Robotic Curiosity and Memory
Spatial and Implicit Approaches to Time in CBIM

Zack Ontiveros and Dougal Sutherland

Abstract

CBIM is an epigenetic robotics system that uses intrinsic motivation and categorization to learn about its relationship with its environment. CBIM, however, fails to adequately deal with time-dependent relationships; predictions made using CBIM utilize only the current timestep data. In the real world, time is an essential part of interaction. For a system to be able to understand and predict complex patterns and relationships, it requires some way of representing time. In this paper, we explore two methods for applying time to CBIM – one spatial and one implicit – and compare the performance of these two methods and CBIM using several simulated robot tasks.

1 Introduction

One key aspect in the development of general-purpose intelligence is curiosity, the desire to learn about new things. Human infants are particularly curious; they are engaged in a continual investigation of their environment and the interaction between their own actions and the outside world. These activities are not motivated by any external source, but rather come from an intrinsic desire to learn. Later in life, although extrinsic motivations play more prominent roles in day-to-day activities, this intrinsic desire to learn remains; we are forever driven to explore the unknown.

A goal of epigenetic robotics is to use human development as a model for implementing systems that mirror a "human" brand of general intelligence. Instead of hard-coding the robot's controller to attempt different activities, the robot is given the opportunity to explore its environment. Its learning is intrinsically motivated, fueled by the desire to correctly predict how its environment will respond to its actions.[1]

The Intelligent Adaptive Curiosity (IAC) system of Oudeyer, Kaplan, and Hafner (2007) uses an intrinsic motivation for curiosity as the driving force in an active learning system. In IAC, the robot categorizes its sensorimotor space into regions, grouping together similar situations in the environment. Each region is associated with an expert that learns to predict how the robot's actions will affect its environment – that is, to predict its sensors values at the next timestep, given the current timestep's sensor values and the motor actions that the robot is about to perform. IAC focuses on events that will result in the maximum learning progress, essentially those cases at "the edge of chaos," as Clark (1998) puts it, which are neither too predictable nor too hard (or impossible) to learn.

IAC incorporates several of the fundamental principles of development espoused by Lungarella, Metta, Pfeifer, and Sandini (2003). Its development is incremental, self-organizing, and is dependent upon the predictive control of the environment based on categorization. It incorporates spontaneous activity and, depending on the level at which sensors and motor actions are defined, allows for self-exploration. Its active nature is essential, as Baranes and Oudeyer (2009) argue. Complex systems are much harder to learn when the learner cannot interact with them; Jay McLellan described attempts to learn language passively as "like trying to learn a language by listening to the radio" (Elman, 1990).

One problem with IAC, however, is its ad-hoc categorization method. It creates new regions according to a simple rule that splits a region once the number of experts has reached some fixed size. Repeatedly performing the same action in the same context forever would thus result in infinitely many identical regions being created. Effective categorization, however, is absolutely essential to successful development: as Harnad (2005) argues, it is fundemental to

[1] We do not claim that a full general intelligence could be developed using *only* curiosity, of course. A robotic system motivated only by curiosity might repeatedly hurl itself out of a window, interested in the way its perception changed as it fell and how its motor controls change once damaged.

the basic activities which all organisms perform. For a system to be robust, it must not be bogged down by minute differences between similar situations; instead it must be able to abstract from those situations a general solution.

Lee, Walker, Meeden, and Marshall (2009) attempt to resolve this with a modification which they call Category-Based Intrinsic Motivation (CBIM), which replaces IAC's categorization method with a more dynamic technique known as Growing Neural Gas (Fritzke, 1995). This system creates new regions only when the sensor data describing the new situation differs sufficently from the present regions, resulting in a better categorization of the environment over time. This improved categorization system makes using neural network experts, rather than the k-nearest-neighbor experts used in IAC, more feasible, as there are fewer redundant regions which each claim part of the relevant training data.

One major drawback of the IAC/CBIM architecture, however, is that predictions are based only on the current sensorimotor information. Time-dependent relationships, which are clearly vital to an understanding of complex real-world situations, are impossible to learn. A human — or, indeed, any complex animal — with no form of short-term memory would have an exceedingly difficult time understanding the world around it, let alone acting in it. Therefore, for the IAC/CBIM approach to be workable in more complex tasks, it must be augmented with time information in some way.

