Machine Learning, 28, 77–104 (1997)

CHILD: A First Step Towards Continual Learning

MARK B. RING mark.ring@gmd.de
Adaptive Systems Research Group, GMD — German National Research Center for Information Technology, Schloß Birlinghoven, D-53 754 Sankt Augustin, Germany

Editors: Lorien Pratt and Sebastian Thrun

Abstract. *Continual learning* is the constant development of increasingly complex behaviors; the process of building more complicated skills on top of those already developed. A continual-learning agent should therefore learn incrementally and hierarchically. This paper describes CHILD, an agent capable of *Continual, Hierarchical, Incremental Learning* and *Development*. CHILD can quickly solve complicated non-Markovian reinforcement-learning tasks and can then transfer its skills to similar but even more complicated tasks, learning these faster still.

Keywords: Continual learning, transfer, reinforcement learning, sequence learning, hierarchical neural networks

1. Introduction

Supervised learning speeds the process of designing sophisticated software by allowing developers to focus on what the system must *do* rather than on how the system should *work*. This is done by specifying the system's desired outputs for various representative inputs; but even this simpler, abstracted task can be onerous and time consuming. Reinforcement learning takes the abstraction a step further: the developer of a reinforcement-learning system need not even *specify* what the system must do, but only *recognize* when the system does the right thing. The developer's remaining challenge is to design an agent sophisticated enough that it can learn its task using reinforcement-learning methods. Continual learning is the next step in this progression, eliminating the task of building a sophisticated agent by allowing the agent to be *trained* with its necessary foundation instead. The agent can then be kept and trained on even more complex tasks.

Continual learning is a continual *process*, where learning occurs over time, and time is monotonic: A continual-learning agent's experiences occur sequentially, and what it learns at one time step while solving one task, it can use later, perhaps to solve a completely different task.

A continual learner:

- Is an autonomous agent. It senses, takes actions, and responds to the rewards in its environment.
- Can learn context-dependent tasks, where previous senses can affect future actions.
- Learns behaviors and skills while solving its tasks.
- Learns incrementally. There is no fixed training set; learning occurs at every time step; and the skills the agent learns now can be used later.
- Learns hierarchically. Skills it learns now can be built upon and modified later.

- Is a black box. The internals of the agent need not be understood or manipulated. All of the agent's behaviors are developed through *training*, not through direct manipulation. Its only interface to the world is through its senses, actions, and rewards.
- Has no ultimate, final task. What the agent learns now may or may not be useful later, depending on what tasks come next.

Humans are continual learners. During the course of our lives, we continually grasp ever more complicated concepts and exhibit ever more intricate behaviors. Learning to play the piano, for example, involves many stages of learning, each built on top of the previous one: learning finger coordination and tone recognition, then learning to play individual notes, then simple rhythms, then simple melodies, then simple harmonies, then learning to understand clefs and written notes, and so on *without end*. There is always more to learn; one can always add to one's skills.

Transfer in supervised learning involves reusing the features developed for one classification and prediction task as a bias for learning related tasks (Baxter, 1995, Caruana, 1993, Pratt 1993, Sharkey & Sharkey, 1993, Silver & Mercer, 1995, Thrun, 1996, Yu & Simmons, 1990). Transfer in reinforcement learning involves reusing the information gained while learning to achieve one goal to learn to achieve other goals more easily (Dayan & Hinton, 1993, Kaelbling, 1993a, Kaelbling, 1993b, Ring, 1996, Singh, 1992, Thrun & Schwartz, 1995). Continual learning, on the other hand, is the transfer of skills developed so far towards the development of new *skills of greater complexity*.

Constructing an algorithm capable of continual learning spans many different dimensions. Transfer across classification tasks is one of these. An agent's ability to maintain information from previous time steps is another. The agent's environment may be sophisticated in many ways, and the continual-learning agent must eventually be capable of building up to these complexities.

CHILD, capable of *Continual, Hierarchical, Incremental Learning* and *Development*, is the agent presented here, but it is not a perfect continual learner; rather, it is a first step in the development of a continual-learning agent. CHILD can only learn in a highly restricted subset of possible environments, but it exhibits all of the properties described above.

1.1. An Example

Figure 1 presents a series of mazes ranging from simple to more complex. In each maze, the agent can occupy any of the numbered positions (states). The number in each position uniquely represents the configuration of the walls surrounding that position. (Since there are four walls, each of which can be present or absent, there are sixteen possible wall configurations.) By perceiving this number, the agent can detect the walls immediately surrounding it. The agent can move north, south, east, or west. (It may not enter the barrier positions in black, nor may it move beyond the borders of the maze.) The agent's task is to learn to move from any position to the goal (marked by the food dish), where it receives positive reinforcement.

The series of mazes in Figure 1 is intended to demonstrate the notion of continual learning in a very simple way through a series of very simple environments — though even such

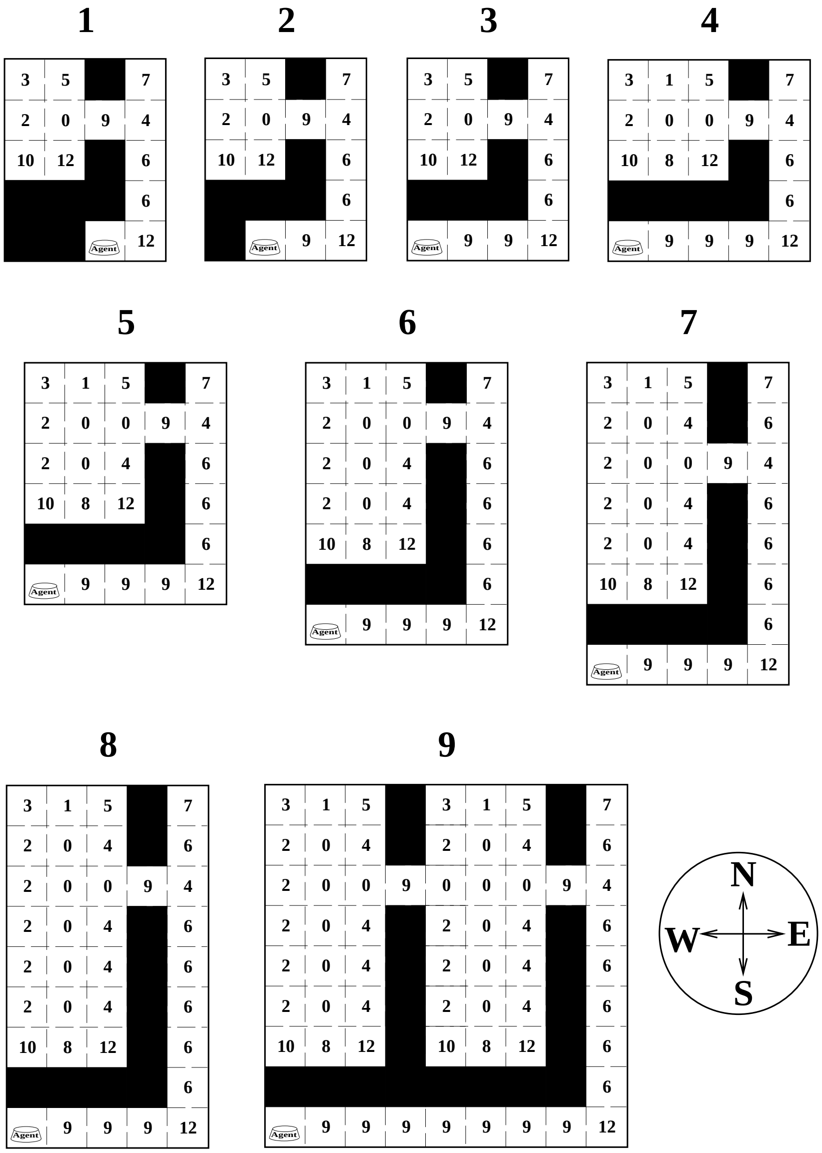

Figure 1. These nine mazes form a progression of reinforcement-learning environments from simple to more complex. In a given maze, the agent may occupy any of the numbered positions (states) at each time step and takes one of four actions: North, East, West, or South. Attempting to move into a wall or barrier (black regions) will leave the agent in its current state. The digits in the maze represent the sensory inputs as described in the text. The goal (a reward of 1.0) is denoted by the food dish in the lower left corner. Each maze is similar to but larger than the last, and each introduces new state ambiguities — more states that share the same sensory input.

seemingly simple tasks can be very challenging for current reinforcement-learning agents to master, due to the large amount of ambiguous state information (i.e., the large number of different states having identical labels). A continual-learning agent trained in one maze should be able to transfer some of what it has learned to the task of learning the next maze. Each maze preserves enough structure from the previous one that the skills learned for one might still be helpful in the next; thus, the basic layout (a room, a doorway, and two hallways) is preserved for all mazes. But each maze also makes new demands on the agent, requiring it to extend its previous skills to adapt to the new environment, and each is therefore larger than the previous one and introduces more state ambiguities.

In the first maze, the agent can learn to choose an action in each state based exclusively upon its immediate sensory input such that it will reach the goal no matter where it begins. To do this, it must learn to move south in all positions labeled “6” and to move west in all positions labeled “12”. In the second maze, however, there is an ambiguity due to the two occurrences of the input “9”. When the agent is transferred to this maze, it should continue to move east when it senses “9” in the upper position, but it should move west upon sensing “9” in the lower position. It must distinguish the two different positions though they are labeled the same. It can make this distinction by using the preceding context: only in the lower position has it just seen input “12”. When placed in the third maze, the agent’s behavior would need to be extended again, so that it would move west upon seeing “9” whenever its previous input was either a “12” *or* a “9”.

