Real-time hebbian learning from autoencoder features for control tasks

Real-time Hebbian Learning from Autoencoder Features for Control Tasks

To appear in: Proc. of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14). Cambridge, MA: MIT Press, 2014.

Justin K. Pugh[1], Andrea Soltoggio[2], and Kenneth O. Stanley[1]

[1]Dept. of EECS (Computer Science Division), University of Central Florida, Orlando, FL 32816 USA
[2]Computer Science Department, Loughborough University, Loughborough LE11 3TU, UK
jpugh@eecs.ucf.edu, a.soltoggio@lboro.ac.uk, kstanley@eecs.ucf.edu

Abstract

Neural plasticity and in particular Hebbian learning play an important role in many research areas related to artficial life. By allowing artificial neural networks (ANNs) to adjust their weights in real time, Hebbian ANNs can adapt over their lifetime. However, even as researchers improve and extend Hebbian learning, a fundamental limitation of such systems is that they learn correlations between preexisting static features and network outputs. A Hebbian ANN could in principle achieve significantly more if it could accumulate *new features* over its lifetime from which to learn correlations. Interestingly, autoencoders, which have recently gained prominence in deep learning, are themselves in effect a kind of *feature accumulator* that extract meaningful features from their inputs. The insight in this paper is that if an autoencoder is connected to a Hebbian learning layer, then the resulting Real-time Autoencoder-Augmented Hebbian Network (RAAHN) can actually learn new features (with the autoencoder) while simultaneously learning control policies from those new features (with the Hebbian layer) in real time as an agent experiences its environment. In this paper, the RAAHN is shown in a simulated robot maze navigation experiment to enable a controller to learn the perfect navigation strategy significantly more often than several Hebbian-based variant approaches that lack the autoencoder. In the long run, this approach opens up the intriguing possibility of real-time deep learning for control.

Introduction

As a medium for adaptation and learning, neural plasticity has long captivated artificial life and related fields (Baxter, 1992; Floreano and Urzelai, 2000; Niv et al., 2002; Soltoggio et al., 2008, 2007; Soltoggio and Jones, 2009; Soltoggio and Stanley, 2012; Risi and Stanley, 2010; Risi et al., 2011; Risi and Stanley, 2012; Stanley et al., 2003; Coleman and Blair, 2012). Much of this body of research focuses on Hebbian-inspired rules that change the weights of connections in proportion to the correlation of source and target neuron activations (Hebb, 1949). The simplicity of such Hebbian-inspired rules makes them easy and straightforward to integrate into larger systems and experiments, such as investigations into the evolution of plastic neural networks (Floreano and Urzelai, 2000; Soltoggio et al., 2008; Risi et al., 2011). Thus they have enabled inquiry into such diverse problems as task switching (Floreano and Urzelai, 2000), neuromodulation (Soltoggio et al., 2008), the evolution of memory (Risi et al., 2011), and reward-mediated learning (Soltoggio and Stanley, 2012).

However, while Hebbian rules naturally facilitate learning correlations between actions and static features of the world, their application in particular to control tasks that require learning *new features* in real time is more complicated. While some models in neural computation in fact do enable low-level feature learning by placing Hebbian neurons in large topographic maps with lateral inhibition (Bednar and Miikkulainen, 2003), such low-level cortical models generally require prohibitive computational resources to integrate into real-time control tasks or especially into evolutionary experiments that require evaluating numerous separate individuals. Thus there is a need for a convenient and reliable feature generator that can accumulate features from which Hebbian neurons combined with neuromodulation (Soltoggio et al., 2008) can learn behaviors in real time.

Interestingly, such a feature-generating system already exists and in fact has become quite popular through the rise of deep learning (Bengio et al., 2007; Hinton et al., 2006; Le et al., 2012; Marc'Aurelio et al., 2007): *the autoencoder* (Hinton and Zemel, 1994; Bourlard and Kamp, 1988). Autoencoders, which can be trained through a variety of algorithms from Restricted Boltzmann Machines (RBMs) (Hinton et al., 2006) to more conventional stochastic gradient descent (Le et al., 2012) (e.g. similar to backpropagation; Rumelhart et al. 1986), aim simply to output the same pattern as they input. By training them to mimic their inputs, they are forced to learn key features in their hidden layer that efficiently encode such inputs. In this way, they accumulate such key features as they are exposed to more inputs. However, in deep learning autoencoders are usually trained for classification tasks through an *unsupervised pre-training phase* and in fact recent results have raised doubts on their necessity for such tasks anyway (Cireşan et al., 2012). The idea in this paper is that in fact autoencoders can instead be put to good use as *feature accumulators* that work synergistically with neuromodulated Hebbian connections that learn from the accumulating features in real time, as an autonomous

agent acts in the world. The resulting structure is the *Real-time Autoencoder-Augmented Hebbian Network* (RAAHN), a novel algorithm for learning control policies through rewards and penalties in real time.

