Hierarchical Reinforcement Learning with Subpolicies Specializing for Learned Subgoals

Bram Bakker[1,2,3] & Jürgen Schmidhuber[1]
[1] IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland, {bram,juergen}@idsia.ch
[2] IAS, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands, bram@science.uva.nl

Abstract

This paper describes a method for hierarchical reinforcement learning in which high-level policies automatically discover subgoals, and low-level policies learn to specialize for different subgoals. Subgoals are represented as desired abstract observations which cluster raw input data. High-level value functions cover the state space at a coarse level; low-level value functions cover only parts of the state space at a fine-grained level. An experiment shows that this method outperforms several flat reinforcement learning methods. A second experiment shows how problems of partial observability due to observation abstraction can be overcome using high-level policies with memory.

Key words
Reinforcement learning, hierarchical reinforcement learning, feedforward neural networks, recurrent neural networks, MDPs, POMDPs, short-term memory

1 Introduction

Reinforcement learning (RL) is an attractive approach for developing controllers for intelligent agents, such as robots, as it allows the agent to *learn* behavior on the basis of sparse, delayed reward signals provided only when the agent reaches desired goals. However, standard RL methods are hard to scale up to larger, more complex tasks. One promising way to scale up RL is *hierarchical* reinforcement learning (HRL). The general idea is that one should plan or learn at multiple levels of abstraction. High-level policies solve the overall task at an abstract level, leaving out many of the details. Low-level policies consider the task at a detailed level, but they solve only parts of the overall task, and they get rewarded for doing just that. In this way, for each level the search space is reduced and temporal credit assignment facilitated. Another advantage is that low-level policies can be re-used easily, either within the same task or in other tasks.

Most studies on HRL assume a given, hardwired hierarchical structure, and they learn policies within this structure [3, 12, 4, 2]. To minimize the designer's role, it is desirable to learn the hierarchy itself as well: "a key open question is how to form task hierarchies automatically" [2]. This paper presents work towards that goal. In our HASSLE algorithm (Hierarchical Assignment of Subgoals to Subpolicies LEarning algorithm), high-level policies identify subgoals that precede overall goals. Simultaneously, low-level policies (subpolicies) learn to reach subgoals set by the higher level. Low-level policies also learn which subgoals they are capable of reaching, and in this way they learn to specialize. The higher levels and the lower levels all learn using essentially standard value function-based RL algorithms. High-level value functions cover the state space at a coarse level; low-level value functions cover only parts of the state space at a fine-grained level.

The next section describes the HASSLE algorithm and relates it to previous work. Section 3 presents experimental results in a simulated office navigation task, with comparisons to several variants of non-hierarchical, "flat" RL. Section 4 focuses on the important problem of partial observability that naturally arises with abstract high-level observations. Here our approach profits from being applicable not only to MDPs (Markov Decision Processes, where memoryless policies and standard value function-based RL are sufficient) but also to POMDPs (partially observable MDPs, requiring policies with memory). Problems of partial observability are overcome through short-term memory in the high-level policy (embodied by an LSTM recurrent neural network [1]). Section 5 concludes with a general discussion of results, limitations, and possible extensions.

2 The HASSLE algorithm

In what follows, for simplicity we assume only two levels in the control hierarchy, and focus on reward-only-at-goal tasks. However, the algorithm should be extendible to more levels and non-episodic tasks and continual rewards.

An attractive feature of the HASSLE algorithm is that both high-level and low-level policies may use standard value function-based RL algorithms such as Q-learning [7]. We use Advantage learning [6] on both levels, because it works well with function approximators. The higher level value function covers the complete state space at a coarse level. It is updated based on "real" rewards r, received via interaction with the environment. Low-level value functions cover only parts of the overall state space, at a fine-grained level. They are updated based on "local" rewards

[3] B.B. was funded by a CSEM research grant to IDSIA and by an NWO research grant to IAS.

r^L provided by the high-level policy.

