PERGAMON

Neural Networks 15 (2002) 1107–1124

Neural Networks

www.elsevier.com/locate/neunet

2002 Special Issue

Applications of the self-organising map to reinforcement learning

Andrew James Smith*

The Division of Informatics, Institute for Adaptive and Neural Computation, University of Edinburgh, 5 Forrest Hill, Edinburgh EH1 2QL, UK

Abstract

This article is concerned with the representation and generalisation of continuous action spaces in reinforcement learning (RL) problems. A model is proposed based on the self-organising map (SOM) of Kohonen [Self Organisation and Associative Memory, 1987] which allows either the one-to-one, many-to-one or one-to-many structure of the desired state–action mapping to be captured. Although presented here for tasks involving immediate reward, the approach is easily extended to delayed reward. We conclude that the SOM is a useful tool for providing real-time, on-line generalisation in RL problems in which the latent dimensionalities of the state and action spaces are small. Scalability issues are also discussed. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Reinforcement learning; Self-organising map; Continuous action spaces; Generalisation; Real-valued actions; Unsupervised learning; Q-learning

1. Introduction

Learning techniques are often divided into *supervised*, *unsupervised* and *reinforcement learning* (RL). Supervised learning requires the explicit provision of input–output pairs and the task is one of constructing a mapping from one to the other. Unsupervised learning has no concept of target data, and performs processing only on the input data. In contrast, RL uses a scalar reward signal to *evaluate* input–output pairs and hence discover, through trial and error, the optimal outputs for each input. In this sense, RL can be thought of as intermediary to supervised and unsupervised learning since some form of supervision is present, albeit in the weaker guise of the reward signal. This method of learning is most suited to problems where an optimal input–output mapping is unavailable a priori, but where a method for evaluating any given input–output pair is available instead.

Often these three learning paradigms can be combined. For example, supervised learning may be used to achieve forms of unsupervised learning as in the auto-associative multi-layer perceptron (MLP) of Rumelhart, Hinton, and Williams (1986).[1] Conversely, unsupervised learning can be used to support supervised learning as, for example, in radial basis networks (see Bishop (1995, Chapter 5)) and also the SOM–MLP hybrid model of Tani and Fukumura (1994). Alternatively, supervised learning can be embedded within an RL framework. Examples are common and include the SRV units of Gullapalli (1990), the CRBP algorithm of Ackley and Littman (1990), the Q-AHC algorithm of Rummery (1995) (based on the adaptive heuristic critic (AHC) and actor–critic models of Barto, Sutton, and Anderson (1983) and Sutton (1984)), the backgammon learning application of Tesauro (1992, 1994), the QCON architecture of Lin (1993), and the lift scheduler of Crites and Barto (1996), all of which make use of the MLP and the backpropagation training algorithm. Similarly, unsupervised learning may also be used to provide representation and generalisation within RL as in the self-organising map (SOM) based approaches of Wedel and Polani (1996) and Touzet (1997). However, despite these overlaps, the initial classification of learning paradigms based on the strength and quality of the supervision provides an important dimension for distinction which can be used to match the learning technique to the task in hand. It is also as well to think of RL as a distinct method since it has its own bloodline in animal psychology dating back to Thorndike (1911), as well as its own theoretical foundations originating in the optimal control problems of the 1950s.[2]

Although RL is unrelated to connectionism per se, the neural paradigm, both supervised and unsupervised, has been shown to provide a convenient mechanism for representation and generalisation in RL problems. One reason for this alliance is that many of the principles of the neural paradigm also underpin modern RL techniques. For example, Q-learning (see Section 2) is an incremental, iterative, interactive algorithm with a simple update rule,

* Tel.: +44-131-651-1209.
www.dai.ed.ac.uk/homes/andys/

[1] This technique can be used to perform linear or non-linear principle component analysis. See Bishop (1995, p. 314) for a discussion.

[2] Later developing into the field of Dynamic Programming (Bellman, 1957).

implementation and representation. These computational features seem to make neural networks a natural implementational choice for RL applications. Indeed combining RL theory with the generalisation offered by connectionism has already been shown to be able to yield very promising practical applications. See Tesauro (1992, 1994) for a literature favourite in which a system combining back-propagation and RL learns to play backgammon to world-class standard.

This article specifically addresses the problem of the representation and generalisation of continuous action spaces in RL problems. Section 2 reviews the field of RL and the specific problem being addressed. Section 3 then briefly reviews the SOM which forms the basis for the proposed model (introduced in Section 4). A number of experiments are then performed which explore one-to-one state–action mappings (Section 5.1), many-to-one mappings (Section 5.2), one-to-many mappings (Section 5.3), the quantitative benefit of being able to exploit topology preservation in the model (Section 5.4), and the performance of the model in dimensions that exceed those of the underlying SOMs (Section 5.5). A discussion of possible extensions, related work and problems to address is offered in Section 6, followed by some concluding remarks summarising the main contributions of the proposed model in Section 7.

2. Reinforcement learning

Standard RL theory has provided a comprehensive framework in which to solve Markov decision problems where the aim is to maximise a scalar reward signal on the state transitions when the underlying model dynamics are unknown. The theoretical framework offers a suite of techniques based on dynamic programming (Bellman, 1957) which have at their heart a process of estimating the expected return following either each state or each state–action pair (Sutton & Barto, 1998). A popular technique is *Q-learning* (Watkins, 1989) which estimates the return:

$$E(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^h r_{t+h})|s, a, \pi \qquad (1)$$

for every state–action pair, (s, a), of the MDP. Here, r_t is the reward received at time t where t is the time that action a was taken in state s (each time-step corresponds to a state transition), $0 \leq \gamma < 1$ is the *discount factor* which ensures a finite sum in the case of an infinite horizon ($h = \infty$), and π is the policy being evaluated which maps each state–action pair to the probability of taking that action in that state.

The *Q-function*, which maps state–action pairs to estimates of Eq. (1), can be visualised as a simple table of *Q-values* as in Fig. 1. After any action a_k is taken in any state s_j, the appropriate entry in the Q-table is updated

	$\mathbf{s_1}$	$\mathbf{s_2}$	$\cdots$	$\mathbf{s_m}$
$\mathbf{a_1}$	$Q(s_1;a_1)$			
$\mathbf{a_2}$				
$\vdots$			$\ddots$	
$\mathbf{a_n}$				$Q(s_m,a_n)$

Fig. 1. The Q-table. Each entry represents an estimate of the expected return following the corresponding state–action pair.

according to the Q-learning rule:

$$Q(s_j, a_k) := Q(s_j, a_k) + \alpha \times \left(\left\{ r + \gamma \max_i Q(s_j', a_i) \right\} - Q(s_j, s_k) \right) \qquad (2)$$

where r is the reward immediately following the state–action pair (s_j, a_k), s_j' is the new state of the environment encountered immediately after s_j, and α is the Q-learning rate. This learning rule just pushes the relevant Q-value towards the *one-step corrected return* inside the curly brackets, $r + \gamma \max_i Q(s_j', a_i)$. This yields a recursive definition. As long as $Q(s_j', a_i)$ is indeed the expected return of the optimal action in the following time-step, then it can be seen that the contents of the curly brackets in Eq. (2) can be recursively expanded out to restore Eq. (1). Optimising behaviour is then a simple matter of selecting the action at each time-step which has the highest Q-value for the current state. This is referred to as *exploiting* the acquired knowledge. However, to ensure that both any adverse initial conditions in the Q-table are overcome, and also to allow adaptation to a dynamic environment, suboptimal actions must occasionally be selected with a view to collecting new reward information that may improve future performance. This is referred to as *exploration*, which is usually reduced as learning progresses.

Under the conditions that the environment is modelled as an MDP, that the Q-function is implemented as a lookup table, that each state–action pair is visited infinitely often, and that the learning parameters are appropriately annealed, then simultaneous convergence of the Q-values to Eq. (1) and of the policy, π, to the optimal policy (in terms of maximising return) is guaranteed (Watkins & Dayan, 1992). See Sutton and Barto (1998) and Kaelbling, Littman, and Moore (1996) for detailed and comprehensive accounts of RL theory, and Q-learning in particular.

In many practical problems, few or even none of the convergence criteria may be met. An example is when the state and action spaces are large or continuous and therefore some form of generalisation is required. There have been many different approaches to generalisation of large or continuous state spaces in RL problems including hand decomposition of the state space (Mahadevan & Connell, 1991), tile-coding techniques such as CMACS (Albus, 1975; Prescott, 1994), coarse coding (see Santamaria, Sutton, and Ram (1997) for a review), the use of the SOM (Sehad & Touzet, 1994; Touzet, 1997; Wedel & Polani, 1996), and the use of MLPs and the back-propagation training algorithm (see Ackley and Littman (1990), Crites and Barto (1996), Gullapalli (1990), Tesauro (1994) and Ziemke (1996) for a selection).

Of particular interest to this paper are approaches to representation and generalisation over continuous *action* spaces. It may sometimes be possible to handcode a number of discrete actions and then let the agent learn which action to select in each state. This is the approach adopted in the robot learning experiments of Mahadevan and Connell (1991), for example, and also the cart-pole balancing experiment considered by Peng and Williams (1996). But other researchers have noted the potential need for being able to adapt actions from a continuous range as part of the learning process itself. Examples include the SRV units of Gullapalli (1990), the Covariance Learning extension of the Motoric Map of Wedel and Polani (1996), the CRBP algorithm of Ackley and Littman (1990) (and its extension proposed by Ziemke (1996)), the Q-AHC algorithm of Rummery (1995, Chapter 5) based on the AHC and actor–critic models of Barto et al. (1983) and Sutton (1984), as well as the approach of Prescott (1994, Chapter 6) based on the work of Albus (1975) and Williams (1988).

A key motivation for studying continuous action spaces as opposed to fixing discrete actions a priori is that although a continuous response may often be approximated to arbitrary accuracy by sufficiently fast interleaving of discrete actions (for example, controlling the temperature of a fridge with a binary thermostat), latency issues may preclude this approach. For example, consider the aim of attempting to balance the cart-pole system at its non-stable equilibrium point when the controller is restricted to sampling the system state relatively infrequently. Binary actions may be too crude for solving a problem of this kind. Moreover, learning tasks involving uncorrectable one-off actions such as throwing a ball, swinging a racquet, or selecting a height to jump, may necessitate an adaptable real-valued response.

This paper presents a new model which adapts Q-learning[3] so that not only is an optimal mapping sought from a discrete set of states to a discrete set of actions, but also the actions themselves are adapted within a continuous action space as part of the learning process. The model addresses the problem of on-line, real-time learning with provision for real-valued (potentially delayed) reward, and real-valued states and actions. Of particular interest is the ability to capture and exploit the inherent structure of the desired state–action mapping. To illustrate this feature of the model, a number of experiments will be performed. The first will involve a one-to-one mapping from states to actions, the second a many-to-one mapping and the third a one-to-many mapping. We will also consider some of the model's limitations including issues pertaining to scalability.