In short, the key idea behind the RAAHN is that a missing ingredient that can reinvigorate the field of Hebbian learning is the ability of an autonomous agent to learn new features as it experiences the world. Those new features are then simultaneously the inputs to a neuromodulated Hebbian layer that learns to control the agent based on the accumulating features. In effect, the RAAHN realizes the philosophy that real-time reward-modulated learning cannot achieve its full potential unless the agent is simultaneously and continually learning to reinterpret and re-categorize its world. Introducing the RAAHN thereby creates the opportunity to build and study such systems concretely.

The experiment in this paper is intended as a proof of concept designed to demonstrate that it is indeed possible to learn autoencoded (and hence unsupervised) features at the same time as Hebbian connections are dynamically adapting based both on correlations with the improving feature set and neuromodulatory reward signals. In the experiment, simulated robots with two kinds of sensors (one to see the walls and the other to see a non-uniform distribution of "crumbs" on the floor) are guided through a maze to show them the optimal path, after which they are released to navigate on their own. However, supervised learning does not take place in the conventional sense during this guided period. Instead, both the autoencoder and the Hebbian connections are adjusting in real time without supervisory feedback to the autoencoder. Furthermore, to isolate the advantage of the autoencoder, two other variants are attempted – in one the Hebbian connections learn instead from the raw inputs and in the other the Hebbian connections learn from a set of random features (e.g. somewhat like an extreme learning machine; Huang and Wang 2011).

The main result is that only the full RAAHN learns the optimal path through the maze in more than a trivial percentage of runs, showing not only that it is possible to train an autoencoder and Hebbian connections simultaneously, but that in fact the autoencoder component can be essential for incorporating the most important features of the world.

While the RAAHN could be viewed in the context of reinforcement learning (RL), it is important to note that the approach is rather aimed at issues outside typical RL. In particular, the RAAHN can be viewed as a platform for later integrating more advanced and realistic Hebbian learning regimes, e.g. with distal rewards (Soltoggio et al., 2013) to demonstrate more convincingly their full potential.

In effect, the RAAHN is a new way to think about plasticity that goes beyond the proof of concept in this paper. It is among the first methods to suggest the potential for *real-time deep learning for control*. In that way it opens up a large and rich research area for which this paper represents an initial step. If Hebbian connections can be trained from a dynamically adjusting autoencoder, then perhaps one day they will learn in real time from deepening stacked autoencoders or within complex evolved networks that incorporate autoencoders as a basic element. While much remains to be explored, the initial study here thereby hints at what might be possible in the future.

Background

This section reviews Hebbian learning in artificial neural networks (ANNs) and the application of autoencoders in deep learning.

Hebbian ANNs

The plain Hebbian plasticity rule is among the simplest for training ANNs:

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

where w_i is the weight of the connection between two neurons with activation levels x_i and y, and η is the learning rate. As an entirely local learning rule, it is appealing both for its simplicity and biological plausibility. For this reason, many researchers have sought to uncover the full extent of functionality attainable by ANNs only of Hebbian rules.

As researchers have gained insight into such networks, they have also found ways to increase the rule's sophistication by elaborating on its central theme of strengthening through correlated firing (e.g. Oja 1982; Bienenstock et al. 1982). Researchers also began to evolve such ANNs in the hope of achieving more brain-like functionalities by producing networks that change over their lifetime (Floreano and Urzelai, 2000; Niv et al., 2002; Risi and Stanley, 2010; Risi et al., 2011; Stanley et al., 2003).

One especially important ingredient for Hebbian ANNs is *neuromodulation*, which in effect allows Hebbian connections to respond to rewards and penalties. Neuromodulation enables the increase, decrease, or reversal of Hebbian plasticity according to feedback from the environment. Models augmented with neuromodulation have been shown to implement a variety of typical features of animal operant learning such as reinforcement of rewarding actions, extinction of unproductive actions, and behavior reversal (Soltoggio and Stanley, 2012; Soltoggio et al., 2013). The combination of Hebbian ANNs with neuromodulatory signals in recent years has especially inspired neuroevolution and artificial life researchers by opening up the possibility of evolving ANNs that can learn from a sequence of rewards over their lifetime (Soltoggio et al., 2008, 2007; Soltoggio and Jones, 2009; Soltoggio and Stanley, 2012; Risi and Stanley, 2012; Coleman and Blair, 2012).