In this paper, we propose two variations of the CBIM architecture to incorporate time information. FixWinCBIM employs a "spatial" approach to time, extending the robot's sensorimotor context to include information from the last N timesteps as well as the current one. MonoCBIM instead uses an Elman network for memory, which necessitates abandoning per-region experts in favor of a single monolithic expert that predicts over the entire input space. In this paper we will discuss their individual implementations and analyze their performance in comparison to that of the original CBIM on various simulated tasks.

2 System Details

We present the IAC/CBIM architecture only in its CBIM realization, as it is that approach on which our work is based.

2.1 CBIM

The CBIM system has three major parts. The first is a Growing Neural Gas (Fritzke, 1995), an unsupervised vector quantization system which builds a model of the topology underlying its input vectors. Nearby model vectors are shifted slightly to accommodate for each new input; in the Equilibrium-GNG variation (Provost, Kuipers, & Miikkulainen, 2006) which we employ, new model units are created once the overall error in the GNG has reached a certain threshold, allowing the GNG to create just enough units to adequately represent its inputs. Part of the process involves edges between units in the model; CBIM does not use these edges directly. Example GNGs are shown in Figure 1.

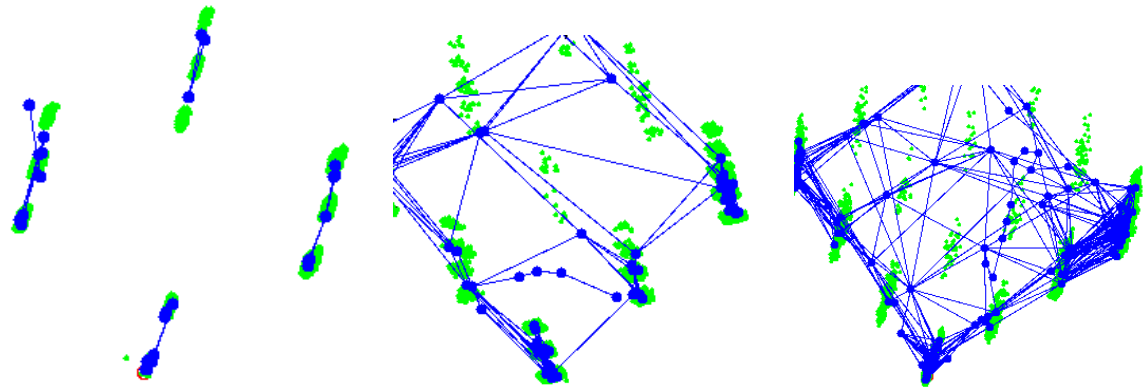
The second component of CBIM is its prediction mechanism. This is comprised of simple feedforward neural networks associated with each GNG unit, which are trained using backpropagation to output a prediction for the next timestep's sensors, $\hat{S}(t+1)$, given the current sensorimotor context $SM(t)$.

The third component is the action selection mechanism. This system keeps track of the error in predictions — Euclidean distance between the prediction $\hat{S}(t+1)$ and the true value $S(t+1)$ — for each region. Learning progress is then computed according to a weighted estimate of the prediction error's derivative. More details on this procedure are available in Oudeyer et al. (2007).

When CBIM chooses an action to take, it begins by considering the current sensor data $S(t)$ and generating potential motor actions $M_i(t)$. It then finds the region associated with $SM_i(t)$ in the GNG and examines the learning progress in each region. Most of the time, it then chooses an action associated with the region currently making the fastest learning progress; some fraction of the time, it chooses a random action. (In our experiments, we used a random action rate of 15%.)

After choosing an action, CBIM asks the associated expert for a prediction of next timestep's sensor values. CBIM then executes the chosen motor action, finds the true value of $S(t+1)$, and uses that value to train the expert. It also stores the prediction error in the region for future learning progress calculations.

In this way, CBIM focuses at first on easily-learnable regions of its sensorimotor space, as their learning progress will be high. Once an expert has been well-trained, it changes to focus on the next-most learnable region. This provides it with a clear developmental trajectory as it focuses on more com-



(a) $N = 1$ (b) $N = 2$ (c) $N = 3$

Figure 1: 2D projections of the GNG models for the experiment described in Section 3.1.1 at various fixed-window sizes. Note the exponential increase in the number of model vectors.

plicated aspects of its environment in phases.

