Unsupervised learning of an embodied representation for action selection

Aapo Hyvärinen
Dept of Computer Science and HIIT
University of Helsinki, Finland

Abstract

We propose a principle on how a computational agent can learn the structure of a classic discrete state space. The idea is to do a kind of principal component analysis on a matrix describing transitions from one state to another. This transforms the space of discrete, completely separate, states into a dimensional representation in a Euclidean space. The representation supports action selection, ideally turning action selection into a trivial problem: the route to a goal state can be directly obtained from the representation. Thus, the computations typically performed by dynamic programming and reinforcement learning are largely replaced by learning the representation. This has the benefit that the representation is not dependent on which state happens to be the goal state; thus, change of goal does not necessitate re-learning, which is in stark contrast to classic reinforcement learning theory.

Introduction

How to find the best course of action? This is the fundamental computational question in embodied cognitive science. Typically, the problem is considered in the framework of reinforcement learning (Sutton and Barto, 1998; Dayan and Abbott, 2001). In the basic setting, the agent finds itself in a state, in which it has a number of actions available. The agent selects an action, and depending on which one it selected, it receives a reward, a punishment, or none. The action taken also determines the new state in which the agent finds itself in the next time step. In the simplest case, a reward is obtained only in a single state, the goal state. For example, a rat might be running around in a maze, and a small portion of cheese is found in a particular location.

Planning and dynamic programming are the two basic approaches to selecting the optimal action. These are closely related to the distinction between model-based and model-free reinforcement learning. If a model of the world is available, one can simply simulate the effects of different actions according to that model, and choose the one that leads to the reward or goal. The problem here, as already realized in classic artificial intelligence, is that the simulation may have to be many steps long before any reward is obtained (it takes some running to get to the cheese), and the number of action sequences to be simulated grows exponentially as a function of the number of steps. Thus, various kinds of partial search strategies have to be developed. The model-free alternative is to compute only the "values" of states[1] or state-action pairs, based on the principle of dynamic programming. The drawback is that such learning tends to be very slow, presumably because it attempts to operate with minimum knowledge of the structure of the world.

We propose here to approach action selection from the viewpoint of finding a suitable representation of the world, i.e. the set of states. This is is stark contrast to the classic theory of reinforcement learning, in which the question of representation is rather much reglected (though see (Dayan, 1993)). In the classic theory, the world is represented as a finite set of different states which are separate and unconnected from each other; or, a representation is given *a priori*, such that the values are computed as linear functions.

We propose a computational theory in which the agent learns a continuous-valued representation of the discrete-state world, and this representation enables a very simple model-based action selection mechanism. More specifically, the learned representation tells the agent which states are "close" to each other in the sense that the states can be reached from each other in a small number of steps. This solves most of the problem of "How to get to state j from state i". Planning is reduced to simply always choosing that action, among the alternatives immediately available, which leads to the new state with minimum distance from the goal. Thus, the exponential explosion in planning is completely avoided.

Learning in our system is unsupervised in the sense that it does not use any kind of reinforcement signal: only observations of state-action sequences. Thus, the learned representation is not bound to any single goal; in general it does not depend on which action in which state gives reward or punishment. This is again in stark contrast to classic (model-free) reinforcement learning, in which the goal is fixed once and for all (but see (Daw et al., 2005)). Thus, if the goal is changed (e.g. the cheese is given in another part of the maze), the learned value functions become useless, and learning has to be started all over again. In our learning system, as long as the state-action structure does not change (e.g. the maze does not change), no re-learning is needed, which gives it great flexibility. Our framework is also applicable to any problem domain, in contrast to some related work which consider navigation only.

[1] i.e. expected reinforcements when starting from that state and following an optimal policy

Learning the Representation

Basic principle

Consider a simple graphical representation of the world as a graph (see Fig. 1 a), where each state corresponds to a node, and each possible transition from one state to another, by a single action, is represented as an arc connecting those two nodes. The agent finds itself in one of the nodes, and selects an action by a method to be specified. The agent then moves along one of the arcs, and finds itself in that new node at the next time point. The agent knows a priori the immediate results of its actions, i.e. to which node it will move after a given action at a given state. What the agent observes is the number (or some other label) of the state in which it is: the states are numbered according to some arbitrary way. There is a single goal state: when the agent moves to that state, it receives a reward.

The agent knows which state is the goal. However, as is typical in reinforcement learning, the agent does not know how to get to that state. From an intuitive viewpoint, the computational problem is that the agent does not have any way of knowing how to get "closer" to the goal because it has no notion of distance.

