16

Solving POMDPs with Automatic Discovery of Subgoals

Le Tien Dung, Takashi Komeda and Motoki Takagi
Shibaura Institute of Technology
Japan

1. Introduction

Reinforcement Learning (RL) is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment (Kaelbling et al., 1996). At any time step, the environment is assumed to be at one state. In Markov Decision Processes (MPDs), all states are fully observable in which the agent can choose a good action based only on the current sensory observation. In Partially Observable Markov Decision Processes (POMDPs), any state can be a hidden state in which the agent doesn't have sufficient sensory observation and the agent must remember the past sensations to select a good action. Q-learning is the most popular algorithm for learning from delayed reinforcement in MDPs, and RL with Recurrent Neural Network (RNN) can solve deep POMDPs.

Several methods have been proposed to speed up learning performance in MDPs by creating useful subgoals (Girgin et al., 2006), (McGovern & Barto, 2001), (Menache et al., 2002), (Simsek & Barto, 2005). Subgoals are actually states that have a high reward gradient or that are visited frequently on successful trajectories but not on unsuccessful ones, or that lie between densely-connected regions of the state space. In MDPs, to attaint a subgoal, we can use a plain table based policy, named a skill. Then these useful skills are treated as options or macro actions in RL (Barto & Mahadevan, 2003), (McGovern & Barto, 2001), (Menache et al., 2002), (Girgin et al., 2006), (Simsek & Barto, 2005), (Sutton et al., 1999). For example, an option named "going to the door" helps a robot to move from any random position in the hall to one of two doors. However, it is difficult to apply directly this approach to RL when a RNN is used to predict Q values. Simply adding one more unit into output layer to predict Q values for an option doesn't work because updating any connection's weight will affect all previous Q values and because it is easy to lose the Q values when the option can't be executed for a long time.

In this chapter, a method named Reinforcement Learning using Automatic Discovery of Subgoals is presented towards this approach but in POMDPs. We can reuse existing algorithms to discover subgoals. To obtain a skill, a new policy using a RNN is trained by experience replay. Once useful skills are obtained by RNNs, these learned RNNs are integrated into the main RNN as experts in RL. Results of experiment in two problems, the E maze problem and the virtual office problem, show that the proposed method enables an agent to acquire a policy, as good as the policy acquired by RL with RNN, with better learning performance.

Source: Theory and Novel Applications of Machine Learning, Book edited by: Meng Joo Er and Yi Zhou,
ISBN 978-3-902613-55-4, pp. 376, February 2009, I-Tech, Vienna, Austria

2. Reinforcement learning and recurrent neural network

RL is learning what to do to maximize a numerical reward signal. The learner is not told which actions to take, as in supervised learning, but instead must discover which actions yield the most reward by trying them (Sutton & Barto, 1998). RL allows a software agent to automatically determine its behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal. This automated learning scheme implies that there is little need for a human expert who knows about the domain of application. Much less time will be spent designing a solution, since there is no need for hand-crafting complex sets of rules as with expert systems (Champandard, 2007). Beyond the agent and the environment, four main sub-elements of a reinforcement learning system are a policy, a reward function, a value function, and, optionally, a model of the environment.

In the general case of a reinforcement learning problem, the agent's actions determine not only its immediate reward, but also the next state of the environment. The agent has to be able to learn from delayed reinforcement: it may take a long sequence of actions, receiving insignificant reinforcement, and then finally arrive at a state with high reinforcement. I.e., the agent must be able to learn which of its actions are desirable based on reward that can take place arbitrarily far in the future.

2.1 Q learning

When a model of fully observable environment is not available, the most widely used RL is Q-learning (Sutton & Barto, 1998), (Watkins, 1989). Q-learning iteratively approximates the state-action value function by updating its plain table based Q values as

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

where $Q(s,a)$ is the state-action value for action a in state s, α is learning rate, and γ is the factor of discount.

Based on Q values, the agent selects an action to execute using a standard exploration method. A simple selection rule, ε-greedy method, is to behave greedily most of the time. The agent selects the action with highest estimated action value with a big probability. But sometimes, the agent selects an action at random with small probability. An alternative solution, softmax selection rule, is to vary the action probabilities as a graded function of estimated value (Sutton & Barto, 1998).