2.1 High-level policy learning

In our algorithm, high-level policies and low-level policies learn simultaneously, and both start without having learned anything. This means that a high-level policy cannot simply select, as its action, one of the low-level policies: in the beginning of learning, a low-level policy does not yet do anything meaningful or consistent. Therefore, each action of a higher-level policy π^H is the selection of a "subgoal" to be reached by a lower-level policy π^L. Possible subgoals are taken from a set o^H of abstract high-level observations which differs from (and is smaller than) the set o^L of "primitive" low-level observations; o^H is also the set of possible inputs to the higher-level policy. We use abstract observations rather than primitive observations, because HRL should reduce the search space of the higher-level policy [3] (details and possible limitations of this approach will be discussed below).

A high-level time step (incrementing t^H by 1) corresponds to a high-level observation change. At any t^H, the high-level policy π^H receives the current high-level observation o^H_s as its input; its action is the selection of another high-level observation o^H_g as the current subgoal, using standard Boltzmann exploration [7]. Essentially, it selects the high-level observation that it wants to see next.

π^H can learn no matter whether the desired subgoal o^H_g or a different $o^H \neq o^H_g$ was reached. In the latter case it simply learns as if the reached o^H was the desired subgoal. In both cases let us denote the reached observation by o^H_r. This difference to standard RL reflects the fact that high-level "actions" are just "desired high-level observations" rather than traditional actions, and allows for efficient use of all experiences. If the high-level policy values are implemented using a table (as in the first experiment described below), the update of the Advantage value $A^H(o^H_s, o^H_r)$ corresponds to

$$\Delta A^H(o^H_{s,t^H}, o^H_{r,t^H}) = \alpha^H[V^H(o^H_{s,t^H}) + \frac{r_{t^H} + \gamma_H V^H(o^H_{s,t^H+1}) - V^H(o^H_{s,t^H})}{\kappa^H} - A^H(o^H_{s,t^H}, o^H_{r,t^H})] \quad (1)$$

where $V^H(o^H_{s,t^H}) = \max_{o^H_g} A^H(o^H_{s,t^H}, o^H_g)$, α^H is a learning rate parameter, γ_H is a discount parameter, and κ^H is a constant scaling the difference between values of optimal and suboptimal subgoals (if $\kappa^H = 1$, the equations reduce to Q-learning). In addition, if the subgoal was not reached,

$$\Delta A^H(o^H_{s,t^H}, o^H_{g,t^H}) = \alpha^H[0 - A^H(o^H_{s,t^H}, o^H_{g,t^H})]. \quad (2)$$

The rationale behind this update is that failure to reach the subgoal indicates that the subgoal may not be a good one at this moment (perhaps because it is unreachable), so it should be penalized.

2.2 C-values

It is the job of the low-level policies to reach the subgoal selected by the high-level policy. There is a limited set of low-level policies π^L_i. None of them is initially associated with any of the subgoals o^H_g or any of the possible o^H_s. Instead, this association is learned. Every π^L_i contains a table of so-called C-values of (o^H_s, o^H_g) pairs, each of which represents the "Capability" of π^L_i to reach subgoal o^H_g (Goal) from high-level observation o^H_s (Start). Following a high-level observation change, one π^L_i is selected (using Boltzmann exploration) based on the C-values of all low-level policies for the current (o^H_s, o^H_g) pair. This one then attempts to reach the current subgoal o^H_g. If it does indeed reach o^H_g, its C-value for this (o^H_s, o^H_g) pair is increased, making future selection of this low-level policy in this context more likely. Otherwise the C-value is decreased.

Just like the high-level Advantage values, C-values can always be updated as if the actually reached o^H_r was the desired subgoal. If low-level policy i was the active one, then

$$\Delta C_i(o^H_{s,t^H}, o^H_{r,t^H}) = \alpha^C_r[\gamma_C^{t^L_r - t^L_s} - C_i(o^H_{s,t^H}, o^H_{r,t^H})] \quad (3)$$

where α^C_r is a learning rate, γ_C a parameter which specifies to what extent a subpolicy's capability of reaching a subgoal should be discounted with each additional time step needed to reach it, t^L_s is the low-level time step at which the current low-level policy was started, and t^L_r the one at which o^H_r was reached. In addition, if the subgoal was not reached,

$$\Delta C_i(o^H_{s,t^H}, o^H_{g,t^H}) = \alpha^C_n[0 - C_i(o^H_{s,t^H}, o^H_{g,t^H})] \quad (4)$$

where α^C_n is another learning rate. In the experiments α^C_n will be an order of magnitude smaller than α^C_r, because in initial learning stages low-level policies rarely reach desired subgoals. If these learning rates were equal, it would be hard to selectively boost C-values of mediocre low-level policies that work just a little bit better than others on this subgoal, because the C-values of all low-level policies would be pulled down more often than up. This would make it more difficult for any of the π^L_i to become the specialist for this subgoal and learn it effectively.