However, one limitation of these cited studies is that the inputs are generally heavily pre-processed to provide meaningful and useful feature to the neural substrate that performs Hebbian plasticity. A natural question is whether such useful features can in principle emerge in real-time during the

agent's lifetime, and in combination with the associative, reward-driven learning provided by Hebbian plasticity. As this paper argues, the autoencoder, reviewed next, is an appealing candidate for playing such a role.

Autoencoders in Deep Learning

The idea behind the autoencoder is to train a network with at least one hidden layer to *reconstruct* its inputs. The autoencoder can be described as a function f that *encodes* a feature vector x (i.e. the inputs) as a set of hidden features $h = f(x)$. A second function g then *decodes* the hidden features h (typically a hidden layer within an ANN) into a *reconstruction* $r = g(h)$ (Bengio et al., 2013). The hope of course is that once trained, r will be as close as possible to x for any input x. While many possible autoencoder models exist, the parameters of the autoencoder (which can be represented as weights in an ANN) are often the same for the encoder and decoder, which means in effect that the weights are bidirectional (Bengio et al., 2013). The main property of the autoencoder that makes it interesting is that by forcing it to learn hidden features h that can reconstruct input instances, under the right circumstances the features of h are forced to encode key features of the input domain. For example, edge detectors might arise in h for encoding images (Hinton et al., 2006).

Autoencoders began to gain in popularity considerably after researchers observed that they can help to train deep networks (i.e. ANNs of many hidden layers) through a pre-training phase in which a stack of autoencoders is trained in sequence, each one from the previous (Bengio et al., 2007; Hinton et al., 2006; Le et al., 2012; Marc'Aurelio et al., 2007), leading to a hierarchy of increasingly high-level features. Because it was thought that backpropagation struggles to train networks of many layers directly, pre-training a stack of such autoencoders and then later completing training through e.g. backpropagation was seen as an important solution to training deep networks. Although later results have suggested that in fact such pre-training is not always needed (Cireşan et al., 2010) (especially in the presence of an abundance of labeled data), it remains a compelling demonstration of unsupervised feature accumulation and remains important for training in the absence of ample labeled data (Bengio et al., 2013). In any case, typically the main application of such deep networks is in classification problems like handwriting recognition.

Another appeal of autoencoders is that there are many ways to train them and many tricks to encourage them to produce meaningful features (Ranzato et al., 2006; Le et al., 2012). While RBMs (a kind of probabilistic model) can play a similar role to autoencoders, classic autoencoders in deep learning are generally trained through some form of stochastic gradient descent (Le et al., 2012; Bengio et al., 2007) (like backpropagation), as is the case in this paper. However, the important issue in the present investigation is not the particular details of the autoencoder; in fact an advantage of the RAAHN formulation is that any autoencoder can be plugged into the RAAHN. Thus as autoencoders are refined and improved, RAAHNs naturally benefit from such improvements and refinements. It is also possible that the real-time context of RAAHNs will provoke more attention in the future to identifying autoencoder formulations most suited to real time.

Approach: Real-time Autoencoder-Augmented Hebbian Network

The Real-time Autoencoder-Augmented Hebbian Network (RAAHN) approach introduced in this paper consists of two distinct components: the autoencoder and the Hebbian learner. The simplest implementation consists of an ANN with a single hidden layer; connections from the inputs to the hidden layer are trained as an autoencoder and connections from the hidden layer to the outputs are trained with a Hebbian rule. Thus the hidden layer represents a set of features extracted from the inputs that a Hebbian rule learns to correlate to the outputs to form an effective control policy.

Both of these components can be implemented in a number of different ways. The particular implementation described in this section, which is tested later in the experiment, serves as a proof of concept. It is designed accordingly to be as simple as possible.

Autoencoder Component

The autoencoder in this experiment is a heuristic approximation of the conventional autoencoder (Bengio et al., 2013) that was chosen for simplicity and ease of implementation. It is important to note that it suffices for the purpose of this experiment because it converges to within 5% of the optimal reconstruction in every run of the experiment in this paper. This consistent convergence validates that the simplified autoencoder effectively approximates the behavior of an ideal autoencoder without loss of generality. Of course, more sophisticated autoencoder implementations can fill the same role in future implementations of the RAAHN.

