The Multiple Roles of Anticipation in Developmental Robotics

Douglas S. Blank[1], Joshua M. Lewis[2], and James B. Marshall[2]

[1] Computer Science Department,
Bryn Mawr College, PA, USA
dblank@cs.brynmawr.edu
[2] Computer Science Program,
Pomona College, CA, USA
{jlewis, marshall}@cs.pomona.edu

Abstract

Anticipatory systems have been shown to be useful in discrete, symbolic systems. However, non-symbolic anticipatory systems are less well understood. In this paper, we explore the use of anticipation within the framework of connectionist networks to bootstrap from an innate behavior; to drive a reinforcement signal; and to provide feedback on the learnability of a task.

Developmental Robotics

Developmental robotics is an approach to artificial intelligence that focuses on the autonomous self-organization of a general-purpose control system. Rather than being programmed to solve a particular task, a robot in the developmental robotics paradigm is programmed to develop sophisticated, self-organized representations and self-motivated behaviors over time. Such a robot begins with nothing but a "seed" program consisting of a simple innate behavior, a motivational system, and a learning system. There is no *a priori* task to master, only a general goal to develop "mentally" and behaviorally.

Developmental robotics brings together several paradigms falling under a range of rubrics, including embodied cognition, biologically-based robotics, artificial animals (animats), reinforcement learning, evolutionary computation, and machine learning. However, developmental robotics is unique in its insistence on goal non-specificity for systems. Therefore, central issues in this field are those at the heart of artificial intelligence: What feature detectors and concepts should be built into the system? How does the system "decide" what to do next? What drives the system to "want" to do anything? Indeed, perceiving the world and deciding which actions to perform are two of the hardest problems of AI. However, developmental robotics advocates combining these two problems into one and letting a solution grow through the experience of the robot.

Many learning mechanisms that are based on optimizing a fitness measure are not useful in this domain because there is no task by which to measure performance. However, the idea of *anticipation* has been found to be useful in multiple ways in the learning and motivation processes. This paper explores three roles of anticipation in a connectionist developmental robotics context.

Roles of Anticipation

Recently, anticipatory learning systems have gained increasing attention in the field. Butz, Sigaud, and Gerard (2002) give an overview of several such systems. However, their summary is limited to those models that can be framed in a Markovian paradigm. That is, anticipation is explored in a *partially observable Markov decision process* (POMDP). However, to have anticipation work in such a framework requires a system to have three properties. First, the Markovian assumption must hold: the next state of the system depends only on the current state. Therefore, the next state cannot be determined by context, unless context becomes part of the previous state. Second, perceptions and actions must be appropriately discretized. Third, these discrete representations cannot change over the course of learning. However, we view these representations of perceptions and actions as something that should be created by the system itself. How can the system's representations of its perceptions be based on the actions made by the system, while the representations of the actions are in turn based on the perceptions?

We believe that the answer to this chicken-and-egg dilemma is to start a system with a set of innate representations of perceptions and actions, and allow the system to gradually modify them over time. Thus, we believe that the space of representations should be continuous and mutable. This leads us in a different direction than, say, Witkowski's Dynamic Expectancy Model (2002). Rather than creating a "hypothesis engine" and a logic for determining when and how to create hypotheses, we have implemented our system as an artificial neural network. The main difference between such a Markovian system and a neural network

(connectionist) system is the form of the hypotheses and actions. As noted, the Markov process requires discrete symbols and actions, and therefore discrete hypotheses. However, artificial neural networks can utilize non-symbolic, distributed representations, and thus implement a continuum of hypotheses and actions. In this manner, distributed, connectionist representations allow many hypotheses to be active and tested at any one time. In addition, such a system also allows for a system to deal naturally with probabilistic and noisy environments.

Connectionist systems offer the ability to exploit information without having a full-fledged hypothesis. We have found that the question of what can be learned from a prediction task is much subtler in a connectionist network than in a POMDP, and potentially much more powerful. In the following sections we present three different methods that can be used in self-organizing anticipatory systems.