Thus, a low-level policy may learn that from some start situation it can reach subgoal A but not subgoal B. Another low-level policy may learn the opposite. This realizes *specialization*. But a single low-level policy may also learn that its capability encompasses multiple (o^H_s, o^H_g) pairs. This realizes *generalization*. The idea is that the low-level policies will specialize when they have to, but generalize when they can. After all, one can easily imagine cases where very similar behaviors will work well for different start/goal combinations. For example, a "leave this room" low-level policy may work well for diverse rooms with diverse neighboring places: generalization. But the "make coffee" subtask should not interfere with the "leave this room" strategy: specialization.

2.3 Low-level policy learning

Besides the update of its C-value, the active low-level policy itself receives a positive reward if it reaches the subgoal, such that it becomes better at reaching it, or it gets zero reward otherwise. In either case the high-level policy can now select a new subgoal, and the process repeats itself.

The learning rule for the low-level policies is even closer to standard Advantage learning than it is for high-level policies. A learning update is done at every primitive, low-level time step t^L. In the experiments we use function approximators to represent the low-level policies' value functions. The weight updates for the currently active low-level policy π_i^L are

$$\begin{aligned}\Delta w_{i,m} =&\alpha^L[V_i^L(o_{t^L}^L) + \frac{r_{t^L}^L + \gamma_L V_i^L(o_{t^L+1}^L) - V_i^L(o_{t^L}^L)}{\kappa^L} \\ &- A_i^L(o_{t^L}^L, a_{t^L}^L)]\frac{\partial A_i^L(o_{t^L}^L, a_{t^L}^L)}{\partial w_{i,m}}\end{aligned} \tag{5}$$

where $V_i^L(o_{t^L}^L) = \max_{a^L} A_i^L(o_{t^L}^L, a^L)$, $a_{t^L}^L$ is the current primitive action (selected using Boltzmann exploration), and and α^L, γ_L, and κ^L are constants. In the experiments we set $r^L = 1$ if the subgoal is reached and $r^L = 0$ otherwise. We use simple linear feedforward neural networks as function approximators: low-level policies compute $A_i^L(o_{t^L}^L, a_k^L)$ for each low-level action a_k^L as a weighted sum of all o_m^L, the elements of the current observation vector. This means that the partial derivative of the Advantage value with respect to weight $w_{i,m}$ in eq. 5 equals o_m^L.

We use function approximators for low-level policies, because in this way each low-level policy can learn to focus on those parts of the low-level observation space which are relevant to its specialization [4], and learn to generalize within that specialization. The current (o_s^H, o_g^H) pair should be part of its input vector, because even though the low-level policy does not know the high-level policy's overall "plan", it should know the high-level policy's current "command" [3]. With the high-level command as part of the input vector, a low-level policy can learn somewhat flexible behavior for various situations in which it is the specialist. For example, a "leave this room" subpolicy may be usable in various rooms although behavior and value function may be somewhat room-dependent.

2.4 Producing high-level observations

So far we have ignored the issue of how to arrive at abstract high-level observations. Here HASSLE is not constrained to any particular method. The main requirement is that a clustering of primitive, low-level observations is accomplished such that neighboring low-level states tend to be clustered together. This allows the high-level policy to select meaningful subgoals, and the low-level policies to develop non-trivial behavior leading the system from subgoal to subgoal.

In the experiments we use a simple unsupervised learning vector quantization technique called ARAVQ (Adaptive Resource Allocation Vector Quantization [8]). It adaptively allocates a new model vector to classify a continuous-valued input vector if the latter's Euclidean distance to any of the existing model vectors exceeds a parameter δ. The version used here is a simplified version without ARAVQ's original stage of determining whether a sequence of consecutive inputs is sufficiently stable, and without its online adaptation of existing model vectors.