2.2 Perceptual aliasing

In a real world environment, not all states are fully observable, named hidden states. I.e., in some world states, the observation is same, but the optimal actions are different. These problems are called POMDPs. Fig. 1 shows an example of POMDPs (Ohta et al., 2003). If the agent selects the same action at a hidden state, it may not reach the goal. To solve this kind of problems, some researchers propose memory-less approach. This method is simply to avoid passing through hidden states or uses stochastic action selection rule at hidden states (Littman, 1994), (Ohta et al., 2003). However, this approach doesn't fit well when there are many perceptual aliasing states in the optimal policy. Memory based approach uses past sensations to predict optimal actions (McCallum, 1995), (Whitehead, 1995).

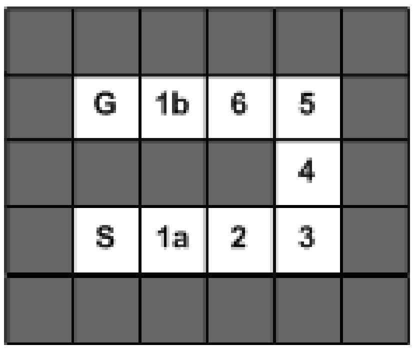

Fig. 1. A maze including perceptual aliasing.

2.3 RL with RNN

A number of researchers use a RNN to predict Q values to solve POMDPs (Bakker, 2002), (Ballini, 2001), (Gomez et al., 2005), (Gomez & Schmidhuber, 2006), (Le et al., 2007), (Le et al., 2008), (Lin & Mitchell, 1993), (Lin, 1993), (Ho & Kamel, 1994), (Onat et al., 1998), (Schafer & Udluft, 2005). The number of input unit is equal to the dimension of the sensory inputs from the environment. The number of output units is equal to the number of possible actions. Each output represents the Q-value of the associated action. For the output units, a linear activation function is used to cope with the required range of Q values. At a time step, RNN inputs are the sensations of the current state and RNN outputs are approximated Q values for that state. Similar as plain table based Q learning, the agent selects an action based on Q values and observes new Q values. However, the agent doesn't update the Q values as in (1). It updates the RNN connections' weights to predict the Q values.

RNN architecture can be Time-Delayed Neural Network, Elman Network, Recurrent Neurofuzzy Network, or Long Short Term Memory Network (LSTM). The RNN used in this chapter is LSTM which has been proved to have a strong ability to solve many difficult tasks (Gers et al., 2000), (Gers et al., 2002), (Hochreiter & Schmidhuber, 1997), (Schmidhuber et al., 2007). The basic element in LSTM is memory cell (Fig. 2), which contains a recurrently self-connected liner unit called the "Constant Error Carousel" or CEC (Hochreiter & Schmidhuber, 1997). A multiplicative input gate unit is employed to protect the memory contents from perturbation by irrelevant inputs. Likewise, a multiplicative output gate unit is used to protect other units from perturbation by currently irrelevant memory contents. F. A. Gers improves the limitation of traditional LSTM by adding a forget gate to memory cell (Gers et al., 2000). Standard LSTM (or LSTM for short) with forget gates can learn to reset the memory contents that are out of date.

The activations are computed as follows:

For each unit i in hidden layer at time t, the net input is

$$net_i(t) = \sum_m w_{im} y_m(t-1) \tag{2}$$

where w_{im} is the weights of the connection from unit m to unit i.

The hidden unit activation y_h, input gate activation y_{in}, output gate activation y_{out}, and the forget gate activation y_φ is calculated by

$$y_i = f(net_i(t)) \tag{3}$$

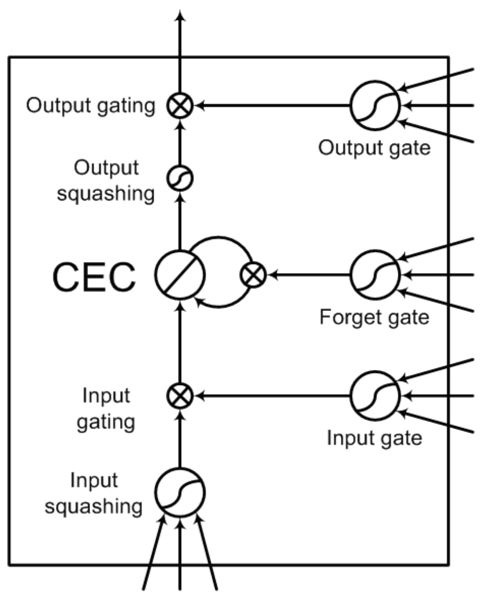

Fig. 2. A memory cell with its gates

where *f* is the standard logistic sigmoid function

$$f = \frac{1}{1+e^{-x}} \tag{4}$$

The CEC activation s_{c_j} is calculated by

$$s_{c_j}(t) = y_{\varphi}(t)s_{c_j}(t-1) + y_{in}(t)g(net_{c_j}(t)) \tag{5}$$

where g is the logistic sigmoid function scaled to [-2, 2]