Anticipation for Bootstrap Learning

In a series of now-classic experiments, Elman (1990) showed that a connectionist network that was trained to predict the next word when given a random sentence could not, of course, guess the next word exactly. However, the prediction task did force the network to develop representations that reflected the syntax and semantics of the grammar of the sentences. His experiments show that no innate knowledge is needed in order to develop grammatical concepts such as noun and verb.

One can ask a similar question in robotics: How much innate information does a developmental robotics system need to know about its own sensors and actuators? Pierce and Kuipers (1997) demonstrated that a simulated robot armed with not much more than the ability to make correlations and predictions can learn most everything that it needs to know about itself, including its sensor types, sensor positions, and its degrees of freedom of movement.

Their system first gathered information from its sensors as the simulated robot made random movements in a room. At first, the robot does not know if a particular sensor reading is a camera pixel, a sonar reading, or something else entirely. However, after analyzing the qualitative properties of sensor readings over time, sensors could then be clustered into similar groups. For example, each sonar sensor varies over a similar range of values as the others, as do each camera pixel, laser sensor, joint reading, etc.

Each sensor in a cluster is then arbitrarily assigned a position in a low-dimensional (often 2D) space (Pierce and Kuipers 1997). These positions are adjusted so as to reflect associated pairwise distances between the sensor readings in a given cluster, in a manner similar to learning in a self-organizing map (Kohonen, 2001).

Starting with no knowledge about its sensor types or sensor topology, such a system now has a foundation to take advantage of higher-order analysis, such as prediction. First, higher-order features (such as discontinuities, local minima, and local maxima) are proposed. Proposed features are evaluated based on stability, predictive power, and extensibility (Pierce and Kuipers, 1997). Evidence is collected for motor commands that cause sensor reading changes in predictable ways.

In such systems, anticipation and correlation are tools used to bootstrap a robot from having zero knowledge about itself, to being able to understand its body, sensor topology, and environment.

Anticipation as a Reinforcement Signal

As we have seen, anticipation can be used to create the foundations of basic concepts of self. We now examine a system that uses anticipation at a higher level. Marshall, Blank, and Meeden (2004) used anticipation as the basis for generating an internal reinforcement signal for training a simulated robot. In this system, a simple recurrent network (SRN) was used to generate motor actions to be performed by the robot, together with predictions of the robot's next sensory state. Both sensory states and motor actions were represented as continuous, distributed patterns of network activation values.

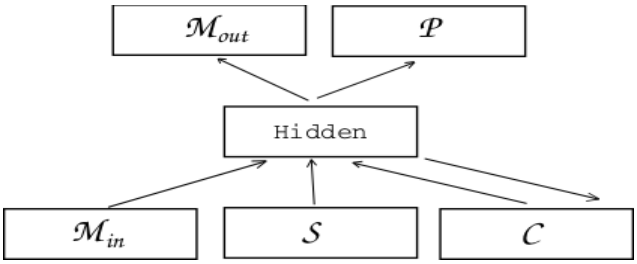

Figure 1. Network architecture for the two-stage error-centering robot controller.

The network architecture used is shown in Figure 1. The robot's predicted next sensory state, P, is determined by its current motor action M_{in}, its current sensory state S, and its past experience, represented by the recurrent context layer C of the network. The robot's next motor action to perform, M_{out}, is determined by S and C, with the M_{in} layer temporarily disabled. The robot's anticipatory subsystem comprises the M_{in}, S, C, and P portions of the network, while its control subsystem comprises the S, C, and M_{out} portions. The weights from the S and C banks of units to the hidden layer are shared between both subsystems. Training proceeds in an interleaved fashion, with the robot learning to anticipate while simultaneously learning to act in its environment.

Sensory states consisted of a two-dimensional spatial representation of the robot's visual field. On each training step, the robot's observed sensory state was compared to the state predicted by the network, and a spatial map of the prediction error was generated. The innate task of the robot was to learn to focus its attention on unanticipated sensory information by centering its visual field on the region of greatest error. On each time step, a positive or negative reinforcement signal was determined by whether the bulk of the prediction error moved toward or away

from the center of the visual field.

