Fully Autonomous Real-Time Autoencoder-Augmented Hebbian Learning through the Collection of Novel Experiences

Joshua A. Bowren[1], Justin K. Pugh[1], and Kenneth O. Stanley[1]

[1]Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA
jbowren@cs.ucf.edu, jpugh@cs.ucf.edu, kstanley@cs.ucf.edu

Abstract

Hebbian plasticity in artificial neural networks is compelling for both its simplicity and biological plausibility. Changing the weight of a connection based only on the activations of the neurons it connects is straightforward and effective in combination with neuromodulation for reinforcing good behaviors. However, a major obstacle to any ambitious application of Hebbian plasticity is that the performance of a layer of Hebbian neurons is highly sensitive to the choice of inputs. If the inputs do not represent precisely the features of the environment that Hebbian connections must learn to correlate to actions, the network will struggle to learn at all. A recently-proposed solution to this problem is the *Real-time Autoencoder-Augmented Hebbian Network* (RAAHN), which inserts an autoencoder between the inputs and the Hebbian layer. This autoencoder then learns in real time to encode the raw inputs into higher-level features while the Hebbian connections in turn learn to correlate these higher-level features to correct actions. Until now, RAAHN has only been demonstrated to work when it is driven by an autopilot during training (in a robot navigation task), which means its experiences are carefully controlled. Progressing significantly beyond this early demonstration, the present investigation now shows how RAAHN can learn to navigate from scratch entirely on its own, without an autopilot. By removing the need for an autopilot, RAAHN becomes a powerful new Hebbian-centered approach to learning from sparse reinforcement with broad potential applications.

Introduction

As a key mechanism behind adaptation in natural organisms, neural plasticity has attracted significant interest in artificial life (alife) (Floreano and Urzelai, 2000; Niv et al., 2002; Soltoggio et al., 2008, 2007; Soltoggio and Jones, 2009; Soltoggio and Stanley, 2012; Risi et al., 2011; Risi and Stanley, 2012; Stanley et al., 2003; Coleman and Blair, 2012). A popular option for studying neural plasticity in artificial neural networks (ANNs) is Hebbian learning, which follows the simple mechanism of increasing connection weights proportionally to the activation strengths of the neurons they connect (Hebb, 1949). For example, researchers often incorporate Hebbian learning into evolutionary algorithms that evolve ANNs to control agents in dynamic or uncertain environments (Floreano and Urzelai, 2000; Soltoggio et al., 2008; Risi et al., 2011). Sometimes such Hebbian networks are accompanied by *neuromodulation* (Soltoggio et al., 2008, 2007; Soltoggio and Jones, 2009; Soltoggio and Stanley, 2012; Risi and Stanley, 2012; Coleman and Blair, 2012), which allows a reward or penalty signal to turn on or off the plasticity of Hebbian connections appropriately. However, a major obstacle to the success of Hebbian ANNs is that the inputs to Hebbian layers must be carefully selected to encompass the right incoming environmental features or the proper correlations will otherwise become too difficult to learn. In domains in which the right features may not be known a priori, or where it may even be necessary to learn them from raw inputs through experience, Hebbian learning thereby becomes brittle or even prohibitive.

Responding to this challenge, Pugh et al. (2014) proposed recently that it might be possible to bridge the gap between the raw inputs to an ANN and a Hebbian layer through an *autoencoder*, which is itself an ANN with at least one hidden layer that is trained to reconstruct its inputs (Bengio, 2009). The value of the hidden layer in the autoencoder is that it typically comes to represent higher-level features of the environment (because they then aid in the reconstruction of the inputs). These higher-level features distilled from raw inputs could be just the features needed by a Hebbian layer to learn correlations between environmental features and appropriate agent actions. The idea behind Pugh et al. (2014) is that in principle both an autoencoder layer and a neuromodulated Hebbian layer can be learned simultaneously, in real time, which would allow an agent to construct a higher-level representation of its environment at the same time as it learns to navigate based on that developing representation. Pugh et al. (2014) call this hybrid combination of autoencoder and Hebbian layers a *Real-time Autoencoder-Augmented Hebbian Network* (RAAHN).

To validate RAAHN, Pugh et al. (2014) showed that it can learn key features of a two-dimensional maze domain at the same time as learning to navigate the domain in real time. Furthermore, a pure Hebbian learner could not effectively learn the same policy, thereby confirming the advantage of Hebbian learning from the autoencoder layer. However, a

major limitation of this demonstration is that the agent was guided during learning by an *autopilot* that ensured that the agent experienced a prescripted succession of inputs identified by the experimenters as appropriate to the task. In this way, the autopilot phase resembles supervised learning more than the kind of autonomous exploratory learning one might hope to see in alife. Ideally, the agent would explore on its own, accumulating higher-level features as it goes, and improving its ability to navigate based on those features at the same time.