The simplified autoencoder component consists of a single layer of bidirectional weights that are trained to match the output of the backwards activation with the input to the forward activation. On each activation of the network, first the inputs I feed into the the regular forward activation of the hidden layer H (the input layer is fully connected to the hidden layer) in the following manner. For each hidden neuron j, forward activation A_j is calculated:

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

where σ is a sigmoid function, A_i is the value of input neuron $i \in I$, $w_{i,j}$ is the weight of the connection between neurons i and j, and b_j is the bias on hidden neuron j. Next, the

forward activation values for hidden layer H are used to calculate the *backwards activation* to input layer I. For each input neuron i, backward activation B_i is calculated:

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

where σ is the same sigmoid activation function as in equation 2, A_j is the forward activation on hidden neuron $j \in H$, and b_i is the bias on input neuron i.

After backwards activation is calculated, for each input neuron i, an error E_i is calculated:

$$E_i = A_i - B_i. \quad (4)$$

Finally, as a simple heuristic for reducing reconstruction error (modeled after the perceptron learning rule), each weight is adjusted according to

$$\Delta w_{i,j} = \alpha E_i A_j, \quad (5)$$

where α is the learning rate (which is set to a small value to prevent convergence before an adequate number of input samples have been seen). This autoencoder was validated on data from the experiment to ensure that it converges with very low error (less than 5% from the optimal reconstruction). It is important again to note that any autoencoder could substitute for this simple model, whose particular mechanics are not essential to the overall RAAHN.

The experiment in this paper applies the proposed RAAHN system to a simulated robot control task. On each simulated time tick, the agent perceives values on its sensors and experiences a network activation. Thus, each time tick constitutes one training sample for the purpose of training the autoencoder connections. In this paper, a batch-style training system is implemented in which training samples are added to a history buffer of size n and autoencoder training is applied several times on the entire history buffer every $\frac{n}{2}$ ticks. Batch-style training is selected because many popular autoencoder training methods such as L-BFGS require batch training, although in preliminary experiments the system was found to perform well with both large and small values of n.

Hebbian Component

In the RAAHN system, connections between learned features and the outputs are trained with a modulated Hebbian learning rule, which is similar to the simple Hebbian rule (equation 1) with an added term to allow for the influence of reward and penalty signals. In this way, connections are only strengthened when a reward signal is received and when a penalty signal is received, the rule switches to anti-Hebbian (which serves to weaken connections). The modulated Hebbian rule is

$$\Delta w_i = m\eta x_i y, \quad (6)$$

where m is the modulation associated with the training sample. The Hebbian rule without modulation is like assuming that all training samples are positive; modulation allows training samples to be marked with varying degrees of positive or negative signal, which is a more flexible learning regime. The details of the modulation scheme have a significant impact on the effectiveness of learning. In the maze-navigating experiment in this paper, a simple modulation scheme is selected in which modulation is positive when the robot turns away from a wall, negative when the robot turns towards a wall, and neutral ($m = 0$, corresponding to no learning) when there is no change. Specifically, the modulation component in this paper is calculated as the difference between the front-pointing rangefinder sensor activation on the previous tick and on the current tick, normalized to the range -1 to 1.

Modulated Hebbian learning following equation 6 is applied to every connection of the Hebbian component of the RAAHN system on each tick. The system in essence learns correlations between the developing feature set discovered by the autoencoder component and an output pattern required for effective control. The Hebbian rule is useful as a learning mechanism to connect autoencoder features to outputs because it is invariant to starting conditions. Thus, if the pattern of features in the learned feature set changes for some reason (e.g. the nature of the task environment shifts significantly), the Hebbian component can simply learn new correlations for the new feature set, enabling calibration in real-time as the feature set itself is refined.

Experiment

To demonstrate the effectiveness of the proposed RAAHN system, a maze-navigating control task is introduced in which a robot agent must make as many laps around the track as possible in the allotted time. The challenges of the task are two-fold. The first and more trivial challenge is for the controller to avoid crashing into walls, which would prevent it from completing any laps at all (robots that crash into walls almost always remain stuck on the wall, preventing any further movement). The second challenge arises because there are multiple round-trip paths around the track (figure 1). In particular, the track consists of an optimal path with two attached "detours" – longer routes that lead the robot off the optimal path before re-joining it. There is one detour attached to the inner wall of the track as well as one detour attached to the outer wall of the track. Both detours take significantly longer to traverse than the optimal path; thus taking either detour (or both detours) reduces the amount of laps the robot can make in the allotted time.