[3] We note that there are other approaches to estimating Eq. (1) for each state–action pair such as SARSA (Rummery & Niranjan, 1994), for example, or Monte Carlo techniques (Sutton & Barto, 1998, Chapter 5), and in fact for the purposes of this paper any of these techniques could be substituted for Q-learning without loss of generality. This paper addresses generalisation and not action–value estimation.

3. The self-organising map

The model proposed in this account is based on the ubiquitous SOM (Kohonen, 1987, 1995), and so this algorithm is now briefly reviewed. For a more detailed reminder of the model see Rojas (1996) and for a book of varied applications and analyses see Oja and Kaski (1999). The interested reader is also referred to SOM-database (2001) for an indexed list of thousands of applications.

The SOM usually consists of an array of units arranged in a grid. Associated with each unit, t, is a weight vector, $\mathbf{w}^t = [w_1^t, w_2^t, \ldots, w_D^t]$ where D is the dimensionality of the input data. The aim is to find a suitable set of weights for each unit so that the network models the distribution of the input data in the input space (also the weight space). In many cases, the *intrinsic* dimensionality of the distribution may be low, even though the dimensionality of the input data itself is high. In these cases, the SOM can be used as a dimensionality reduction technique. As an illustration, consider Fig. 2 in which a map is shown in physical or topological space on the left, and in weight (input) space on the right. In this example, the two-dimensional SOM is an appropriate choice for representing the three-dimensional data because the data apparently lies on a two-dimensional manifold inside the input space.

The learning rule responsible for finding a suitable set of weights is simple: given an input vector, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$, the distance between each unit, t, of the SOM and that vector is calculated by:

$$\sum_{d=1}^{D} (x_d - w_d^t)^2 \tag{3}$$

The unit with the smallest distance is that which most closely represents the current input, and is thus considered

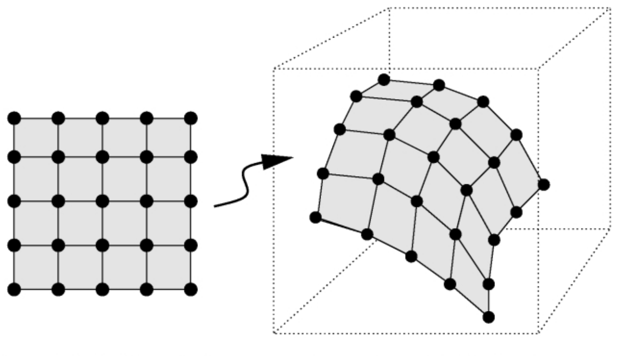
Kohonen Map in Physical Space
Kohonen Map in Weight/Input Space

Fig. 2. A two-dimensional Kohonen map is shown in topological space on the left and in weight space (or input space) on the right. The input data lies on a two-dimensional manifold despite the input space being three-dimensional.

the *winner* for that input. The weights of the winning unit are now updated towards that input:

$$\mathbf{w}^{\text{winner}} = \mathbf{w}^{\text{winner}} + \beta(\mathbf{x} - \mathbf{w}^{\text{winner}}) \tag{4}$$

where β is the learning rate. In addition to the winning unit being updated towards the current input vector, the winning unit's *neighbours* are also moved in this direction but by an amount that decays with the distance of those neighbours from the winning unit. The *neighbourhood function* which controls this is often normally distributed around the winning unit, but for the purposes of the following experiments a very simple linear neighbourhood will be used.

The weights of the map are initialised to random values and then the above process is iterated for each input vector in the data set, effectively resulting in a competition between different regions of the input space for units of the map. Dense regions of the input space will tend to attract more units than sparse ones, with the distribution of units in weight space ultimately reflecting the distribution of the input data in the input space. Neighbourhood learning also encourages *topology preservation* with units close in the topology of the map ending up close in the weight space too, and therefore responding to similar inputs (as in Fig. 2). Fig. 3 shows the effect of mapping various two-dimensional spaces with both a one- and two-dimensional SOM.

4. The model

The problem under consideration can be paraphrased as the search for an optimal mapping from a continuous input or state space to a continuous action space, where optimal means maximising the expected total reward following each possible state. Following the approaches of Wedel and Polani (1996) and Touzet (1997), the proposed model uses a SOM to quantise a continuous input space into a discrete representation. The SOM maps the input space in response to the real-valued state information, and each unit is then interpreted as a discrete state of the environment occupying its own *column* in the Q-table. A second SOM is used to represent the action space, with each unit of this second map corresponding to a discrete action occupying its own *row* in the Q-table. For each state–action pair, an estimate of expected return is maintained using any action value estimation technique.

The first SOM (we will call this the *input map*) resides in the state space and adapts in the usual SOM way in response to each state vector so that the state space is represented at the highest resolution in the most active regions of the space. The second SOM (the *action map*), which resides in the action space, must be more pro-active in its representation of this space because actions must be explored by trial and error in order to discover those that yield the highest return for the whole range of observed inputs. To achieve this exploration, the following algorithm is used: for any real-valued state vector, the unit of the input map with smallest Euclidean distance from that state vector is identified as the winner. Next, one of the units of the action map is selected according to the usual Q-learning criterion—i.e. the one with the highest Q-value for the current state if *exploiting*, and a random action if *exploring*. The weight vector associated with this winning action unit is then used as the basis for the real-valued action to be taken. We can refer to this action weight vector as the *proposed action*. The proposed action is then perturbed using random noise and it is this *perturbed action* which is actually output by the learning system. If the return received following the perturbed action is greater than the estimated expected return associated with the winning state–action pair, then the exploration in the action map appears to have been successful and so the action map is updated (in the usual SOM way) towards the perturbed action. Otherwise, no learning takes place in the action map. In either case, the Q-value of the winning state–action pair is updated towards the actual one-step corrected return.

To summarise, the algorithm can be interpreted as standard Q-learning with the discrete states being represented by the dynamic units of the input map, and the discrete actions being represented by the dynamic units of the action map. The input map forms in response to the sampled state information, and the action map randomly explores the action space with updates favouring the representation of actions which apparently improve performance over the status quo.

Because of the topology preserving nature of the SOM, we expect neighbouring input units to tend to maintain similar estimates of expected return for neighbouring action units. Therefore, a simple amendment to expedite the algorithm is to not only update the Q-value of the winning state–action pair towards the one-step corrected return, but to update *every* state–action pair towards this value proportionally to the product of the two neighbourhood functions (of the input and action maps). We call this *neighbourhood Q-learning*. The complete algorithm is summarised below for the case of a two-dimensional input space and a two-dimensional action space.

1. Present an input vector, I, to the system, and identify the winning unit in the *input map*, s_j.
2. Identify a unit in the *action map*, $a_k =$

$$\begin{cases} \text{One with best Q-value for state, } s_j & \text{with probability } 1-p \\ \text{Random action} & \text{with probability } p \end{cases}$$

3. Identify the *proposed action* as the weights of unit a_k, $\langle u_{k1}, u_{k2} \rangle$.
4. Independently perturb each element of the proposed action by a small random amount to yield a new

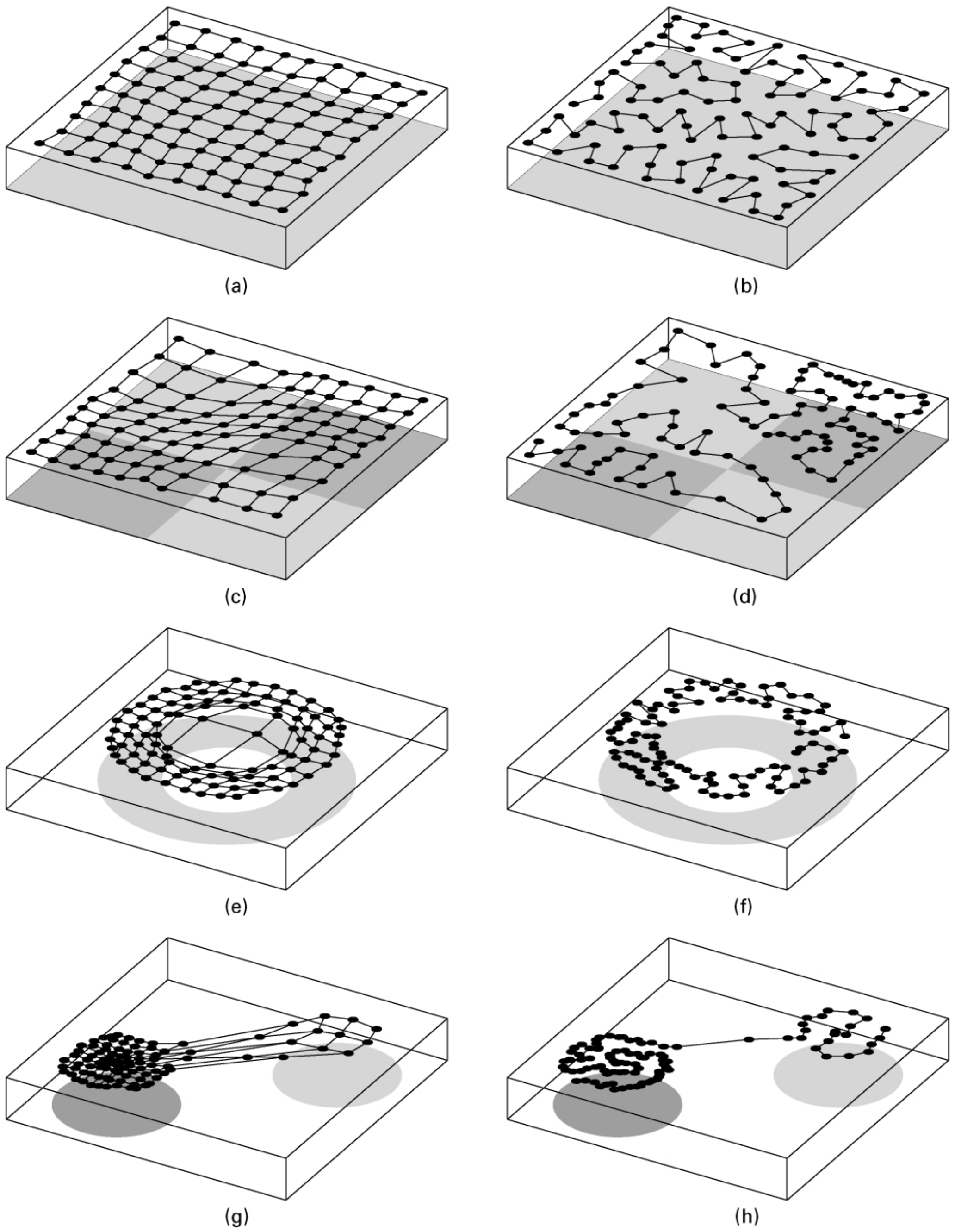

Fig. 3. The figures on the left show two-dimensional Kohonen mappings (units plotted in weight space) of various two-dimensional distributions. The shading underneath each map reflects the actual density of the input distribution (dark grey = $4 \times$ light grey) from which input vectors are drawn at random and presented to the network: (a) uniform contiguous, (c) non-uniform contiguous, (e) irregular, and (g) discontiguous, non-uniform and irregular. The figures on the right show the mapping of the same distributions obtained using a one-dimensional map.

perturbed action:

$$\langle u'_{k1}, u'_{k2} \rangle = \langle u_{k1} + random(-1,1) \times \varepsilon, u_{k2} + random(-1,1) \times \varepsilon \rangle$$

5. Take the perturbed action by outputting the action vector $\langle u'_{k1}, u'_{k2} \rangle$.
6. Receive a reward, r, from the environment.
7. If $r + \gamma \max_i Q(s'_j, a_i) > Q(s_j, a_k)$, then the perturbed action appears to be an improvement over the proposed action, so update the action map towards the perturbed action, $\langle u'_{k1}, u'_{k2} \rangle$, according to the usual SOM update rule:

$$u_{m1} := u_{m1} + \lambda_A \times \psi_A(k, m, N_A)(u'_{k1} - u_{m1})$$

$$u_{m2} := u_{m2} + \lambda_A \times \psi_A(k, m, N_A)(u'_{k2} - u_{m2})$$

for all action units, m.