The aim of this paper is to take that next step, demonstrating that RAAHN is indeed sufficiently capable of doing all the learning on its own, without an autopilot to guide it. Such a result would open up a broad range of possible experiments and applications, where agents can be released into a world to explore and learn without explicit guidance, more in the spirit of reinforcement learning (RL) (Watkins and Dayan, 1992; Rummery and Niranjan, 1994) than supervised learning. To enable this capability, RAAHN is slightly elaborated through a new kind of novelty-based history buffer (which decides from which experiences it is trained) and neural noise to encourage autonomous exploration. The result, demonstrated in a two-dimensional maze domain, is ultimately that RAAHN can learn effectively on its own, and furthermore that RAAHN can learn even when the provided sensors are insufficient for neuromodulated Hebbian learning.

With RAAHN's ability to learn control policies and higher-level features in real time established, RAAHN can begin to be employed in more sophisticated unguided scenarios. RAAHN also goes beyond traditional RL algorithms (Watkins and Dayan, 1992; Rummery and Niranjan, 1994) because it has the potential to learn increasingly high-level features through stacking autoencoders (Le et al., 2012) in the future. Such autoencoder stacks and entire RAAHN architectures potentially can even be evolved in the future through neuroevolution (Stanley and Miikkulainen, 2002; Floreano et al., 2008; Yao, 1999). As a first step towards such ends, this study accordingly establishes best practices for successfully running RAAHN without the need for an autopilot.

Background

Before previewing the original work on RAAHN, this section begins with a review of Hebbian learning and autoencoders, which are the two core components of RAAHN.

Hebbian Learning

Basic Hebbian learning is implemented in ANNs with the simple learning rule

$$\Delta w_i = \eta x_i y, \tag{1}$$

where w_i is the weight connecting two neurons with activations x_i and y, and η is the learning rate. This learning rule has the advantage of being completely local, making its application flexible. It is also biologically motivated, reflecting basic principles of neural plasticity.

Researchers interested in evolving ANNs in particular took interest in the Hebbian rule as a means to allowing evolved ANNs to exhibit plasticity during their lifetime (Floreano and Urzelai, 2000; Niv et al., 2002; Risi and Stanley, 2010; Risi et al., 2011; Stanley et al., 2003). Furthermore, by adding *neuromodulation* to the basic Hebbian learning rule, Hebbian ANNs are able to be trained with rewards and penalties similar to reinforcement learning algorithms (Watkins and Dayan, 1992). Neuromodulation allows an experience to strengthen, weaken, or have no effect on learned Hebbian correlations by associating a *modulatory signal* with the experience. The modulatory signal can be calibrated to guide learning based on an agent's behavior within its environment. Interestingly, neuromodulation has been shown to elicit agent behavior reminiscent of operant conditioning in animals (Soltoggio and Stanley, 2012; Soltoggio et al., 2013). Researchers in neuroevolution and artificial life have also shown that evolutionary algorithms benefit from Hebbian learning combined with neuromodulation by allowing their discovered ANNs to learn from reward signals (Soltoggio et al., 2008, 2007; Soltoggio and Jones, 2009; Soltoggio and Stanley, 2012; Risi and Stanley, 2012; Coleman and Blair, 2012).

A major obstacle to building a general learning system around the Hebbian rule is that its success depends greatly upon receiving inputs that correspond to precisely the domain features necessary to learn the right correlations for the task (Field, 1994). RAAHN introduced the idea of placing an autoencoder, reviewed next, before the Hebbian layer so that such essential features can be learned from raw inputs without the need for human engineering.

Autoencoders

An autoencoder is an ANN that is trained to learn a feature representation of its inputs that is conceptually at a higher level. For example, edge detectors are a higher-level feature of images than raw pixels (Hinton and Salakhutdinov, 2006). The autoencoder achieves such representation by *encoding* its inputs in a hidden layer (the learned feature representation) that is then *decoded* by an output layer (of the same dimensionality as the input layer) representing the autoencoder's *reconstruction* of its inputs. The autoencoder is trained to minimize the error between its reconstruction and its inputs (Bengio et al., 2013). Rather than a different set of weights representing the encoder and the decoder, the same set of weights can compute both the encoding and reconstruction, which is called *tied weights* (Vincent, 2011).

Deep learning researchers attracted fresh interest in autoencoders by showing that they can be stacked into layers that learn increasingly high-level features (Le et al., 2012). That way, for example, raw inputs can be encoded into edge detectors, which can be encoded into increasingly high-level

concepts until face detectors arise. There are many ways to train autoencoders, and many heuristics to help them learn meaningful feature representations (Ranzato et al., 2006; Le et al., 2012), but for RAAHN the precise autoencoder implementation is not the key concern because in theory any autoencoder can be plugged into RAAHN, so as autoencoders improve, RAAHN also improves.