This progression of complexity continues in the remaining mazes in that each has more states than the last and introduces new ambiguities. The continual-learning agent should be able to transfer to the next maze in the sequence skills it developed while learning the previous maze, coping with every new situation in a similar way: old responses should be modified to incorporate new exceptions by making use of contextual information that disambiguates the situations.

2. Temporal Transition Hierarchies

CHILD combines Q-learning (Watkins, 1989) with the *Temporal Transition Hierarchies* (TTH) learning algorithm. The former is the most widely used reinforcement-learning method and will not be described here. The Temporal Transition Hierarchies algorithm is a constructive neural-network-based learning system that focuses on the most important but least predictable events and creates new units that learn to predict these events.

As learning begins, the untrained TTH network assumes that event probabilities are constants: “the probability that event A will lead to event B is x_{AB}.” (For example, the probability that pressing button C2 on the vending machine will result in the appearance of a box of doughnuts is 0.8). The network’s initial task is to learn these probabilities. But this is only the coarsest description of a set of events, and knowing the context (e.g., whether the correct amount of change was deposited in the slot) may provide more precise probabilities. The purpose of the new units introduced into the network is to search the preceding time steps for contextual information that will allow an event to be predicted more accurately.

In reinforcement-learning tasks, TTH's can be used to find the broader context in which an action *should* be taken when, in a more limited context, the action appears to be good sometimes but bad other times. In Maze 2 of Figure 1 for example, when the agent senses "9", it cannot determine whether "move east" is the best action, so a new unit is constructed to discover the context in which the agent's response to input 9 should be to move east.

The units created by the TTH network resemble the "synthetic items" of Drescher (1991), created to determine the causes of an event (the cause is determined through training, after which the item represents that cause). However, the simple structure of the TTH network allows it to be trained with a straightforward gradient descent procedure, which is derived next.

2.1. Structure, Dynamics, and Learning

Transition hierarchies are implemented as a constructive, higher-order neural network. I will first describe the structure, dynamics, and learning rule for a given network with a fixed number of units, and then describe in Section 2.3 how new units are added.

Each unit (u^i) in the network is either: a sensory unit ($s^i \in S$); an action unit ($a^i \in A$); or a high-level unit ($l^i_{xy} \in L$) that dynamically modifies w_{xy}, the connection from sensory unit y to action unit x. (It is the high-level units which are added dynamically by the network.)[1] The action and high-level units can be referred to collectively as non-input units ($n^i \in N$). The next several sections make use of the following definitions:

$$S \overset{def}{=} \{u^i \,|\, 0 \leq i < ns\} \tag{1}$$

$$A \overset{def}{=} \{u^i \,|\, ns \leq i < ns + na\} \tag{2}$$

$$L \overset{def}{=} \{u^i \,|\, ns + na \leq i < nu\} \tag{3}$$

$$N \overset{def}{=} \{u^i \,|\, ns \leq i < nu\} \tag{4}$$

$$u^i(t) \overset{def}{=} \text{the value of the } i\text{th unit at time } t \tag{5}$$

$$T^i(t) \overset{def}{=} \text{the target value for } a^i(t), \tag{6}$$

where ns is the number of sensory units, na is the number of action units, and nu is the total number of units. When it is important to indicate that a unit is a sensory unit, it will be denoted as s^i; similarly, action units will be denoted as a^i, high-level units will be denoted as l^i, and non-input units will be denoted when appropriate as n^i.

The activation of the units is very much like that of a simple, single-layer (no hidden units) network with a linear activation function,

$$n^i(t) = \sum_j \hat{w}_{ij}(t) s^j(t). \tag{7}$$

The activation of the i^{th} action or higher-level unit is simply the sum of the sensory inputs multiplied by their respective weights, $\hat{w}_{ij}$, that lead into n^i. (Since these are higher-order connections, they are capable of non-linear classifications.) The higher-order weights are produced as follows:

$$\hat{w}_{ij}(t) = \begin{cases} w_{ij} + l_{ij}(t-1) & \text{if unit } l_{ij} \text{ for weight } w_{ij} \text{ exists} \\ w_{ij} & \text{otherwise.} \end{cases} \tag{8}$$

If no l unit exists for the connection from i to j, then w_{ij} is used as the weight. If there is such a unit, however, its previous activation value is added to w_{ij} to compute the higher-order weight $\hat{w}_{ij}$.[2]

2.2. Deriving the Learning Rule

A learning rule can be derived that performs gradient descent in the error space and is much simpler than gradient descent in recurrent networks; so, though the derivation that follows is a bit lengthy, the result at the end (Equation 29) is a simple learning rule, easy to understand as well as to implement.

As is common with gradient-descent learning techniques, the network weights are modified so as to reduce the total sum-squared error:

$$\begin{aligned} E &= \sum_t E(t) \\ E(t) &= \frac{1}{2}\sum_i (T^i(t) - a^i(t))^2. \end{aligned} \tag{9}$$

In order to allow incremental learning, it is also common to approximate strict gradient-descent by modifying the weights at every time step. At each time step, the weights are changed in the direction opposite their contribution to the error, $E(t)$:

$$\Delta w_{ij}(t) \overset{def}{=} \sum_{\tau=0}^{t} \frac{\partial E(t)}{\partial w_{ij}(\tau)} \tag{10}$$

$$w_{ij}(t+1) = w_{ij}(t) - \eta \Delta w_{ij}(t)\ , \tag{11}$$

where η is the learning rate. The weights, w_{ij}, are time indexed in Equation 10 for notational convenience only and are assumed for the purposes of the derivation to remain the same at all time steps (as is done with all on-line neural-network training methods).

It can be seen from Equations 7 and 8 that it takes a specific number of time steps for a weight to have an effect on the network's action units. Connections to the action units affect the action units at the current time step. Connections to the first level of high-level units — units that modify connections to the action units — affect the action units after one time step. The "higher" in the hierarchy a unit is, the longer it takes for it (and therefore its incoming connections) to affect the action units. With this in mind, Equation 10 can be rewritten as:

$$\Delta w_{ij}(t) \overset{def}{=} \sum_{\tau=0}^{t} \frac{\partial E(t)}{\partial w_{ij}(\tau)} = \frac{\partial E(t)}{\partial w_{ij}(t-\tau^i)} \ , \tag{12}$$

where τ^i is the constant value for each action or high-level unit n^i that specifies how many time steps it takes for a change in unit i's activation to affect the network's output. Since this value is directly related to how "high" in the hierarchy unit i is, τ is very easy to compute:

$$\tau^i = \begin{cases} 0 & \text{if } n^i \text{ is an action unit, } a^i \\ 1 + \tau^x & \text{if } n^i \text{ is a higher-level unit, } l^i_{xy}. \end{cases} \tag{13}$$

The derivation of the gradient proceeds as follows. Define δ^i to be the partial derivative of the error with respect to non-input unit n^i:

$$\delta^i(t) \overset{def}{=} \frac{\partial E(t)}{\partial n^i(t-\tau^i)} \ . \tag{14}$$

What must be computed is the partial derivative of the error with respect to each weight in the network:

$$\begin{aligned} \frac{\partial E(t)}{\partial w_{ij}(t-\tau^i)} &= \frac{\partial E(t)}{\partial n^i(t-\tau^i)} \frac{\partial n^i(t-\tau^i)}{\partial w_{ij}(t-\tau^i)} \\ &= \delta^i(t) \frac{\partial n^i(t-\tau^i)}{\partial w_{ij}(t-\tau^i)} \ . \end{aligned} \tag{15}$$

From Equations 7 and 8, the second factor can be derived simply as:

$$\frac{\partial n^i(t-\tau^i)}{\partial w_{ij}(t-\tau^i)} = s^j(t-\tau^i) \frac{\partial \hat{w}_{ij}(t-\tau^i)}{\partial w_{ij}(t-\tau^i)} \tag{16}$$

$$= s^j(t-\tau^i) \begin{cases} 1 + \frac{\partial l_{ij}(t-\tau^i-1)}{\partial w_{ij}(t-\tau^i)} & \text{if } l_{ij} \text{ exists} \\ 1 & \text{otherwise.} \end{cases} \tag{17}$$

Because $w_{ij}(t-\tau^i)$ does not contribute to the value of $l_{ij}(t-\tau^i-1)$,

$$\frac{\partial n^i(t-\tau^i)}{\partial w_{ij}(t-\tau^i)} = s^j(t-\tau^i). \tag{18}$$

Therefore, combining 12, 15 and 18,

$$\Delta w_{ij}(t) = \frac{\partial E(t)}{\partial w_{ij}(t-\tau^i)} = \delta^i(t) s^j(t-\tau^i)\ . \tag{19}$$

Now, $\delta^i(t)$ can be derived as follows. First, there are two cases depending on whether node i is an action unit or a high-level unit:

$$\delta^i(t) = \begin{cases} \dfrac{\partial E(t)}{\partial a^i(t-\tau^i)} & \text{if } n^i \text{ is an action unit, } a^i \\ \dfrac{\partial E(t)}{\partial l^i_{xy}(t-\tau^i)} & \text{if } n^i \text{ is a higher-level unit, } l^i_{xy}. \end{cases} \tag{20}$$