8. Update *all* Q-values towards the corrected return proportionally to the Q-learning rate and the product of the two neighbourhoods (neighbourhood Q-learning):

$$Q(s_m, a_n) := Q(s_m, a_n) + \alpha \times \psi_S(j, m, N_S) \times \psi_A(k, n, N_A) \times \left(r + \gamma \max_i Q(s'_j, a_i) - Q(s_m, a_n) \right)$$

for all states units m, and actions units n.
9. Update the input map towards vector I according to the usual SOM update rule:

$$w_{m1} := w_{m1} + \lambda_S \times \psi_S(j, m, N_S)(I_1 - w_{m1})$$

$$w_{m2} := w_{m2} + \lambda_S \times \psi_S(j, m, N_S)(I_2 - w_{m2})$$

for all state units, m.
10. return to 1.

where u_{ki} is the ith weight of the kth unit of the action map (w are the weights of the input map), $random(-1, 1)$ yields a random number drawn evenly from the range [− 1,1] (Gaussian noise could also be used for example), ε controls the amount of exploration in the action space, s'_j is the state immediately following s_j, λ_A is the learning rate of the action map, λ_S is the learning rate of the input map, α is the Q-learning rate, and $\psi_S(j, m, N_S)$ is the value of the neighbourhood function of the input map at unit m given winning unit j and a neighbourhood size N_S (similar for ψ_A, only for the action map). A simple linear neighbourhood is used so that $\psi_S(j, m, N_S) = \max(0, 1 - (d/(N_S + 1)))$ where d is the distance between units j and m in the topology of the map. Learning rates α, λ_S and λ_A, and neighbourhood sizes, N_S and N_A, pertain to the amount of plasticity in the system, while p and ε control the amount of exploration. All these parameters must be derived empirically and annealed together throughout learning. Note that learning effectively occurs in all parts of the system simultaneously. That is, the input map forms at the same time as the action map is exploring the action space, and the values of the Q-table are being estimated. The model is illustrated in Fig. 4.

5. Experiments

5.1. One-to-one mapping

Although the algorithm has been presented for the general case of delayed reward, in the following experiments the special case of immediate reward is considered and so the one-step corrected return in steps 7 and 8 above can be simplified to just the immediate reward, r. See Smith (2001b) for an application of the model to delayed reward, and also the discussion in Section 6.

The first control problem requires a mapping to be

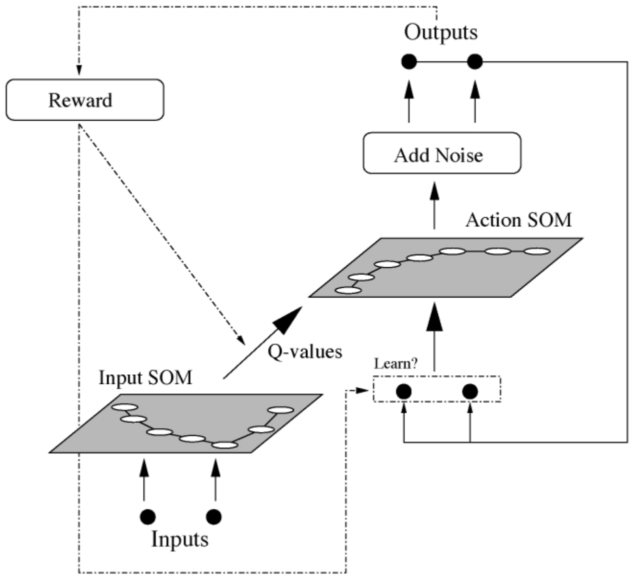

Fig. 4. The proposed model. The *input map* sits in the input space and its units are interpreted as dynamic states of the environment. The *action map* sits in the action space and its units are interpreted as dynamic actions available to the learning system. A *Q-value* (estimate of expected return) is maintained for each state–action pairing. The input map forms in response to the raw state information so that only states relevant to the problem are represented. The action map explores the action space with random noise which is reduced as learning progresses. The action map is updated towards *perturbed actions* which improve on the estimate of expected return. *Q-learning* proceeds in the usual way between the units of the input and action maps.

learned from a continuous two-dimensional state space to a continuous two-dimensional action space. Fig. 5 shows the task to be learned. A goal is generated at random on the circle shown, and the Cartesian coordinates of the goal provided as input to the learning agent. The agent must then output a suitable set of joint angles so that the tip of the arm touches the goal. After an action is taken, reward is immediate and is simply the negative of the distance between the tip of the arm and the goal. Hence the task is to

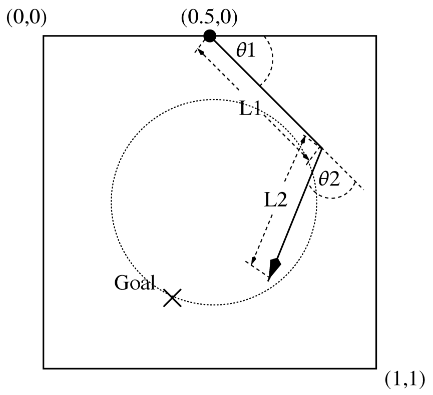

Fig. 5. A simulated two-jointed arm residing inside the unit square. The base of the arm is fixed at (0.5,0), and the position of the end point of the arm is defined by the lengths of the two arm segments ($L1$ and $L2$), and two relative angles, $\theta1$ and $\theta2$, measured in radians and restricted to the range [− π,π]. Note that $\theta1$ is given relative to the 'x-axis'.

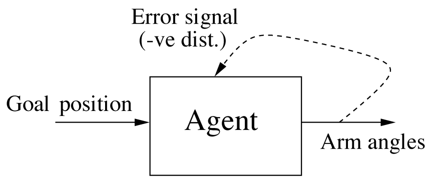

Fig. 6. The task is to learn a mapping from goal space to joint-angle space using an immediate reward signal. The outputs of the agent are $\theta 1$ and $\theta 2$. Note that these are not angles to move *through*, but angles to move *to*. $\theta 1$ and $\theta 2$ uniquely describe the arm configuration.

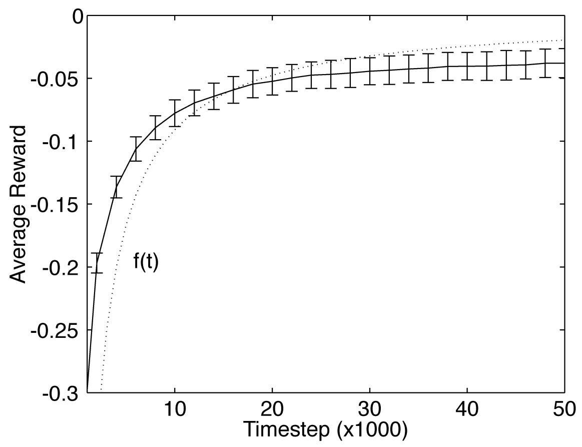

Fig. 7. Average reward against time using the parameters of Table 1. Each data point is averaged over the last 1000 time-steps for each trial, and the plot shows the mean of these data points over 20 independent trials. The error bars show the standard deviation of this mean. The dotted curve labelled $f(t)$ shows the fastest annealing schedule that was empirically judged not to significantly compromise final performance (see Table 1).

learn a mapping from goal space to arm space as shown in Fig. 6. It is important to note that this is not supervised learning because the reward signal does not provide the actual solution to the agent, only an impoverished signal conveying the distance (but not direction) from optimality.

Table 1 shows the set of empirically derived parameters used to achieve the results in Fig. 7. Performance depends crucially on the speed with which exploration and plasticity are annealed. Clearly the faster exploratory noise can be reduced the better, but this must be balanced against the need to maintain sufficient exploration for adequate search of the action space. The graph shown is for the fastest annealing schedule (also shown) which was observed not to significantly compromise final mean performance. Fig. 8 shows performance for each of the 20 trials. Note that a single trial fails to reach close to mean performance. The most likely reason for this anomaly is that on that particular trial, adverse initial conditions meant that plasticity and exploration were annealed too quickly and before an adequate mapping had had chance to form.

While the graph of Fig. 7 gives an idea of quantitative performance, the following qualitative illustrations are rather more interesting. Figs. 9 and 10

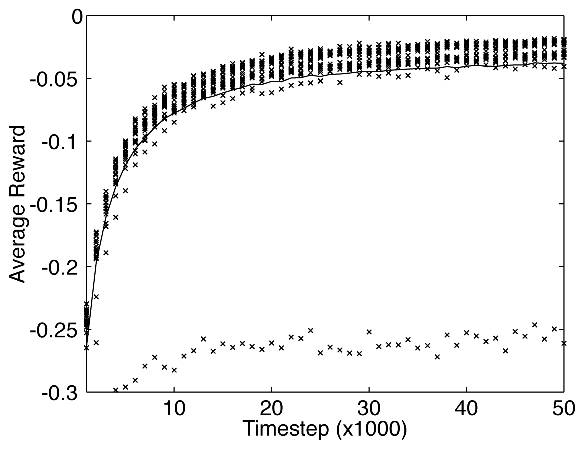

Fig. 8. As for Fig. 7 except data from each trial is plotted. For the given annealing schedule, one trial out of 20 failed to achieve close to mean performance.

Table 1
An empirically derived set of appropriate parameters for the proposed model's application to the multi-jointed arm problem. Initially, all Q-values are set to zero and all action SOM weights are generated randomly and evenly in the range $[-\pi,\pi]$ (input SOM weights in the range [0,1]). t is the time-step in which a single cycle of the algorithm is performed

Parameter	Value
Input map size	50×1 units
Action map size	50×1 units
Input map neighbourhood size, N_S	$20 \times f(t)$
Action map neighbourhood size, N_A	$20 \times f(t)$
Q-learning rate, α	$f(t)$
Learning rate of input map, λ_S	$f(t)$
Learning rate of action map, λ_A	$f(t)$
Probability of Q-learning exploration, p	$f(t)$
Max. exploration distance around action unit, ε	$0.1 \times f(t)$
Annealing schedule, $f(t)$	$1/((t/1000)+1)$

show the input and action maps after learning on a typical trial. The input map has, predictably, responded to the input distribution (goal positions). The action map, through a process of trial and error, has learned to represent those actions which offer optimal solutions to the states represented in the input map. The greedy strategy (mapping input units to action units) is indicated by the shading. It can be seen that topology is preserved not only in the two maps, but also in the Q-table, since neighbouring input units propose neighbouring action units. The performance graph of Fig. 7 suggests that the action units do indeed occupy highly rewarded regions of the action space for the whole range of inputs.