RAAHN

RAAHN was recently introduced by Pugh et al. (2014), who were motivated by the limitations of Hebbian learning to combine an autoencoder with the Hebbian layer so that Hebbian correlations could be learned from the features extracted by the autoencoder. The idea is that in theory both the features and the neuromodulated Hebbian correlations can be learned in real time, as an agent navigates its environment, thereby offering an appealing new approach to reinforcement-like learning. However, this original work relied on an initial *autopilot* phase for RAAHN to learn a feature set representative of the domain. The autopilot in effect ensures that the feature set learned by the autoencoder is reliable and consistent because the learner is forced to encounter a prescripted chronology of experiences. While this setup helps to demonstrate that RAAHN can learn in principle, it is in effect a form of supervised learning, which leaves open the question of whether RAAHN can really learn *on its own* in real time.

Approach

The key hypothesis driving this paper is that RAAHN can still perform well in real time even without an initial autopilot phase if the autoencoder component learns a good feature set. Most of the original RAAHN setup from Pugh et al. (2014) does not need to change, but there are several proposed implementation differences to support fully autonomous learning. The experiments in this paper will apply RAAHN to a similar two-dimensional agent navigation task to that in Pugh et al. (2014), where the agent learns on every simulation tick.

Autoencoder Component

The autoencoder implementation follows conventional practice (Hinton and Salakhutdinov, 2006; Bengio, 2009). In particular, it computes neural activations with tied weights trained with stochastic gradient descent and error backpropagation. Recall that the aim of an autoencoder is to reproduce its own inputs. That way, its hidden layer becomes an *encoding* of the input space that captures its essential underlying features. The forward activation A_j for a hidden neuron j with input neurons I is calculated with

$$A_j = \sigma\left(\sum_{i \in I}(A_i \cdot w_{i,j})\right), \tag{2}$$

where σ is the activation function (in this paper the logistic function), A_i is the activation of a given input neuron $i \in I$, and $w_{i,j}$ is the weight between input neuron i and hidden neuron j. After calculating the forward activations, the backward activation B_i (whereby the reconstruction is computed) for every input neuron is calculated with

$$B_i = \sigma\left(\sum_{j \in H}(A_j \cdot w_{i,j})\right), \tag{3}$$

where σ is still the logistic function, and A_j is the previously computed activation for a hidden neuron $j \in H$.

The backward activations thus serve as reconstructions of the original inputs. With these reconstructions, the reconstruction error E_i can be calculated for each input i:

$$E_i = A_i - B_i. \tag{4}$$

From these error values the delta δ_i is calculated for each input neuron i, which will help to compute backpropagated error:

$$\delta_i = E_i \cdot \sigma'(B_i), \tag{5}$$

where σ'_i is the derivative of the logistic function at the reconstruction B_i for the input neuron i. Now the delta for a given hidden neuron j can be calculated as

$$\delta_j = \left(\sum_{i \in I}(\delta_i \cdot w_{i,j})\right)\sigma'(A_j), \tag{6}$$

where δ_i is from equation 5, $w_{i,j}$ is from equation 2, and $\sigma'(A_j)$ is the derivative of the logistic function at the forward pass A_j for hidden neuron j.

With the original error deltas δ_i and backpropagated error deltas δ_j, the tied weights of the autoencoder component can be updated. The change in weight $\Delta w_{i,j}$ for the connection from input neuron i to hidden neuron j is then

$$\Delta w_{i,j} = \alpha\left(\delta_i A_j + \delta_j A_i\right), \tag{7}$$

where α is the learning rate (held constant in later experiments at 0.1), and the remaining variables as described above.

In RAAHN this autoencoder is trained in real time as the agent explores its world, raising the question of on what data it should be trained at any given tick. To address this question a *history buffer* saves n experiences (sets of rangefinder values) that the agent encounters in the domain. However, the method through which these n experiences are chosen turns out instrumental in facilitating the novel real-time exploratory autonomy of RAAHN investigated in this paper, as explained next.

History Buffer Management

A simple way to manage the history buffer is to save every experience as it is discovered and drop the oldest ones when the buffer is at capacity. This method is called **Queue-RAAHN** because the history buffer is in effect managed as

a queue. It gives RAAHN the ability to learn a feature set from past experiences in the domain, but it also has the significant downside that the history buffer may accumulate too much of a single kind of experience (e.g. crash experiences), eventually causing it to "forget" other important aspects of the domain. While this approach worked in research by Pugh et al. (2014) under the controlled environment of the autopilot, when RAAHN is allowed to run autonomously the chance of a succession of negative experiences like crashes is much higher.