Robots in this task have access to two different types of sensors (figure 2). First, robots are equipped with a set of 11 wall-sensing rangefinders, each 200 units in length (slightly longer than the narrowest portions of the track) and equally spaced across the frontal 180 degrees of the robot (with one rangefinder pointing directly towards the front). Notice also

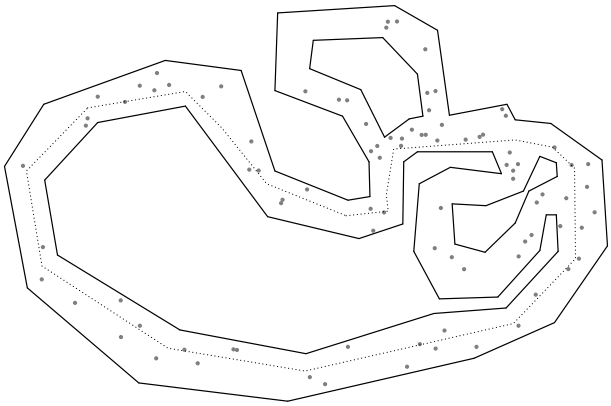
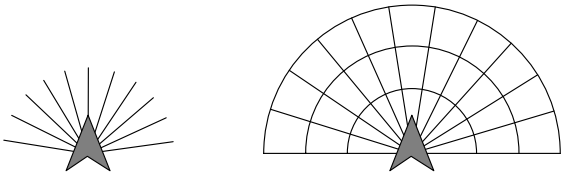


Figure 1: **Multiple path environment.** Robots navigate this cyclical track that consists of an optimal path (in terms of the shortest lap time) with two attached suboptimal detours. The training phase autopilot is denoted by a dotted line. Crumbs (gray dots) are scattered non-uniformly around the track to enable the identification of unique locations.



(a) Wall Sensors (b) Crumb Sensors

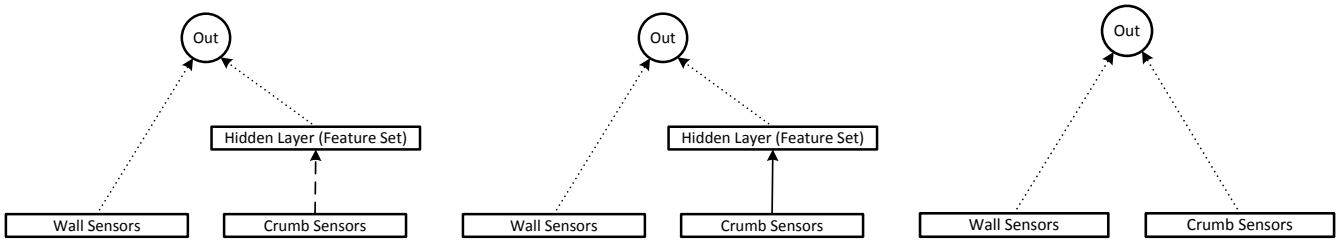
Figure 2: **Agent sensor types.** Robots have a set of 11 rangefinder sensors (a) for detecting the presence of walls (up to a maximum distance of 200 units). Robots also have an array of 33 non-overlapping range-limited pie slice sensors (b) to sense crumbs in the environment. Each sensor can detect up to three crumbs with increasing levels of activation, at which point sensor activation is capped. These crumb sensors form a semicircular grid up to a distance of 400 units across the frontal 180 degrees of the robot.

in figure 1 that there are "crumbs" scattered nonuniformly across the track. The random distribution of these crumbs means the robot can in principle identify unique locations. For this purpose, robots are equipped with 33 crumb-density sensors that sense the density of crumbs within a range-limited pie slice. The crumb-density sensors are divided into three sets of 11 (*near*, *mid*, and *far*), each set equally spaced across the frontal 180 degrees of the robot. Near-type crumb density sensors sense crumbs between 0 and 132 units in distance, mid-type between 133 and 268 units, and far-type between 269 and 400 units. If one crumb is present within a crumb density sensor's area, then the sensor experiences 0.33 activation; it experiences 0.67 activation for two crumbs and 1.0 activation for three or more crumbs (it is rare for more than three crumbs to be present in a sensor's area). Robots have a single effector output, corresponding to the ability to turn right or left. Otherwise, robots are always moving forward at a constant velocity (5 units per tick).