Figure 2 shows the simulated robot in its environment. The robot is at the center of an empty circular arena, with an extra "decoy" robot on the periphery (the straight lines indicate the robot's visual field). The trained robot has only one degree of freedom: it is capable of rotating left or right, but is fixed at the center of the arena. The decoy roams around the arena's edge under control of a simple obstacle-avoiding behavior.

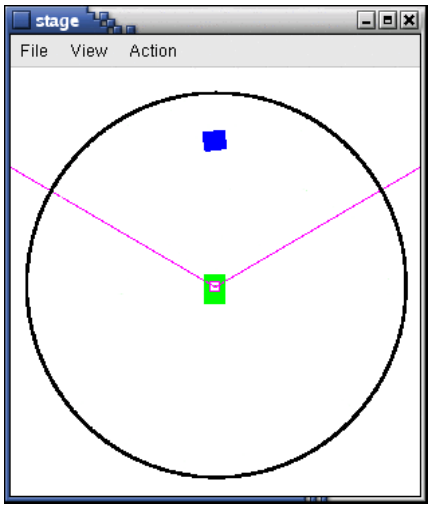

Figure 2. Environment for the error-centering robot.

The goal of the experiment was for the system to learn to predict the next state, and to center its view on the region of maximum error. These two tasks, however, cause interesting dynamics in the system. For example, as the system learns to make better predictions, the error decreases, and the system begins to look away. However, if a motor action causes an unanticipated prediction, then the system again attempts to center on the error.

Through the interaction of these two competing tasks, the robot first moves around randomly, then begins to "track" the decoy robot, and finally begins to anticipate the movements of the robot. In effect, the robot learns to track the decoy using only the self-generated reinforcement signal based on prediction error as feedback. The expectation is that as learning progresses, the robot will gradually "get bored" and continually search the environment for new sources of novel stimuli.

Anticipation as a Learning Accelerator

In the previous section we described a system that attempts to center on the prediction error while simultaneously learning to decrease error. One drawback of this approach is that it can become "fixated" on random patterns, such as white noise on a television screen (Oudeyer *et al*. 2005). In such a case, the robot would repeatedly attempt to focus on the noise without ever learning to predict it accurately. To explore this problem, we again consider how prediction can help. If the robot could predict that a particular situation was unlearnable, then it might break free of the situation.

Consider the following variation on the classic exclusive-or (XOR) problem. In this version, we have two XOR problems side by side. That is, there are four inputs and two outputs, for a total of 16 patterns. A standard backpropagation network (Rumelhart *et al*. 1986) can learn this double-XOR task. However, we now make the problem harder by embedding it within a larger set of "noisy" patterns. We do this by duplicating the input patterns and substituting randomly-generated training targets in place of the XOR targets for these inputs. These random targets are generated anew on every training step, instead of remaining static.

To distinguish noisy from non-noisy patterns, we add a 2-bit "flag" to each input. The 2-bit flag defines four possible categories, allowing the proportion of unpredictable patterns in the dataset to be varied between 0%, 25%, 50%, and 75%. For example, a dataset with 75% noise would consist of the 16 patterns with flag bits 00 shown in Table 1, together with three similar sets of patterns with flag bits 01, 10, and 11 whose targets are randomly generated on each training step, for a total of 64 patterns.

Input	***Output Target***
00 00 00	00
00 00 01	01
00 00 10	01
00 00 11	00
00 01 00	10
00 01 01	11
00 01 10	11
00 01 11	10
00 10 00	10
00 10 01	11
00 10 10	11
00 10 11	10
00 11 00	00
00 11 01	01
00 11 10	01
00 11 11	00

Table 1. Inputs and non-random targets for the Noisy XOR problem. The input consists of a 2-bit noise flag followed by 4 inputs to two XOR problems. The outputs are the exclusive-or of the respective pairs of non-flag inputs.