To address this problem the history buffer can instead be managed by saving only the most *novel* experiences encountered during the agent's lifetime. That way, commonly repeated experiences (such as crashing into a wall), will not come to occupy the entire buffer. This new method is called **Novelty-RAAHN**. To determine the novelty of an experience the Euclidean distance is calculated between it and all the other experiences in the buffer. Inspired by the calculation of novelty in the novelty search algorithm (Lehman and Stanley, 2011), the experience is then assigned a *novelty score*, which is the sum of the 20 smallest such distances. The current experience is then added to the buffer only if its novelty score is greater than that of the least novel experience in the buffer, which it replaces. That way, the buffer fills with diverse rather than redundant experiences that give the autoencoder a representative sample of the entire domain.

Hebbian Component

The Hebbian component of RAAHN, which learns a controller based on the features from the autoencoder, trains with the *reconfigure and saturate* method described by Soltoggio and Stanley (2012). This method adds a modulatory signal (which can either be positive or negative) to the basic Hebbian learning rule, thus giving reward and penalty feedback to the otherwise-naive Hebbian component. The modulatory signal m influences the weights through the update equation

$$\Delta w_i = m\eta x_i y, \tag{8}$$

where η is the learning rate, and x_i and y are the activations of the two neurons connected by the weight w_i. As prescribed by reconfigure and saturate, noise is added to each output and weight delta to facilitate exploration (which is essential when there is no autopilot). Thus the noisy activation A_j for each output neuron is computed as

$$A_j = \sigma\left(\sum_{i\in I}(x_i w_{i,j})\right) + \xi_j, \tag{9}$$

where ξ_j is the neural noise associated with A_j. Noise is added to each weight delta with

$$\Delta w_i = m\eta x_i y + \xi_i, \tag{10}$$

where ξ_i is noise sampled from the same distribution as ξ_j in equation 9 (in this paper: a uniform distribution in the range $[-0.1, 0.1]$).

Hebbian and RAAHN Architectures

The experiments in this paper compare pure Hebbian learning and RAAHN in a two-dimensional agent navigation domain. The raw inputs for both pure Hebbian and RAAHN come from 11 simulated wall-sensing rangefinders. Both network architectures output a single value denoting the fraction of the maximal turn angle to steer the agent. The pure Hebbian architecture is simple in that it merely connects the raw inputs directly to the output with a single layer of Hebbian-trained connections (figure 1a). The RAAHN architecture includes two layers of weights and an intervening five-neuron hidden layer (figure 1b), where the first layer of weights is trained by the autoencoder learning rule and the second layer is Hebbian-trained. Therefore, RAAHN's Hebbian component essentially learns driving behavior from a set of five higher-level features extracted from the raw inputs by the autoencoder component. The number five, which preliminary experiments suggest is not sensitive to minor variation, is chosen to avoid learning only the identity function.

Experiments and Results

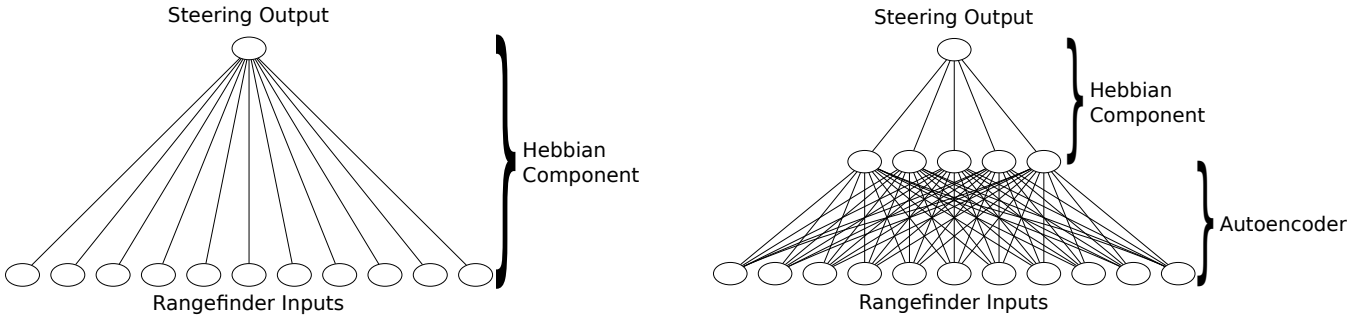
Recall that the hope in this paper is to advance beyond the previous finding that RAAHN can navigate a two-dimensional maze domain with an initial autopilot-driven training phase (Pugh et al., 2014). Although that study established that RAAHN can learn features as it learns to control, the more exciting potential of RAAHN is the ability to learn by itself, without an autopilot, in the spirit of real organisms.