In the experiment, robots first experience a *training phase*. During this phase an autopilot drives the robot around the track for $30{,}000$ ticks. The robot is shown a path that never deviates onto suboptimal detours. However, the driving within the chosen path is not perfect. The autopilot, whose path (shown in figure 1) is deterministic, does not crash but it also does not always drive in the precise middle of the road; that way it is exposed to the penalty for moving too close to walls. During this training phase the autopilot overrides the robot's outputs, forcing it to move along the autopilot's route, while both the autoencoder and Hebbian learning are turned on. After the training phase, autopilot is turned off, learning is stopped, and agents are released to follow their learned controller for $10{,}000$ ticks (the evaluation phase).

It is important to note that while this experiment could have been performed with a supervised learning framework, RAAHN is not restricted to supervised learning. Because RAAHN organizes its feature set and learns a control policy in real-time as it accumulates information about its environment, it can in principle perform when there is no autopilot training phase and robots are simply released into the world under their own control from the first tick. However, this type of application of RAAHN would require a more advanced modulation scheme that also rewards making laps and perhaps would require confronting the distal reward problem. In the interest of introducing the core learning mechanism without other potentially confusing variables, such a study is reserved for future investigation. It is also critical to note that the experiment even as devised is *not* supervised learning because the autoencoder is accumulating features in real-time with no error feedback whatsoever, just as would happen if the agent were allowed to control its own movements while learning. The autopilot simply ensures that the *experience* of the robot is consistent in this initial study so we can understand what is typically learned by a RAAHN when experience is consistent (though of course from different initial random starting weights).

Preliminary experiments revealed that a robot controller consisting of only rangefinder sensors connected directly to the output with Hebbian connections (i.e. without an autoencoder) was able to navigate the track with a trivial wall-following behavior that keeps close to and parallel to a wall on either the right or the left side while moving forward around the track. Because there is a detour attached to both the inner wall and the outer wall, robots performing such a trivial wall-following behavior will inevitably take one of the two detours each lap. The optimal behavior, which involves *avoiding* both detours while moving around the track at maximum speed and avoiding crashing into walls, therefore requires extra information and neural structure than



(a) RAAHN (b) RNDFEAT (c) HEBB

Figure 3: **Variant network structures.** Three variant networks are compared in the main experiment. Individual neurons within sensor array layers and hidden layers have been omitted for clarity. Layers shown as connected are fully connected. Dotted lines represent connections trained with modulated Hebbian plasticity. Dashed lines represent connections trained as an autoencoder. Solid lines represent static (non-trained) connections.

Hebbian learning from the raw rangefinder information. For the purposes of the main experiment, the crumb sensors and a layer of features extracted from the crumb sensors serve as this extra information. With the crumb sensors, robots in principle have the ability to distinguish between different parts of the track that have different "density signatures" (e.g. the opening for the outer detour causes a very different activation pattern on the crumb sensors than the opening for the inner detour). This distinguishing information makes it possible to enact different policies at different parts of the track (e.g. switching to following walls on the left side rather than the right side after passing the outer detour going around the track clockwise), which is essential for avoiding both detours and proceeding around the track along the optimal path. Thus an optimal agent *must* somehow encode these higher-level features.

The main experiment consists of a comparison between three very different learning regimes: RAAHN, RNDFEAT (random features), and HEBB. These methods differ only in the way that they process the extra information from the crumb sensors; all three methods include direct Hebbian connections from the wall-sensing rangefinders to the output. **RAAHN** (figure 3a) includes an autoencoder-trained feature set of 7 neurons drawn from the 33 crumb sensors. This feature set then feeds into the output via Hebbian connections. **RNDFEAT** (figure 3b) has the same structure as RAAHN, except autoencoder training is turned off. This configuration means that RNDFEAT has a set of 7 static *random* features. Results for RNDFEAT with 30 and 100 random features are also included (as **RNDFEAT30** and **RNDFEAT100**, respectively), which resemble the idea behind extreme learning machines (Huang and Wang, 2011). Finally, **HEBB** (figure 3c) consists of *all* inputs directly connected to the output via Hebbian connections. In all variants, connection weights are randomly initialized with a uniform distribution of small magnitude (with average magnitude equal to 5% of the maximum weight of 3.0); increasing the magnitude of initial weights in preliminary experiments did not significantly impact the results.