The first part of our proposal is that it is possible to learn a notion of distance in this setting. The basic principle is that the agent observes that it is now in state j and that it was, in the previous time step, in state i. Thus, it can observe which states are "close" to each other in the sense that it is possible to move from state i to state j in a single step. Observing many state-to-state transitions, the agent could in principle just use the minimum number of steps required to move from state i to state j as a measure of distance between states i and j. The second part of our proposal is a computational scheme which learns something like this but with computational advantages.

Once the agent knows how to compute such a distance, action selection becomes rather trivial. First, the agent computes the distance to the goal state from all states to which the agent can go in one step. Second, the agent chooses the action which leads to the state minimizing that distance. (In practice, it may be necessary to introduce some randomness in this choice because the distance is learned only approximately.) Thus, the action of the agent typically reduces the distance at each step and eventually the agent will reach the goal.

Such computation of distance could, in principle, be achieved by classic shortest-path algorithms. However, it would demand a lot of computation, and possibly memory as well, to actually compute such distances for all the candidate states to which the agent might want to move. Thus, we propose to learn a dimensional representation for the points, i.e. to associate each state with a point in an n-dimensional real space. The distance between two points can then be simply computed as the Euclidean distance (sum of the differences of squares) of those representational points. Such a distance can be very quickly computed when needed, and the system only needs to store the n coordinates for each state; n is typically rather small, perhaps even one or two.

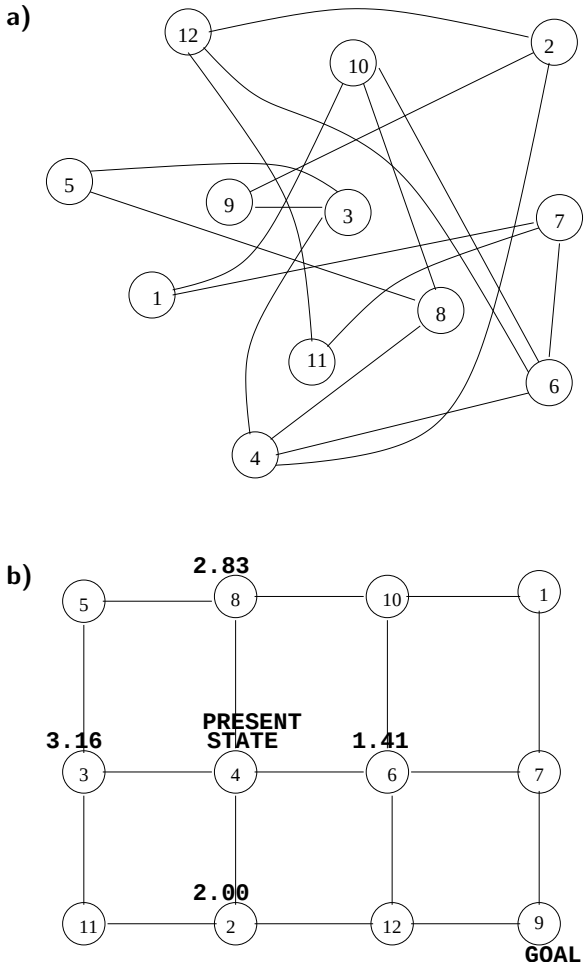

Figure 1: Representing states properly can make action selection trivial. a) A graph which represents the world so that each state is a node and each arc between two nodes means that there is an action which takes the agent from one state to the other. For simplicity of illustration, it is assumed in this figure that actions are reversible, so the arc can be travelled in both direction. (This representation does not show which action corresponds to which arc.) b) If the states can be arranged on a two-dimensional plane (or any other simple part of an n-dimensional real space), one can compute the distance of two nodes in that representation and use that as a guide to action selection. In particular, if the goal node is number 9 (bottom right-hand corner), and the present state is number 4, the Euclidean distances computed for the nodes to which one can move from the present node (shown next to those nodes) indicate that the agent should move down or right (the Euclidean distance favours "right" here but this is not the case for all distances).

Intuitively, it is clear how much the problem is simplified if instead of a general graph, the states can be arranged as grid of points on a two-dimensional plane (Fig. 1 b) so that the actions correspond to moving up, down, left or right. In fact, the graph in Fig. 1 a) can be arranged so. The agent can now use some notion of distance which is closely related to the conventional distance of the nodes on the graph on this graph.