$$g = \frac{4}{1+e^{-x}} - 2 \tag{6}$$

The memory cell output activation y_{c_j} is calculated by

$$y_{c_j}(t) = y_{out_j}(t)h(s_{c_j}(t)) \tag{7}$$

where h is the logistic sigmoid function scaled to [-1, 1]

$$h = \frac{2}{1+e^{-x}} - 1 \tag{8}$$

Finally, the output unit activation y_k is calculated by

$$y_k = f_k(net_k(t)) \tag{9}$$

where f_k is the standard logistic sigmoid function or the identity function.

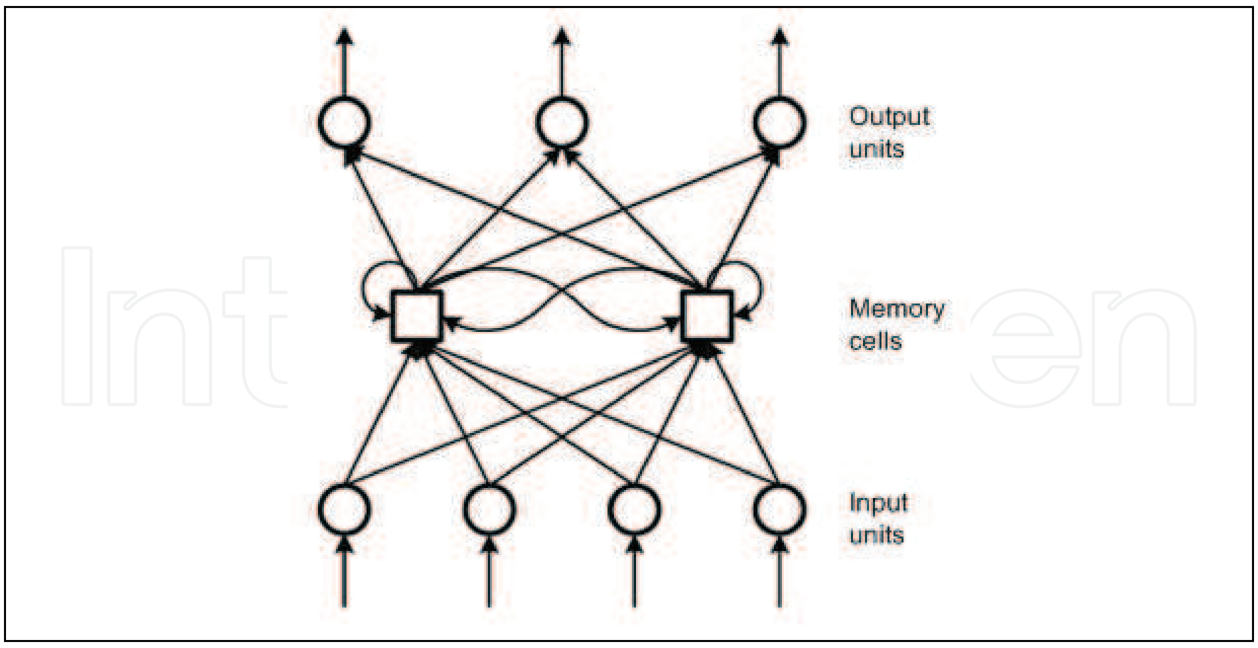

Fig. 3. A typical LSTM network with 4 input units, 2 memory cells, and 3 output units.

Several LSTM network topologies are proposed to solve some particular problems (Gers et al., 2000), (Gers et al., 2002), (Hochreiter & Schmidhuber, 1997), (Schmidhuber et al., 2007). Fig. 3 shows a typical LSTM network architecture with 4 input units, 2 memory cells, and 3 output units.