That the action map is capable of proposing sensible actions for a range of inputs is confirmed in Fig. 11 which shows the behaviour of the agent in response to a number of different goal positions in a similar experiment involving a 20-dimensional arm. Of interest here is that as an

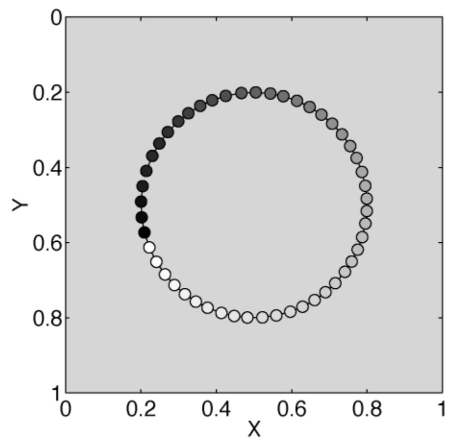

Fig. 9. Input map plotted in input space after learning on a typical trial. Units are shaded according to the action unit with the highest Q-value for that input unit (see Fig. 10).

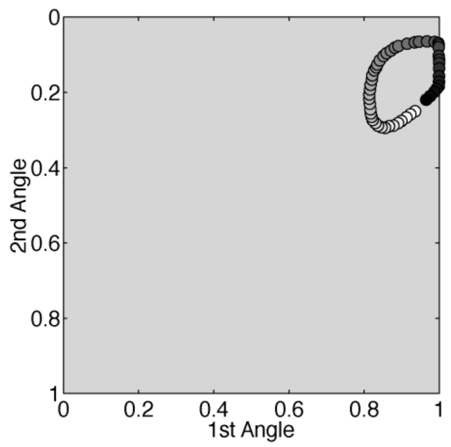

Fig. 10. Action map plotted in action space after learning on a typical trial. Each angle is normalised to the range [0,1] (from the range $[-\pi,\pi]$). Units are shaded according to their topological index in the action map.

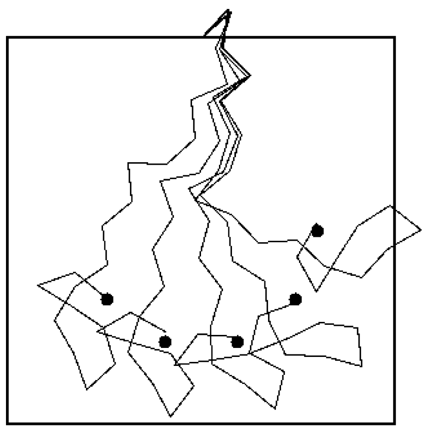

Fig. 11. Topological preservation is apparent in a 20-dimensional action space after learning in a typical trial of a second experiment.

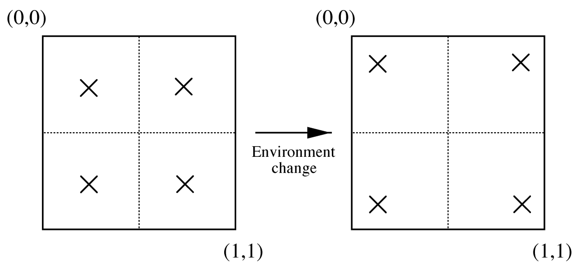

Fig. 12. The agent is presented with real-valued input drawn randomly and evenly from the unit square. This time the agent must learn to reach one of four discrete goal positions depending on which quadrant of the input space the input vector is drawn from. For example, if the input is drawn from the top-left quadrant, then the goal position corresponds to the top-left cross. Reward is given as before, as the negative distance between the arm tip and the relevant goal. During training, the positions of the four goals are moved as shown in the diagram on the right. The structure of the task suggests a many-to-one mapping from states to actions would be desirable.

automatic consequence of topology preservation in the Q-table, proximal goals are reached with similar arm configurations. Such a property may be useful since moving from one goal to another nearby goal can be achieved with minimal changes to joint angles.[4] Also note that in this example, if we assume that we do not know the optimal arm-configurations beforehand, then it would be infeasible to speculatively populate the high dimensional action space with a sufficient number of a priori fixed actions for the agent to choose between. Here the RL agent has to be able to adapt its response from a continuous range in order to solve the problem to any acceptable level of precision.

5.2. Many-to-one mapping

A second experiment demonstrates the ability of the proposed model to capture and exploit the structure of the task providing this structure is known beforehand. Fig. 12 outlines the experiment, which returns for convenience to the two jointed arm. This time the input is drawn randomly and evenly from the whole of the unit square, but the target is only ever one of four discrete goals. The structure of the task suggests a many-to-one mapping from states to actions is desirable. Representing this many-to-one structure will be useful for adapting to the environmental change shown in the second diagram where the four goal positions are moved, but the mapping from state-space to goals remains the same.

The quantitative results (using the parameters in Table 2) are shown in Fig. 13. The environment is changed at $t = 20{,}000$ (see Fig. 12), and it can be seen in graph 'A' how the residual plasticity in the system allows the proposed model to adapt its representations to the new

[4] The SOM's tendency to minimise its final length is a feature that has been exploited to optimise performance in other control applications. See the application to the Travelling Salesman Problem in Favata and Walker (1991), for example.

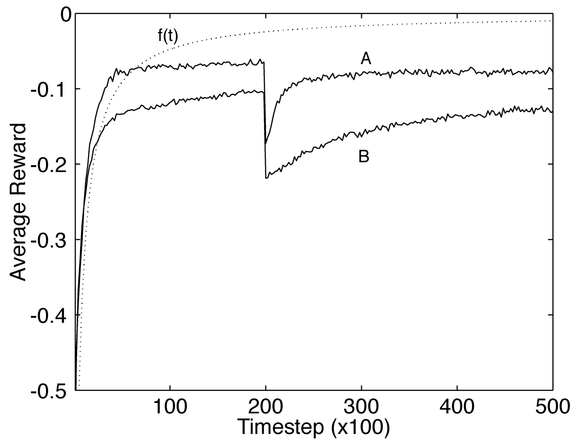

Fig. 13. Average reward over 20 trials plotted against time for the proposed model (graph 'A') and also the *Motoric Map* approach reviewed by Wedel and Polani (1996) (graph 'B'). The proposed model learns more quickly than the Motoric Map in both the initial stages and after the change in the environment because it is better able to take advantage of the structure of the task. The SD of the mean is small and not shown. Data points are averaged over the 100 previous time-steps.

Table 2
An empirically derived set of appropriate parameters for the proposed model's application to the multi-jointed arm problem of Fig. 12. A two-dimensional SOM is now used to map the input space, and a smaller action map is used to reflect the smaller number of discrete actions required. Minimum values for some parameters are also specified in order to maintain a small amount of residual plasticity that will permit continual adaptation in response to a dynamic environment

Parameter	Value	Min.
Input map size	10×10 units	
Action map size	8×1 units	
Input map neighbourhood size, N_S	$10 \times f(t)$	
Action map neighbourhood size, N_A	$3 \times f(t)$	
Q-learning rate, α	$f(t)$	0.1
Learning rate of input map, λ_S	$f(t)$	
Learning rate of action map, λ_A	$f(t)$	0.1
Probability of Q-learning exploration, p	$0.25 \times f(t)$	
Max. exploration around action unit, ε	$0.1 \times f(t)$	0.01
Learning schedule, $f(t)$	$1/((t/500) + 1)$	

task. Graph 'B' was obtained by using an efficient version of the *Motoric Map*[5] in which the input space is again mapped with a SOM, but where each input unit maintains a single, private action which is learned in the same way as the proposed model—i.e. by using random noise as exploration. The disadvantage of this approach is that a separate action must be learned for each state, discovered actions may not easily be shared across the input space, and the implicit 'many-to-one' structure of this particular task cannot be captured. This is why learning is slower in graph 'B'. For completeness, Figs. 14 and 15 show the

[5] First proposed by Ritter, Martinetz, and Schulten (1990), although the account is taken from Wedel and Polani (1996). See Smith (2001a,b) for a full description, and Wedel and Polani (1996) for an extended application.

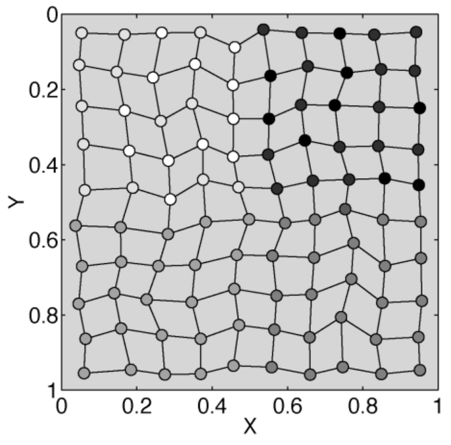

Fig. 14. The input map plotted in input space just before the environment change on a typical trial of the structured task. As before, each unit is shaded according to the action unit for which it has the highest Q-value (see Fig. 15).

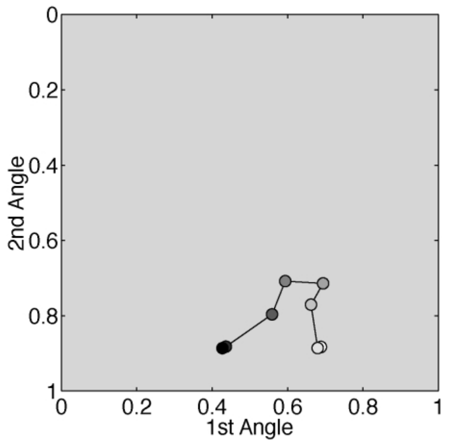

Fig. 15. The corresponding action map for Fig. 14. Each of the eight action units is shaded according to its topological index in the action map. Angles are normalised as before. After the environmental change, only the action map need be adapted and these changes can then be shared across the units of the input map.

input and action maps immediately prior to the environment change. After the change, only the free parameters in the action map need be adapted in order to re-optimise performance. Note that although eight units are provided in the action map, not all units end up being involved in the solution. We also note from this experiment that the final state–action mapping need not be contiguous (although neighbourhood Q-learning will encourage this phenomena) and that the system can cope with non-continuous reward functions.[6] See Smith (2001b, Chapter 5) for more examples of non-continuous reward functions and non-contiguous state–action mappings (in terms of SOM topology).

[6] The reward function here is $C(1)$ discontinuous because a small transition in input space can lead to a different goal becoming the target.