As pointed out by several researchers [3, 4], state/observation abstraction can easily lead to partial observability or hidden state at higher levels. Hidden state can be both a blessing and a curse [9]. On the one hand, it reduces the search space and aids generalization; on the other hand, it may confound different states in which different actions are required. We will consider instances of both cases below.

2.5 Related work

HASSLE shares the idea of finding subgoals associated with observations with several other HRL algorithms which learn the hierarchical structure [5, 13, 11, 10]. These algorithms, however, associate subgoals with primitive, low-level observations rather than high-level observations. As explained above, there are important advantages to having the higher levels abstract in observation space and action space. Those HRL algorithms that do exploit the advantages of high-level observations (notably Feudal RL [3]) are limited to *predesigned* hierarchies, and do not offer solutions to problems of partial observability on higher levels.

Previous HRL approaches also differ in how they assign low-level policies to specific subgoals. For example, Feudal RL [3] distributes low-level policies evenly over the entire state space, such that at each level each possible start/subgoal pair has exactly one responsible low-level policy. One disadvantage is that this usually implies that many low-level policies and value functions need to be stored and learned. Other approaches identify a limited set of subgoals or subtasks in an initial design phase [12, 4] or learning phase [10], and assign one low-level policy to each subgoal or subtask. Thereafter low-level policies are used by high-level policies as though they were normal (but temporally extended) actions. However, this approach depends on significant knowledge of or initial experience with the overall task.

In contrast, HASSLE and HQ-learning [13] and the SSS algorithm [11] simultaneously learn subgoals and low-level policies specializing on certain subgoals (low-level observations for HQ and SSS but not for HASSLE). Only a limited set of low-level policies is available for the whole state space and all subgoals. However, HQ and SSS do not use local reward functions. Instead the value functions of their low-level policies strongly depend on those of subsequently invoked low-level policies. For this reason, HQ's low-level policies can be used only once per episode, and SSS's can be used multiple times only when the various

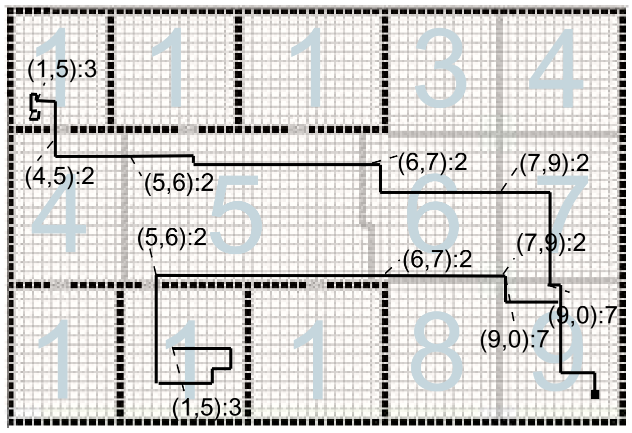

Figure 1. The first office task. The goal is the black dot in the lower right corner. There are 6 rooms, connected by doors to a central corridor. Large grey numbers indicate 9 possible high-level observations extracted by ARAVQ; 14 fields with grey boundaries mark areas in which high-level observations do not change. See text for details.

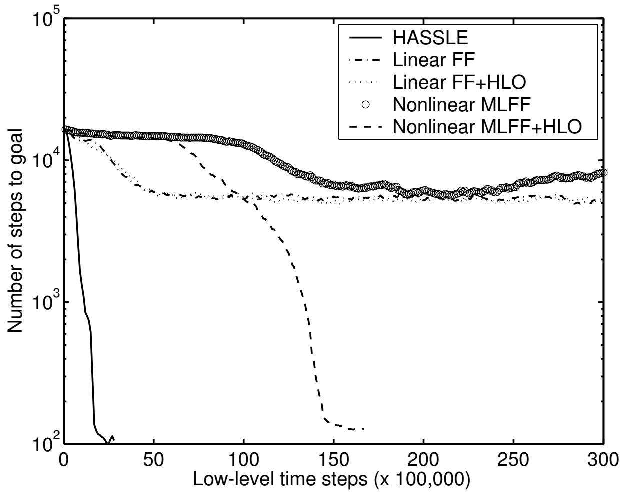

Figure 2. Average numbers of low-level actions (note the log scale) needed to reach the goal in the first office task, as functions of learning time (low-level time steps).

trigger situations are very similar in value and appearance. This is regrettable as one would like to be able to reuse low-level policies, say "leave this room", everywhere in the environment, close to and far from the goal, etc. HASSLE can easily invoke its subpolicies again and again.