3. Solving POMDPs with automatic discovery of subgoals

Reinforcement Learning using Automatic Discovery of Subgoal (RLSG) is a framework to accelerated learning ability in non-Markovian problems. There are two phases in RLSG. The first phase is to obtain online generated sub-policies. One sub-policy is a useful skill to attain subgoals. Each sub-policy has its own RNN which plays a role of Recurrent Neural SubNetwork (RSN) in RLSG. The second phase in RLSG is to integrate online generated RSNs into the main RNN. The new RNN will be trained to predict the Q values for the original problem.

3.1 Automatic discovery of subgoal

Several methods have been proposed to discover subgoals (Girgin et al., 2006), (McGovern & Barto, 2001), (Menache et al., 2002), (Simsek & Barto, 2005). Subgoals are actually states that have a high reward gradient or that lie between densely-connected regions of the state space. In our system, a state will be considered as a subgoal if it is visited frequently on successful trajectories but not on unsuccessful trajectories. A simple method to find a subgoal is to calculate the probability of being a subgoal for all states by:

$$P(s) = P(s \in B^{+}) * \left[1 - P(s \in B^{-})\right] \tag{10}$$

where B^{+} and B^{-} are successful bag and unsuccessful bag. The states with high probability and not surrounding the starting and ending states are considered as subgoals.

3.2 Learning a skill

After finding a subgoal, a new policy should be trained to attain it. Most of previous researchers assume that the environment is observable therefore they can use a plain table to store Q values for the new policy (Girgin et al., 2006), (McGovern & Barto, 2001), (Menache et al., 2002), (Simsek & Barto, 2005). However, it can’t be done in POMDPs. In our system, a policy using a RNN is trained to attain this subgoal by experience replay as described in (Lin, 1992) with a pseudo reward function once we can find a subgoal.

3.3 Integration

In full observable environments, we can use skills as options in RL by adding these options to the action set of the agent. These options are considered as macro actions. New elements are inserted into the table to predict Q values of these macro actions. Update one value in a table doesn’t affect the others. However, it is hard to apply directly these methods in RL with RNN because update a neural network connection’s weight will affect all other previous Q values. Furthermore, only in some parts of the whole state space, we can execute an option. That means if we add one more output unit for an option, it is very easy to lose the previous learned Q values for that option.

In this chapter, our proposed method Reinforcement Learning using automatic discovery of SubGoals (RLSG) doesn’t use generated skills as options but as components of the RNN to predict Q value for the main policy. All elements of previous learned neural networks except input and output units are integrated into the main RNN. Our previous works show that it is possible to speed up learning performance by reuse previous policies in similar tasks (Le et al., 2007). Similar as in supervised learning, the new RNN is composed from all learned RNNs with or without new hidden units (Carroll & Peterson, 2002), (Carroll et al., 2001), (Kirschning et al., 1995), (Jordan et al., 1994). The previous connections can be frozen or trainable. Our previous work (Le et al., 2007) also shows that Mixture of Experts System (MES) is the best among several integration methods. After integration, the agent continues to learn to accomplish its task using the new policy.

3.5 Mixture of Experts System (MES)

In MES, we merge the original network and all learned sub-networks in order to make a new network as shown in Fig. 4. Learned connections are considered as experts in the new network to speed up learning performance (Jordan et al., 1994). All connections are trainable. I.e., we can change the weight of any connection.

4. Experiment

In order to examine the learning ability of RLSG, we performed two experiments in the E maze problem and in the Virtual Office problem. In the first and the second experiments, RLSG uses online generated RSNs with one and two useful skills respectively.

4.1 E maze problem

An agent must learn to move from a starting position S to a goal position G. Observation at each position is shown in Fig. 5. The agent can choose one of four actions: North, East, South, and West. Executing any action, the agent receives a reward -1. If the agent can reach the goal it receives a reward 10. An episode is terminated when the agent reaches the goal or the agent has executed 100 actions.