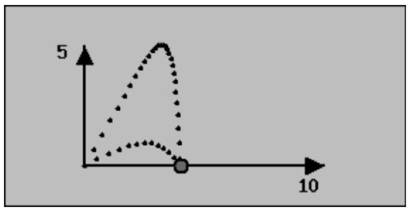

Fig. 16. Two sample trajectories out to a target positioned four units along the horizontal axis.

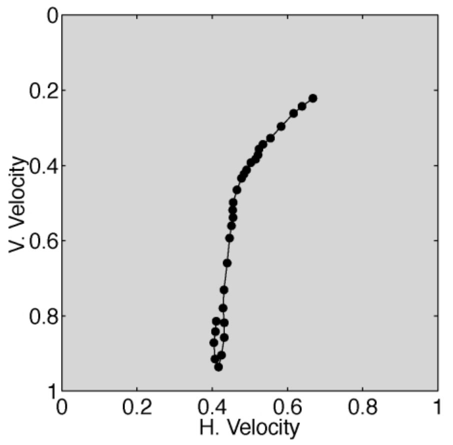

Fig. 17. The action map after learning in a typical trial under the environment parameters of Table 3 and the model parameter of Table 4.

5.3. One-to-many mapping

So far, problems involving one-to-one and many-to-one state–action mappings have been presented. In the latter case, the many-to-one structure of the task could be exploited through the sharing of information in the action map across different units of the input map. The next experiment considers a different problem in which the optimal mapping from states to actions is one-to-many; in other words, there are multiple optimal actions for the same state.

Consider the problem of throwing a projectile to a target. Imagine a javelin thrower standing at the origin of a Cartesian axis, and a target lying somewhere along the x-axis. The javelin thrower must select an appropriate horizontal and vertical throwing speed (u_x and u_y, respectively) at which to launch the projectile so that it lands as close to the target as possible. Fig. 16 shows an example of two different trajectories to a target positioned four units along the horizontal axis. The initial horizontal and vertical velocity components are restricted to the range [0,1] which limits how far the projectile can be thrown. The labelling on the axes reflects this. The environment model is (arbitrarily) based on the mechanics of the physical world with the parameters of Table 3.

Purely for convenience air resistance is only modelled in the horizontal direction, so we can describe the vertical

Table 3
The parameters of the environment along with some initial values. Units of measurement are implicit as the environment is not specifically intended to follow any particular real physical system. Terms like *air resistance* and *gravity* enjoy only metaphoric sense here

Parameter	Value
Mass of projectile, M	1
Gravity, G	0.05
Coefficient of friction due to air resistance, Fr	0.1
Horizontal wind speed against the thrower, W	0
Horizontal position of target, X	4

position, s_y, of the projectile at time T by:

$$s_y(T) = \int_0^T u_y - t \times G \, \mathrm{d}t = u_y T - \frac{1}{2} G T^2 \tag{5}$$

Similarly, the horizontal distance travelled at time T can be described by[7]

$$s_x(T) = \int_0^T (u_x + W)\mathrm{e}^{-\mathrm{Fr}t} - W \, \mathrm{d}t$$

$$= \frac{(u_x + W)(\mathrm{e}^{-\mathrm{Fr}T} - 1)}{-\mathrm{Fr}} - WT \tag{6}$$

From Eq. (5), the time at which the projectile lands is given by $T_{\mathrm{landing}} = ((2u_y)/G)$, and then substituting this into Eq. (6) gives the landing position:

$$s_x(T_{\mathrm{landing}}) = \frac{(u_x + W)(\mathrm{e}^{-\mathrm{Fr}(2u_y/G)} - 1)}{-\mathrm{Fr}} - W\left(\frac{2u_y}{G}\right) \tag{7}$$

The model details are included here for reproducibility rather than specific interest, since the choice of mechanics is somewhat arbitrary. The motivation is to create a suitable problem with which to investigate the behaviour of the proposed model.

The role of agent is to learn to hit the target using a reinforcement signal. After each throw, a reward is given that is equal to the negative of the distance between the landing point of the projectile and the target, i.e. $r = -|X - s_x(T_{\mathrm{landing}})|$. Note that in this first problem the system only has a single state (corresponding to being in a throwing position) and so only a single input map unit is needed. Fig. 17 shows the action map plotted in the two-dimensional action space after 4000 throws[8] for the parameters in Table 4. The units of the map have identified 30 different trajectories for reaching the target ranging from low, hard throws to high lobs. Note that the unusually high value of p in Table 4 encourages all units of the action map to be involved in solving the problem.

By rearranging Eq. (7), the relationship between u_x and u_y can be derived for which the target is hit by the

[7] Now including the effect of wind resistance, which generates a horizontal force in the opposite direction to motion that is proportional to the speed of the projectile.

[8] No attempt is made to optimise the speed of learning in this experiment.

Table 4
An empirically derived set of suitable parameters for the proposed model's application to the projectile throwing problem. 'throw' is the number of the current throw and replaces the parameter, t, in the previous experiments. The ad hoc annealing schedule was empirically derived and could be replaced by a more conventional schedule of the form 1/throw without compromising performance. Note that although the latter schedule has some useful convergence properties in the limit of infinite trial length, in the case of finite trials other schedules are found to work equally well. Minimum values are maintained for some parameters in order to achieve the best looking maps

Parameter	Value	Min.
Action map size	30×1 units	
Action map neighbourhood size	$15 \times f$(throw)	2
Q-learning rate, α	f(throw)	0.05
Learning rate of action map, λ_A	f(throw)	0.1
Probability of Q-learning exploration, p	f(throw)	0.5
Max. exploration distance around action unit, ε	f(throw)	0.1
Annealing schedule, f(throw)	0.999^{throw}	

projectile:

$$u_x = -\mathrm{Fr}\frac{X + \dfrac{2Wu_y}{G}}{\mathrm{e}^{-\mathrm{Fr}(2u_y/G)} - 1} - W \quad (8)$$

Fig. 18 shows this relationship plotted for all real values of u_x and u_y (where both lie in the range [0,1]), which neatly corresponds to the set of solutions found by the learning system. Interestingly, although the RL system clearly adheres to the analytical solution, it appears to be reluctant to use the very hardest and lowest throws which correspond to the missing part of the curve in Fig. 17. It is likely that this is due to the variation in the reliability of different types of throw. To illustrate this, consider first Fig. 19, which shows the landing position of the projectile as a function of throwing velocity (same model parameters from Table 3), and then Fig. 20, which shows the first derivative (angle of steepest descent) of this function. The observation is that for the very lowest and hardest throws, the rate of change in the landing position with respect to the throwing velocity is largest. This means that small, evenly distributed deviations in the throwing velocity as a result of the random noise generated by exploration will have the greatest impact on reward at this point. In other words, given that there is noise in the system, low hard throws are the least reliable way to hit the target. The Q-values will reflect reliability because they are estimating *expected* reward.

Note that generally there is no reason why the entire range of actions should be discovered since any subset of points, or indeed any single point on the solution curve, will be sufficient for achieving optimal behaviour. The action map in Fig. 17 shows the potential for discovering multiple actions given the stochastic nature of the search process, but if *all* possible solutions are required with certainty then extra care must be taken to ensure adequate exploration of

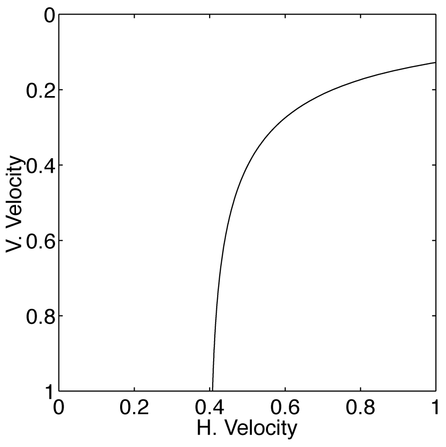

Fig. 18. Analytical solution to the problem of maximising reward under the environment parameters of Table 3.

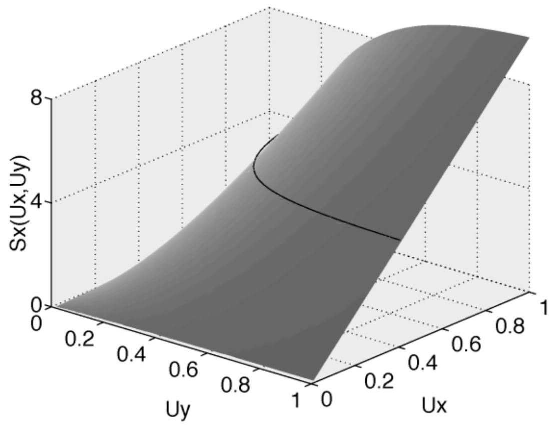

Fig. 19. Landing position of the projectile as a function of the throwing velocity. The solid line denotes all trajectories that reach a target at $X = 4$.

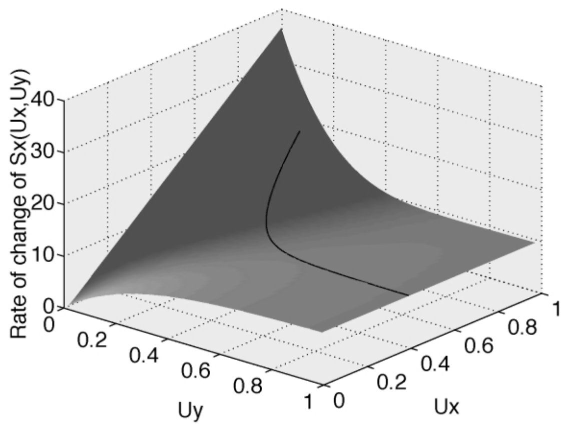

Fig. 20. Numerical approximation of the first derivative (angle of steepest descent) of the landing position with respect to the throwing velocity. Again the solid line denotes all trajectories reaching a target at $X = 4$.

the space (achieved here through a large value of the exploration parameter, p).