3 Experiment 1: office navigation

3.1 Task setup and comparison with flat RL

To demonstrate and test HASSLE we consider a navigation task in a simulated "office" gridworld. The agent should learn to move from any possible start position to a fixed goal position (see Figure 1). Its possible orientations are north, east, south, west. It has three primitive low-level actions: make a step in the current direction, turn left 90°, turn right 90°. It can move through doors but not through walls. The number of states is 10,973. Each episode starts with a random position and orientation. It ends once the agent reaches the goal (where it receives the only reward $r = 4$), or after at most 20,000 low-level actions (episode time-out). Random walk finds the goal in around 24% of the episodes.

Four simulated directed "sonar" sensors (one for each direction relative to the current orientation) measure the current distance (in number of grid cells) to the nearest wall/door. Four additional directed binary sensors detect the presence of doors, and four more the presence of the goal, provided it is at most 8 grid cells away. The agent's orientation is part of the primitive low-level observation vector. With these sensors the task is an MDP, because the distance to walls or doors has a unique value for each position.

The high-level observations are produced online by applying ARAVQ to the 4-dimensional output of the distance sensors, ignoring the current orientation, using $\delta = 32$. Figure 1 shows how ARAVQ distributes high-level observations over the state space (large grey numbers). There is also one high-level observation that corresponds to the goal, such that the high-level policy can learn to select it as a subgoal once the agent is close enough. This yields a total of 10 high-level observations.

To make the task more realistic, sensor and actuator noise was added. Each of the 4 distance sensors adds independent Gaussian noise (standard deviation .05) to the actual distance at each low-level time step, and with probability .01 one of the sensors detecting door or goal has the wrong value. With probability .05, a random high-level observation is perceived rather than the correct one. Furthermore, with probability .01 primitive turn left/right actions turn too far or fail to turn, and primitive move forward actions move in a random direction.

π^H consists of a table of 10 (o_s^H) by 10 (o_g^H) A^H-values ($\alpha^H = .02$, $\gamma_H = .95$, and $\kappa^H = .2$). We use 8 π_i^L, each using a table of 10 by 10 C_i-values ($\alpha_r^C = .002$, $\alpha_n^C = .00004$, $\gamma_C = .99$) and a linear function approximator with 3 outputs encoding the A_i^L-values of the low-level actions, and a 36-element input vector (the 12 sensors described above, 4 bits to encode the agent's orientation, 10 bits to encode o_s^H, and 10 bits to encode o_g^H). We use $\alpha^L = .002$, $\gamma_L = .95$, $\kappa^L = .2$. The time-out value for low-level policies is 100, i.e. a low-level policy may execute at most 100 low-level actions to reach its assigned subgoal. All parameters here and below were found by a coarse search in parameter space.

We compare the HASSLE system to 4 flat RL systems. The first is a single linear feedforward neural net similar to HASSLE's low-level policies, also trained using Advantage learning. Note that the nature of the primitive observations makes a table-based system inappropriate. This system's input and output vectors are like those for a HASSLE low-level policy (see above), except that the 20 input bits for encoding o_s^H and o_g^H are omitted. The

