/article/pii/S0893608013000968 doi: http://dx.doi.org/10.1016/j.neunet.2013.03.011

Riedmiller, M., Gabel, T., Hafner, R., & Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots*, *27*(1), 55-73. Retrieved from http://dx.doi.org/10.1007/s10514-009-9120-4 doi: 10.1007/s10514-009-9120-4

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Prentice Hall. Retrieved from http://aima.cs.berkeley.edu

Stewart, T. C., Choo, X., & Eliasmith, C. (2010). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In *Proceedings of the 10th international conference on cognitive modeling.* Retrieved from http://compneuro.uwaterloo.ca/files/publications/stewart.2010.pdf

Stone, P., Sutton, R. S., & Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, *13*(3), 165-188. Retrieved from http://adb.sagepub.com/content/13/3/165.abstract doi: 10.1177/105971230501300301

Zico Kolter, J., & Ng, A. Y. (2011). The stanford littledog: A learning and rapid replanning approach to quadruped locomotion. *The International Journal of Robotics Research*, *30*(2), 150-174. Retrieved from http://ijr.sagepub.com/content/30/2/150.abstract doi: 10.1177/0278364910390537