The first case is simply the immediate derivative of the error with respect to the action units from Equation 9. Since τ^i is zero when n^i is an action unit,

$$\begin{aligned} \frac{\partial E(t)}{\partial a^i(t-\tau^i)} &= \frac{\partial E(t)}{\partial a^i(t)} \\ &= a^i(t) - T^i(t). \end{aligned} \tag{21}$$

The second case of Equation 20 is somewhat more complicated. From Equation 8,

$$\frac{\partial E(t)}{\partial l^i_{xy}(t-\tau^i)} = \frac{\partial E(t)}{\partial \hat{w}_{xy}(t-\tau^i+1)} \frac{\partial \hat{w}_{xy}(t-\tau^i+1)}{\partial l^i_{xy}(t-\tau^i)} \tag{22}$$

$$= \frac{\partial E(t)}{\partial \hat{w}_{xy}(t-\tau^i+1)}. \tag{23}$$

Using Equation 7, this can be factored as:

$$\frac{\partial E(t)}{\partial \hat{w}_{xy}(t-\tau^i+1)} = \frac{\partial E(t)}{\partial n^x(t-\tau^i+1)} \frac{\partial n^x(t-\tau^i+1)}{\partial \hat{w}_{xy}(t-\tau^i+1)}. \tag{24}$$

Because n^i is a high-level unit, $\tau^i = \tau^x + 1$ (Equation 13). Therefore,

$$\frac{\partial E(t)}{\partial l^i_{xy}(t-\tau^i)} = \frac{\partial E(t)}{\partial n^x(t-\tau^x)} \frac{\partial n^x(t-\tau^x)}{\partial \hat{w}_{xy}(t-\tau^x)} \tag{25}$$

$$= \delta^x(t) s^y(t-\tau^x), \tag{26}$$

from Equations 7 and 14. Finally, from Equation 19,

$$\frac{\partial E(t)}{\partial l^i_{xy}(t-\tau^i)} = \Delta w_{xy}(t). \tag{27}$$

Returning now to Equations 19 and 20, and substituting in Equations 21 and 27: The change $\Delta w_{ij}(t)$ to the weight w_{ij} from sensory unit s^j to action or high-level unit n^i can be written as:

$$\Delta w_{ij}(t) = \delta^i(t) s^j(t-\tau^i) \tag{28}$$

$$= s^j(t-\tau^i) \begin{cases} a^i(t) - T^i(t) & \text{if } n^i \text{ is an action unit, } a^i \\ \Delta w_{xy}(t) & \text{if } n^i \text{ is a higher-level unit, } l^i_{xy}. \end{cases} \tag{29}$$

Equation 29 is a particularly nice result, since it means that the only values needed to make a weight change at any time step are (1) the error computable at that time step, (2) the input recorded from a specific previous time step, and (3) other weight changes already calculated. This third point is not necessarily obvious; however, each high-level unit is higher in the hierarchy than the units on either side of the weight it affects: $(i > x) \wedge (i > y)$, for all l^i_{xy}. This means that the weights may be modified in a simple bottom-up fashion. Error values are first computed for the action units, then weight changes are calculated from the bottom of the hierarchy to the top so that the $\Delta w_{xy}(t)$ in Equation 29 will already have been computed before $\Delta w_{ij}(t)$ is computed, for all high-level units l^i_{xy} and all sensory units j.

The intuition behind the learning rule is that each high-level unit, $l^i_{xy}(t)$, learns to utilize the context at time step t to correct its connection's *error*, $\Delta w_{xy}(t+1)$, at time step $t+1$. If the information is available, then the higher-order unit uses it to reduce the error. If the needed information is not available at the previous time step, then new units may be built to look for the information at still earlier time steps.

While testing the algorithm, it became apparent that changing the weights at the bottom of a large hierarchy could have an explosive effect: the weights would oscillate to ever larger values. This indicated that a much smaller learning rate was needed for these weights. Two learning rates were therefore introduced: the normal learning rate, η, for weights without higher-level units (i.e., w_{xy} where no unit l^i_{xy} exists); and a fraction, η_L, of η for those weights whose values *are* affected by higher-level units, (i.e., w_{xy} where a unit l^i_{xy} does exist).

2.3. Adding New Units

So as to allow transfer to new tasks, the net must be able to create new higher-level units *whenever* they might be useful. Whenever a transition varies, that is, when the connection weight should be different in different circumstances, a new unit is required to dynamically set the weight to its correct value. A unit is added whenever a weight is pulled strongly in opposite directions (i.e., when learning forces the weight to increase and to decrease at the same time). The unit is created to determine the contexts in which the weight is pulled in each direction.

In order to decide when to add a new unit, two long-term averages are maintained for every connection. The first of these, $\Delta\bar{w}_{ij}$, is the average change made to the weight. The second, $\Delta\tilde{w}_{ij}$, is the average *magnitude* of the change. When the average change is small but the average magnitude is large, this indicates that the learning algorithm is changing the weight by large amounts but about equally in the positive as in the negative direction; i.e., the connection is being simultaneously forced to increase and to decrease by a large amount.

Two parameters, Θ and ϵ, are chosen, and when

$$\Delta\tilde{w}_{ij} > \Theta|\Delta\bar{w}_{ij}| + \epsilon, \tag{30}$$

a new unit is constructed for w_{ij}.

Since new units need to be created when a connection is unreliable *in certain contexts*, the long-term average is only updated when changes are actually made to the weight; that is, when $\Delta w_{ij}(t) \neq 0$. The long-term averages are computed as follows:

$$\Delta \bar{w}_{ij}(t) = \begin{cases} \Delta \bar{w}_{ij}(t-1) & \text{if } \Delta w_{ij}(t) = 0 \\ \sigma \Delta w_{ij}(t) + (1-\sigma)\Delta \bar{w}_{ij}(t-1) & \text{otherwise,} \end{cases} \tag{31}$$

and the long-term average magnitude of change is computed as:

$$\Delta \tilde{w}_{ij}(t) = \begin{cases} \Delta \tilde{w}_{ij}(t-1) & \text{if } \Delta w_{ij}(t) = 0 \\ \sigma |\Delta w_{ij}|(t) + (1-\sigma)\Delta \tilde{w}_{ij}(t-1) & \text{otherwise,} \end{cases} \tag{32}$$

where the parameter σ specifies the duration of the long-term average. A smaller value of σ means the averages are kept for a longer period of time and are therefore less sensitive to momentary fluctuations. A small value for σ is therefore preferable if the algorithm's learning environment is highly noisy, since this will cause fewer units to be created due strictly to environmental stochasticity. In more stable, less noisy environments, a higher value of σ may be preferable so that units will be created as soon as unpredictable situations are detected.

When a new unit is added, its incoming weights are initialized to zero. It has no output weights: its only task is to anticipate and reduce the error of the weight it modifies. In order to keep the number of new units low, whenever a unit l^n_{ij} is created, the statistics for all connections into the destination unit (u^i) are reset: $\Delta \bar{w}_{ij}(t) \leftarrow -1.0$ and $\Delta \tilde{w}_{ij}(t) \leftarrow 0.0$.

2.4. The Algorithm

Because of the simple learning rule and method of adding new units, the learning algorithm is very straightforward. Before training begins, the network has no high-level units and all weights (from every sensory unit to every action unit) are initialized to zero. The outline of the procedure is as follows:

```
For (Ever)
    1) Initialize values.
    2) Get senses.
    3) Propagate Activations.
    4) Get Targets.
    5) Calculate Weight Changes;
       Change Weights & Weight Statistics;
       Create New Units.
```

The second and fourth of these are trivial and depend on the task being performed. The first step is simply to make sure all unit values and all delta values are set to zero for the next forward propagation. (The values of the l units at the last time step must, however, be stored for use in step 3.)

1) Initialize values

```
Line  /* Reset all old unit and delta values to zero.                          */
1.1   For each unit, u(i)
1.2     u(i) ← zero;
1.3     delta(i) ← zero;
```

The third step is nearly the same as the forward propagation in standard feed-forward neural networks, except for the presence of higher-order units and the absence of hidden layers.

3) Propagate Activations

```
Line  /* Calculate new output values.                                           */
3.1   For each Non-input unit, n(i)
3.2     For each Sensory unit, s(j)

          /* UnitFor(i, j) returns the input of unit $l_{ij}$ at the *last* time step.  */
          /* Zero is returned if the unit did not exist. (See Equation 8.)       */
3.3       l ← UnitFor(i,j);

          /* To $n^i$'s input, add $s^j$'s value times the (possibly modified)    */
          /* weight from $j$ to $i$. (See Equation 7.)                            */
3.4       n(i) ← n(i) + s(j)*(l + Weight(i, j));
```

The fifth step is the heart of the algorithm. Since the units are arranged as though the input, output, and higher-level units are concatenated into a single vector (i.e., $k < j < i$, for all s^k, a^j, l^i), whenever a unit l^i_{jk} is added to the network, it is appended to the end of the vector; and therefore $(j < i) \wedge (k < i)$. This means that when updating the weights, the δ^i's and Δw_{ij}'s of Equation 29 must be computed with i in ascending order, so that Δw_{xy} will be computed before any Δw_{ij} for unit l^i_{xy} is computed (line 5.3).