Although this is a harder problem with the addition of the noisy distractor patterns, a standard connectionist network can still learn to produce valid outputs for the patterns in Table 1. However, the performance of the

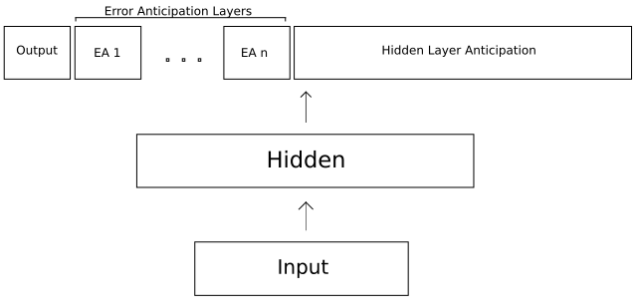

Figure 3: Network architecture.

network can be substantially improved using two learning strategies, which fall into the category of "hints."

Abu-Mostafa (1990) explored the idea of using "hints" in learning systems. He defined a hint to be any information that could be used to learn more effectively in a neural network. Abu-Mostafa examined two types of hints: specially crafted sets of input-output pairs, and additional constraints on the weights. However, both of these types of hints had to be carefully constructed by the designer. Following Abu-Mostafa, Suddarth and Holden (1991) developed the notion of a "catalytic hint." These hints formed an easier version of the main task, which was learned first. After the network had mastered this simplified task, it was then switched to the harder task. Suddarth and Holden showed that by using the catalytic hint technique, some problems could be learned more quickly. However, again the simplified task had to be carefully chosen by the designer.

We wish to expand on the idea of catalytic hints, by incorporating them into the developmental robotics paradigm. This requires, however, that they not be crafted by the modeler. Therefore, we will investigate a type of catalytic hint that is provided by the network itself. We call these "autocatalytic hints." We have developed two such techniques: Error Anticipation (EA) and Hidden Layer Anticipation (HLA).

Both strategies use a three-layer backpropagation network consisting of an input layer, a hidden layer, and an output layer split into several components. The first component of the output layer, labeled Output in Figure 3, attempts to produce the correct target patterns for each input pattern, as in a standard neural network.

The next components of the output layer, labeled EA 1 through EA *n* in Figure 3, perform Error Anticipation. EA involves training the network to anticipate the error of its own Output component. Before updating weights, the input activations are propagated through the network and the activation of the Output component is compared with its training target, resulting in an error vector. The target for the EA1 component of the output layer is that error vector. This process can be repeated multiple times. Another EA component can predict the error vector of the previous EA component (EA2 predicts the error of EA1), and so on. Our experiments used two EA components.

The final component of the output layer is the Hidden Layer Anticipation component. This component attempts to reproduce the activation of the hidden layer for the current input. During the same propagation step used for EA above, we record the activation of the hidden layer and set it as the target for the HLA component of the output layer. This effectively trains the network to predict its own internal representations of the input patterns.

On some trials we disable the effects of EA or HLA by using fixed target values of 0.5 for the EA or HLA components, without changing the structure of the network itself.

Figure 4a shows 30 training runs on a dataset with 75% unpredictable patterns: 10 runs with EA and HLA disabled (solid lines), 10 runs with EA enabled (dashed lines), and 10 runs with HLA enabled (dotted lines). Training epochs are plotted on the x-axis, and the Output component's total sum squared error (TSS) for the predictable patterns is plotted on the y-axis. Standard backpropagation is able to learn the predictable patterns even in the presence of random distractions, although not

