Q-error as a Selection Mechanism in Modular Reinforcement-Learning Systems[*]

Mark Ring, Tom Schaul
IDSIA
Galleria 2
6928 Manno-Lugano, Switzerland
Email: {mark,tom}@idsia.ch

Abstract

This paper introduces a novel multi-modular method for reinforcement learning. A multi-modular system is one that partitions the learning task among a set of experts (modules), where each expert is incapable of solving the entire task by itself. There are many advantages to splitting up large tasks in this way, but existing methods face difficulties when choosing which module(s) should contribute to the agent's actions at any particular moment. We introduce a novel selection mechanism where every module, besides calculating a set of action values, also estimates its own error for the current input. The selection mechanism combines each module's estimate of long-term reward and self-error to produce a score by which the next module is chosen. As a result, the modules can use their resources effectively and efficiently divide up the task. The system is shown to learn complex tasks even when the individual modules use only linear function approximators.

1 Background, Motivation and Overview

Most reinforcement-learning solutions assume a single, monolithic mechanism for learning and estimating the value function. In practice, there are many reasons to decompose the learning task and spread it among many component learners, where each learner has a limited domain of expertise.

Many modular approaches to reinforcement learning have been proposed from many different perspectives and with many different motivations. All recognize the need to decompose large reinforcement-learning tasks into smaller, more manageable components. The different approaches often reflect different dimensions along which the task can be split.

Each method expects that the modules are limited in some way and that only together can they solve the task as a whole. The principal limitations can be categorized as: goal, state, observation, and capacity.

Goal limitation. The first modular reinforcement-learning systems had goal-limited modules [Singh, 1992; Tenenberg *et al.*, 1993; Dayan and Hinton, 1993]. These systems have multiple possible subgoals, where each subgoal is assigned to a particular module either manually or automatically. The agents attempt to learn the appropriate policy for each subgoal and also attempt to learn appropriate switching behavior between subgoals. Once a module takes control, it is generally expected to retain control until a subgoal is reached [Wiering and Schmidhuber, 1998; Sutton *et al.*, 1999; Dietterich, 2000; Bakker and Schmidhuber, 2004].

Observation limitation. Observation-limited modules assign different parts of the observation space to different modules [Kohri *et al.*, 1997], which is useful if the observation space can be divided into orthogonal subspaces such that the global value function depends on the features available in each subspace independently. A global arbitrator is needed to choose the appropriate module and corresponding subspace.

State limitation. Systems with state-limited modules generally try to solve problems where the observation alone is insufficient for determining the state. Typically, each module presides over different parts of the state space such that in certain regions, one module is expert at dealing with a certain observation, while in other regions, a different module may be expert in dealing with the same observation but in a different way [Wolpert and Kawato, 1998; Samejima *et al.*, 2003; Doya *et al.*, 2002]. These modules often include a prediction component that evaluates the modules' fitness to the current (unseen) state by measuring the accuracy of its predictions. The module that best predicted the recent observations gets control (meaning that the actions it recommends are most likely to be taken by the agent).

Capacity limitation. Capacity-limited systems assume nothing about how the modules will break up the task. They assume only that each module is not powerful enough to perform the entire task by itself, though it may each have access to all the necessary data. A simple example is a linear function approximator (LFA), which might receive the complete input from the XOR problem, but still cannot always choose the appropriate output because of its own capacity limitations.

This paper considers capacity limitation as the determining factor for breaking up the task, following the insight that if a single module is powerful enough to solve the task, then,

[*] This work was funded by the following grants to J. Schmidhuber: EU project FP7-ICT-IP-231722 (IM-CLeVeR) and SNF Project 200020-122124.

maybe it should. This insight is perhaps deeper than it seems: a complex system composed of simple subsystems offers several benefits. First, the simpler subsystem may be significantly less computationally expensive than a monolithic system, and it might also have a less computationally expensive learning mechanism. Second, the modular solution is highly and immediately parallelizable. Third, and most important, the capacity limitation can be useful in forcing specialization of modules in specific parts of the state space, which is difficult to achieve autonomously in other modular systems.