The problem is now extended by parameterising the task with respect to the position of the target along the x-axis. This changes the problem to one involving a single-

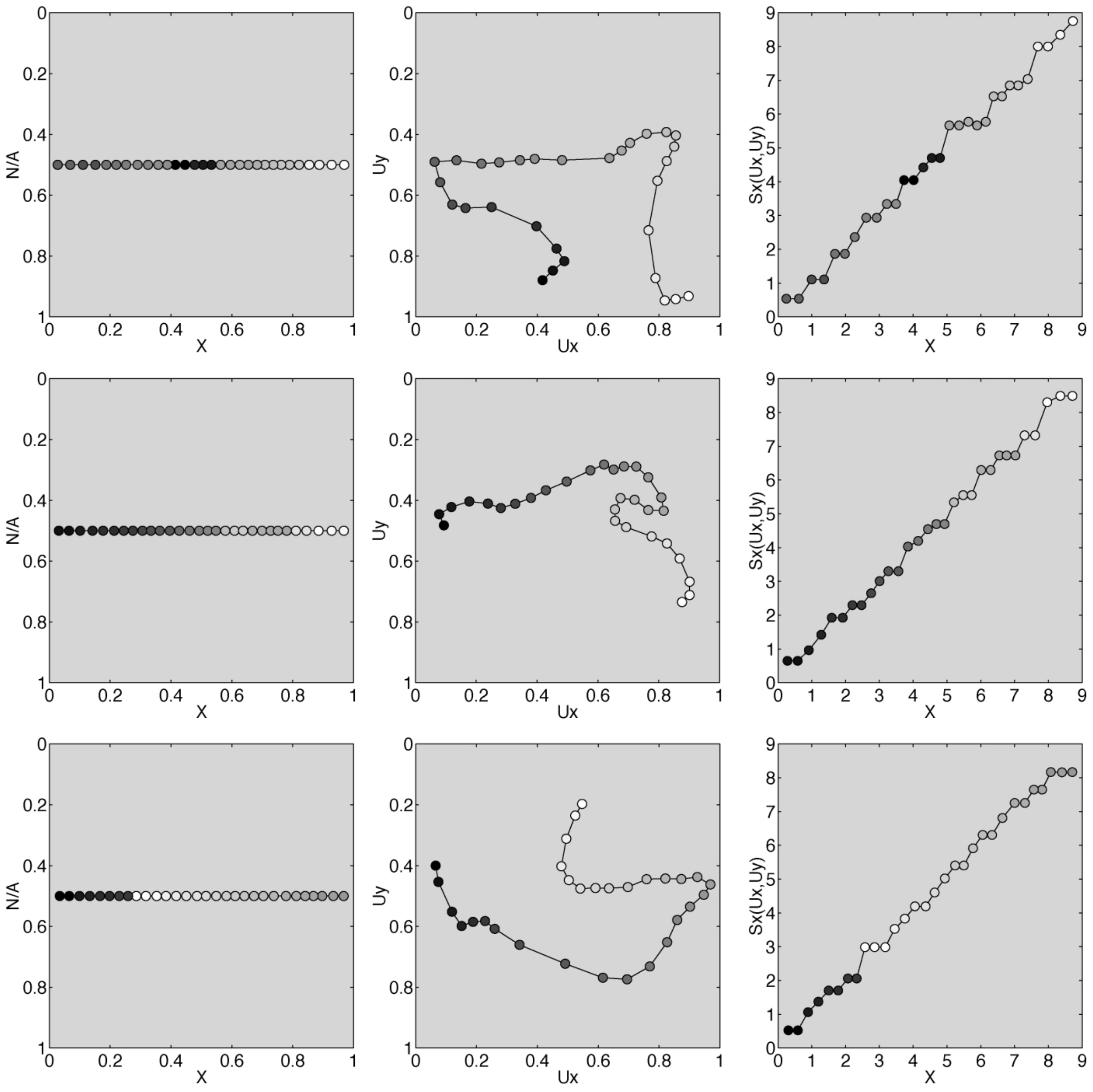

Fig. 21. Input maps (left) (input normalised to the range [0,1]) and corresponding action maps (centre) after learning in three independent trials of the experiment involving a varied target position. Action units are shaded simply according to their index within the map. As before, input units are shaded according to the action unit for which they have the highest Q-value. The rightmost plot in each row shows the landing position of the projectile (y-axis) for the best action associated with each unit of the input map (the input map is plotted along the x-axis). In this plot the input space is no longer normalised and it is evident that the favourite action for each state does a good job of reaching the targets to which that state unit responds. In these rightmost plots each input unit is shaded as in the leftmost plot—i.e. according to the favourite action unit for that input.

dimensional state space where the inputs to the system are goal positions drawn randomly and evenly from the range [0,9]. To map this space, a one-dimensional input map of 30 units is used, with an initial neighbourhood of 15, annealed in the usual way. In this experiment, minimum values are no longer used for any of the parameters. The new (or altered) parameters are given in Table 5.

The input and action maps after learning in three independent trials of this new experiment are shown in Fig. 21. In all cases, the input map accurately and smoothly represents the range of possible target positions by equally spacing the 30 states in the single dimensional state-space. The action maps vary across the trials, but recall that each target position is satisfied by a range of throws within the

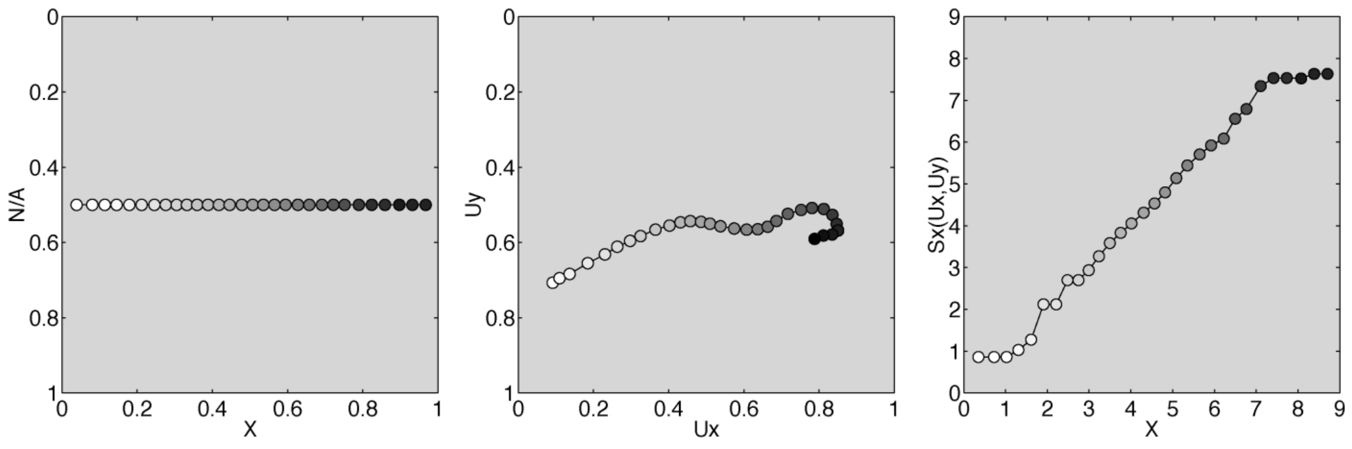

Fig. 22. Input and action maps for the same setup as Fig. 21, except that a *larger* neighbourhood of $50 \times f(\text{throw})$ is used.

space. This gives the action map a large amount of freedom to represent this space, and in fact all runs performed well in terms of maximising reward (the graph labelled +NL/0.9985 in Fig. 24 shows the average performance over 20 runs). We can also see from the shading in the input maps that although topology preservation is encouraged in the Q-table, the system does not necessarily always arrive at a contiguous mapping—i.e. sometimes neighbouring input units do not map to neighbouring action units. The final diagram in each row shows the landing position of the projectile (y-axis) for the best action associated with each unit of the input map (the input map is plotted along the x-axis). It is evident that the favourite action for each state does a good job of reaching the targets to which that state unit responds.

There are essentially two forces present in the action map. The first is the attraction of units towards highly rewarded regions of the space, and the second is the pull that neighbours exert on each other as imposed by the familiar SOM learning rule.[9] This second force is necessary because it ensures competition between regions of the action space for action units, thereby encouraging *all* units to occupy popular and profitable regions. This second force also encourages topology preservation which can then be used to expedite the passing of reward information around the Q-table through neighbourhood Q-learning.

However, the force exerted on an action unit by its neighbours is also disruptive since it will tend to drag that unit away from its preferred position—a position that it is attempting to discover for itself through a costly trial and error process. It is therefore important to strive for a balance between these two forces that respects both the individual unit and the society of the map as a whole. This will then allow for a suitable tradeoff between competition in the action space and the ability of units to independently explore this space.

As a brief illustration consider Figs. 22 and 23. The graphs show the input and action maps for similar experiments to those reported in Fig. 21, except that the size of the neighbourhood is varied. Fig. 22 shows the effect of increasing the neighbourhood and therefore the second force referred to above. The consequence is a more continuous action map with greater topology preservation in both the maps and the Q-table (indicated by the contiguous mapping). From this we note that increasing the strength of the neighbourhood is one way to apply additional constraints to an under-constrained mapping. Fig. 23 shows the effect of using a smaller neighbourhood. Here the action map is less well organised, topology is compromised in the action map, and there is greater discontiguity in the mapping between input and action units. However, we note that the actual performances (in terms of the final reward averages) associated with Figs. 21–23 are very similar (as can be deduced from the rightmost plots), and we should not be unduly prejudiced by the appearance of Fig. 23.

5.4. Neighbourhood Q-learning

This section briefly quantifies the benefit afforded by neighbourhood Q-learning. Fig. 24 shows three learning curves. The first to consider is the one labelled +NL/0.9985 which corresponds to the previous experiment of Fig. 21, which utilises neighbourhood Q-learning and an annealing schedule of 0.9985^{throw}. This was empirically found to be the fastest annealing rate that did not compromise final performance. The same experiment without neighbourhood

Table 5
Additional (or altered) empirically derived parameters for a one-dimensional state space

Parameter	Value
Input map size	30×1 units
Input map neighbourhood size, N_S	$15 \times f(\text{throw})$
Action map neighbourhood size, N_A	$25 \times f(\text{throw})$
Learning rate of input map, λ_S	$0.3 \times f(\text{throw})$
Annealing schedule, $f(\text{throw})$	0.9985^{throw}

[9] Note that this second force is quite distinct from *neighbourhood Q-learning* which is only concerned with the Q-table, not the positions of action units.

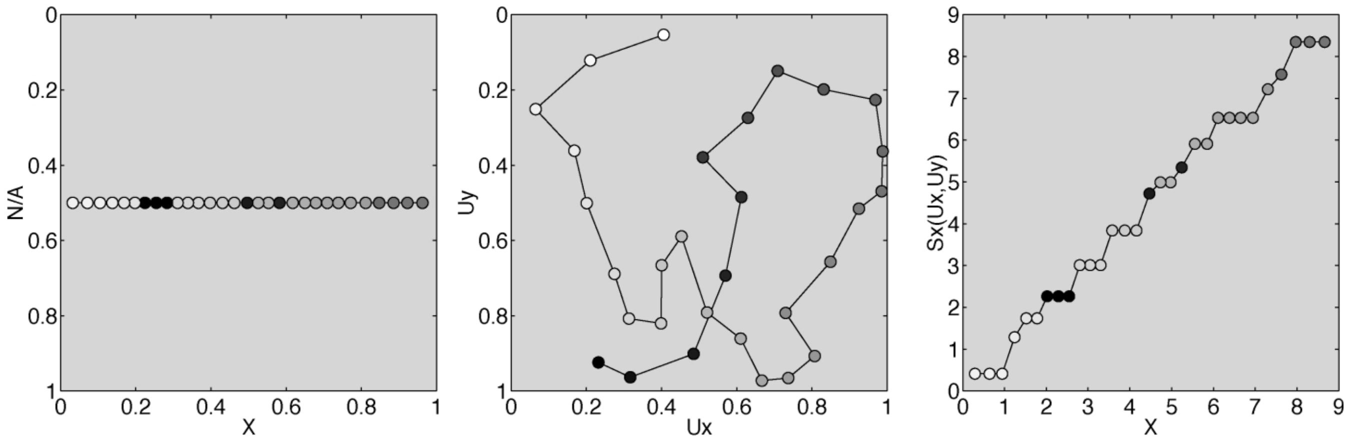

Fig. 23. Input and action maps for the same setup as Fig. 21, except that a *smaller* neighbourhood of $5 \times f(\text{throw})$ is used.