If a weight change is not zero (line 5.6), it is applied to the weight (line 5.9 or 5.19). If the weight has no higher-level unit (line 5.8), the weight statistics are updated (lines 5.10 and 5.11) and checked to see whether a higher-level unit is warranted (line 5.12). If a unit is warranted for the weight leading from unit j to unit i (line 5.12), a unit is built for it (line 5.13), and the statistics are reset for all weights leading into unit i (lines 5.14–5.16). If a higher-level unit already exists (line 5.17), that unit's delta value is calculated (line 5.18) and used (at line 5.5) in a following iteration (of the loop starting at line 5.3) when *its* input weights are updated.

5) Update Weights and Weight Statistics; Create New Units.

```
/* Calculate $\delta_i$ for the action units, $a^i$. (See Equations  20 and 21).            */
For each action unit, a(i)
  delta(i) = a(i) - Target(i);

/* Calculate all $\Delta w_{ij}$'s, $\Delta \bar{w}_{ij}$'s, $\Delta \tilde{w}_{ij}$'s.          */
/* For higher-order units $l^i$, calculate $\delta^i$'s.                                     */
/* Change weights and create new units when needed.                                          */
For each Non-input unit, n(i), with $i$ in **ascending** order
  For each Sensory unit, s(j)

    /* Compute weight change (Equations 28 and 29).                                          */
    /* Previous(j, i) retrieves $s^j(t - \tau^i)$.                                           */
    delta_w(i, j) $\leftarrow$ delta(i) * Previous(j, i);

    /* If $\Delta w_{ij} \neq 0$, update weight and statistics. (Eqs. 31 and 32).            */
    if (delta_w(i,j) $\neq$ 0)

      /* IndexOfUnitFor(i, j) returns $n$ for $l^n_{ij}$; or -1 if $l^n_{ij}$ doesn't exist.  */
      n $\leftarrow$ IndexOfUnitFor(i, j);

      /* If $l^n_{ij}$ doesn't exist: update statistics, learning rate is $\eta$.          */
      if (n = -1)

        /* Change weight $w_{ij}$. (See Equation 11.)                                        */
        Weight(i, j) $\leftarrow$ Weight(i, j) - ETA * delta_w(i, j);

        /* Update long-term average, $\Delta \bar{w}_{ij}$. (See Equation 31)                */
        lta(i,j) $\leftarrow$ SIGMA * delta_w(i,j) + (1-SIGMA) * lta(i,j);

        /* Update long-term mean absolute deviation $\Delta \tilde{w}_{ij}$. (Eq. 32)        */
        ltmad(i,j) $\leftarrow$ SIGMA * abs(delta_w(i,j)) +
                      (1-SIGMA) * ltmad(i,j);

        /* If Higher-Order unit $l^n_{ij}$ should be created (Equation 30) ...               */
        if (ltmad(i, j) $>$ THETA * abs(lta(i, j)) + EPSILON)

          /* ... create unit $l^N_{ij}$ (where $N$ is the current network size).             */
          BuildUnitFor(i, j);

          /* Reset statistics for all incoming weights.                                      */
          For each Sensory unit, s(k)
            lta(i, k) $\leftarrow$ -1.0;
            ltmad(i, k) $\leftarrow$ 0.0;

      /* If $l^n_{ij}$ does exist ($n \neq -1$), store $\delta^n$ (Equation 20 and 27).       */
      /* Change $w_{ij}$, learning rate $= \eta_L * \eta$.                                    */
      else
        delta(n) $\leftarrow$ delta_w(i, j);
        Weight(i, j) $\leftarrow$ Weight(i, j) - ETA_L*ETA * delta_w(i, j);
```

3. Testing CHILD in Continual-Learning Environments

This section demonstrates CHILD's ability to perform continual learning in reinforcement environments. CHILD combines Temporal Transition Hierarchies with Q-learning (Watkins, 1989). Upon arriving in a state, the agent's sensory input from that state is given to a transition hierarchy network as input. The output from the network's action units are Q-values (one Q-value for each action) which represent the agent's estimate of its discounted future reward for taking that action (Watkins, 1989). At each time step an action unit, $a^{C(t)}$, is chosen stochastically using a Gibbs distribution formed from these values and a *temperature* value T. The agent then executes the action associated with the chosen action unit $a^{C(t)}$. The temperature is initially set to 1.0, but its value is decreased at the beginning of each trial to be $1/(1+n\Delta T)$, where n is the number of trials so far and ΔT is a user-specified parameter.

The network is updated like the networks of Lin (1992):

$$T^i(t-1) = \begin{cases} r(t-1) + \gamma \max\limits_k(a^k(t)) & \text{if } a^i \equiv a^{C(t-1)} \\ a^i(t-1) & \text{otherwise,} \end{cases} \tag{33}$$

where $T^i(t-1)$ is the target as specified in Equation 29 for action unit a^i at time step $t-1$; $r(t-1)$ is the reward the agent received after taking action $a^{C(t-1)}$; and γ is the discount factor. Only action unit $a^{C(t-1)}$ will propagate a non-zero error to its incoming weights.

The sequence of mazes introduced in Figure 1 are used as test environments. In these environments there are 15 sensory units — one for each of the 15 possible wall configurations surrounding the agent; therefore exactly one sensory unit is on (has a value of 1.0) in each state. There are 4 action units — N (move one step north), E (east), W (west), and S (south).

CHILD was tested in two ways: learning each maze from scratch (Section 3.1), and using continual learning (Section 3.2). In both cases, learning works as follows. The agent "begins" a maze under three possible conditions: (1) it is the agent's first time through the maze, (2) the agent has just reached the goal in the previous trial, or (3) the agent has just "timed out", i.e, the agent took all of its allotted number of actions for a trial without reaching the goal. Whenever the agent begins a maze, the learning algorithm is first reset, clearing its short-term memory (i.e., resetting all unit activations and erasing the record of previous network inputs). A random state in the maze is then chosen and the agent begins from there.

3.1. Learning from Scratch

100 agents were separately created, trained, and tested in each maze (i.e., a total of 900 agents), all with different random seeds. Each agent was trained for 100 trials, up to 1000 steps for each trial. The agent was then *tested* for 100 trials; i.e., learning was turned off and the most highly activated action unit was always chosen. If the agent did not reach the goal on every testing trial, training was considered to have failed, and the agent was trained for 100 more trials and tested again. This process continued until testing succeeded or until the agent was trained for a total of 1000 trials.

Since CHILD is a combination of Q-learning and Temporal Transition Hierarchies, there are seven modifiable parameters: two from Q-learning and five from TTH's. The two from Q-learning are: γ — the discount factor from Equation 33, and ΔT, the temperature decrement. The five from the TTH algorithm are: σ, Θ, and ϵ from Equations 30, 31 and 32, η — the learning rate from Equation 11, and η_L — the fraction of η for weights with high-level units. Before training began, all seven parameters were (locally) optimized for each maze independently to minimize the number of trials and units needed. The optimization was done using a set of random seeds that were not later used during the tests reported below.

3.2. The Continual-Learning Case

To measure its continual-learning ability, CHILD was allowed in a separate set of tests to use what it had learned in one maze to help it learn the next. This is a very tricky learning problem, since, besides the added state ambiguities, the distance from most states to the goal changes as the series of mazes progresses, and the Q-values for most of the input labels therefore need to be re-learned.

There were three differences from the learning-from-scratch case: (1) after learning one maze, the agent was transferred to the next maze in the series; (2) the agent was tested in the new maze *before* training — if testing was successful, it was moved immediately to the following maze; and (3) the parameters (which were not optimized for this approach) were the same for all mazes:

$$\begin{array}{rcl} \eta = \beta & = & 0.25 \\ \eta_L & = & 0.09 \\ \gamma & = & 0.91 \\ \Delta T & = & 2.1 \end{array} \qquad\qquad \begin{array}{rcl} \sigma & = & 0.3 \\ \Theta & = & 0.56 \\ \epsilon & = & 0.11 \\ & & \end{array}$$

T was reset to 1.0 when the agent was transferred to a new maze.

3.3. Comparisons

For both the learning-from-scratch case and the continual-learning case, the total number of steps during training was averaged over all 100 agents in each maze. These results are shown in Figure 2A. For the continual-learning approach, both the average number of steps and the average accumulated number of steps are shown. The average number of units created during training is given in Figure 2B. Figure 2C compares the average number of test steps. Since it was possible for an agent to be tested several times in a given training run before testing was successful, only the final round was used for computing the averages (i.e., the last 100 testing trials for each agent in each maze).