Thus, this paper answers the question: can a modular learning system made up of simple learning components cooperate to perform a task that none of the simple components could achieve by themselves? To answer as directly as possible, we choose modules composed exclusively of linear function approximators and then test the system on tasks that include non-linear decision boundaries. The results demonstrate that a very simple ε-greedy mechanism for choosing one module at a time is sufficient for solving non-linear tasks.

The method requires each module to have two components: a controller, which generates action values, and an error predictor, which predicts what the module's TD error will be at the next time step.

2 System description

In the standard reinforcement-learning framework (see Sutton and Barto, 1998), a learning agent interacts with a Markov Decision Process (MDP) over a series of time steps $t \in \{0, 1, 2, ...\}$. At each time step the agent takes an action $a_t \in \mathcal{A}$ from its current state $s_t \in \mathcal{S}$. As a result of the action the agent transitions to a state $s_{t+1} \in \mathcal{S}$ and a reward $r_t \in \Re$ is received. The dynamics underlying the environment are described by the state-to-state transition probabilities $\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ and expected rewards $\mathcal{R}^a_{ss'} = \mathbb{E}\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$. The agent's decision-making process is described by a policy, $\pi(s, a) = Pr\{a_t = a | s_t = s\}$, which the agent refines through repeated interaction with the environment so as to maximize $Q(s_0, a_0) = \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s_0, a_t = a_0\}$, the total discounted reward it can expect to receive by taking action a_0 in state s_0 and following policy π thereafter.

This paper introduces SERL, which stands for *Selected Expert Reinforcement Learner*. For purposes of this paper, SERL's state is provided as an observation vector $\mathbf{o} \in \mathcal{O}$, which uniquely identifies the state. SERL consists of a set of modules, $\mathcal{M}$. Each module $i \in \mathcal{M}$ contains two components: a controller function,

$$f^{c,i} : \mathcal{O} \to \mathbb{R}^{|\mathcal{A}|},$$

which generates a vector of action-value estimates; and a predictor function,

$$f^{p,i} : \mathcal{O} \to \mathbb{R}^{|\mathcal{A}|},$$

which generates a vector of predicted action-value errors. At every time step, each module generates values from the current observation vector, $\mathbf{o}_t$:

$$\mathbf{q}^i_t = f^{c,i}(\mathbf{o}_t)$$
$$\mathbf{p}^i_t = f^{p,i}(\mathbf{o}_t)$$

These are combined for each module to create an $|\mathcal{M}| \times |\mathcal{A}|$ matrix L_t of *lower confidence* values such that

$$L^i_t = \mathbf{q}^i_t - |\mathbf{p}^i_t|,$$

where L^i_t is the i^{th} row of L_t. These are used for action selection in place of the action values. Note that this prudent tradeoff derives from the requirement that action values never be overestimated. This is because, while every module has an area of expertise where its value estimates should be roughly correct, in all other places, the estimates may be wildly off. In these inaccurate cases, the action-value estimate that is far below the correct value does not interfere with the correct estimates from the other modules, whereas if the value is estimated far above its correct value, it could be selected instead of the correct value.

SERL then forms a global vector of action-value estimates $\hat{Q}_t$ corresponding to the best L^i for each action. Thus, if L^{ia}_t is the entry in vector L^i_t corresponding to action a, then

$$\hat{Q}^a_t = \max_i L^{ia}_t$$

Winner. At every time step there is a winning module, w_t. In most cases the winner is the module with the highest L value. But this rule is modified in an ε-greedy fashion to allow occasional random selection of winners, based on a random value, $x_t \sim U(0, 1)$:

$$W = \{i \in M : \max_a L^{ia}_t = \max_a \hat{Q}^a_t\}$$

$$Pr\{w_t = i | L^i_t\} = \begin{cases} \frac{1}{|M|} & \text{if } x_t < \varepsilon \\ \frac{1}{|W|} & \text{if } x_t \geq \varepsilon \text{ and } i \in W \\ 0 & \text{otherwise.} \end{cases}$$

SERL then calculates an ε-greedy policy [Sutton and Barto, 1998] based on the winner's L values: $L^{w_t}_t$. (In principle, the two ε values for action and module selection can be different but were the same in the experiments of Section 3).