2.2 Fixed Window CBIM

One approach that we took in adding time information to CBIM is to extend the sensorimotor context used at each timestep with a buffer that remembers the previous $N-1$ timesteps. We call this approach FixWinCBIM. Using a fixed window size of two, then, the sensorimotor $SM(t)$ used by CBIM is then replaced with the concatenation $SM(t)\,SM(t-1)$. These longer sensorimotor vectors are categorized by the GNG and provided as input to each region's experts.

This approach allows us to retain the local experts of the original IAC/CBIM architecture – a major advantage, in that it keeps each expert's prediction space simple and (hopefully) easy to learn. The GNG categorization will then seperate the space based on not only the last timestep but also each timestep before that; if there would be k categories when considering only one timestep's sensorimotor data, there would be approximately k^2 categories using a fixed window of 2, and k^N for a fixed window of N. This is illustrated in Figure 1.

This categorization may well be more than is actually required for the task. If previous timesteps' sensor data are irrelevant to predicting the effects of motor actions in a certain context, the GNG will still create separate categories for each combination of past timestep states with the current one, thus significantly overcategorizing the space. The robot will then have to learn the prediction space for each of those categories. Each region may be easy to learn, but it still certainly slows down the process; different experts must learn essentially the same thing.

The robot's memory is further limited by the value of N: it would be unable to learn time dependencies that span a longer time window. This is a problem in that the designer must predetermine how large N should be; if it is too large, development will be much slower and the number of categories (and hence the number of experts) will increase exponentially, while if it is too small, certain aspects of its environment will be simply unlearnable.

2.3 Monolithic CBIM

MonoCBIM, our other approach in adding time information to CBIM, is a more drastic change. We use an Elman network (Elman, 1990) as an expert to add an implicit memory, which can (in theory) autonomously decide what is important to remember and how long it should be remembered for. (An Elman network is a simple three-layer neural network in which the hidden layer is self-recurrent, so that last timestep's hidden layer is used as a so-called context layer for the next timestep's evaluation.)

This requires, however, that the *same* network be consulted at each timestep — otherwise its memory would only have access to the timesteps where sensorimotor perceptions fell into that expert's region. We therefore implemented it as a single "monolithic" expert, whose task is to predict the effects of motor actions over the entire input space of the task.

MonoCBIM still uses the GNG to categorize sen-

sorimotor inputs and choose which motor action will result in the most learning progress, using them to drive the active learning system. The monolithic expert's prediction space, however, is far more complex than that of any of the experts in the original CBIM or FixWinCBIM, and so MonoCBIM abandons many of the categorization advantages of the IAC/CBIM architecture. It is also unclear in advance how large the context layer of the Elman net should be.

Simple neural network architectures like the one we use tend to be bad at predicting many unrelated tasks; after mastering one task, the expert may well forget it while learning another. Alternatively, if the learning rate is too low, the network may learn the first task and then be too "stuck in its ways" to learn another unrelated task. In complex or general-purpose environments, more complex topologies or possibly other machine learning systems may be needed.

Some have found success in training networks by resetting the context layer, first at short intervals and then gradually extending the resetting schedule. We attempted this on the time-dependent experiment described in Section 3.3, but found that performance was essentially identical.

Using a monolithic expert does have another advantage, though, in that the training process never needs to restart. In CBIM, when a region is split, the neural network expert is started from random connection weights.[2]

3 Experiments

To examine the characteristics of our new CBIM variations, we compared the performance of FixWinCBIM and MonoCBIM to the original CBIM in several tasks, including the original CBIM experiment and new explicitly time-dependent tasks.

All experiments were performed in the PyRobot simulator (Blank, Kumar, Meeden, & Yanco, 2006). We used a learning rate of 0.5 and a momentum of 0 for our experts; the time window of progress calculation was 10, while the smoothing factor was 15.

In all experiments, we tested FixWinCBIM with window sizes of both 2 and 4, and MonoCBIM with Elman context sizes of both 4 and 15. This allows us to determine their relative performance on tasks of varying complexity.

3.1 Color Vision Task

We reproduced the original CBIM experiment performed by Lee et al. (2009), as well as investigating a simplification of that task.

3.1.1 Original CBIM Experiment

The original CBIM experiment consisted of one developing robot placed in the middle of an environment with green walls. The robot shares its environment with a stationary red robot and a blue robot which oscillates back and forth at a regular rate. The developing robot is used essentially as a rotating camera; it is constrained to stay in-place but can rotate arbitrarily (controlled by a motor output separated into seven bins for various rotation speeds). The simulated situation is pictured in Figure 2.