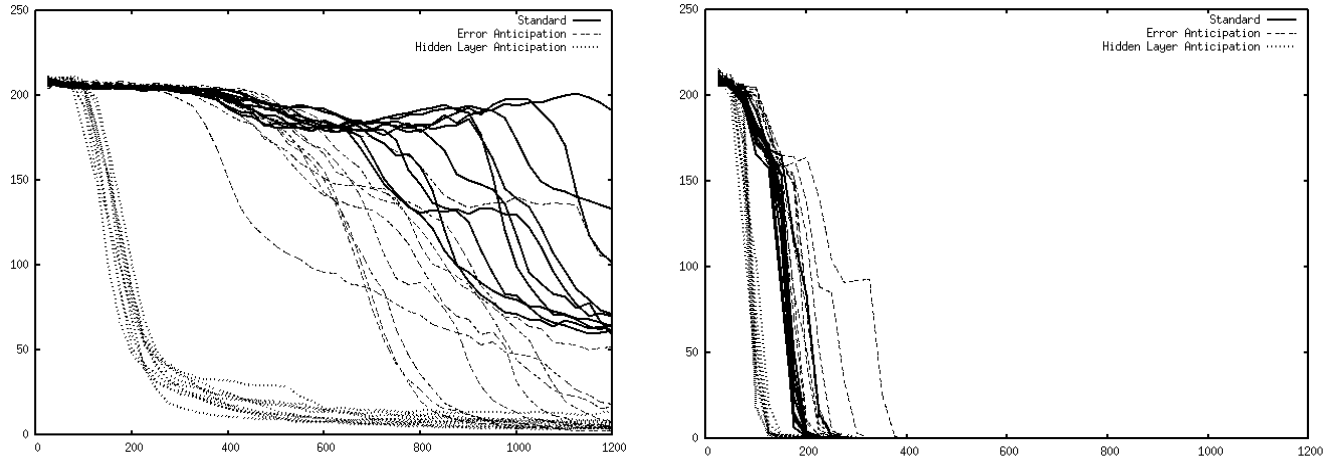

Figure 4. Plot of TSS error vs. training epoch for datasets containing (a) 75% noise and (b) 0% noise, showing 10 runs using standard backprop, 10 runs using EA, and 10 runs using HLA.

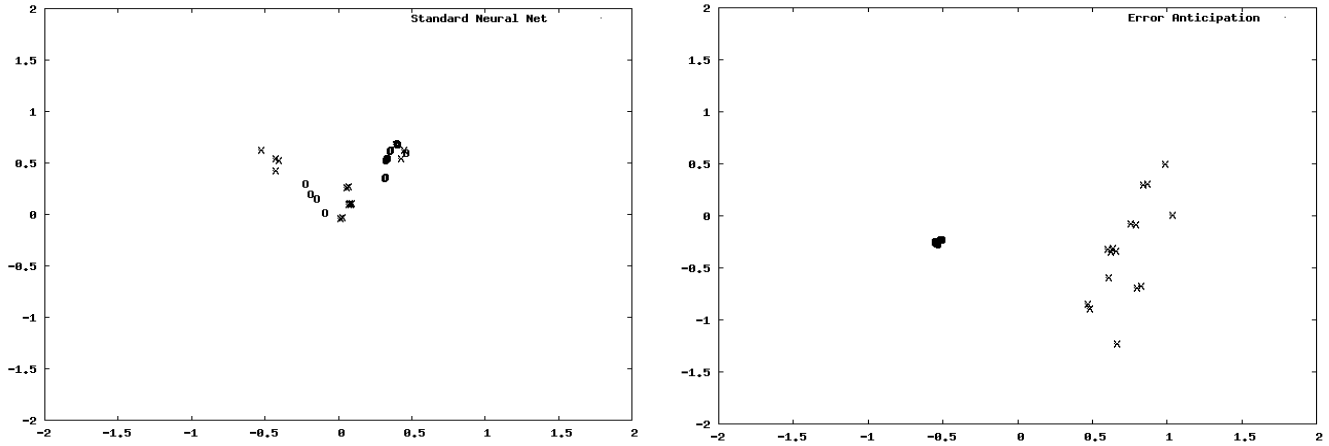

Figure 5. Plot of first two principal components of hidden representations developed using standard backprop (left) and EA (right), showing predictable inputs (X) and unpredictable inputs (O).

all of the runs have achieved a low TSS error by 1200 epochs. Turning on EA improves the speed and accuracy of learning. Eight of the ten EA runs attained a low TSS error by 1200 epochs. Using HLA instead of EA results in an even more dramatic improvement. With HLA, very low error is achieved on all runs by 600 epochs.