Learning. The function approximators are updated with targets generated by TD-learning [Sutton, 1988]. The only controller to be updated is that of the winning module from the previous time step. The update is done using an action-value method such as Sarsa [Rummery and Niranjan, 1994] or Q-learning [Watkins, 1989]. We obtained good results with no divergence issues by using Q-learning; thus the target for $\mathbf{q}^{w_t a_t}$ was: $r_t + \gamma \max_a \hat{Q}^a_{t+1}$.

Every module's predictor is updated at every step; the target is the magnitude of the module's TD error:

$$\delta^i_t = r_t + \gamma \max_a \hat{Q}^a_{t+1} - \mathbf{q}^{ia_t}_t.$$

Linear function approximation. For the demonstration and experimental section below, the predictors and controllers are all LFAs. The controllers and predictors are represented as $|A| \times |O|$ weight matrices, $\Theta^{c,i}$ and $\Theta^{p,i}$, so that the $\mathbf{q}$ and $\mathbf{p}$ vectors are calculated as matrix multiplications:

$$\mathbf{q}^i_t = \Theta^{c,i}\mathbf{o}_t$$
$$\mathbf{p}^i_t = \Theta^{p,i}\mathbf{o}_t,$$

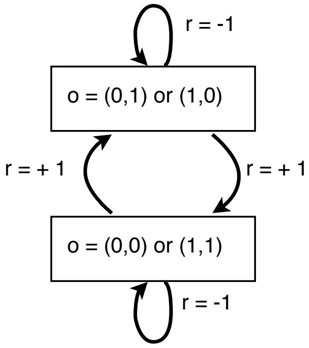

Figure 1: The XOR MDP. A tiny reinforcement-learning problem that can only be learned by solving XOR. There are only two states. In the top state the agent observes (0,1) or (1,0) with equal likelihood, and must learn to move South; in the bottom state the agent observes (0,0) or (1,1) and must learn to move North. (All observations also include a bias input, which is always 1.)

where all vectors $\mathbf{o}, \mathbf{q}$, and $\mathbf{p}$ are column vectors. Learning is done by online least squares in the standard fashion:

$$\Theta_{t+1}^{c,w_t a_t} = \Theta_t^{c,w_t a_t} + \alpha\delta_t^{w_t}\mathbf{o}^T$$
$$\Theta_{t+1}^{p,ia_t} = \Theta_t^{p,ia_t} + \alpha_t^{p,i}(|\delta_t^i| - \mathbf{p}_t^{ia})\mathbf{o}_t^T,$$

where $\Theta^{c,ij}$ refers to the weight vector (of size $|\mathcal{O}|$) in the j^{th} row of $\Theta^{c,i}$; and α is the learning rate. The learning rate for the predictor is modified to emphasize small errors: $\alpha_t^{p,i} = \alpha/(1 + 10|\delta_t^i|)$. This keeps predictors from forming highly accurate predictions of very high errors (whose exact magnitude is essentially irrelevant) at the expense of maintaining accurate predictions of low errors, which are critical.

3 Experiments

The experiments were devised to answer the following questions:

1. Can capacity-limited modular systems automatically divide the task into smaller components?
2. Can the modules learn to cooperate to solve the task?
3. Can the system as a whole learn to solve tasks that the individual modules cannot?
4. How does performance scale with number of modules?

The following two different experiments address these issues. The first is a simple XOR (exclusive-or) reinforcement-learning environment; the second is a small artificial eye task.

3.1 XOR

The XOR task is shown in Figure 1. The task answers questions 1 and 3 on a very small scale. A single LFA cannot solve the XOR task. Therefore to solve the task, each module must specialize in a different part of the space. In the XOR task, the agent has two actions available, one to move North and one to move South. There are only four states. The binary state numbers are given to the agent as the observation vector (together with a bias unit whose value is always 1). From the even parity states (those with observations of 00 and 11), the agent moves South for a reward of $+1$ and North for a reward of -1. From the odd-parity states (observations of 01 and 10), the rewards are the opposite (+1 for South, -1 for North). For the optimal policy, the agent must learn to take opposite actions in odd- and even-parity states.