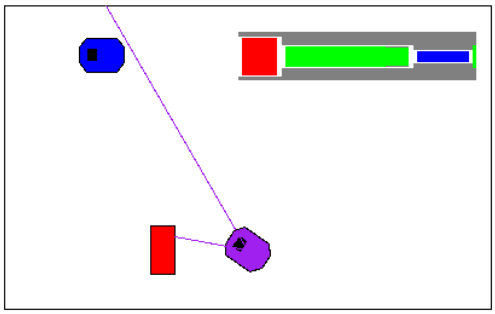


Figure 2: Our simulation of the original CBIM task. The overlay represents the robot's camera.

The robot is given five sensors. The first three are binary flags indicating the presence or absence of each of red, green, and blue in its field of vision. The robot also chooses with its second motor action to focus on a specific color (red, green, or blue) at each timestep. If the robot's color-focus motor value was between 0 and $\frac{1}{3}$ in the previous timestep, it focuses on red, between $\frac{1}{3}$ and $\frac{2}{3}$ on green, or between $\frac{2}{3}$ and 1 on blue. The robot's fourth sensor represents the size of the bounding box around the largest area of the focused color in its vision, and the fifth the position of that box: 0 for it being in the left portion of its vision, 0.5 for the middle, or 1 for the right.

[2]This is, to some degree, simply an implementation issue: one could choose to retain all past sensorimotor training data, or a representative sample, and then train new experts based on the data that fall into their new region. This is the approach taken by Oudeyer et al. (2007), though because they use k-nearest-neighbor experts, they have to retain all past training data in any case.

We ran two versions of this experiment, with the blue robot moving at different speeds.

3.1.2 Bluebot experiment

The bluebot experiment is a simplification of the original CBIM experiment. We extracted only the part to which our temporal modifications are relevant: trajectory-following on the oscillating robot. In order to predict where an object is about to be, one needs to know not just where it currently is, but also which way it is travelling – information that could be provided by knowing where it was a fraction of a second earlier, or, perhaps, by some sort of switch state in the context layer of an Elman network. It could reasonably be expected, therefore, that time-aware versions of CBIM will have improved performance on this task.

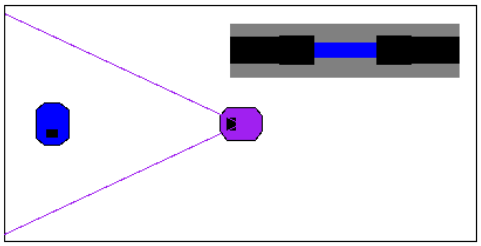


Figure 3: The environment for the bluebot simplification of the original CBIM task.

The environment, as pictured in Figure 3, consists of just the developing robot and a bluebot oscillating across its field of vision. (Walls are no longer green in the simulator.) The developing robot is a stripped-down version of the original color-vision robot: the red and green detection sensors are removed, as neither will ever be present, and the robot's rotation motor has also been removed; it is stuck in place. The color location sensor has also been made continuous rather than chunked, in an effort to make the task slightly more time-dependent. To maintain the active nature of CBIM, however, the color-focus motor has been retained. If the robot chooses to focus on red or green – ie, if its motor output is less than $\frac{2}{3}$ – its sensor values will all be 0.

We also ran two versions of this experiment with the same two speeds for the bluebot.

Results

Somewhat surprisingly, the new implementations of CBIM performed no better than the original in either the bluebot or original experiments, as shown in Figures 4 and 5. (Error is only reported to 10,000 timesteps, but runs were generally made to 20,000 steps, and a few were tried out to 60,000, with no improvement in performance.) Results for the two speeds of bluebots were similar.

We believe that this lack of improvement has to do with the nature of the experiment. Although trajectory-following is in theory a time-dependent task, our timesteps are quite short compared to the speed of the bluebot, even when it moves at the maximum speed allowed by the simulator. As such, predicting that the robot will simply occupy in the same place as it did the timestep before will result in quite low error already; there is not enough "incentive" to learn the nuances of that sensor value.

3.2 Jumping robot experiment

We next altered the bluebot task to be more explicitly time-dependent. In this "jumping robot" experiment, the experimental setup is identical to that of Figure 3. Rather than moving back and forth, however, the blue robot jumps in and out of the developing robot's field of vision. Sensors are further simplified to a single binary value that tests whether the focused color is present in the robot's field of vision. The color focus motor is maintained.