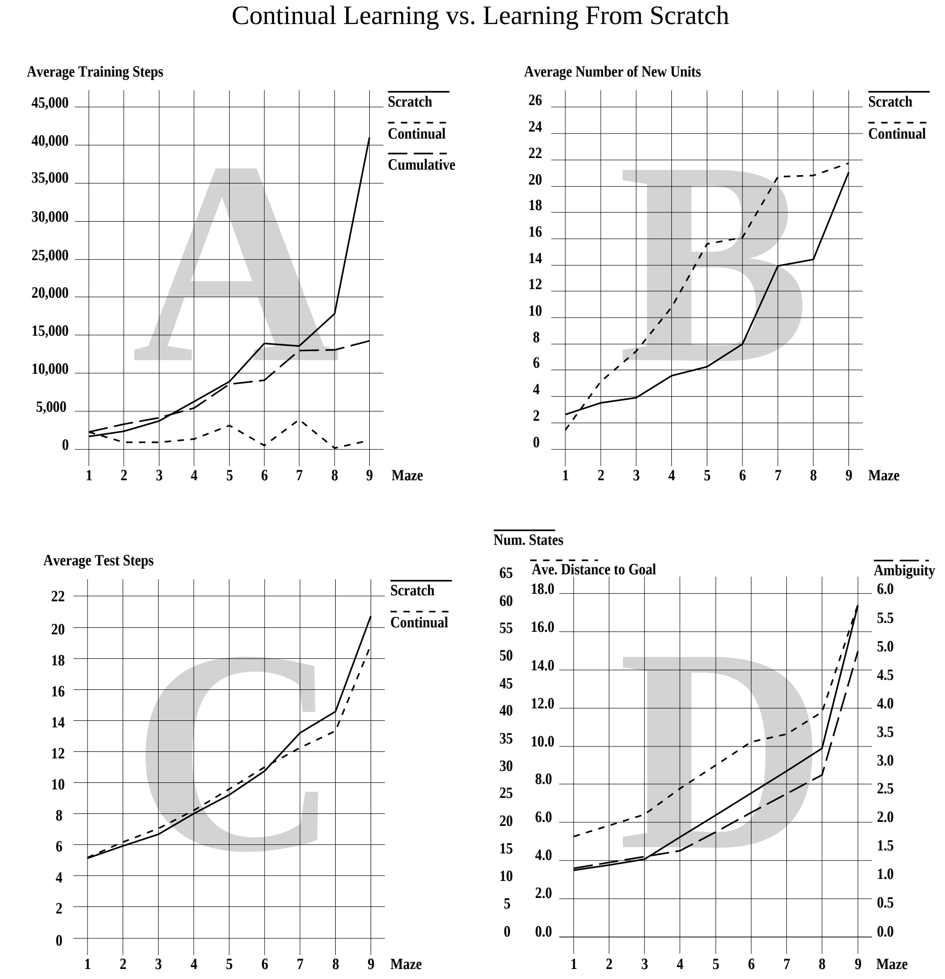

Figure 2. Graph (A) compares Learning from Scratch with Continual Learning on the nine Maze tasks. The middle line shows the average cumulative number of steps used by the continual-learning algorithm in its progression from the first to the ninth maze. Graph (B) compares the number of units created. The line for the continual-learning case is, of course, cumulative. Graph (C) compares the performance of both methods during testing. The values shown do not include cases where the agent failed to learn the maze. Graph (D) shows the number of states in each maze (solid line, units shown at far left); the average distance from each state to the goal (dotted line, units at left); and the "ambiguity" of each maze, i.e., the average number of states per state label (dashed line, units at right).

There were five failures while learning from scratch — five cases in which 1000 training trials were insufficient to get testing correct. There were two failures for the continual-learning case. All failures occurred while learning the ninth maze. When a failure occurred, the values for that agent were not averaged into the results shown in the graphs.

4. Analysis

Due to the large amount of state ambiguity, these tasks can be quite difficult for reinforcement-learning agents. Even though a perfect agent could solve any of these mazes with a very small amount of state information, *learning* the mazes is more challenging. Consider for example the fact that when the agent attempts to move into a barrier, its position does not change, and it again receives the same sensory input. It is not in any way informed that its position is unchanged. Yet it must learn to avoid running into barriers nevertheless. On the other hand, the bottom row in Maze 9 is exactly the opposite: the agent must continue to move east, though it repeatedly perceives the same input. Also, once a maze has been learned, the sequence of states that the agent will pass through on its path to the goal will be the same from any given start state, thereby allowing the agent to identify its current state using the context of recent previous states. However, while *learning* the task, the agent's moves are inconsistent and erratic, and information from previous steps is unreliable. What can the agent deduce, for example, in Maze 9 if its current input is 4 and its last several inputs were also 4? If it has no knowledge of its previous actions and they are also not yet predictable, it cannot even tell whether it is in the upper or lower part of the maze.[3]

The progression of complexity over the mazes in Figure 1 is shown with three objective measures in Figure 2D.The number of states per maze is one measure of complexity. Values range from 12 to 60 and increase over the series of mazes at an accelerating pace. The two other measures of complexity are the average distance from the goal and the average ambiguity of each maze's state labels (i.e., the average number of states sharing the same label). The latter is perhaps the best indication of a maze's complexity, since this is the part of the task that most confuses the agent: distinguishing the different states that all produce the same sensory input. Even in those cases where the agent should take the same action in two states with the same label, these states will still have different Q-values and therefore will need to be treated differently by the agent. All three measures increase monotonically and roughly reflect the contour of graphs A–C for the learning-from-scratch agents.

The performance of the learning-from-scratch agents also serves as a measure of a maze's complexity because the parameters for these agents were optimized to learn each maze as quickly as possible. Due to these optimized parameters, the learning-from-scratch agents learned the first maze faster (by creating new units faster) than the continual-learning agents did. After the first maze, however, the continual-learning agents always learned faster than the learning-from-scratch agents. In fact, after the third maze, despite the disparity in parameter optimality, even the *cumulative* number of steps taken by the continual-learning agent was less than the number taken by the agent learning from scratch. By the ninth maze the difference in training is drastic. The number of extra steps needed by the continual-learning agent was tiny in comparison to the number needed without continual learning. The cumulative total was about a third of that needed by the agent learning from scratch. Furthermore, the trends shown in the graphs indicate that as the mazes get larger, as the size and amount of ambiguity increases, the difference between continual learning and learning from scratch increases drastically.

Testing also compares favorably for the continual learner: after the fourth maze, the continual-learning agent found better solutions as well as finding them faster. This is perhaps attributable to the fact that, after the first maze, the continual-learning agent is always in the process of making minor corrections to its existing Q-values, and hence to its policy. The corrections it makes are due both to the new environment and to errors still present in its Q-value estimates. The number of units needed for the continual learner escalates quickly at first, indicating both that the parameters were not optimal and that the skills learned for each maze did not transfer perfectly but needed to be extended and modified for the following mazes. This number began to to level off after a while, however, showing that the units created in the first eight tasks were mostly sufficient for learning the ninth. The fact that the learning-from-scratch agent created about the same number shows that these units were also necessary.

That CHILD should be able to transfer its skills so well is far from given. Its success at continual learning in these environments is due to its ability to identify differences between tasks and then to make individual modifications even though it has already learned firmly established rules. Moving from Maze 4 to Maze 5, for example, involves learning that the tried and true rule $4 \rightarrow S$ (move south when the input is “4”), which was right for all of the first four mazes, is now only right within a certain *context*, and that in a different context a move in the opposite direction is now required. At each stage of continual learning, the agent must widen the context in which its skills were valid to include new situations in which different behaviors are indicated.

4.1. Hierarchy Construction in the Maze Environments

It is interesting to examine what CHILD learns: which new units are built and when. The following progression occurred while training one of the 100 agents tested above. For the first maze, only one unit was constructed, $l^{20}_{W,12}$. This was unit number 20, which modifies the connection from sensory unit 12 to action unit W (move west). (Units 1–15 are the sensory units; units 16–19 are the action units; and the high-level units begin at number 20). Unit 20 resolved the ambiguity of the two maze positions labeled 12. With the unit in place, the agent learned to move east in the position labeled 10. It then could move north in position 12 when having seen a 10 in the previous step, but west when having seen a 6. Thus, the weight from s^6 to $l^{20}_{W,12}$ was positive, and the weight from s^{10} to $l^{20}_{W,12}$ was negative. The weight from s^0 to $l^{20}_{W,12}$ was also negative, reflecting the fact that, though the optimal route does not involve moving south from the position labeled 0, during training this would happen nevertheless. For the same reason, there was also a negative weight from s^{12} to $l^{20}_{W,12}$. If the agent was placed in one of the two positions labeled 12 at the beginning of the trial, it would move north, since it had no way of distinguishing its state. (It would then move south if it saw a 6 or east if it saw a 0.)

The second maze contains two new ambiguities: the positions labeled 9. Two new units were created: $l^{21}_{W,9}$, and $l^{22}_{N,12}$. The first, $l^{21}_{W,9}$, was needed to disambiguate the two 9 positions. It had a strong positive weight from s^{12} so that the agent would move west from a position labeled 9 if it had just seen a 12. The second, $l^{22}_{N,12}$, complemented $l^{20}_{W,12}$. It was apparently needed during training to produce more accurate Q-values when the new 9 position was introduced, and had a weak positive connection from s^9 and a weaker negative connection from s^{10}.

The third maze introduces another position labeled 9. This caused a strong weight to develop from s^9 to $l^{21}_{W,9}$. Only one new unit was constructed, however: $l^{23}_{20,12}$. The usefulness of this unit was minimal: it only had an effect if in a position labeled 12, the agent moved south or east. As a consequence, the unit had a weak negative connection from s^{10} and all other connections were zero.

The fourth maze introduces new ambiguities with the additional positions labeled 0 and 9 and two new labels, 1 and 8. No new units were constructed for this maze. Only the weights were modified.

The fifth maze introduces the additional ambiguities labeled 0 and 4. The label 4 definitely requires disambiguation, since the agent should choose different actions (N and S) in the two positions with this label. Since the agent can still move to the goal optimally by moving east in positions labeled 0, no unit to disambiguate this label is necessary. Five new units were created: $l^{24}_{S,4}, l^{25}_{21,9}, l^{26}_{25,9}, l^{27}_{26,9}$, and $l^{28}_{E,9}$. The first disambiguates the positions labeled 4. It has a positive weight from s^9 and negative weights from s^0, s^4, and s^{12}. The second, third, and fourth new units, $l^{25}_{21,9}, l^{26}_{25,9}$, and $l^{27}_{26,9}$ all serve to predict the Q-values in the states labeled 9 more accurately. The last new unit, $l^{28}_{E,9}$, also helps nail down these Q-values and that of the upper position labeled 9.