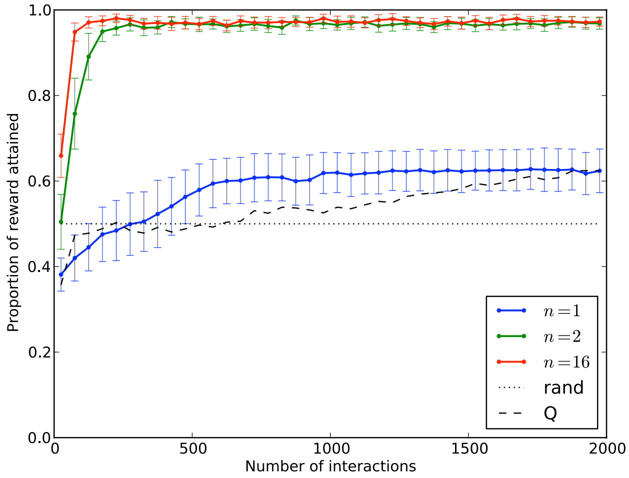

Figure 2: Results from running SERL on the XOR task with the number of modules shown. The vertical axis shows reward received, normalized between 0 and 1 (optimal policy), while the horizontal axis shows the number of actions taken. The dashed line displays the results for a simple Q-learning agent using an LFA and the same parameters as with the modular system. All plots are averaged over 50 runs and errorbars correspond to half a standard deviation, for readability. The dotted line shows the baseline performance of the uniform random policy.

Figure 2 shows SERL's performance when run in this environment. Learning was non-episodic, and the graph shows reward received as a percentage of the optimal as a function of the total number of actions taken. For this task, γ was set to 0 so that only the immediate reward had any effect; estimates of future reward did not. The learning-rate α was 0.05, $\varepsilon = 0.02$ and all weights are initialized by randomly drawing from $\mathcal{N}(0, 0.1)$. The graph shows the results using 1, 2, and 16 modules (n), as well as a simple linear Q-learner (without a predictor). The one-module system and the linear Q-learner show roughly the same performance and do not solve the task, as expected. The two-module system solves it very quickly. Adding more modules than necessary does not interfere with success, in fact it even speeds up convergence.

3.2 Artificial Eye

In the artificial-eye task, SERL's observation is determined by the location of its "eye" (a 4×4 square—shaded gray) above a 15×15 random black and white bitmap (Figure 3). Each pixel in the bitmap represents a value of 1 (black) or 0 (white) within a 16-dimensional observation vector. This

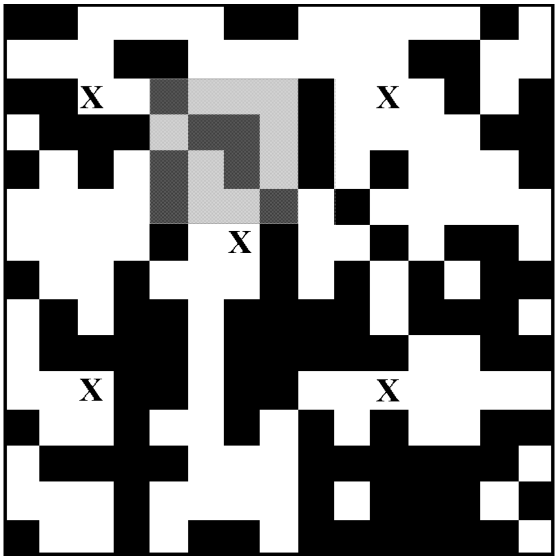

Figure 3: The moving-eye task. Through its "eye" (shown in gray) SERL observes a 4×4 (16 bit) piece of the underlying black-and-white image. The agent can move the eye one black or white square to the North, South, East or West. It receives a reward of +1 when the upper-left corner of its "eye" moves over an "X", after which the eye "saccades" to a random position.

observation vector is always expanded with a bias feature of 1. The image was generated randomly but was checked to satisfy the constraint that every 4×4 square is unique. In this task, SERL has four actions that move its eye one pixel North, East, West, or South. If there is no room to move in the chosen direction, the eye remains in place. There are 5 rewarding squares (shown as an X) which, when reached with the upper left-hand corner of the eye, provide a reward of +1 and force the eye to saccade to a random position in the image. All other actions result in a reward of 0. For this task $\gamma = 0.9$, and again $\alpha = 0.05$ and $\varepsilon = 0.02$.