By promising to learn *new* features at the same time as it learns a control policy even without any preliminary training, RAAHN adds a novel capability over and above what modulated Hebbian plasticity can offer. However, that new capability also raises the possibility that RAAHN might be overall more difficult to train without the help of the autopilot. For that reason, the experiments that follow establish first that RAAHN remains competitive with Hebbian on problems that Hebbian can solve. Once that is established, showing in effect that the new capabilities of RAAHN cost very little, the next logical step is an experiment that shows that on some problems (where feature learning is essential), RAAHN becomes critical to making effective learning possible.

Queue-RAAHN Experiment

To explore the potential of RAAHN to learn through its own exploration, a two-dimensional maze experiment similar to the one carried out by Pugh et al. (2014) is conducted without the initial autopilot. Instead, the agent is allowed to learn from its own decisions as it explores the world, as described in the Approach section.

Queue-RAAHN manages its history buffer as a queue. Every experience is saved in real time and the oldest experiences are deleted when the buffer is at capacity. Recall that the experiences in the buffer will periodically train the autoencoder in real time. This simple queue-based approach to storing



(a) Hebbian Architecture

(b) RAAHN Architecture

Figure 1: **Hebbian and RAAHN Architectures.** The Hebbian architecture (a) simply connects the 11 rangefinder inputs to the one output corresponding to the agent's steering. The RAAHN architecture (b) introduces an autoencoder layer between the rangefinder inputs and the steering output. The first layer of connections is trained as the autoencoder with tied weights, to learn (in the experiment) five high-level features from the rangefinder inputs. These features are then connected in RAAHN to the steering output with Hebbian-trained connections.

experiences gives the agent memory of previous experiences so it does not immediately forget them.

To test this approach (and others to be introduced shortly), the agent is given 11 rangefinders (depicted in figure 2) to detect how close it is to nearby walls. The rangefinders are separated from each other by an angle of 18 degrees. The two rangefinders at the edges of the range are 90 degrees from the middle rangefinder. Each rangefinder in this first experiment is 350 units long; the whole track is 3,140 units from left to right and 2,160 units from top to bottom. Rangefinders produce a minimum activation of 0.0 when they do not intersect walls, and a maximum activation of 1.0 when the entire rangefinder intersects a wall. Intermediate intersections produce an activation between 0.0 and 1.0.

Queue-RAAHN is compared to a single-layer neural network with Hebbian connections. The aim is to show that the autoencoder layers in RAAHN, which are lacking in the Hebbian network, do not diminish the ability of RAAHN to learn on its own in real time compared to the Hebbian layer alone. The agents controlled by Queue-RAAHN and the Hebbian neural network are referred to as **Queue-RAAHN** and **Hebbian**, respectively. Both neural network topologies take the 11 inputs and produce one output denoting the turning angle, with a range of [-2.0, 2.0]. A turn angle output of 2.0 changes the agent's direction by 2.0 degrees for the given tick. The neural network topology of Queue-RAAHN includes an autoencoder with a hidden layer of five neurons to learn features from the 11 rangefinder inputs, as depicted in figure 1b. The history buffer size for Queue-RAAHN is 500. For both methods Hebbian training occurs once every tick based on only the most recent experience. In Queue-RAAHN the autoencoder component also trains on 20 randomly-selected experiences from the history buffer (which for Queue-RAAHN is of course managed as a queue). The learning rates for autoencoder and Hebbian training (for the Hebbian layer of RAAHN and the pure Hebbian network) are held constant at 0.1 and 1.0 respectively. Both the single-layer Hebbian network and the Hebbian component of Queue-RAAHN receive positive modulation for turning away from walls, and negative modulation for turning towards walls, in the range of [-1.0, 1.0], where the magnitude of modulation is proportional to the magnitude of the turn. The wall chosen for this calculation is the closest wall colliding with an imaginary line projecting from the center of the agent 400 units in the direction the agent is facing. If the imaginary line does not intersect any wall then modulation is zero.

The simulation is run 200 times for both methods, each time for 10,000 ticks (i.e. simulation state updates). Agents that fail to complete at least one lap are *complete failures*. Agents that complete more than one lap but fewer than four

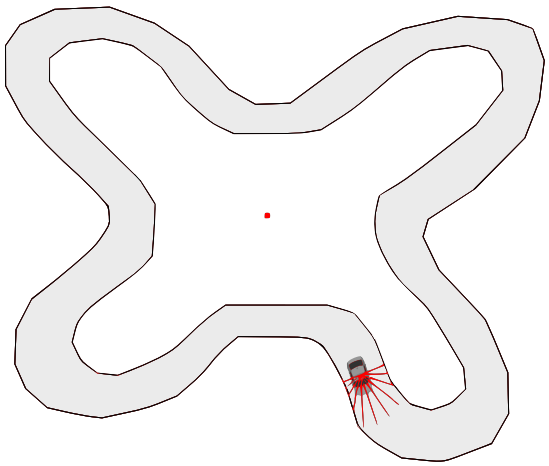


Figure 2: **X-Shaped Domain.** The track is not uniform to ensure the agent does not simply just repeat one behavior several times. The red dot in the center determines agent performance; every time the agent completes a circle around the dot, it completes one lap.

laps are *partial failures* because their performance is less than half that of correctly-driving agents.