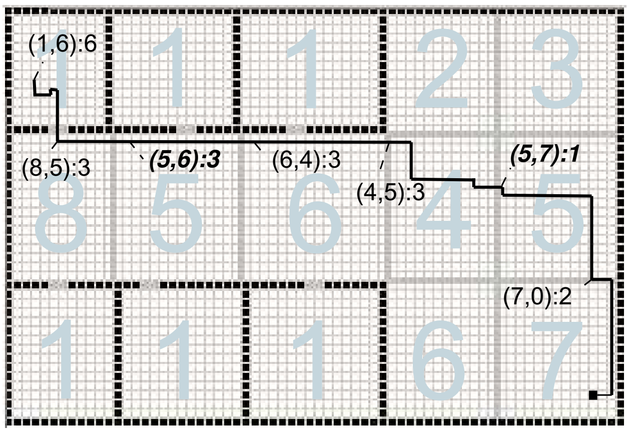

Figure 3. The second office task. Like in Figure 1, but with different HLOs that introduce problematic partial observability on the higher level (high-level observations 5 and 6).

learning parameters are $\alpha = .0001$, $\gamma = .99$, and $\kappa = .2$. One may argue that the task is just too complex for a simple linear function approximator. For this reason, we replace it with a nonlinear one: a multilayer feedforward neural network (MLFF) approximating the value function. The network is trained using the same learning algorithm and inputs and outputs as the linear function approximator. It has 15 hidden units and uses $\alpha = .002$, $\gamma = .99$, and $\kappa = .2$. Finally, we extend the input vectors of both flat systems by 10 additional bits encoding o_s^H as computed by ARAVQ, to investigate the utility of one HASSLE aspect, namely, spatial and temporal abstraction of observations, without interference from HASSLE's other features. Learning parameters are left unchanged.

3.2 Results

Figure 2 plots the average number of primitive actions needed to reach the goal as a function of the total number of low-level time steps. All curves are averages of 10 runs. HASSLE converges to roughly 100 low-level actions per episode within around 1.5 million low-level time steps. Such policies are close to optimal, as determined by studying the agent's trajectories. Flat linear systems with and without high-level observations (HLOs) do achieve significant performance improvements compared to a random walk (they reach the goal in many more cases), but never come close to HASSLE. The first nonlinear system (MLFF) performs poorly, but the one *with* high-level observations (MLFF + HLO) does quite well and its final performance is not much worse than HASSLE's. It takes much longer though. But this shows that a single memoryless policy can indeed learn the task reasonably well, and that high-level observations are helpful even for an otherwise flat RL system, probably because they provide useful "global" information about the agent's position relative to the goal.

Figure 1 shows trajectories of a HASSLE agent after

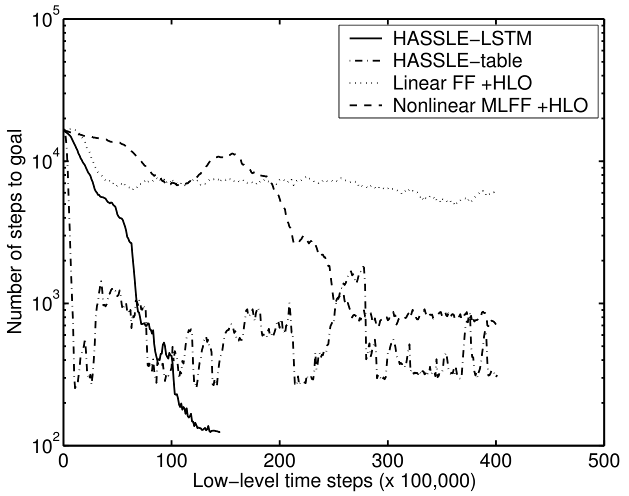

Figure 4. Average numbers of low-level actions (note the log scale) needed to reach the goal in the second office task, as functions of learning time (low-level time steps).

learning. The invocation of new subgoals and subpolicies is indicated by labels of the form (o_s^H, o_g^H): *subpolicy index*. For example, in the upper left room, $o_s^H = 1$, subgoal $o_g^H = 5$ is selected, and specialized subpolicy 3 is started. This subpolicy has learned to look for doors and go through them. Following the trajectories, one can see how the high-level policy selects proper subgoals following each high-level observation change, as well as examples of specialization and generalization of subpolicies. A single subpolicy always specialized on leaving rooms, and usually this was all it could do. Other specialists learned to move through the corridor, and usually there was a specialist for moving to the goal once it was close enough.