Q-learning yields the curve labelled −NL/0.9985, showing a marked decrease in both learning speed and final performance. It turns out that learning is so handicapped by not using neighbourhood Q-learning that the annealing schedule must be decreased to 0.9997^{throw} before final performance comes close to achieving that of the original experiment (see curve −NL/0.9997). The annealing schedule numbers of 0.9997 and 0.9985 are rather meaningless on their own, so the actual schedules are superimposed on the graph. The main result is that in this experiment, neighbourhood Q-learning is able to expedite convergence to near-optimal performance by a factor of at least six. Interestingly, this speedup factor is consistent with the approximation obtained by Smith (2001b, Chapter 5), even though that experiment involved a very different task.

Table 6
Four experiments involving different numbers of variables, with each variable selected randomly and evenly from within a specified range. Both variable ranges and, where appropriate, constant values are indicated. Care must be taken to ensure that the target can always be reached (in principle) from anywhere within the input space

Experiment	Distance	Wind	Co. Fr.	Mass	No. of variables
A	[0,9]	0	0.1	1	1
B	[0,9]	[−3,0]	0.1	1	2
C	[0,9]	[−3,0]	[0,0.2]	1	3
D	[0,9]	[−3,0]	[0,0.2]	[1,2]	4

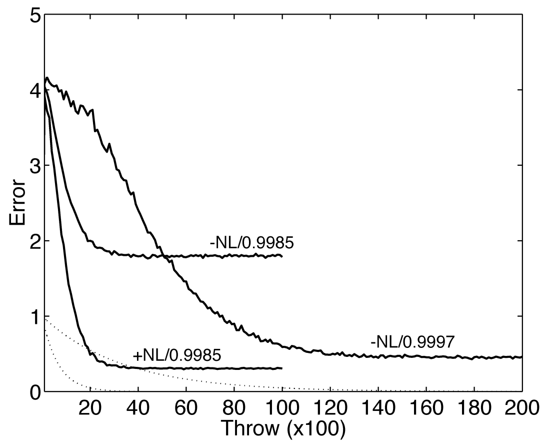

Fig. 24. Three learning curves are shown plotting the error, averaged over 20 trials, against the number of throws for the experiment of Fig. 21. Here, the error is defined as the negative of reward. Each curve is labelled according to whether neighbourhood Q-learning is used (+NL) or not (−NL), and according to the annealing schedule used. The two different annealing schedules (0.9997^{throw} and 0.9985^{throw}) are also plotted as the dotted lines. Error bars can be considered negligible for this comparison and are not shown.

5.5. Higher dimensional spaces

The problem can be extended further by increasing the dimensionality of the input space. In a final experiment, different numbers of parameters from Table 3 are varied and provided as a multi-dimensional input vector to the learning system. Four different experiments are performed, each one adding a new variable dimension to the problem. As before, reward is immediate and is simply the negative of the distance between the landing position and the target.[10] Table 6 summarises the experiments, Table 7 summarises the model parameters, and Fig. 25 shows the results. Performance predictably deteriorates as a result of over generalisation as the latent dimensionality of the input space is increased. It is also expected that as either the input or action maps fold into a higher dimensional space than that of their own topology, then neighbourhood learning will become less effective. These experiments highlight a potential drawback associated with using a SOM in high latent dimensional input and action spaces. However, the experiments also show

[10] To best demonstrate the effect of higher dimensional input spaces and to remove a way to 'cheat' discovered by the learning system, an additional punishment of −2 is added to the reward signal if the throw is below 45°. This detail is of no significance to the nature of the results presented here and is explained in detail in Smith (2001b, Chapter 8).

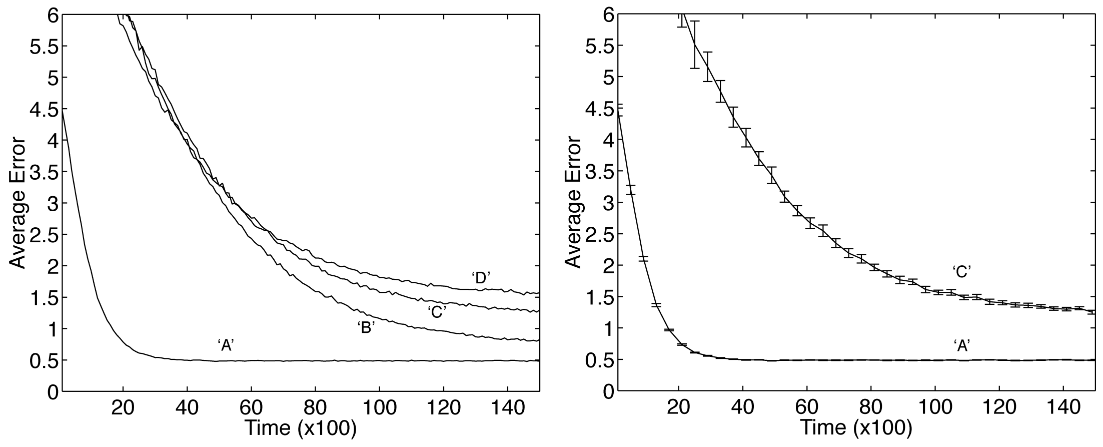

Fig. 25. (Left) Performance graphs for the four experiments of Table 6 under the model parameters of Table 7 averaged over 100 independent trials. Within each trial, each data point is also averaged over the last 100 throws. Each curve represents the best set of parameters found for that experiment. The standard deviations of the mean are negligible, but (right) shows the variance of the actual data for two of the curves. The variance of 'C' is typical also of 'B' and 'D'.

that the model can still acquire reasonable behaviour even when the latent dimensionalities of the input and action spaces exceed those of the respective maps. See Smith (2001b, Chapter 8) for more experiments and a more detailed analysis of the drawbacks of the model in high dimensional state spaces. This issue is returned to in Section 6.2.

6. Discussion

The proposed model was originally introduced to address the general case of delayed reward problems, although in the experiments performed here only immediate reward has been considered. The model has previously been successfully applied to a delayed reward problem involving simulated robot navigation (Smith, 2001b, Chapter 5) in which the aim of the agent was to learn a mapping from real-valued distance sensors to real-valued motors in order to maximise a reward signal which punished the agent for close contact with obstacles. In this experiment, the state space was six-dimensional and the action space two-dimensional. However, doubt remains as to whether this particular problem actually requires dynamic real-valued actions as opposed to using a small set of discrete actions which are fixed a priori (as in Mahadevan and Connell (1991) for example).

Another delayed reward problem to which the proposed model has been successfully applied is the cart-pole balancing problem. Here, the goal is to apply horizontal forces to a cart on a track in order to keep a pole from falling.[11] In the standard version of this task a four-dimensional state space[12] must be mapped to a single dimensional action space (corresponding to the force to be applied to the cart) in order to maximise a reward signal which only punishes the system when the pole falls below a given angular threshold. But it has already been established that this problem is easily solved using general RL techniques in which there are just two discrete actions corresponding to a right and left push of the cart (Peng & Williams, 1996), and so the usefulness of adaptable real-valued actions in this context is again questionable. If latency is introduced into the system so that the agent can only sample the state of the environment relatively infrequently, then a more subtle real-valued response may be useful for maintaining the pole at its non-stable equilibrium. But even now, because the action space is low-dimensional, it may just be easier to hardwire fixed actions at regular intervals within the space and proceed with standard Q-learning. The upshot of this discussion is that although preliminary experiments suggest that the model *is* suited to the general case of delayed rewards, finding problems of this kind which can benefit from adaptable real-valued actions is not so easy. This is because

Table 7
The model parameters for the four experiments of Table 6. Experiment A exceptionally used a single dimension input map (30×1, neighbourhood $= 7 \times f(\text{throw})$) and was able to utilise a faster annealing schedule, $f(\text{throw}) = 0.9985^{\text{throw}}$

Parameter	Value
Input map size	10×10 units
Action map size	30×1 units
Input map neighbourhood size, N_S	$7 \times f(\text{throw})$
Action map neighbourhood size, N_A	$7 \times f(\text{throw})$
Q-learning rate, α	$f(\text{throw})$
Learning rate of input map, λ_S	$0.3 \times f(\text{throw})$
Learning rate of action map, λ_A	$f(\text{throw})$
Probability of Q-learning exploration, p	$f(\text{throw})$
Max. exploration distance around action unit, ε	$f(\text{throw})$
Annealing schedule, $f(t)$	0.9997^{throw}

[11] The pole is hinged at its base to the cart and free to rotate around this hinge in the direction of movement of the cart.

[12] Containing information about the position and velocity of the cart, and the angle and angular velocity of the pole.

in delayed reward tasks it is usually possible to correct mistakes in subsequent time-steps and thus achieve an approximate continuous response by interleaving discrete actions.

Although the representation of the state–action mapping is discrete, it is simple to extend the model so that it produces a truly continuous response by interpolating over the Q-table. One approach would be to use smoothing basis functions over the units of both maps. If a search of the action space at action selection time is admissible, then an approach similar to the coarse-coding techniques reviewed by Santamaria et al. (1997) could be adopted with the 'cases' or prototypes being read directly from the Q-table. One key difference with the proposed model is that because the action space is searched during the learning process rather than at action selection time, actions can be reliably selected with a search of $O(|A|)$ where A is a small set of action units. The proposed model also only represents the Q-table in relevant regions of the input space and rewarding regions of the action space. See Smith (2001b, Chapter 6) for a more in-depth discussion of generating a continuous response from discrete representations.

6.1. Related work

The use of real-valued actions in RL has a number of precedents in the literature. Gullapalli (1990) uses a very simple backpropagation network (a single linear unit) to map state information to a mean and variance which are used to define an action distribution. A separate network estimates the expected immediate reward (delayed reward problems are not considered) and if the received reward is higher than the expected reward then the mean of the distribution is pushed towards the sampled action, and away from the sampled action otherwise. However, the system does not achieve robust performance even though the problems considered are relatively simple. Another issue is that the system can only suggest a single action for any given input, so there is no parallel search of the action space, and no provision for flexible state–action mappings (particularly of a one-to-many nature). A similar approach was adopted for the Q-AHC model of Rummery (1995, Chapter 5) which was based on the early RL work of Williams (1988) as well as the AHC and actor–critic models of Barto et al. (1983) and Sutton (1984). Actions are again represented by Gaussian distributions with mean and variance represented by an MLP and trained with back-propagation, but here multiple actors are used so that there is a parallel search of the action space and multiple responses can be generated for the same state. As with Gullapalli (1990), results reported are somewhat disappointing, this time because other models using a fixed hand-coded action set outperformed Q-AHC on the tasks attempted. Problems are also reported with multiple Q-AHC modules learning the same actions even though there may be other better actions still to be found. This may be a general symptom of insufficient exploration and competition in the action space.