Though the above example was one of the agent's more fortunate, in which CHILD tested perfectly in the remaining mazes without further training, similar unit construction occurred in the remainder of the mazes for the agent's slower-learning instantiations.

4.2. Non-Catastrophic Forgetting

Continual learning is a process of growth. Growth implies a large degree of change and improvement of skills, but it also implies that skills are retained to a certain extent. There are some skills that we undoubtedly lose as we develop abilities that replace them (how to crawl efficiently, for example). Though it is not the case with standard neural networks that forgotten abilities are regained with less effort than they had originally demanded, this tends be the case with CHILD.

To test its ability to re-learn long-forgotten skills, CHILD was returned to the first maze after successfully learning the ninth. The average number of training steps needed for the 100 cases was about 20% of what were taken to learn the maze initially, and in two-thirds of these cases, no retraining of any kind was required. (That is, the average over the cases where additional learning *was* required was about 60% of the number needed the first time.) The network built on average less than one new unit. However, the average testing performance was 20% worse than when the maze was first learned.

4.3. Distributed Senses

One problem with TTH's is that they have no hidden units in the traditional sense, and their activation functions are linear. Linear activation functions sometimes raise a red flag in the connectionist community due to their inability in traditional networks to compute non-linear mappings. However, transition hierarchies use higher-order connections, involving the multiplication of input units with each other (Equations 7 and 8). This means that the network can in fact compute non-linear mappings. Nevertheless, these non-linearities are constructed from inputs at previous time steps, not from the input at the current time step alone. This is not a problem, however, if the current input is repeated for several time steps, which can be accomplished in reinforcement environments by giving the agent a "stay" action that leaves it in the same state. The agent can then stay in the same position for several time steps until it has made the necessary non-linear discrimination of its input data — simulating the temporal *process* of perception more closely than traditional one-step, feed-forward computations. Since the stay action is treated the same as other actions, it also develops Q-values to reflect the advantage of staying and making a non-linear distinction over taking one of the other, perhaps unreliable actions. Again, context from preceding states can help it decide in which cases staying is worthwhile.

With the additional stay action, CHILD was able to learn the mazes using a *distributed* sense vector consisting of five units: bias, WN (wall north), WW (wall west), WE (wall east), and WS (wall south). The bias unit was always on (i.e., always had a value of 1.0). The other units were on only when there was a wall immediately to the corresponding direction of the agent. For example, in positions labeled 12, the bias, WE, and WS units were on; in positions labeled 7, the bias, WN, WW, and WE units were on, etc. The agent was able to learn the first maze using distributed senses, but required much higher training times than with local senses (averaging nearly 6000 steps).

It turned out, however, that the agent was still able to learn the mazes effectively even without the stay action. In all positions except those labeled 0, the effect of the stay action could be achieved simply by moving into a barrier. Furthermore, the agent could often make the necessary discriminations simply by using the context of its previous senses. Whenever it moved into a new state, information from the previous state could be used for disambiguation (just as with the locally encoded sense labels above). In the distributed case, however, previous sensory information could be used to distinguish states that were in principle unambiguous, but which were in practice difficult to discriminate.

After the agent had learned the first maze, it was transferred to the second in the same continual-learning process as described above. An interesting result was that CHILD was able to generalize far better, and in some training episodes was able to solve *all* the mazes after being trained on only the first *two*. It did this by learning a modified right-hand rule, where it would follow the border of the maze in a clockwise direction until it reached the goal; or, if it first hit a state labeled 0, it would instead move directly east. In one case it did this having created only six high-level units. In most training episodes, more direct routes to the goal were discovered; but the cost was the creation of more units (usually 15–20).

4.4. Limitations

As was mentioned in the introduction, CHILD is not a perfect continual learner and can only learn in a restricted subset of possible environments. Though the TTH algorithm is very fast (exhibiting a speedup of more than two orders of magnitude over a variety of recurrent neural networks on supervised-learning benchmarks (Ring, 1993)), it can only learn a subset of possible finite state automata. In particular, it can only learn Markov-k sequences: sequences in which all the information needed to make a prediction at the current time step occurred within the last k time steps if it occurred at all. Though TTH's can learn k and indefinitely increase it, they cannot keep track of a piece of information for an *arbitrary* period of time. Removing this limitation would require adding some kind of arbitrary-duration memory, such as high-level units with recurrent connections. Introducing the ability to learn push-down automata (as opposed to arbitrary FSA) would even more greatly relax CHILD's restrictions, but would be far more challenging.

Transition hierarchies also have a propensity for creating new units. In fact, if the parameters are set to allow it, the algorithm will continue building new units until all sources of non-determinism are removed, under the assumption that ignorance is the sole cause of all unpredictable events. When the parameters are properly set, only the needed units are created. Since there is no *a priori* way of knowing when an unpredictable event is due to ignorance or to true randomness, different kinds of environments will have different optimal parameter settings. Unnecessary units may also result when tasks change and previously learned skills are no longer useful. Finding a method for removing the extra units is non-trivial because of the difficulty involved in identifying which units are useless. This is more complicated than the problem of weight elimination in standard neural networks. A large hierarchy may be very vital to the agent, and the lower-level units of the hierarchy may be indispensable, but before the hierarchy is completed, the lower units may appear useless.

One problem that faced CHILD in the previous section was that the environments of Figure 1 kept changing. This resulted in constant modification of the reinforcement landscape, and the agent had to re-learn most or all of its Q-values again and again. It might be better and more realistic to use a single environment composed of *multiple layers* of complexity such that the agent, once it has learned some simple skills, can use them to achieve an ever increasing density of reward by continually uncovering greater environmental complexities and acquiring ever more sophisticated skills.

The mazes in Figure 1 were designed to give CHILD the opportunity to perform continual learning. An ideal continual learner would be capable of learning in any environment and transferring whatever skills were appropriate to any arbitrary new task. But, of course, it will always be possible to design a series of tasks in which skills learned in one are not in any way helpful for learning the next — for example, by rearranging the state labels randomly or maliciously. There will always be a certain reliance of the agent on its trainer to provide it with tasks appropriate to its current level of development. This is another argument in favor of the "multiple layers" approach described in the previous paragraph, which may allow the agent smoother transitions between levels of complexity in the skills that it learns.

5. Related Work

Continual learning, CHILD, and Temporal Transition Hierarchies bring to mind a variety of related work in the areas of transfer, recurrent networks, hierarchical adaptive systems, and reinforcement-learning systems designed to deal with hidden state.

5.1. Transfer

Sharkey and Sharkey (1993) discuss *transfer* in its historical, psychological context in which it has a broad definition. In the connectionist literature, the term has so far most often referred to transfer across *classification tasks* in non-temporal domains. But, as with the other methods described in this volume, continual learning can also be seen as an aspect of task transfer in which transfer across classification tasks is one important component.

There seem to be roughly two kinds of transfer in the connectionist literature. In one case, the network is simultaneously trained on related classification tasks to improve generalization in its current task (Caruana, 1993, Yu & Simmons, 1990). In the other case, feed-forward networks trained on one classification task are modified and reused on a different classification task (Baxter, 1995, Pratt, 1993, Sharkey & Sharkey, 1993, Silver & Mercer, 1995, Thrun, 1996). Sometimes the new tasks lie in contradiction to the old tasks, that is, the old task and the new are inconsistent in that a classification made in one task would necessarily be made differently in the other. The autoassociation and bit-displacement tasks explored by Sharkey and Sharkey (1993), as one example, require different outputs for the same input. In other cases there may not be any necessary inconsistency and the tasks might be seen as separate parts of a larger task. For example, the two tasks of recognizing American speech and recognizing British speech, as described by Pratt (1993), are each sub-tasks of recognizing English speech. Continual learning and CHILD introduce into this discussion the issue of context: In certain cases, one mapping may be correct, in others the identical mapping may be incorrect. The issue is thus one of identifying the larger context in which each is valid. Transfer across classification tasks is therefore one aspect of continual learning, where temporal context, incremental development, and hierarchical growth are some of the others.

Another kind of transfer that is not confined to classification tasks is the very general "learning how to learn" approach of Schmidhuber (1994) , which not only attempts transfer in reinforcement domains, but also attempts to *learn how* to transfer (and to learn how to learn to transfer, etc.).

5.2. Recurrent Networks

Transition hierarchies resemble recurrent neural networks such as those introduced by Elman (1993), Jordan (1986), Robinson and Fallside (1987), Pollack (1991), etc., in that they learn temporal tasks using connectionist methods. Most relevant here, Elman showed that his network was capable of learning complex grammatical structures by "starting small" — by imposing storage limitations on the recurrent hidden units and then gradually relaxing them. Elman's interest in learning complex sequences by first learning simpler sequences is closely related to continual learning, though there are a few important differences between his system and CHILD. The first is subtle but important: Elman discovered that his network could *only* learn certain complex sequences if he imposed artificial constraints and then gradually relaxed them; CHILD on the other hand was designed specifically for the purpose of continual learning. As a result, the internals of Elman's net must be externally altered as learning progresses, whereas CHILD detects for itself when, how, and to what degree to increase its capacity. Also, Elman used a fixed-sized network and only two kinds of inputs: simple and complex, whereas CHILD adds new units so that it can grow indefinitely, learning a never-ending spectrum of increasing complexity. Elman's net is therefore limited to a certain final level of development, whereas for CHILD, there is no final level of complexity. It must be emphasized, however, that Elman's intention was to shed light on certain aspects of human cognition, not to develop a learning agent.