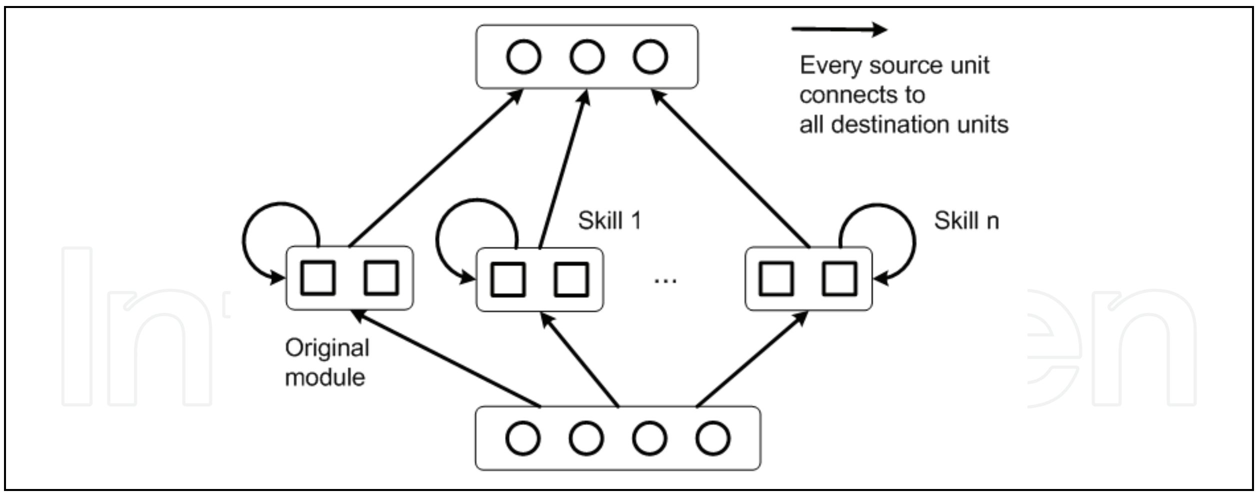

Fig. 4. Mixture of Expert System.

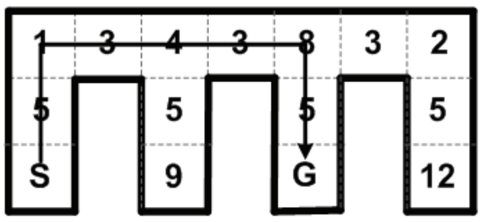

Fig. 5. E Maze Problem: The agent must learn to move from S to G.

We executed RLSG and RL with RNN using softmax exploration method to compare the learning performance. The following parameters were used: discount factor γ = 0.98, exploration temperature τ = 1, RNN learning rate α = 0.1, 4 input units, 6 memory cells, 4 output units. After every 10 episodes, the system evaluates the current RNN. Learning process is terminated when the agent can reach the goal in 10 moves using greedy method. In this experiment, only one subgoal was allowed to create after 100 first learning episodes. For each method, 5 runs were performed.

Results

In some runs, state 8 is detected as a subgoal. In the others, state 2 is considered as a subgoal. We found that even if the subgoal discovery process wasn't very perfect, it didn't affect the learning ability. All runs were convergent giving a good policy. Learning performance of two methods is shown in Fig. 6. The figure shows RLSG outperforms RL with RNN.

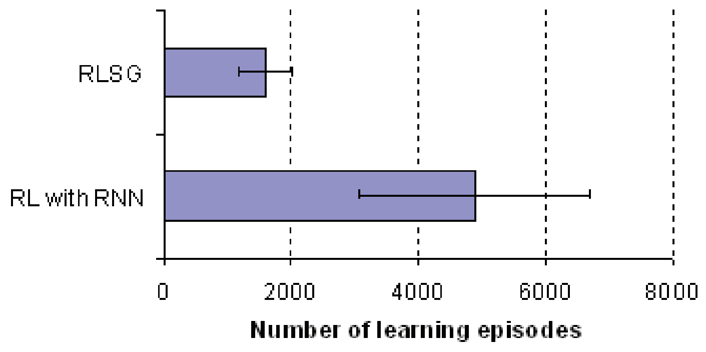

Fig. 6. Learning Performance in the E Maze Problem.

4.2 The virtual office problem

An agent must learn to move from the any random starting position in the hall H (the left room) to one of the goals in the right rooms (Fig. 7). Observations in the upper right room are same as observations in the lower right room except the goal positions. Four actions are available: North, East, South, and West. If the agent reaches the goal, it receives a reward 10. An episode is terminated when the agent reaches the goal or it has executed 100 actions.

Again, we executed RLSG and RL with RNN using softmax exploration method to compare the learning performance. The following parameters were used: discount factor γ = 0.98, exploration temperature τ = 1, RNN learning rate α = 0.1, 6 input units, 6 memory cells, 4 output units. After every 10 episodes, the system evaluates the current RNN. Learning process is terminated when the agent can reach one goal from any starting position in 10 moves using greedy method. Two subgoals were allowed to create after 30 first learning episodes. For each method, 10 runs were performed.