Other MLP based approaches include the Complementary Reinforcement Backpropagation Learning algorithm (CRBP) of Ackley and Littman (1990), its adaptation found in Ziemke (1996), and also the work of Touzet (1997). In all these models, binary reward is used to drive an MLP (which is responsible for generating actions) towards one output extreme or another. Apart from being dependent on immediate and binary reward, these approaches make another key departure from the general theoretical foundations by not maintaining an explicit estimate of expected reward. The generality of these approaches is therefore questionable.

The SOM has also been used for continuous action space RL. Of significant note is the work of Wedel and Polani (1996) which presents an extension to the Motoric Map (Ritter & Schulten, 1987). Their idea is to map the input space with a SOM, and then attach a single action to each unit which must learn an appropriate action for that state. Their innovation, which they call *covariance learning*, is to sample the reward of a number of perturbed actions around the proposed action for each state (unit of the SOM). Each state then builds a Gaussian model of the reward function around the current action, and uses this model to select a suitable direction in action space in which to explore. Their premise is that a simple stochastic search of the action space will be insufficient in high dimensional action spaces. Although presented for the case of immediate reward, their algorithm is also easily extended to delayed rewards. However, problems reported include difficulty sampling (online) enough actions around a given state to fuel the modelling process, difficulty in forming the SOM and exploring actions concurrently, and problems with achieving their version of neighbourhood Q-learning in which states would share information. The architecture is also constrained to producing one-to-one state–action mappings. Touzet (1997) also applies the SOM to RL by mapping the combined input–action–reward space with a single SOM. Dimensionality problems are expected to strike particularly early with this approach. Action selection needs special attention in this model because although the reward function is mapped explicitly, a search of the action space is implied at action selection time, especially if smoothing is used. The use of immediate, binary rewards is also restricting although the model may be extendible to the more general case of real-valued, delayed reward.

Other noteworthy approaches to RL generalisation include the work of Prescott (1994) which uses the CMAC (Albus, 1975) approach to decomposing the state-space and then linear output units to generate a normally distributed action (similar to Williams (1988), Gullapalli (1990) and Rummery (1995)), and also the coarse-coding techniques reviewed by Santamaria et al. (1997) which use basis functions over a number of sampled state–action–reward prototypes to approximate the Q-function. One important difference between this and the proposed model is that with coarse-coding techniques, a search of the action space is required at action selection time, whereas in the proposed model, only a search O($|A|$) is required. Also, in the proposed model, the state–action–reward prototypes (represented by each element of the Q-table) are dynamic which means that only relevant regions of state space and rewarding regions of action space are actually represented.

6.2. Problems to address

In the later stages of learning, when the exploration parameter, ε, is small, the model is shown to approximate a hill-climb of the reward surface over the action space (Smith, 2001b, Chapter 6). This necessarily leaves the model vulnerable to local minima on this surface. However, in the early stages of learning, when ε is large, local minima need not be a problem since larger exploratory steps may be made. But a consequence of large ε and the stochastic nature of the search process[13] is that large regions of action space that yield above average reward may be favoured over smaller regions of optimal reward. According to Smith (2001b, Chapter 6), it appears that a malicious return surface could recurse this process so that arbitrarily low regions of reward are favoured. In theory this presents a significant drawback to the robustness of the model, but in practice it seems likely that reasonably well behaved return surfaces can be expected. In any case, the parallel nature of the search, achieved by having multiple units in the action map, should ameliorate these problems.

A second drawback relates to the important issue of scalability. In the examples considered so far, the latent dimensionality of both the input and action spaces has been low. Even in the experiment involving a 20-jointed arm, the latent action space representation was one-dimensional because it involved a one-to-one mapping from a one-dimensional latent state-space. Clearly the SOM approach is expected to scale badly if it is required to represent spaces of significantly higher dimensionality than the map itself, and indeed this intuition is supported in the results of Fig. 25.[14] Experiments performed by Smith (2001b, Chapter 8) also suggest that the proposed model will be vulnerable to poor performance in high latent dimensional state-spaces even when only a small number of those dimensions are relevant to solving the problem. In such cases, approaches based on distributed representations, and in particular the MLP, may be favoured since the response of these models to the curse of dimensionality is known to be more robust than local representation models such as radial basis functions, the elastic net (Durbin & Willshaw, 1987), and the SOM, for example. This intuition is partly borne out in the comparison of the proposed model with MLP based generalisation (based on the SRV units of Gullapalli (1990), and the Q-AHC model of Rummery (1995)) in Smith (2001b, Chapter 8). However, Smith (2001b, Chapter 8) identifies a number of drawbacks associated with distributed representation techniques. Of particular significance are potential problems with unpredictably distributed training data in which learned representations may be quickly overwritten by more frequent data, and also problems pertaining to flexibility and the representation of non-continuous or highly oscillatory functions. The advantage of using local representation is that different parts of the target mapping can be achieved relatively independently of other parts.

[13] The process is stochastic in the sense that the action map is only updated *towards* randomly explored actions and the Q-values only updated *towards* actual reward. This is necessary because the feedback from the environment is expected to be noisy for a number of reasons including the action space exploration itself.

[14] Similar problems will pertain for high latent dimensional action spaces.

7. Conclusion

A new model has been presented for representation and generalisation in model-less RL which is based on the SOM and standard Q-learning and which provides a number of desirable properties: real-valued states and actions are adapted dynamically and online, a real-valued and potentially delayed reward signal is accommodated, RL theory is adhered to in the sense that at the core of the model is an explicit notion of estimated expected return, the model has both an efficient learning and action selection phase, and the inherent structure of the task can be captured (i.e. one-to-one, many-to-one, one-to-many) as long as the nature of the desired state–action mapping is known beforehand. The topology preserving property of the SOM has also been shown to be able to add useful constraints to under-constrained tasks (Fig. 11), and to expedite learning through the application of neighbourhood Q-learning.

The two main drawbacks are the theoretical possibility that arbitrarily poor actions could be discovered by the system given a sufficiently malicious reward function, and the inherent scalability issues resulting from the fact that the representation uses local parameters. Distributed models may offer a solution to the scalability issue, but these models have been shown to introduce their own problems pertaining to stability, flexibility, and robustness in the face of unpredictably distributed training data. Hybrid local/distributed models such as those suggested by Smith (2001b, Chapter 9) may be of future interest to the kinds of applications considered here.

Acknowledgments

I gratefully acknowledge and thank Professor David Willshaw and Dr John Hallam for their guidance throughout this project, along with Jeremy Wyatt and Gillian Hayes for their reviews. This work was completed with the support of a research grant from the Engineering and Physical Sciences Research Council. Award number: 98318242.

References

Ackley, D. H., & Littman, M. L. (1990). Generalisation and scaling in reinforcement learning. In D. Touretzky (Ed.), *Advances in neural information processing systems* (pp. 550–557). San Mateo, CA: Morgan Kaufmann.

Albus, J. (1975). A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Dynamic Systems, Measurement and Control*, *97*(3), 220–227.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*, 835–846.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bishop, C. (1995). *Neural networks for pattern recognition*. New York: Oxford University Press.

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *Neural Information Processing Systems*, *8*.

Durbin, R., & Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, *326*, 689–691.

Favata, F., & Walker, R. (1991). A study of the application of kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, *64*, 463–468.

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, *3*, 671–692.

Kaelbling, L., Littman, M., & Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence*, *4*, 237–285.

Kohonen, T. (1987). *Self organisation and associative memory* (2nd ed.). Berlin: Springer.

Kohonen, T. (1995). *Self organising maps*. Berlin: Springer.

Lin, L. J. (1993). *Reinforcement learning for robots using neural networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, CMU-CS-93-103.

Mahadevan, S., & Connell, J. (1991). Automatic programming of behaviour-based robots using reinforcement learning. *Proceedings of the Ninth International Conference on Artificial Intelligence (AAAI 91)*, Anaheim, CA (pp. 768–773).

Oja, E., & Kaski, S. (Eds.), (1999). *Kohonen maps*. Amsterdam: Elsevier.

Peng, J., & Williams, R. J. (1996). Incremental multi-step q-learning. *Machine Learning*, *22*, 283–290.

Prescott, A. J. (1994). *Explorations in reinforcement and model-based learning*. PhD thesis, University of Sheffield.

Ritter, H., Martinetz, T., & Schulten, K. (1990). *Neuronal netze*. Reading, MA: Addison-Wesley.

Ritter, H., & Schulten, K. (1987). Extending kohonen's self-organising mapping algorithm to learn ballistic movements. *Neural Computers*, *F41*, 393–406.

Rojas, R. (1996). *Neural networks: a systematic introduction*. Berlin: Springer.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing* (*vol. 1*). Cambridge, MA: MIT Press.

Rummery, G. A. (1995). *Problem solving with reinforcement learning*. PhD thesis, Cambridge University.

Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionist systems*. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University.

Santamaria, J. C., Sutton, R. S., & Ram, A. (1997). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, *6*(2), 163–217.

Sehad, S., & Touzet, C. (1994). Self-organising map for reinforcement learning: Obstacle avoidance with khepera. *Proceedings of From Perception to Action*, Lausanne, Switzerland. IEEE Computer Society Press.

Smith, A. J. (2001a). *Dynamic actions in reinforcement learning*. http://www.dai.ed.ac.uk/homes/andys/.

Smith, A. J. (2001b). *Dynamic generalisation of continuous action spaces in reinforcement learning: A neurally inspired approach*. PhD dissertation, Division of Informatics, Edinburgh University, UK.

SOM-database (2001). http://www.cis.hut.fi/research/som-bibl/.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning*. Cambridge, MA: MIT Press.

Tani, G. J., & Fukumura, N. (1994). Learning goal-directed sensory-based navigation of a mobile robot. *Neural Networks*, *7*(3), 553–563.

Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277.

Tesauro, G. J. (1994). Td-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, *6*(2), 215–219.

Thorndike, E. (1911). *Animal intelligence*. Darien, CT: Hafner.

Touzet, C. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems. Special Issue on Learning Robots: The New Wave*, *22*, 251–281.

Watkins, C. J. (1989). *Learning from delayed rewards*. PhD thesis, Cambridge University.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3/4), 279–292.

Wedel, J., & Polani, D. (1996). *Critic-based learning of actions with self-organising feature maps*. Technical Report 5/96, Institute for Informatics, Johannes-Gutenberg University.

Williams, R. J. (1988). *Towards a theory of reinforcement-learning connectionist systems*. Technical Report NU-CCS-88-3, College of Computer Science of Northeastern University, Boston, MA.

Ziemke, T. (1996). Towards adaptive behaviour system integration using connectionist infinite state automata. In P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, & S. W. Wilson (Eds.), *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulated and Adaptive Behavior* (pp. 145–154). Cambridge, MA: MIT Press.