4 Experiment 2: office navigation with partial observability problem

4.1 Task setup

Experiment 2 is designed to address the most important objection against observation abstraction at higher levels (see Introduction and Section 2.4): it naturally leads to problems of partial observability where important sensory information is thrown away, such that the optimal high-level action is no longer a function of the current high-level observation (POMDP) [3, 9, 4, 1]. We use exactly the same office gridworld navigation task as before, but handcraft new high-level observations (see Figure 3) in which this problem occurs. Now there are two high-level states which cause high-level observation 5. The first requires selection of subgoal 6, which should move the agent east. The second requires selection of subgoal 7, which should move the agent south (likewise for high-level observation 6).

A memoryless high-level policy cannot deal with this ambiguity. Therefore, we use an RL Long Short-Term Memory (RL-LSTM) recurrent neural network [1] as the function approximator for the high-level value function. Using the recurrent activations, the network can learn to memorize important past high-level observations which disambiguate the current high-level state. For instance, if the agent just came out of a room in our office environment and it sees high-level observation 5, it can use its memory to infer where it is and produce the correct high-level values. This high-level LSTM policy uses the same Advantage RL algorithm as the high-level table-based policy of Experiment 1. It has 27 standard hidden units and 3 memory cells. Its learning parameters are: $\alpha^H = .01$, $\gamma_H = .95$, and $\kappa^H = .2$. See [1] for details of RL-LSTM and the update equations. All other parameters of the HASSLE system remain unchanged.

4.2 Results

Figure 4 shows learning results of HASSLE-LSTM. Also shown are results of HASSLE with a table-based high-level policy, and the two flat RL systems with high-level observations (for the flat systems without high-level observations the situation does not change; see figure 2). In all runs HASSLE with LSTM learned very good policies, almost as good as those found in Experiment 1 (but not quite, mainly because an episode may begin with one of the ambiguous high-level observations, so the system cannot yet know an optimal subgoal). Learning (until 120 low-level actions per episode were reached) took around 10 times longer than in experiment 1, which is not surprising as now appropriate internal memories must also be learned. Figure 3 shows a typical agent's trajectory, demonstrating its ability to select correct subgoals despite ambiguous high-level observations. Note the italic bold-faced labels of the form (o^H_s, o^H_g): *subpolicy index*: the same $o^H_s = 5$ invokes different subgoals and subpolicies.

The table-based HASSLE system performs reasonably well in this task, considering its limitations, but obviously not as well as the LSTM-based HASSLE system, because it often could not select the optimal subgoal. None of the flat systems has good performance, not even the non-linear system with high-level observations (MLFF + HLO), presumably because the usefulness of the high-level observations is reduced by the increased ambiguity.

5 Conclusions

HASSLE, our new hierarchical RL algorithm, outperformed variants of plain RL in two experiments. It did so by learning to create both useful subgoals and the corresponding specialized subtask solvers. The subgoals classify states under loss of information. In the second experiment, the naturally arising problem of partial observability was overcome by memory-based policies at higher levels. Current limitations include the large number of parameters, the lack of strict convergence guarantees (although the system uses standard RL algorithms for which some convergence theorems exist), and the dependence on identifying reasonable high-level observations.

Ongoing work aims at *adapting* high-level observations online so as to improve their utility as subgoals and to possibly prevent the problem of partial observability arising with state abstraction. Finally, we are looking at ways to show convergence, probably for the slightly simplified case of using table-based representations at each level.

References

[1] B. Bakker. Reinforcement learning with Long Short-Term Memory. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

[2] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems, Special issue on reinforcement learning*, 13:41–77, 2003.

[3] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, 1993.

[4] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[5] B. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In *Proc. Conf. Simulation of Adaptive Behavior*, 1996.

[6] M. E. Harmon and L. C. Baird. Multi-player residual advantage learning with general function approximation. Technical report, Wright-Patterson Air Force Base, 1996.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[8] F. Linåker and H. Jacobsson. Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond. In *Proc. of IJCAI'2001*, 2001.

[9] R. A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Computer Science Department, July 1997.

[10] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proc. 18th International Conf. on Machine Learning*, 2001.

[11] R. Sun and C. Sessions. Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors. *IEEE Trans. on Systems, Man, and Cybernetics*, 30:403–418, 2000.

[12] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[13] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6:2, 1997.