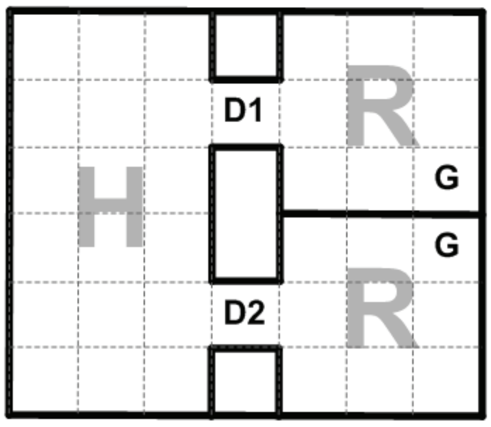

Fig. 7. Virtual Office Problem: D1, D2 are doors between hall and rooms.

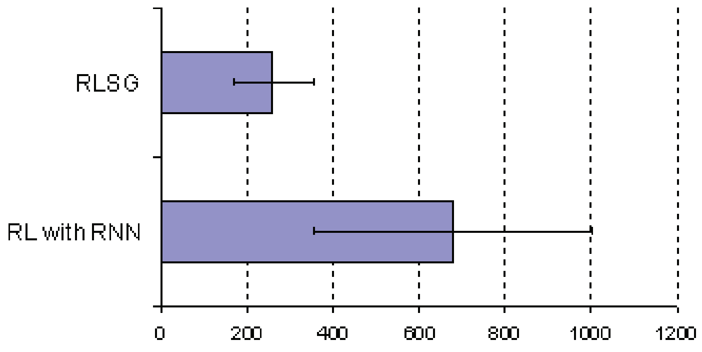

Fig. 8. Learning Performance in the Virtual Office Problem.

Results

In this experiment, all the states in the rights rooms, located near the goals, weren't considered as subgoals. Two doors were detected as subgoals. Again, all runs were convergent giving a good policy. Learning performance of two methods is shown in Fig. 8. The figure also shows RLSG outperforms RL with RNN.

5. Conclusion

In this chapter, we have proposed Reinforcement Learning using Automatic Discovery of Subgoals to accelerate learning of RL with RNN by profiting useful skills. Hidden units and their connections of RNNs, which are used by generated skills, are integrated into the RNN of the main policy. Experiment results of the E maze problem and the virtual office problem show the potential of this method.

6. References

Bakker, B. Reinforcement Learning with Long Short Term Memory, *Advances in Neural Information Processing Systems*, Vol. 14, pp. 1475-1482, 2002.

Barto, A. G. & S. Mahadevan. Recent Advances in Hierarchical Reinforcement Learning, *Discrete Event Systems, Special Issue on Reinforcement Learning*, Vol. 13, pp. 41-77, 2003.

Ballini, R., S. Soares & F. Gomide. A Recurrent Neurofuzzy Network Structure and Learning Procedure, *Proceedings of the IEEE International Conf. on Fuzzy Systems*, pp. 1408-1411, 2001.

Carroll, J. L. & T. S. Peterson. Fixed vs. Dynamic Sub-transfer in Reinforcement Learning, *Proceedings of the 2003 International Conference on Machine Learning and Applications*, 2002.

Carroll, J. L., T. S. Peterson & Nancy E. Owens. Memory-guided Exploration in Reinforcement Learning, *Proceedings of the International Joint Conference on Neural Network*s, 2001.

Champandard, A. J. *Reinforcement Learning*. Available at http://reinforcementlearning.ai-depot.com/Intro.html. Last access 9/25/2007.

Gers, F. A., J. Schmidhuber & F. Cummins. Learning to Forget: Continual Prediction with LSTM, *Neural Computation*, Vol. 12, pp. 2451-2471, 2000.

Gers, F., N. Schraudolph, J. Schmidhuber. Learning Precise Timing with LSTM Recurrent Networks, *Journal of Machine Learning Research*, Vol. 3, pp. 115-143, 2002.

Girgin S., F. Polat & R. Alhajj Learning by Automatic Option Discovery from Conditionally Terminating Sequences, *Proceedings of the 17th European Conference on Artificial Intelligence*, 2006

Gomez, F., J. Schmidhuber & R. Miikkulainen. Efficient Non-Linear Control through Neuroevolution, *Proceedings of the European Conference on Machine Learning*, Berlin, 2006.