Hebbian on average completes 9.2 laps and exhibits no complete failures nor partial failures. However, Queue-RAAHN on average completes 7.8 laps (which is significantly less at $p < 0.01$; Student's t-test) and yields 14% complete failures and a further 1% partial failures. The failure of Queue-RAAHN to approximate the performance of the plain Hebbian network suggests that the queue-based learning approach does not enable RAAHN to approach optimal performance.

Observing runs visually in real time reveals that after colliding with a wall, Queue-RAAHN agents either escape after a few hundred ticks or else become stuck for the duration of the run. Interestingly, an analysis of Queue-RAAHN neural networks suggests that this behavior results from poor driving compounded with poor representation. That is, sometimes agents drive poorly initially, which leads them to fill their history buffer with only crash experiences. This misadventure then leads to learning a poor feature set also representative only of crash experiences. The poor feature set then makes it difficult for such agents to escape their perpetual crashing and learn the types of better driving behaviors employed by agents who complete more laps.

Thus one hypothesis is that the main problem with Queue-RAAHN is that the autoencoder is unable to learn a feature set that is representative of the entire domain. If this hypothesis is true then if the feature set is constrained to be *novel* and therefore representative of the entire domain, RAAHN should perform about as well as Hebbian, which is tested next.

Novelty-RAAHN Experiment

By constraining the history buffer of RAAHN to contain only the most novel experiences, the data set on which RAAHN trains can become representative of the entire domain. For example, when the agent crashes into a wall, because crash experiences tend to be similar, they are not able to flood the buffer the way they do in Queue-RAAHN. By thereby avoiding a buffer with only one or few kinds of experience, the history buffer accumulates experiences of the entire domain as the agent explores it. This experiment uses the same parameters as the previous experiment aside from the difference in history buffer management. The novelty constraint on the history buffer still allows the simulation to run in real-time, as can be observed in the source available at:

http://eplex.cs.ucf.edu/uncategorised/software

The simulation is run 200 times for 10,000 ticks each. On average Novelty-RAAHN completes 8.8 laps, which is significantly above the 7.8 laps achieved by Queue-RAAHN ($p < 0.05$; Student's t-test). While the 8.8 laps of Novelty-RAAHN is still significantly below the 9.2 of Hebbian alone ($p < 0.01$; Student's t-test), this difference is small (less than one lap), and moreover *some* small disparity is essential in practice because Novelty-RAAHN must consume some extra time at the beginning of each run acquiring a novel set of experiences. Novelty-RAAHN also suffers only 0.5% complete failures and no partial failures. Thus it is likely close to performing as well as possible for a method that learns both features and policy at the same time. In conclusion, in this task in which the Hebbian network is provided a good input representation, RAAHN can learn in real time to approximate the same performance all while learning its feature representation in real time as well.

Increased Rangefinder Length Experiment

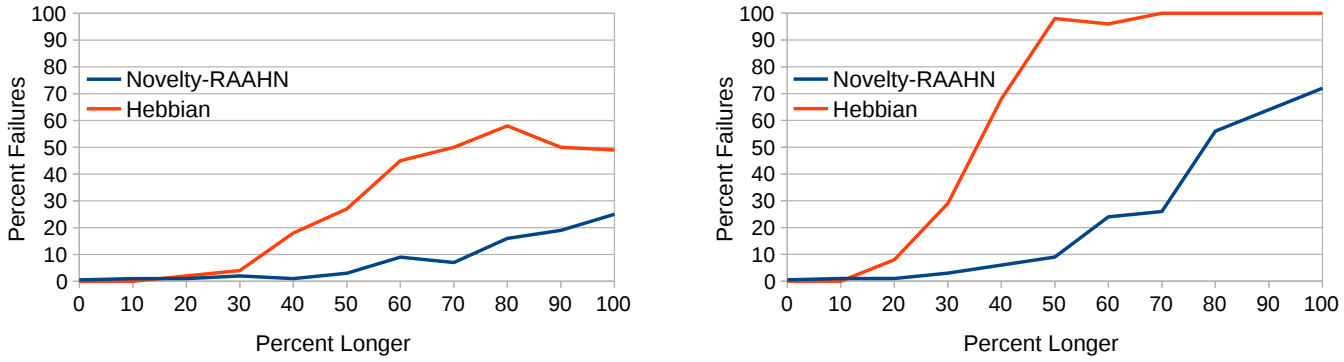
While it is important to establish that RAAHN can learn a representation in real time competitive with Hebbian alone, RAAHN's real promise is to learn *better* feature representations that overcome the limitations of the raw sensors. One way to investigate this idea is to *degrade* the quality of the rangefinders by increasing their length. Such an increase makes distinguishing different situations more difficult by forcing more sensory input into the agent as the sensors intersect walls more frequently and with greater activation. In theory RAAHN can overcome this challenge to some extent because it learns a new representation from the sensory input. However, Hebbian is forced to learn from degraded inputs.