The results from the eye task are shown in Figure 4. All results are shown in proportion to the maximum achievable reward for this task (an average of 0.36 per step), which is represented by a value of 1 on the vertical axis. The horizontal axis shows the total number of steps taken. Learning with a simple Q-learner using an LFA (and the same parameters) was similar to that of a single module and did not achieve high reward. As the number of modules increases, SERL achieves consistently higher reward, nearing the theoretical maximum.

This experiment addressed questions 2 and 4 above. Coding by hand, we were unable to find a solution with less than eight modules that could achieve the optimal policy. Therefore, the modules must cooperate to solve the task, each making updates based on the values of the later winners. Each (autonomously) learns to become expert in a small part of the state space while leaving the rest to the other modules. With more modules, more of the task is accomplished.

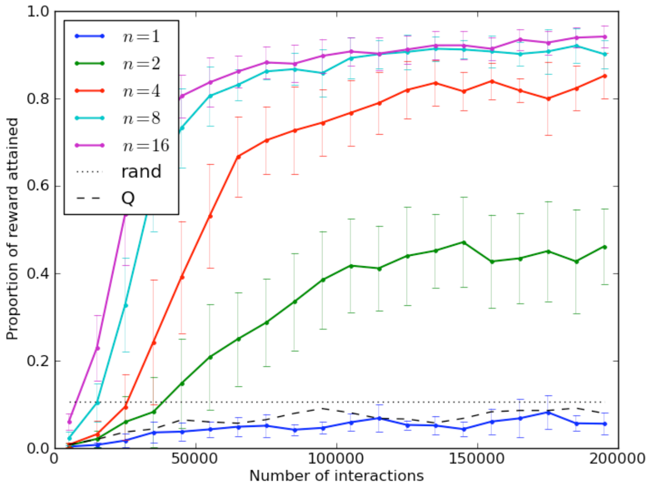

Figure 4: SERL's results on the eye task with the number of modules shown. The vertical axis shows reward received, normalized between 0 and 1 (optimal policy), while the horizontal axis shows the number of actions taken. The dashed line displays the results for a simple Q-learning agent using a LFA and the same parameters as the modular system. All plots are averaged over 25 runs; error bars correspond to half a standard deviation, for readability. The dotted line shows the baseline performance of the uniform random policy.

4 Discussion

It is instructive to analyze what would happen if each module contained only linear, action-value function approximators and no error-prediction component. Could the same intelligent behavior be achieved by simply selecting the action with the highest estimated Q-value? In principle, the answer is yes, given enough modules. Consider for the moment (without loss of generality) a single action per state so that state-values and action-values are identical. If every state has a dedicated module, then each module need only calculate a single correct Q-value, while calculating an arbitrarily low value in all other states. This is always possible for binary feature vectors when a bias unit is present. However, in general it is more interesting and efficient for each module to properly calculate the value for multiple states.

An d-dimensional LFA (with a bias unit) can perfectly map any d linearly independent vectors to any arbitrary values, but in doing so, over-generalization becomes problematic. Let $\mathcal{O}$ be the complete set of binary vectors of length d; let $\mathcal{C} \subset \mathcal{O}$ be a linearly independent subset of size d, where each element is correctly mapped to its Q value by a modules' controller LFA; and let $\mathcal{G} = \mathcal{O} - \mathcal{C}$. For the system to function correctly, all $g \in \mathcal{G}$ must be mapped to values not greater than the correct value, or else the module will be chosen preferentially over the module with the correct value. As d increases, $\frac{|\mathcal{O}|}{|\mathcal{C}|}$ increases exponentially, as does the difficulty of mapping $\mathcal{G}$ to low values. In contrast, the predictor LFA only needs to be correct when the controller's error is zero. As long as SERL has enough controllers to map every state that it encounters to the correct value, then the vectors in $\mathcal{G}$ may be mapped to any non-zero value. This constraint is much easier to satisfy than the previous one, because it only requires that vectors in $\mathcal{G}$ not be in the nullspace of the vectors in $\mathcal{C}$.