Gomez, F. & J. Schmidhuber, Co-evolving Recurrent Neurons Learn Deep Memory POMDPs, *Proceedings of the Conference on Genetic and Evolutionary Computation*, Washington, D. C., pp. 1795-1802, 2005.

Hochreiter, S. and J. Schmidhuber. Long Short-Term Memory, *Neural Computation*, Vol. 9, pp. 1735-1780, 1997.

Ho, F. & Kamel, M. Reinforcement Learning using a Recurrent Neural Network, *Proceedings of the ICNN*, pp. 437-440, 1994.

Jordan, M. I. & Jacobs R. A. Hierarchical mixtures of experts and the EM algorithm, *Neural Computation*, Vol. 6, pp.181-214, 1994.

Kaelbling, L. P., M. L. Littman & A. W. Moore. Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237-285, 1996.

Kirschning, I., H. Tomabechi & J.I. Aoe. A Parallel Recurrent Cascade-Correlation Neural Network with Natural Connectionist Glue, *Proceedings of the IEEE ICNN*, pp. 953-956, Australia, 1995.

Le, T. D., T. Komeda & M. Takagi. Knowledge-Based Recurrent Neural Networks In Reinforcement Learning, *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing*, pp. 179-184, Spain, 2007.

Le, T. D., T. Komeda & M. Takagi. Reinforcement Learning for POMDP Using State Classification, *Journal of Applied Artificial Intelligence*, 2008, in press.

Lin, L. J. Self-improving Reactive Agents based on Reinforcement Learning, Planning and Teaching, *Machine Learning*, Vol. 8, pp. 293-321, 1992.

Lin, L. J. and T. Mitchell. Reinforcement Learning with Hidden States, *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*, 1993.

Lin, L. J. *Reinforcement Learning for Robots Using Neural Networks*, PhD. thesis, Carnegie Mellon, School of Computer Science, 1993.

Littman, M. Memoryless Policies: Theoretical Limitations and Practical Results, *From Animal to Animats 3: Proceedings of the 3rd International Conference on Simulation and Adaptive Behavior*, 1994.

McCallum, R. A. Instance-based State Identification for Reinforcement Learning, *Advances in Neural Information Processing Systems*, pp. 377-384, 1995.

McGovern, A. & A. G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density, *Proceedings of the 18th International Conference on Machine Learning*, pp. 361- 368, San Francisco, CA, USA, 2001.

Menache, I., Mannor, S. & Shimkin, N. Q-Cut Dynamic discovery of sub-goals in reinforcement learning, *Proceedings of the European Conference on Machine Learning*, pp. 295-306, 2002.

Ohta, M., Y. Kumada, I. Noda. Using Suitable Action Selection Rule in Reinforcement Learning, *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, pp. 4358-4363, 2003.

Onat, A., H. Kita & Y. Nishikawa. Recurrent Neural Networks for Reinforcement Learning: Architecture, Learning Algorithms and Internal Representation, *Proceedings of the International Joint Conference on Neural Networks*, pp. 2010-2015, Anchorage, AK, USA, 1998.

Schafer, A. M. & S. Udluft. Solving Partially Observable Reinforcement Learning Problems with Recurrent Neural Networks, *Proceedings of the 16th European Conference on Machine Learning*, Portugal, 2005.

Schmidhuber, J., D. Wierstra, M. Gagliolo, F. Gomez. Training Recurrent Networks by Evolino, *Neural Computation*, Vol. 19, No. 3, pp. 757-779, 2007.

Simsek, O., A. & A. G. Barto. Identifying Useful Sub Goals in Reinforcement Learning by Local Graph Partitioning, *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

Sutton, R. S., Precup, D. & Singh, S. P. Between MDPs and Semi-MDPs: A Framework For Temporal Abstraction in Reinforcement Learning, *Artificial Intelligence*, Vol. 112, pp. 181-211, 1999.

Sutton, R. & Andrew Barto. *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

Taylor, M. E., P. Stone & Y. Liu. Value Functions for RL-based Behavior Transfer: A Comparative Study, *Proceedings of the 12th National Conference on Artificial Intelligence*, 2005.

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

Whitehead S. D., Lin L. J. Reinforcement Learning of Non-Markov Decision Processes, *Artificial Intelligence*, Vol. 73, pp. 271-306, 1995.