To investigate whether RAAHN can indeed gain an advantage by learning a new feature representation, this experiment tests Hebbian and Novelty-RAAHN over ten variations of rangefinder lengths. These variants range from 10% longer to 100% longer with an interval of 10% between each variant.

The simulation is run for each variant 100 times, each for 10,000 ticks. As the rangefinder length increases both Hebbian and Novelty-RAAHN experience significantly more failures. However, Novelty-RAAHN indeed exhibits far fewer failures (figure 3). Hebbian begins to experience dozens of failures as early as 30%, 40%, and 50% longer while Novelty-RAAHN experiences fewer than four complete failures and fewer than seven partial failures at the same lengths. In addition, the number of laps completed by Novelty-RAAHN is significantly greater from 30% onward ($p < 0.05$). Novelty-RAAHN is only affected eventually by dramatically increasing the rangefinder lengths to values that provide little discernible information. Thus Novelty-RAAHN is significantly less sensitive to the precise sensory setup than Hebbian.

Discussion and Future Work

The experimental results establish for the first time that RAAHN is able to learn effective maze navigation behavior without the need for an autopilot. This achievement opens up a wide range of application domains because it means RAAHN does not need knowledge of the problem domain a priori. Interestingly, the implication is also that RAAHN can now be applied to conventional RL problems where training data is not labeled because now RAAHN only needs a modulation scheme to learn Hebbian correlations. Of course,



(a) Complete Failures

(b) Complete + Partial Failures

Figure 3: **Complete and Partial Failures (lower is better).** The number of complete failures (a) and complete plus partial failures (b) is shown for Hebbian and Novelty-RAAHN for rangefinder length increments between 0% and 100%. In both cases, the fact that Novelty-RAAHN completes significantly more laps in all cases from 30% longer onward ($p < 0.01$ for all 30% and above) is reflected in the higher number of runs that fail for Hebbian.

because RAAHN is significantly different from conventional RL algorithms such as Q-learning (Watkins and Dayan, 1992) or SARSA (Rummery and Niranjan, 1994) its strengths and weaknesses are likely different as well, but the opportunity for an entirely new path of research in this direction offers the potential for new discoveries and insights that would not emerge from conventional RL.

For example, already in this paper we begin to glimpse some principles behind learning a useful feature set in real time. In particular, it is likely that Queue-RAAHN performs significantly worse than Novelty-RAAHN because it has a tendency to "forget" salient aspects of the domain if they are not constantly revisited, as is the case when an agent crashes into a wall for an extended period of time: eventually its buffer becomes filled entirely with crash experiences that offer little utility beyond the scenario of a crash. Novelty-RAAHN avoids this trouble by only forgetting experiences that are redundant, instead attempting to maintain a set of experiences representative of the entire domain. The success of the novelty-driven buffer in Novelty-RAAHN thus hints at the importance of gathering and retaining experience in a principled manner.

Advances in autoencoders, which provide RAAHN the ability to represent higher-level concepts related to its domain, also lead to possible enhancements to RAAHN as well. In this way, RAAHN's potential reaches beyond simply calibrating sensitivity to a range of input parameters. Rather, as the domains in which RAAHN is applied become more complex, so does the possibility for more interesting feature sets. RAAHN can potentially stack several autoencoders (Hinton and Salakhutdinov, 2006) to learn high-level features from the complex data of vision, audition, or any other sensory modalities. The correlations learned by the Hebbian component can then serve to respond to those features in real time. It may also be possible to adapt RAAHN architectures through neuroevolution (Stanley and Miikkulainen, 2002; Floreano et al., 2008; Yao, 1999). If the topology of a RAAHN architecture can change over evolution, the process of finding an effective architecture (including autoencoder stacks) could be automated.

Furthermore, by pairing Hebbian learning with an autoencoder, much more becomes possible through Hebbian modulation than would be possible in a simple Hebbian network alone (without an autoencoder), thereby breathing new life into research focused on Hebbian learning. Supporting this view, when rangefinder lengths are extended beyond what is optimal, RAAHN exhibits significantly fewer failures than Hebbian alone. The performance of Hebbian alone degrades quickly without the autoencoder because Hebbian learns correlations best with sparse input activations (Olshausen and Field, 2004). With very long rangefinders, most of the inputs are highly active at any given time, so Hebbian cannot learn meaningful correlations. RAAHN's performance does not degrade as quickly because its autoencoder transforms the highly active inputs into more distributed features better-suited for the Hebbian component to learn effective driving behavior. *Sparse autoencoders* (Le et al., 2012) might help to limit such degradation even further.