We ran two variations of this experiment: one where the bluebot jumped every 3 steps and one where it jumped every 5.

Ideally, this task should be impossible for memoryless CBIM to learn to better than a $\frac{2}{3}$ probability ($\frac{4}{5}$ in the 5-step version) when it chooses to focus on blue, while fixed windows of two steps provide enough information for perfect predictability in the 3-step case, and 4 enough in the 5-step case.

Results

The fixed-window runs did not achieve perfect prediction status. They did, however, perform significantly better than the original CBIM. In the 3-step version, the 2- and 4-window systems performed similarly; in the 5-step version, the 2-window system did only slightly better than original CBIM, while the 4-window system performed rather well. This is as expected.

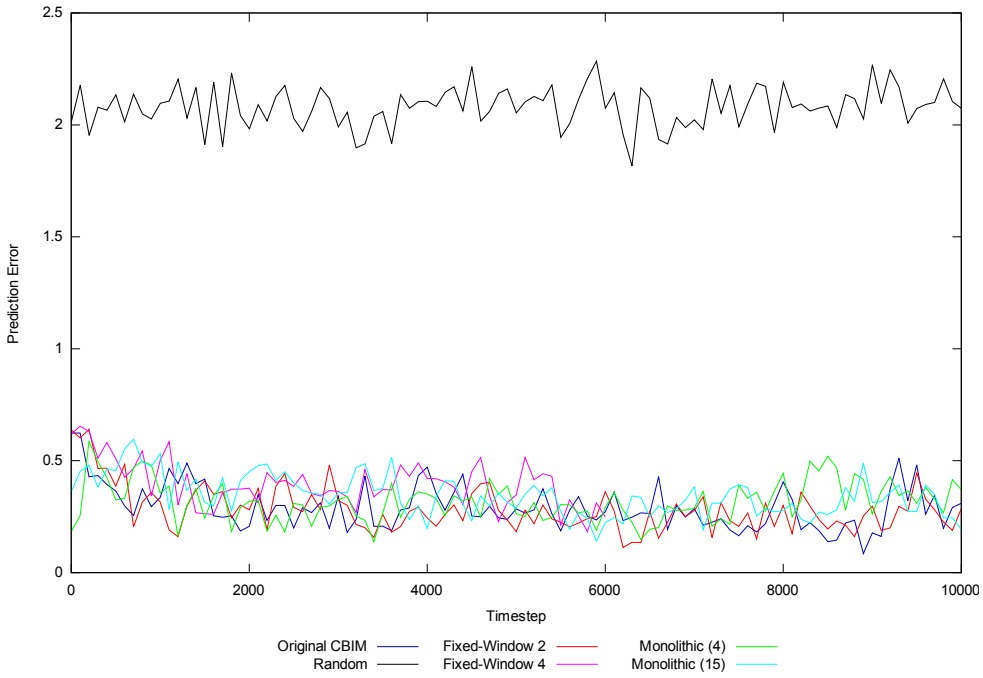


Figure 4: Prediction errors for the full color vision task. The black "random" line shows prediction error from a controller which simply made random guesses as to the sensor values, indicating baseline performance for the task.