Transfer of Control and Reward. An interesting aspect of SERL is how easily it allows control to pass from module to module. In most modular systems, transfer of control is a core dilemma. But in SERL, modules do not truly assume control; they only make suggestions. Each suggestion is followed depending on its strength (Q-value) and expected accuracy (prediction value). Module A may pass control to module B without ever seeing a reward itself. Nevertheless, module A shares in the credit for reaching the goal that B achieves because module A learns to minimize the TD error for transfer to a state represented by B. More specifically: each winning module's controller is updated by the next. The maximum Q-value at the current step is used to update all the previous modules' predictors and the previous winner's controller.

The system is therefore able to successfully combine the intelligence of its components and demonstrates robustness to the number of modules used, showing only positive benefit to having more modules than are necessary and degrading gracefully when the number of modules is insufficient.

Breaking up is hard to do. If reinforcement learning is going to scale to larger problems, then there will need to be ways of scaling to large state spaces. One of the most promising ways to address this issue is through multi-module systems. Yet one of the most vexing problems with modular RL systems has been getting them to split up a large task in meaningful ways. Many methods rely on hand-built parceling of the space, either by specifying in advance what the modules should do (such as seeking certain goals), giving each module a constrained representation of the input space. Methods that attempt to get modules to split up the task autonomously by developing expertise in the prediction of the task dynamics have been at best only marginally successful and have used only a small number of modules [Wolpert and Kawato, 1998; Samejima *et al.*, 2003; Doya *et al.*, 2002].

The results in the previous section show that capacity limitation can provide a natural way for modules to organize themselves when they are augmented with predictions of their own error. Figure 5 shows how the task has been distributed among the modules. (It should be noted that no particular spatial or temporal contiguity constraints were imposed to force the system to group neighboring states together, though this is a topic of ongoing work.)

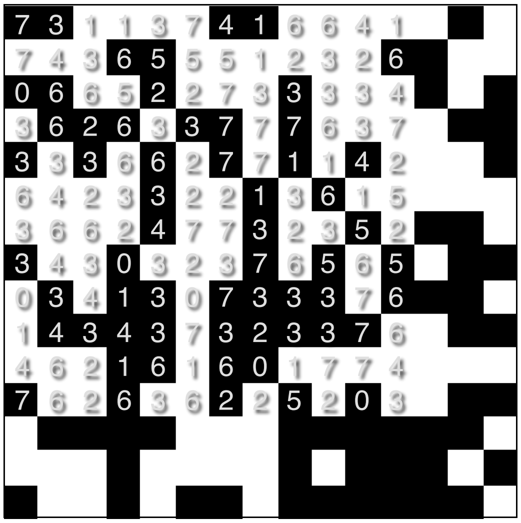

Figure 5: The eye task with each eye position labeled by the winning module for that position after training with eight modules. The labels shown correspond to the upper left corner of each eye position.

Refinements. This paper has presented a proof-of-concept for the capacity-limited approach to multi-module reinforcement learning. This proof of concept has used linear function approximators as an example class of function approximators. The reason for doing so is primarily to clarify the exposition, but also because this class is particularly important. LFAs are straightforward to describe and analyze, have clean learning algorithms, scale well to high dimensions, are easy to parallelize, are not computationally demanding, and are theoretically well understood. As a consequence, they have been greatly studied and much is known about their behavior and techniques for optimization. Some of that knowledge could be used to advantage with SERL. In particular, methods such as LSTD(λ) [Boyan, 1999] and LSPI [Lagoudakis and Parr, 2003] can be used in combination with them, though some adaption to those algorithms (beyond the scope of this paper) is required.

On the other hand, more powerful function approximators may also be used for either controller, or predictor, or both. Though not explicitly tested in this paper, in the general case, all actual instantiations of learning algorithms are limited, and because of that, SERL can potentially make use of these limitations to split a more complex problem into a set of simpler solutions; or, said in another way, it can combine complex mappings into even more complex mappings. Some interesting examples to consider are:

- a nonlinear predictor with a linear controller. This may have some advantages in continuous domains.
- a linear controller combined with a recurrent predictor network (which maintains internal state information), thus allowing recognition and parceling hidden states to the correct controller (i.e., mapping POMDPs to a collection of MDP solvers.)