Experimental Parameters

Batch autoencoder training occurs every 800 ticks on a history buffer of training samples spanning the past $1{,}600$ ticks, which is roughly equivalent to the amount of time required to make a full lap during the training phase (recall that training encompasses $30{,}000$ total ticks). The learning rate α for autoencoder training is 0.001. Each time batch training occurs, the buffer of training samples is spun through 10 times, with backwards activations recalculated after each pass. The result is that the autoencoder mostly converges by the end of a single training pass (autoencoder error is reduced to less than 5% of the optimal reconstruction). The Hebbian learning rate η for all variants is 0.2. These parameter settings were found to be robust to moderate variation through preliminary experimentation.

Results

In the results reported in this section, performance is based on the kinds of paths followed over the $10{,}000$ tick evaluation period, averaged across $1{,}000$ trials. Learned behaviors were found to be consistent, that is, the behavior observed during the first lap is very similar to the behavior on all other laps, especially with respect to which detours are taken (if any). Therefore, it is possible to place behaviors that navigate the maze into two key categories: First are robots that follow the *optimal path*; these best-performing robots never take a detour and thereby stay on the best path. Second are *wall-avoiding* robots who deviate from the optimal path by taking at least one detour but still effectively avoid crashing and thereby make several clean laps around the track. Learned behaviors that do not fall into either of these two categories do not complete the course because they crash into walls. Experimental results are reported in figure 4 as the percentage of robots trained with each scheme that fall into either category.

The main result is that the RAAHN learns the optimal path four times more often than the closest variant (RNDFEAT30). HEBB never learns the optimal path. However, the results

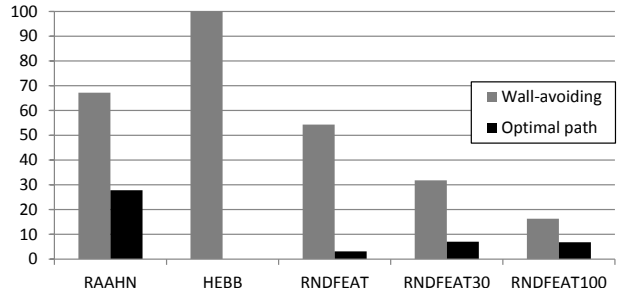


Figure 4: **Percentage of wall-avoiding and optimal path followers.** Results for each variant are percentages calculated over 1 000 evaluations.

confirm that pure Hebbian learning (HEBB) is invariant to starting conditions – it performs exactly the same in each of 1,000 trials (each with different initial weights). HEBB robots always follow the inner detour. Both RAAHN and RNDFEAT variants experience some failed trials, though RAAHN experiences the least. These failures come in many forms: robots that get stuck on sharp corners when exiting a detour are a common observed failure. Some robots fail immediately after leaving autopilot by spinning in tight circles – this type of failure (immediate failure) is not consistent across all methods. Interestingly, immediate failure occurs in 1.7% of RAAHN runs, 15.3% of RNDFEAT runs, 33.4% of RNDFEAT30 runs, and 58.6% of RNDFEAT100 runs.

Discussion and Future Work

The HEBB variant, which attempts to incorporate the crumb sensor inputs directly as a static feature set, never achieves optimal behavior. Thus, raw crumb sensor information appears to be insufficient for Hebbian learning to discover optimal behaviors on this task, suggesting the necessity of higher-level features. While HEBB's performance could in principle be manipulated by manually changing the set of static features (e.g. preprocessing the inputs), doing so is like the human playing the role of the RAAHN. This paper focuses instead on the automated generation of features with the intent of building a learning system that is generally applicable to future domains that may be too complicated for manual preprocessing of features or where adaptation of the feature set is required (e.g. large, open-ended artificial life worlds).

One naive way to automatically generate higher level features is through a hidden layer with random incoming weights, such as in RNDFEAT. While such random features bring optimal behaviors in this task into the realm of possibility, they do not constitute a strong method for achieving such behaviors – even the best RNDFEAT-type variant (RNDFEAT30) achieves the optimal behavior only 7% of the time. Adding more random features (as in RNDFEAT100) only causes performance to deteriorate. Indeed, the rate of immediate failures increases sharply as more random features are added to the feature set (compared to 7 features in RNDFEAT), which suggests that additional random features are destructive as they wash out the useful signal from the wall sensors (e.g. leading to many immediate failures).