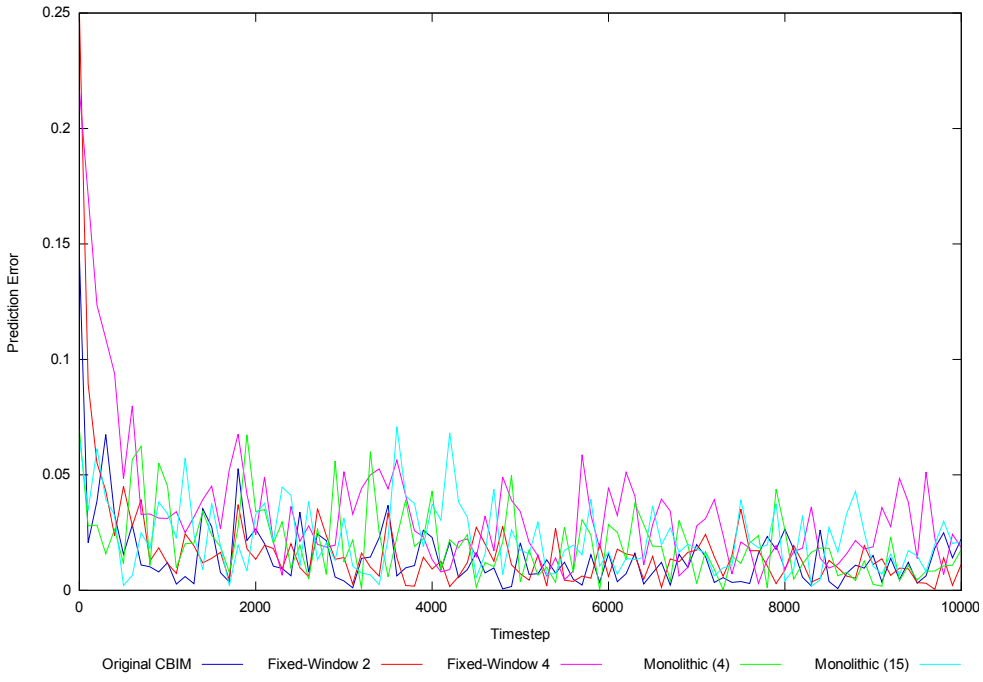


Figure 5: Prediction errors for the bluebot simplification of the color vision task.

The Elman networks did not fare as well; in fact, the 15-node context version performed worse than the original CBIM.

Part of the poor performance may be due to a bug in our simulation. It appeared that almost 10% of the time, our method for moving the bluebot did not line up exactly with the timesteps of the simulation; it sometimes appeared in the camera results either the timestep before or the timestep after. This would make the task as a whole significantly less predictable.

3.3 Overheating Toy Example

Our experiment with the most complex time dependence involved a developing robot interacting with a toy of sorts. The toy changes colors according to an internal state; the robot has a camera which views this color, and one motor action that determines whether or not it interacts with the toy. The world itself is otherwise identical to Figure 3.

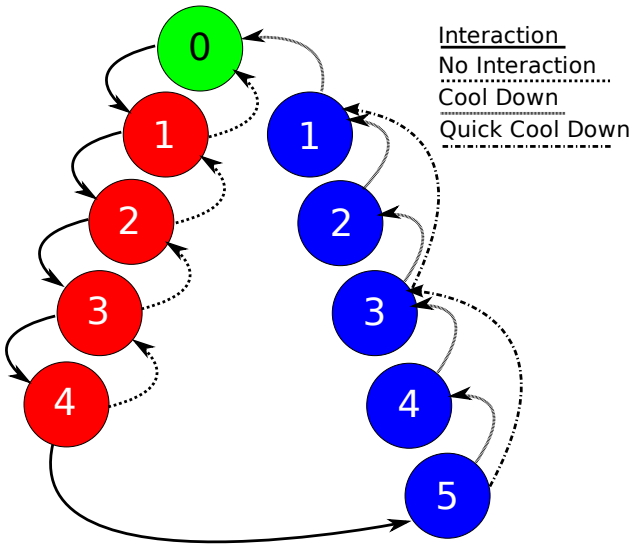


Figure 6: The state transitions of the color-changing toy.

The toy's color relies on the value of an internal counter. Initially, the counter is set to 0 and the toy's color is green. If the robot interacts with the toy, its counter increments; if not, the counter decrements. The toy turns red when its counter is more than 0; if it reaches 0 again, it reverts to green.

If the toy's counter reaches 5, it "overheats" and turns blue. While the toy is overheated, the robot's interactions (or lack of interactions) have no effect; it instead cools down at a rate which is a parameter of the experiment. We did experiments with cooling rates of 1 and 2, resulting in toys that cool down in 5 or 3 steps, respectively. Once the cooldown period has ended, it returns to green.

Results

Prediction errors from representative runs of the task, with cooldown rates of 1 and 2, are presented in Figures 7 and 8, respectively. Error rates are only shown through 10,000 steps, but they were run through 60,000 and showed no significant changes not represented in the first 10,000 steps.

FixWinCBIM clearly outperformed its competitors on this task. The differences between the sizes of the fixed windows are interesting, however. In the experiment with a cooldown rate of 1 (Figure 7), the fixed-window systems with windows of 2 and 4 performed similarly, although the 4-window one takes significantly longer to reach that level of performance (as expected). This makes sense; neither one has a long enough window to know when the cooldown process or, to a lesser extend, the overheating process, began; they are working with essentially the same information. In the experiment where the cooldown process takes three steps (Figure 8), however, the system with a window of 4 greatly outperformed the one with a window of 2. In this case, the cooldown process is completely predictable, so error is reduced drastically.