We ran the same experiment on datasets with 50%, 25%, and 0% unpredictable patterns. As the dataset becomes more deterministic, the effects of EA and HLA on learning become less pronounced, although HLA retains a slight edge over the other two methods. Figure 4b shows the results for a dataset containing no unpredictable patterns.

We believe that EA and HLA assist the learning process by causing the network to develop hidden representations that better distinguish the predictable inputs from the unpredictable inputs. To test this idea, we performed principal components analysis on the hidden representations developed by networks using EA and by networks with EA disabled. Figure 5 plots the first two principal components for an EA and a non-EA network after 1200 training epochs. The EA network does a much better job of segregating the predictable inputs (X) from the unpredictable inputs (O). However, representations created by HLA did not seem to achieve as clean a separation as in the case of EA.

HLA has a more pronounced impact on the speed of learning. However, it is not clear why HLA has such an impact. Our current hypothesis is that the effect of HLA varies over three stages of the learning process, as follows.

In the first stage, the majority of the network's output layer error comes from the Output component (see Figure 6). Since the weights from the input layer to the hidden layer are randomly initialized, as are the weights from the hidden layer to the HLA component, the activations of the HLA component and the hidden layer are already near 0.5 and thus there is not much error from this component. In addition, the majority of the error comes from the predictable input patterns—since their targets are either 0 or 1, whereas the unpredictable inputs have real-valued targets ranging from 0 to 1 and averaging 0.5. Thus the initial expected error from predictable inputs is 0.5 and from unpredictable inputs is 0.25. The predictable input samples account for the majority of learning from the input to hidden layer at this stage. Even if the unpredictable samples had discrete targets (resulting in an equivalent expected error), half of them would be "correct" and the "incorrect" samples would still have less influence. In fact, HLA worked similarly in our tests with discrete random targets.

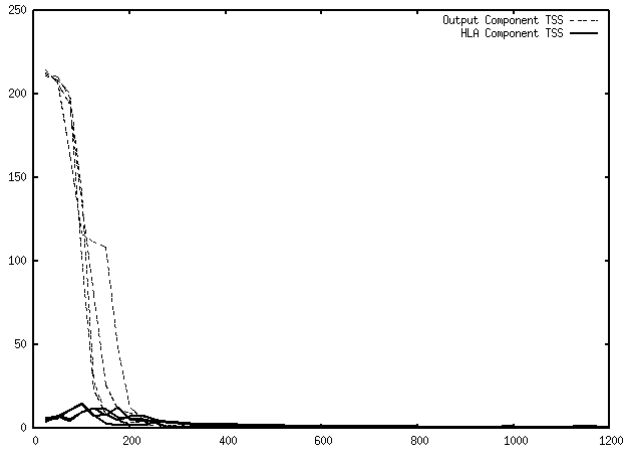

Figure 6. Plot of TSS error on the Output and HLA components of the output layer.

In the second stage, the TSS of the Output component plummets, and the TSS from the HLA component rises. This is the critical stage where the HLA strategy does much better than the other strategies. In this stage we believe that the HLA component is minimizing damaging weight changes between the input and hidden layers.

Recall that backpropagation works by adjusting the weights based on the amount of error on the output layer. Therefore, the network is continually changing the weights whenever it encounters unpredictable patterns.

Consider the first unit of both the hidden layer and the HLA component of the output layer. During the first stage of learning, the hidden layer unit will have become either activated (closer to 1) or inactivated (closer to 0) when presented with a certain input pattern. The HLA unit will learn to reproduce this, but will likely lag behind slightly, since it will always be learning before the weight changes from the Output component are applied for a particular input. For example, for some arbitrary input, the hidden unit might be activated at 0.8, and the HLA unit might be activated at 0.7. In this case (and the corresponding low activation case) the backpropagation step will cause the hidden layer unit to become even more activated the next time around, assuming the weight between the two units is positive. In this way, HLA speeds up the learning between the input and hidden layers —reinforcing the initial movements of the hidden layer activations. This learning rate increase is also observable in training sets with 0% unpredictable patterns—the HLA networks still perform somewhat better initially than their standard counterparts.