RAAHN does not suffer from the pathology of immediate failures because training the feature set as an autoencoder encourages the discovery of a more balanced feature set. RAAHN also discovers optimal behaviors four times as often (28%) as the best RNDFEAT-type variant (RNDFEAT30), which demonstrates the autoencoder's ability to learn useful information about the environment. Furthermore, RAAHN's feature set is dynamic, enabling it in principle to adapt to changing environmental conditions, while RNDFEAT's feature set cannot. Future RAAHNs might even add new layers over time, yielding a kind of real-time deep learning for control. A deeper analysis of neural activity within the autoencoder layer as well as experiments with larger quantities of autoencoded features will indicate the extent of the RAAHN's potential to be expanded in real-time in this way. Furthermore, as the state of the art in Hebbian learning advances, such as by addressing distal rewards (Soltoggio and Steil, 2013), the RAAHN benefits from the advancing capabilities as well.

Finally, while 28% optimality still may appear to leave room for improvement, it is important to note that it is actually impressive for the simple training regimen in this experiment. In particular, during training, robots were never actually shown the detours that they are expected to avoid, so there can be little to no representation of the inside of detours within the autoencoder. Furthermore, robots were not explicitly rewarded for taking the optimal path or penalized for taking detours. Rather, the reward scheme only rewards steering away from walls. Thus optimal paths were acquired *entirely implicitly* through observing the path taken by the autopilot and encoding its key features, suggesting the power of the RAAHN to derive behaviors from such features based on sparse and indirect feedback. In the future, when RAAHN-controlled networks are released to explore completely *on their own* while rewards are experienced, they have the potential to acquire a significantly wider breadth of abilities.

Conclusion

The experiment in this paper showed that a Hebbian layer can learn during ongoing behavior in real time from an autoencoder placed below it under controlled conditions. The implication of this initial step is that the RAAHN is a synergistic union that opens up many opportunities for new investigations. For example, what is possible to achieve when the RAAHN is allowed to acquire new features while exploring on its own without an autopilot? Can increasingly complex skills be acquired if the depth of the autoencoder is allowed to expand during learning? Can ANNs be evolved to incorporate both autoencoders and Hebbian plasticity? These are among the intriguing possibilities created by the RAAHN.

Acknowledgments

This work was partially supported through a grant from the US Army Research Office (Award No. W911NF-11-1-0489). This paper does not necessarily reflect the position or policy of the government, and no official endorsement should be inferred. This work was also partially supported by the European Communitys Seventh Framework Programme FP7/2007-2013, Challenge 2 Cognitive Systems, Interaction, Robotics under grant agreement No. 248311 - AMARSi.

References

Baxter, J. (1992). The evolution of learning algorithms for artificial neural networks. In Green, D. and Bossomaier, T., editors, *Complex Systems*, pages 313–326. IOS Press, Amsterdam.

Bednar, J. A. and Miikkulainen, R. (2003). Self-organization of spatiotemporal receptive fields and laterally connected direction and orientation maps. In De Schutter, E., editor, *Computational Neuroscience: Trends in Research, 2003*, pages 473–480.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, pages 1798–1928.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS)*, Cambridge, MA. MIT Press.

Bienenstock, L. E., Cooper, L. N., and Munro, P. W. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *The Journal of Neuroscience*, 2(1):32–48.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294.

Cireşan, D., Meier, U., Gambardella, L., and Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220.

Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2012). Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338.

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.

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. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems (NIPS 1994)*.

Huang, G.-B. and Wang, D. H. (2011). Extreme learning machines: a survey. *International Journal of Machine Learning & Cybernetics*, 2(107-122).

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 in Machine Learning (ICML-2012)*.

Marc'Aurelio, R., Boureau, L., and LeCun, Y. (2007). Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1185–1192, Cambridge, MA. MIT Press.

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.

Oja, E. (1982). A Simplified Neuron Model as a Principal Component Analyzer. *Journal of Mathematical Biology*, 15(3):267–273.

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 *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB2010)*, 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.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing*, pages 318–362.

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, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, 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. R., and Steil, J. J. (2013). Rare neural correlations implement robotic conditioning with reward delays and disturbances. *Frontiers in Neurorobotics*, 7:6(Research Topic: Value and Reward Based Learning in Neurobots).

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.

Soltoggio, A. and Steil, J. J. (2013). Solving the distal reward problem with rare correlations. *Neural computation*, 25(4):940–978.

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.