The Elman networks performed at about the level of the original CBIM.

4 Discussion

The results from our experiments overwhelmingly suggest that FixWinCBIM is the best approach to adding time to CBIM, at least in these simple situations. Although it does over-categorize the input space and thus takes longer to train to its final level of performance, its asymptotic performance does not suffer even in non-time-dependent tasks such as the original CBIM experiment.

Unfortunately, MonoCBIM failed to outperform the original CBIM in any of our experiments. This is probably due to several factors, perhaps the most important of which is that Elman networks simply are not very good at counting (Elman, 1990). In the simple domains in which we can practically test and an-

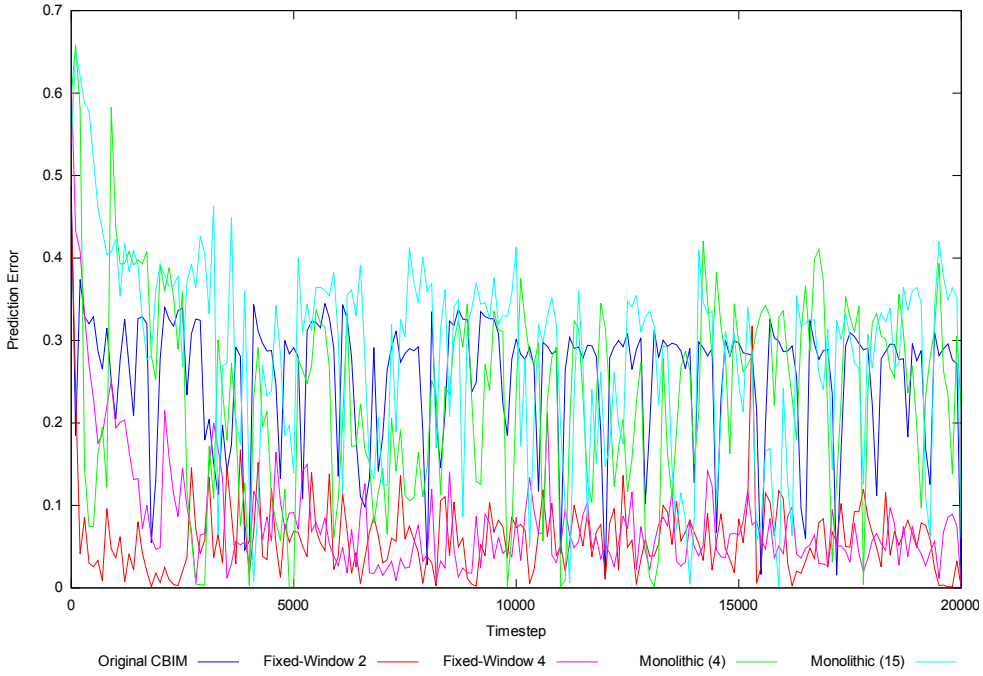


Figure 7: Prediction errors for the color toy task of Section 3.3, with a cooldown rate of 1.