If we imagine that the training sample is random, however, and that the Output component target is the opposite of what it should be, then the Output component will likely propagate back error that would reduce the activation of the hidden unit. However, at the same time, the HLA component will propagate back error that would increase that same activation. This is the key assistance that HLA provides in the second stage. Once the network starts learning its problem partially, the discrepancy in expected error mentioned above begins to disappear, and predictable inputs no longer carry more weight. In this situation HLA helps the network "remember" the direction its learning was proceeding in. This memory may not always be beneficial. If the initial training samples presented to a network push it toward a local minimum in the error space, HLA will likely make it more difficult for the network to break away from that path and find the absolute minimum.

In the third stage, the TSS error from both the Output component and the HLA component is very small, but the HLA error is higher in part due to the greater number of units in the HLA component. Here, HLA is simply a distraction, and it prevents HLA networks from achieving the same low error that non-HLA networks can achieve. One possible improvement on HLA would be to turn it off once the Output error reaches some predetermined target value.

We performed several follow-up experiments to verify the impact of HLA on other noisy data sets.

First, we wondered whether we would see similar effects with datasets of randomly-generated input patterns. We created a new dataset consisting of 50 unique input patterns of size 10 paired with 50 target patterns (not necessarily unique) of size 5. All bits were determined by random "coin tosses". A few example patterns are shown below.

Input	*Output Target*
0 1 0 1 1 1 1 0 0 1	1 0 0 0 1
1 1 0 1 1 1 1 0 0 0	0 0 1 1 0
0 0 0 0 1 0 1 0 1 1	1 1 1 1 1
1 1 1 0 0 0 0 0 0 0	0 1 1 0 1
1 0 1 1 0 1 0 0 0 0	1 1 1 1 1
0 1 0 0 0 1 1 0 0 1	0 1 1 0 0
. . .	. . .

For each experiment, a portion of the input patterns were designated as unpredictable noise; for these inputs, new binary target patterns were generated on each training step as before, rather than using the original randomly-generated target pattern of the dataset. We tried several different levels of noise: 75%, 50%, 25%, and 0% of the dataset. Unlike in the XOR dataset, input patterns had no explicit flag bits to indicate whether or not an input was predictable, since all bits were randomly generated. Because HLA seems to have a greater impact on learning, we did not use EA in these experiments.

The network architecture consisted of 10 input units, 15 hidden units, 5 output units, and 15 hidden layer prediction units. Prediction could be disabled, as a control, in which case training targets of 0.5 were used for the prediction units instead of the hidden layer activations, as before.

We found no significant difference between the performance of standard backpropagation and HLA in this case. In fact, prediction seemed to actually hinder the learning process. One possible explanation is that this randomly-generated dataset may be too easy for the network to learn, and thus hidden layer prediction affords no clear advantage over regular backpropagation. With 10-bit patterns, the input space consists of $2^{10} = 1024$ possible patterns, but only 50 were included in the dataset. This gives the network ample freedom to "draw" separating hyperplanes between the predictable and unpredictable input patterns. In other words, it is relatively easy for the network to learn to distinguish the "signal" from the "noise" of this dataset, without relying on the extra machinery of hidden layer prediction. Given that conclusion, HLA performs much as we would expect. It slightly accelerates learning in the early epochs, but the noise created by HLA error in later epochs prevents the network from achieving the low TSS error of the standard approach.