As an example of how recent work in Hebbian learning can enhance RAAHN, new ideas on augmenting Hebbian connections to react to distal rewards (Soltoggio, 2015) (i.e. rewards from far away in time) can potentially shift RAAHN from its current limited temporal context to learning long-term causal dependencies. Recurrent connections might further allow learning to react to experiences from the past. These possibilities in effect draw on advances in Hebbian learning in general, and provide fuel for further research into improving Hebbian learning.

Finally, as a novel approach to RL, RAAHN aligns naturally with research in alife because agent behavior is shaped

in RAAHN through low-level neuromodulation as opposed to high-level value-function approximation (as in Q-learning and SARSA), making its analogy to low-level biological processes (in particular Hebbian plasticity) more accessible and open to study. Future work will focus on more complex domains that require for example behaving in location-dependent contexts by learning in real time the identifying features of different locations.

Conclusion

Moving beyond the original demonstration of RAAHN (Pugh et al., 2014) that depended on an initial autopilot phase, this paper showed how RAAHN can explore and learn on its own without an autopilot phase. By maintaining a buffer of experiences representative of the domain, such real-time learning becomes realistic. This new capability was demonstrated in a robot control domain where RAAHN learned to steer a robot through hallways on its own from scratch. The benefit of the architecture was further demonstrated by showing how much less its performance degrades compared to a simple Hebbian network when the sensory inputs become less optimal. The long-term implication is that RAAHN is a new sandbox and a new model for experimenting with modulation and reinforcement learning, which in the future can benefit from advances in both Hebbian learning and autoencoders alike.

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 1798–1928.

Coleman, O. J. and Blair, A. D. (2012). Evolving plastic neural networks for online learning: review and future directions. In *AI 2012: Advances in Artificial Intelligence*, pages 326–337. Springer.

Field, D. J. (1994). What is the goal of sensory coding? *Neural Computation*, 6(4):559–601.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1:47–62.

Floreano, D. and Urzelai, J. (2000). Evolutionary robots with online self-organization and behavioral fitness. *Neural Networks*, 13:431–4434.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*.

Hinton, G. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A. (2012). Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning (ICML-2012)*.

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

Niv, Y., Joel, D., Meilijson, I., and Ruppin, E. (2002). Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*, 10(1):5–24.

Olshausen, B. A. and Field, D. J. (2004). Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487.

Pugh, J. K., Soltoggio, A., and Stanley, K. O. (2014). Real-time hebbian learning from autoencoder features for control tasks. In *Proceedings of the Fourteenth International Conference on Artificial Life (Alife XIV)*, Cambridge, MA. MIT Press.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2006). Efficient learning of sparse representations with an energy-based model. In et al., J. P., editor, *Advances in Neural Information Processing Systems (NIPS 2006)*, volume 19. MIT Press.

Risi, S., Hughes, C., and Stanley, K. (2011). Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491.

Risi, S. and Stanley, K. O. (2010). Indirectly encoding neural plasticity as a pattern of local rules. In *Proc. of the 11th Intl. Conf. on Simulation of Adaptive Behavior*, Berlin. Springer.

Risi, S. and Stanley, K. O. (2012). A unified approach to evolving plasticity and neural geometry. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN-2012)*, Piscataway, NJ. IEEE.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, UK.

Soltoggio, A. (2015). Short-term plasticity as cause–effect hypothesis testing in distal reward learning. *Biological Cybernetics*, 109(1):75–94.

Soltoggio, A., Bullinaria, A. J., Mattiussi, C., Drr, P., and Floreano, D. (2008). Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In Bullock, S., Noble, J., Watson, R., and Bedau, M., editors, *Proc. of the 11th Intl. Conf. on Artificial Life*, Cambridge, MA. MIT Press.

Soltoggio, A., Dürr, P., Mattiussi, C., and Floreano, D. (2007). Evolving neuromodulatory topologies for reinforcement learning-like problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2471–2478.

Soltoggio, A. and Jones, B. (2009). Novelty of behaviour as a basis for the neuro-evolution of operant reward learning. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 169–176, New York, NY, USA. ACM.

Soltoggio, A., Lemme, A., Reinhart, F., and Steil, J. J. (2013). Rare neural correlations implement robotic conditioning with delayed rewards and disturbances. *Frontiers in neurorobotics*, 7(6).

Soltoggio, A. and Stanley, K. O. (2012). From modulated hebbian plasticity to simple behavior learning through noise and weight saturation. *Neural Networks*, 34:28–41.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2003). Evolving adaptive neural networks with and without adaptive synapses. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Piscataway, NJ. IEEE.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.