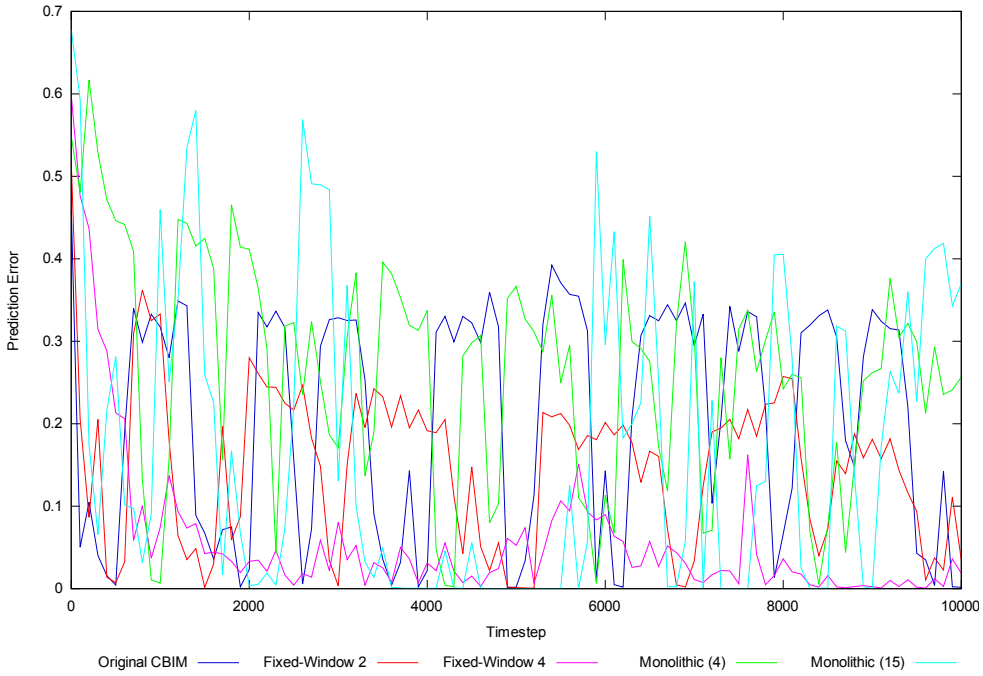


Figure 8: Prediction errors for the color toy task of Section 3.3, with a cooldown rate of 2.

alyze CBIM-like architectures, however, most time-dependent tasks seem to rely essentially on counting.

Furthermore, the simple topology of the network may be holding it back. More complex topologies and a better learning algorithm than backpropogation might be advantageous. In particular, training Elman networks can be a sticky wicket, and CBIM's approach to guiding training is not necessarily particularly well-suited to training Elman nets.

Perhaps abandoning the categorization aspect of CBIM to the extent that MonoCBIM does, however, is simply too much. Essentially, MonoCBIM is just an Elman network being trained on its environment according to how quickly it is learning various aspects of it. More tweaking of the system to accommodate for the idiosyncrasies of the Elman nets would be productive.

5 Future Work

There are many aspects of the CBIM architecture which could be profitably improved, many of which have nothing to do with knowledge of time. Some form of accounting for other motivations than just curiosity would be necessary to a fuller system; work on constraining development in the early stages and gradually opening up possibilities is also very interesting.

In terms of the systems discussed in this paper, though, there is also much interesting work to be done. Clearly MonoCBIM needs improvements if it is to be used effectively. There are several lines of work that could be done in this area; most obviously, one could use a different learning system than Elman networks — one that learns faster, is better at predicting, and/or has a more robust memory system. The Long Short-Term Memory system of Hochreiter and Schmidhuber (1997) seems one promising avenue to investigate, or perhaps their related recurrent SVM approach.

Alternatively, it would be ideal to come up with a system that can somehow combine local experts with a global memory. Perhaps there could be a monolithic memory system that would pass extra inputs to the local experts, or each expert could be provided with extra outputs to signal one another. There are many interesting issues to work out in such systems, but retaining highly specific experts seems desirable.

In terms of FixWinCBIM, the largest problem is clearly setting the parameter N. Perhaps a system could be developed that would dynamically determine how much context is needed for different top-level categories of the current sensorimotor contexts. Once it had been determined that a given node needs more context (by having high enough error), it could be provided with another level of context. There would then be a sub-GNG that only applies to certain top-level nodes, a process which would be reapeated as necessary. Some experts, then, might be provided with six levels of context, while others get just one. This would also be very interesting to investigate.

References

Baranes, A., & Oudeyer, P.-Y. (2009). R-IAC: Robust intrinsically motivated active learning. *Proceedings of the IEEE International Conference on Learning and Development*, 1–6.

Blank, D. S., Kumar, D., Meeden, L., & Yanco, H. (2006). The Pyro toolkit for AI and robotics. *AI Magazine*, *27*(1).

Clark, A. (1998). *Being there: Putting brain, body, and world together again.* Cambridge, MA: The MIT Press.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

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

Harnad, S. (2005). To cognize is to categorize: Cognition is categorization. In H. Cohen & C. Lefebvre (Eds.), *Handbook of categorization.* Amsterdam: Elsevier.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Lee, R., Walker, R., Meeden, L., & Marshall, J. (2009). Category-based intrinsic motivation. *Proceedings of the Ninth International Conference on Epigenetic Robotics*.

Lungarella, M., Metta, G., Pfeifer, R., & Sandini, G. (2003). Developmental robotics: A survey. *Connection Science*, *15*(4).

Oudeyer, P.-Y., Kaplan, F., & Hafner, V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, *11*(2).

Provost, J., Kuipers, B. J., & Miikkulainen, R. (2006). Developing navigation behavior through self-organizing distinctive state abstraction. *Connection Science*, *18*(2), 159–172.