In our final experiment, we created a new dataset consisting of 64 unique 6-bit input patterns covering the entire input space, paired with 2-bit target patterns. Since the input patterns cover the entire space, the problem

should be "difficult" and HLA should once again assist learning. This setup is very similar to the XOR training set above, except there are no flag bits and the targets are randomly generated (instead of being determined by XOR operations). As before, we varied the percentage of the input patterns designated as noise. For these experiments, we used 9 hidden units (1.5 times the number of input units, as before). The results for 75% noise are shown in Figure 7. The five HLA runs all have significantly lower TSS error than the five runs with HLA turned off. With this training set, HLA once again provides a healthy boost to learning.

In summary, we believe that HLA will assist learning for many difficult and noisy training sets. EA may also assist learning, but its effects tend to be more subtle.

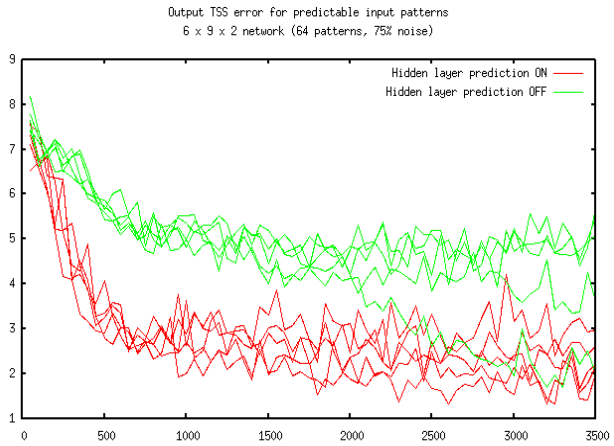

Figure 7. Plot of TSS error vs. training epoch for a 64-pattern dataset containing 75% noise, showing 5 runs with HLA on and 5 runs with HLA disabled.

Conclusion

This paper has explored the multiple uses of anticipation in developmental robotics, especially in systems that incorporate continuous, non-symbolic representations of states and actions. We have seen that anticipation can be a subtle but effective tool, useful for multiple purposes, and in ways very different from those uses in symbolic Markov Decision Processes. Yet, these experiments are only suggestive of the potential power that may lie behind connectionist anticipatory systems.

References

Abu-Mostafa, Y. (1990). Learning from hints in neural networks. *Journal of Complexity* 6, pp. 192-198.

Butz, M., Sigaud, O., and Gerard, P. (2002). Internal models and anticipations in adaptive systems. In *Proceedings of the Workshop on Adaptive Behavior in Anticipatory Learning Systems.* Seventh International Conference on Simulation and Adaptive Behavior, Edinburgh, Scotland.

Elman, J. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, 48:1, pp. 71-79.

Kohonen, T. (2001). *Self-Organizing Maps, Third edition*. Springer, Berlin.

Marshall, J., Blank, D., and Meeden, L. (2004). An emergent framework for self-motivation in developmental robotics. *Proceedings of the 3rd International Conference on Development and Learning*, pp. 104-111, Salk Institute for Biological Studies, La Jolla, CA.

Oudeyer, P.-Y., Kaplan, F., Hafner, V., and Whyte, A. (2005). The Playground experiment: Task-independent development of a curious robot. In *Developmental Robotics*, Blank, D. and Meeden, L. (eds.), AAAI Spring Symposium Technical Report, AAAI Press.

Pierce, D. M. and Kuipers, B. J. (1997). Map learning with uninterpreted sensors and effectors. *Artificial Intelligence* 92, pp. 169-227.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning in- ternal representations by error propagation. In McClelland, J. and Rumelhart, D., eds., *Parallel Distributed Processing: Explor-ations in the Microstructure of Cognition, Volume I*, pp. 318-362. MIT Press, Cambridge, MA.

Suddarth, S. and Holden, A. (1991). Symbolic-neural systems and the use of hints for developing complex systems. *International Journal of Man-Machine Studies*, 35(3). pp. 291-311.

Witkowski, M. (2002). Anticipatory learning: The animat as discovery engine. In *Proceedings of the Workshop on Adaptive Behavior in Anticipatory Learning Systems*, Butz, M, Sigaud, O., and Gerard, P. (eds). Seventh International Conference on Simulation and Adaptive Behavior, Edinburgh, Scotland.