Recurrent networks are in general capable of representing a greater range of FSA than are TTH's, but they tend to learn long time dependencies extremely slowly. They also are not typically constructive learning methods, though RCC, Recurrent Cascade Correlation (Fahlman, 1991), and the network introduced by Giles, et al.(1995) are exceptions and introduce new hidden units one at a time during learning. Both these networks are unsuitable for continual learning in that they must be trained with a fixed data set, instead of incrementally as data is presented. In addition, these nets do not build new units into a structure that takes advantage of what is already known about the learning task, whereas TTH units are placed into exact positions to solve specific problems.

Wynne-Jones (1993), on the other hand, described a method that examines an existing unit's incoming connections to see in which directions they are being pulled and then creates a new unit to represent a specific area of this multidimensional space. In contrast, Sanger's network adds new units to reduce only a single weight's error (Sanger, 1991), which is also what the TTH algorithm does. However, Both Wynne-Jones' and Sanger's network must be trained over a fixed training set, and neither network can be used for tasks with temporal dependencies.

5.3. Hierarchical Adaptive Systems

The importance of hierarchy in adaptive systems that perform temporal tasks has been noted often, and many systems have been proposed in which hierarchical architectures are developed by hand top down as an efficient method for modularizing large temporal tasks (Albus, 1979, Dayan & Hinton, 1993, Jameson, 1992, Lin, 1993, Roitblat, 1988, Roitblat, 1991, Schmidhuber & Wahnsiedler, 1993, Singh, 1992, Wixson, 1991). In the systems of Wixson (1991), Lin (1993), and Dayan and Hinton (1995), each high-level task is divided into sequences of lower-level tasks where any task at any level may have a termination condition specifying when the task is complete. Jameson's system (Jameson, 1992) is somewhat different in that higher levels "steer" lower levels by adjusting their goals dynamically. The system proposed by Singh (1992), when given a task defined as a specific sequence of subtasks, automatically learns to decompose it into its constituent sequences. In all these systems, hierarchy is enlisted for task modularization, allowing higher levels to represent behaviors that span broad periods of time. Though Wixson suggests some possible guidelines for creating new hierarchical nodes, none of these systems develop hierarchies bottom-up as a method for learning more and more complicated tasks.

Wilson (1989) proposed a hierarchical classifier system that implied but did not implement the possibility of automatic hierarchy construction by a genetic algorithm. The schema system proposed by Drescher (1991) supports two hierarchical constructs: "composite actions" (sequences of actions that lead to specific goals) and "synthetic items" (concepts used to define the pre-conditions and results of actions). Drescher's goal — simulating early stages of Piagetian development — is most congruous with the philosophy of continual learning, though the complexity of his mechanism makes learning more cumbersome.

Another constructive bottom-up approach is Dawkins' "hierarchy of decisions" (Dawkins, 1976), similar to Schmidhuber's "history compression" (Schmidhuber, 1992), where an item in a sequence that reliably predicts the next several items is used to represent the predicted items in a reduced description of the entire sequence. The procedure can be applied repeatedly to produce a many-leveled hierarchy. This is not an incremental method, however — all data must be specified in advance — and it is also not clear how an agent could use this method for choosing actions.

Macro-operators in STRIPS (see Barr & Feigenbaum, 1981) and *chunking* in SOAR (Laird et al., 1986) also construct temporal hierarchies. Macro-operators are constructed automatically to memorize frequently occurring sequences. SOAR memorizes solutions to subproblems and then reuses these when the subproblem reappears. Both methods require the results of actions to be specified in advance, which makes learning in stochastic environments difficult. For real-world tasks in general, both seem to be less suitable than methods based on closed-loop control, such as reinforcement learning.

5.4. Reinforcement Learning with Hidden State

Chrisman's Perceptual Distinctions Approach (PDA) (Chrisman, 1992) and McCallum's Utile Distinction Memory (UDM) (McCallum, 1993) are two methods for reinforcement learning in environments with hidden state. Both combine traditional hidden Markov model (HMM) training methods with Q-learning; and both build new units that represent states explicitly. UDM is an improvement over PDA in that it creates fewer new units. In an environment introduced by McCallum (1993), CHILD learned about five times faster than UDM (Ring, 1994) and created about one third the number of units. In larger state spaces, such as Maze 9 above, transition hierarchies can get by with a small number of units — just enough to disambiguate which action to take in each state — whereas the HMM approaches need to represent most or all of the actual states. Also, both UDM and PDA must label every combination of sensory inputs uniquely (i.e., senses must be locally encoded), and the number of labels scales exponentially with the dimensionality of the input. Distributed representations are therefore often preferable and can also lead to enhanced generalization (as was shown in Section 4.3).

CHILD learns to navigate hidden-state environments without trying to identify individual states. It takes a completely action-oriented perspective and only needs to distinguish states well enough to generate a reasonable Q-value. If the same state label always indicates that a particular action needs to be taken, then none of the states with that label need to be distinguished. Very recently, McCallum has introduced the U-Tree algorithm (McCallum, 1996), which also takes an action-oriented approach. Like Temporal Transition Hierarchies, U-tree is limited to learning Markov-k domains. Also like TTH's, U-tree learns to solve tasks by incrementally extending a search backwards in time for sensory information that helps it disambiguate current Q-values. It then (like TTH's) keeps track of this information and uses it while moving forward in time to choose actions and update Q-values. But unlike TTH's, this method stores its information in an explicit tree structure that reflects the dependencies of the Q-values on the previous sensory information. It also makes assumptions about the environment's Markov properties that the TTH algorithm does not. In particular, it assumes that all Markov dependencies manifest themselves within a fixed number of time steps. As a result, it can miss dependencies that span more than a few steps. This apparent weakness, however, enables the algorithm to perform statistical tests so that it only builds new units when needed (i.e., only when a dependency within this fixed period actually does exist). The algorithm can thereby create fewer units than the TTH algorithm, which is inclined (depending on parameter settings) to continue building new units indefinitely in the search for the cause of its incorrect predictions.

6. Discussion and Conclusions

Given an appropriate underlying algorithm and a well-developed sequence of environments, the effort spent on early training can more than pay for itself later on. An explanation for this is that skills can be acquired quickly while learning easy tasks that can then speed the learning of more difficult tasks. One such skill that CHILD developed was a small dance that it always performed upon beginning a maze in certain ambiguous positions. The dance is necessary since in these ambiguous states, neither the current state nor the correct actions can be determined.[4] Once an agent had performed the dance, it would move directly to the goal. Often the dance was very general, and it worked just as well in the later mazes as in the earlier ones. Sometimes the skills did not generalize as well, and the agent had to be trained in each of the later environments. However, before training began it would have been extremely difficult for the trainer to know many (if any) of the skills that would be needed. In real-world tasks it is even more difficult to know beforehand which skills an agent will need. This is precisely why continual learning is necessary — to remove the burden of such decisions from the concerns of the programmer/designer/trainer.

The problems that CHILD solves are difficult and it solves them quickly, but it solves them even more quickly with continual learning. The more complicated the reinforcement-learning tasks in Section 3 became, the more CHILD benefited from continual learning. By the ninth maze, the agent showed markedly better performance. (It also showed no sign of catastrophic interference from earlier training, and was in fact able to return to the first maze of the series and solve the task again with only minimal retraining.) Taking a fast algorithm and greatly improving its performance through continual learning shows two things. It shows the usefulness of continual learning in general, and it demonstrates that the algorithm is capable of taking advantage of and building onto earlier training.

CHILD is a particularly good system for continual learning because it exhibits the seven properties of continual learning listed in the introduction.

- It is an autonomous agent. It senses, takes actions, and responds to the rewards in its environment. It handles reinforcement-learning problems well, since it is based on the Temporal Transition Hierarchy algorithm, which can predict continuous values in noisy domains (learning supervised learning tasks more than two orders of magnitude faster than recurrent networks (Ring, 1993)).

- It learns context-dependent tasks where previous senses affect future actions. It does this by adding new units to examine the temporal context of its actions for clues that help predict their correct Q-values. The information the agent uses to improve its predictions can eventually extend into the past for any arbitrary duration, creating behaviors that can also last for any arbitrary duration.

- It learns behaviors and skills while solving its tasks. The "dance" mentioned above, for example, is a behavior that it learned in one maze that it then was able to use in later mazes. In some cases the dance was sufficient for later tasks; in other cases it needed to be modified as appropriate for the new environment.

- Learning is incremental. The agent can learn from the environment continuously in whatever order data is encountered. It acquires new skills as they are needed. New units are added only for strongly oscillating weights, representing large differences in Q-value predictions and therefore large differences in potential reward. Predictions vary the most where prediction improvements will lead to the largest performance gains, and this is exactly where new units will be built first.

- Learning is hierarchical. New behaviors are composed from variations on old ones. When new units are added to the network hierarchy, they are built on top of existing hierarchies to modify existing transitions. Furthermore, there is no distinction between learning complex behaviors and learning simple ones, or between learning a new behavior and subtly amending an existing one.

- It is a black box. Its internals need not be understood or manipulated. New units, new behaviors, and modifications to existing behaviors are developed automatically by the agent as training progresses, not through direct manipulation. Its only interface to its environment is through its senses, actions, and rewards.