Other Related Approaches. All modular approaches to reinforcement learning considered here are value-function approximation methods based on the standard reinforcement-learning framework as described by Sutton and Barto (1998). However, classifier systems are also methods for learning from reinforcement [Holland, 1985]. One such method, XCS [Wilson, 1995] bears an interesting relation to our work by virtue of the fact that it incorporates a method for predicting Q-error, which is then used as part of the computation for assessing the fitness of classifiers. This value in that system helps to enhance accuracy and provide greater stability. We speculate that prediction of TD-error may in general be a particularly useful quantity for reinforcement-learning systems.

5 Conclusions

This paper has described the first modular RL system that uses capacity limitation as the exclusive mechanism for automating module assignment. We have shown that this system has the ability to allocate the state space across different modules, to train each module on part of the state space, and to consistently choose from among all the modules the one best suited for the task. The results also showed that such division of labor can be done automatically and follows as a direct consequence of each module's inability to solve the entire task by itself.

Because each module keeps track of its areas of expertise, and because it is an inherently easier task to keep track of where one is expert than to keep track of the expertise itself, SERL is able to combine expertise among the modules and to solve problems greater than the ability of any individual module. The system is robust to the number of modules used, showing only positive benefit to having more modules than are necessary and degrading gracefully when the number of modules is insufficient.

While we only show a proof of concept here, there is good reason to believe that this technique will generalize to other function approximators as well as to other reinforcement learning algorithms.

The system is highly parallelizable and stable, showing no sign of divergence despite the combination of LFA and off-policy TD updates.

Acknowledgments

We thank Vincent Graziano, Jürgen Schmidhuber, and Faustino Gomez for many helpful discussions, and we also thank Faustino Gomez, as well as the anonymous reviewers, for helpful suggestions on earlier drafts.

References

[Bakker and Schmidhuber, 2004] B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In F. Groen et al., editor, *Proc. 8th Conference on Intelligent Autonomous Systems IAS-8*, pages 438–445, Amsterdam, NL, 2004. IOS Press.

[Boyan, 1999] Justin A. Boyan. Least-squares temporal difference learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, 1999.

[Dayan and Hinton, 1993] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 271–278, San Mateo, California, 1993. Morgan Kaufmann Publishers.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res. (JAIR)*, 13:227–303, 2000.

[Doya *et al.*, 2002] Kenji Doya, Kazuyuki Samejima, Ken ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural Computation*, 14(6):1347–1369, 2002.

[Holland, 1985] J. H. Holland. Properties of the bucket brigade. In *Proceedings of an International Conference on Genetic Algorithms*. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[Kohri *et al.*, 1997] Takayuki Kohri, Kei Matsubayashi, and Mario Tokoro. An adaptive architecture for modular q-learning. In *IJCAI (2)*, pages 820–825, 1997.

[Lagoudakis and Parr, 2003] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[Rummery and Niranjan, 1994] G. A. Rummery and M. Niranjan. Online Q-learning using connectionist systems (Tech. Rep. No. CUED/F-INFENG/TR 166). *Cambridge University Engineering Department*, 1994.

[Samejima *et al.*, 2003] Kazuyuki Samejima, Kenji Doya, and Mitsuo Kawato. Inter-module credit assignment in modular reinforcement learning. *Neural Networks*, 16(7):985–994, 2003.

[Singh, 1992] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8, 1992.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Tenenberg *et al.*, 1993] Josh Tenenberg, Jonas Karlsson, and Steven Whitehead. Learning via task decomposition. In J. A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 337–343. MIT Press, 1993.

[Watkins, 1989] Christopher J. C. H Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, May 1989.

[Wiering and Schmidhuber, 1998] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1998.

[Wilson, 1995] S.W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[Wolpert and Kawato, 1998] Daniel M. Wolpert and Mitsuo Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7-8):1317–1329, 1998.