- It has no ultimate, final task. What the agent learns now may or may not be useful later, depending on what tasks come next. If an agent has been trained for a particular task, its abilities afterwards may someday need to be extended to include further details and nuances. The agent might have been trained only up to maze number eight in Figure 1 for example (its training thought to be complete), but later the trainer might have needed an agent that could solve the ninth maze as well. Perhaps there will continue to be more difficult tasks after the ninth one too.

In the long term, continual learning is the learning methodology that makes the most sense. In contrast to other methods, continual learning emphasizes the transfer of skills developed so far towards the development of new *skills of ever greater complexity*. It is motivation for training an intelligent agent when there is no final goal or task to be learned. We do not begin our education by working on our dissertations. It takes (too) many years of training before even beginning a dissertation seems feasible. It seems equally unreasonable for our learning algorithms to learn the largest, most monumental tasks from scratch, rather than building up to them slowly.

Our very acts of producing technology, building mechanisms, writing software, is to make automatic those things that we can do ourselves only slowly and with the agony of constant attention. The process of making our behaviors reflexive is onerous. We build our technology as extensions of ourselves. Just as we develop skills, stop thinking about their details, and then use them as the foundation for new skills, we develop software, forget about how it works, and use it for building newer software. We design robots to do our manual work, but we cannot continue to design specific solutions to every complicated, tiny problem. As efficient as programming may be in comparison with doing a task ourselves, it is nevertheless a difficult, tedious, and time-consuming process — and program modification is even worse. We need robots that learn tasks without specially designed software solutions. We need agents that learn and don't forget, keep learning, and continually modify themselves with newly discovered exceptions to old rules.

Acknowledgments

This paper is a condensation of the reinforcement-learning aspects of my dissertation (Ring, 1994), and I would therefore like to acknowledge everyone as I did there. Space, however, forces me just to mention their names: Robert Simmons, Tim Cleghorn, Peter Dayan, Rick Froom, Eric Hartman, Jim Keeler, Ben Kuipers, Kadir Liano, Risto Miikkulainen, Ray Mooney, David Pierce. Thanks also to Amy Graziano, and to NASA for supporting much of this research through their Graduate Student Researchers Program.

Notes

1. A connection may be modified by at most one l unit. Therefore l^i, l^i_{xy}, and l_{xy} are identical but used as appropriate for notational convenience.
2. With a bit of mathematical substitution, it can be seen that these connections, though additive, are indeed higher-order connections in the usual sense. If one substitutes the right-hand side of Equation 7 for l_{ij} in Equation 8 (assuming a unit $n^x \equiv l^x_{ij}$ exists) and then replaces $\hat{w}_{ij}$ in Equation 7 with the result, then

$$\begin{aligned} n^i(t) &= \sum_j s^j(t)[w_{ij}(t) + \sum_{j'} s^{j'}(t-1)\, \hat{w}_{xj'}(t-1)] \\ &= \sum_j [s^j(t) w_{ij}(t) + \sum_{j'} s^j(t) s^{j'}(t-1)\, \hat{w}_{xj'}(t-1)]. \end{aligned}$$

   As a consequence, l units introduce higher orders while preserving lower orders.
3. For this reason, it is sometimes helpful to provide the agent with the proprioceptive ability to sense the actions it chooses. Though this entails the cost of a larger input space, it can sometimes help reduce the amount of ambiguity the agent must face (Ring, 1994).
4. The dance is similar to the notion of a *distinguishing sequence* in automata theory (which allows any state of a finite state machine to be uniquely identified) except that CHILD does not need to identify every state; it only needs to differentiate those that have significantly different Q-values.

References

Albus, J. S. (1979). Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences*, 45:247–293.

Barr, A. & Feigenbaum, E. A. (1981). *The Handbook of Artificial Intelligence*, volume 1. Los Altos, California: William Kaufmann, Inc.

Baxter, J. (1995). Learning model bias. Technical Report NC-TR-95-46, Royal Holloway College, University of London, Department of Computer Science.

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the tenth International Conference*, pages 41–48. Morgan Kaufmann Publishers.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*. Cambridge, MA: AAAI/MIT Press.

Dawkins, R. (1976). Hierarchical organisation: a candidate principle for ethology. In Bateson, P. P. G. and Hinde, R. A., editors, *Growing Points in Ethology*, pages 7–54. Cambridge: Cambridge University Press.

Dayan, P. & Hinton, G. E. (1993). Feudal reinforcement learning. In Giles, C. L., Hanson, S. J., and Cowan, J. D., editors, *Advances in Neural Information Processing Systems 5*, pages 271–278. San Mateo, California: Morgan Kaufmann Publishers.

Drescher, G. L. (1991). *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, Massachusetts: MIT Press.

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

Fahlman, S. E. (1991). The recurrent cascade-correlation architecture. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 190–196. San Mateo, California: Morgan Kaufmann Publishers.

Giles, C., Chen, D., Sun, G., Chen, H., Lee, Y., & Goudreau, M. (1995). Constructive learning of recurrent neural networks: Problems with recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 6(4):829.

Jameson, J. W. (1992). Reinforcement control with hierarchical backpropagated adaptive critics. Submitted to *Neural Networks*.

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. ICS Report 8604, Institute for Cognitive Science, University of California, San Diego.

Kaelbling, L. P. (1993a). Hierarchical learning in stochastic domains: Preliminary results. In *Machine Learning: Proceedings of the tenth International Conference*, pages 167–173. Morgan Kaufmann Publishers.

Kaelbling, L. P. (1993b). Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1094–1098. Chambéry, France: Morgan Kaufmann.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321.

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University. Also appears as Technical Report CMU-CS-93-103.

McCallum, A. K. (1996). Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior, (SAB'96)*.

McCallum, R. A. (1993). Overcoming incomplete perception with Utile Distinction Memory. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 190–196. Morgan Kaufmann Publishers.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, 7:227–252.

Pratt, L. Y. (1993). Discriminability-based transfer between neural networks. In Giles, C., Hanson, S. J., and Cowan, J. D., editors, *Advances in Neural Information Processing Systems 5*, pages 204–211. San Mateo, CA: Morgan Kaufmann Publishers.

Ring, M. B. (1993). Learning sequential tasks by incrementally adding higher orders. In Giles, C. L., Hanson, S. J., and Cowan, J. D., editors, *Advances in Neural Information Processing Systems 5*, pages 115–122. San Mateo, California: Morgan Kaufmann Publishers.

Ring, M. B. (1994). *Continual Learning in Reinforcement Environments*. PhD thesis, University of Texas at Austin, Austin, Texas 78712.

Ring, M. B. (1996). Finding promising exploration regions by weighting expected navigation costs. Arbeitspapiere der GMD 987, GMD — German National Research Center for Information Technology.

Robinson, A. J. & Fallside, F. (1987). The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department.

Roitblat, H. L. (1988). A cognitive action theory of learning. In Delacour, J. and Levy, J. C. S., editors, *Systems with Learning and Memory Abilities*, pages 13–26. Elsevier Science Publishers B.V. (North-Holland).

Roitblat, H. L. (1991). Cognitive action theory as a control architecture. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 444–450. MIT Press.

Sanger, T. D. (1991). A tree structured adaptive network for function approximation in high-dimensional spaces. *IEEE Transactions on Neural Networks*, 2(2):285–301.

Schmidhuber, J. (1992). Learning unambiguous reduced sequence descriptions. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 291–298. San Mateo, California: Morgan Kaufmann Publishers.

Schmidhuber, J. (1994). On learning how to learn learning strategies. Technical Report FKI–198–94 (revised), Technische Universität München, Institut für Informatik.

Schmidhuber, J. & Wahnsiedler, R. (1993). Planning simple trajectories using neural subgoal generators. In Meyer, J. A., Roitblat, H., and Wilson, S., editors, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 196–199. MIT Press.

Sharkey, N. E. & Sharkey, A. J. (1993). adaptive generalisation. *Artificial Intelligence Review*, 7:313–328.

Silver, D. L. & Mercer, R. E. (1995). Toward a model of consolidation: The retention and transfer of neural net task knowledge. In *Proceedings of the INNS World Congress on Neural Networks*, volume III, pages 164–169. Washington, DC.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8: 323-340.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In Touretzky, D. S., Mozer, M. C., and Hasselno, M. E., editors, *Advances in Neural Information Processing Systems 8*. MIT Press.

Thrun, S. & Schwartz, A. (1995). Finding structure in reinforcement learning. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7*, pages 385–392. MIT Press.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College.

Wilson, S. W. (1989). Hierarchical credit allocation in a classifier system. In Elzas, M. S., Ören, T. I., and Zeigler, B. P., editors, *Modeling and Simulation Methodology*. Elsevier Science Publishers B.V.

Wixson, L. E. (1991). Scaling reinforcement learning techniques via modularity. In Birnbaum, L. A. and Collins, G. C., editors, *Machine Learning: Proceedings of the Eighth International Workshop (ML91)*, pages 368–372. Morgan Kaufmann Publishers.

Wynn-Jones, M. (1993). Node splitting: A constructive algorithm for feed-forward neural networks. *Neural Computing and Applications*, 1(1):17–22.

Yu, Y.-H. & Simmons, R. F. (1990). Extra ouput biased learning. In *Proceedings of the International Joint Conference on Neural Networks*. Hillsdale, NJ: Erlbaum Associates.