Learning algorithm

Now we describe our learning algorithm. It can be considered a variant of classic principal component analysis (PCA) which has been used in many different situations for learning a low-dimensional representation. It is also very closely related to spectral methods as used in clustering and seriation (Shi and Malik, 2000; Ng et al., 2002; Atkins et al., 1998).

We assume that the agent chooses completely random actions during an initial learning period. During this period, the agent collects information on the transition probabilities, i.e. what is the probability of going from each state to any other state, when the actions are completely random. These probabilities essentially contain the information on which arcs are present in the graph[2] in Fig. 1.

Let us denote by p_{ij} the probability of going from state i to j, and by k the total number of states. The probabilities can be collected in a matrix with k rows and columns, denoted by $\mathbf{P}$:

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & \dots & p_{1k} \\ p_{21} & p_{22} & \dots & p_{2k} \\ \dots & & & \\ p_{k1} & p_{k2} & \dots & p_{kk} \end{pmatrix} \quad (1)$$

Now, we want to find n vectors which represent most of the "variation" in this matrix. A precise formulation of such variation can be obtained using the theory of singular value decomposition. Different variants are possible here; we proceed as follows.

First, we add an identity matrix $\mathbf{I}$ to $\mathbf{P}$, i.e. we add one to all the diagonal elements of the matrix; this does not change the eigenvectors but makes sure that their real parts are all non-negative, which avoids cumbersome absolute value computations. Denote the new matrix by $\tilde{\mathbf{P}} = \mathbf{P} + \mathbf{I}$. We compute the eigenvectors corresponding to the $n+1$ eigenvalues with the largest real parts, ignoring their imaginary parts. Note that all eigenvalues in this paper are right eigenvectors instead of left. The eigenvalues are related to the amount of variation each eigenvector explains. Now, with any transition probability matrix, the largest eigenvalue is equal to 1, and corresponds to an eigenvector which has all constant entries.[3] We discard this degenerate eigenvector. We then take the n eigenvectors which correspond to the n next eigenvalues with largest real parts. These vectors form our representation.

A complication with this kind of eigenvectors computations is that since the transition matrix need not be symmetric, the eigenvalues and vectors can be complex-valued. Some mathematical analysis which is outside of the scope of this paper shows that the interesting eigenvectors and eigenvalues are, under some theoretical assumptions, all real-valued. This justifies considering only the real parts of the eigenvalues. In practice, the eigenvalues might be complex-valued due to violations of those theoretical assumptions, but we avoid this problem by using an eigenvector calculation method, the power method, which does not give complex values if it is initialized with real values.

Each of those obtained n eigenvectors of $\tilde{\mathbf{P}}$, denote them by $\mathbf{v}_q, q = 1, \dots, n$, associates a real number with each state: the i-th entry in an eigenvector, $\mathbf{v}_q(i)$, gives one of the coordinates of that state in our representation. Altogether, we obtain n such coordinates, one for each eigenvector, so we associate to each state n real coordinates. This is our representation.

The actual computation of the eigenvectors can be done by classic methods. The situation is quite simple if the agent actually computes and stores all the transition probabilities. The computation of the probabilities is very simple: the agent just has to count how many times it went from state i to state j, and divide this frequency by the total number of times it found itself in state i. It could be argued that the storage of the probabilities is a major problem because the number of states k can be very large, and the number of probabilities to be stored (i.e. the number of entries in the matrix $\mathbf{P}$) is equal to the square of k. However, this need not be a problem because the matrix is typically very sparse: most states are accessible from only a few other states, so most of the entries p_{ij} are zero, and one needs to store only those which are non-zero. This can radically reduce the amount of memory needed by the agent. If the matrix $\mathbf{P}$ is stored in the memory of the agent, a number of classic methods for computing the eigenvectors can be used. We used the power method (Golub and van Loan, 1996) in our simulations, since it has the additional benefit of constraining the search to real values if it is initialized with real values. The off-line learning algorithm which results from this choice is described in Table 1.

If one wants to investigate the neurobiological plausibility of such learning, a simple on-line algorithm may be more interesting. Using the classic theory of on-line learning, based on stochastic approximation, we have developed a simple online algorithm for learning this representation. This algorithm does not store the transition probabilities in memory but uses each observed transition immediately for learning. The algorithm is adapted from previously proposed online methods for PCA (Oja,

[2] Actually, the probabilities contain more information because probabilities is not simply binary, 0/1. Moreover, the probability from going from state i to state j might not be equal to the probability of going from j to i, so the arcs would actually be directed in the general case.

[3] This holds for the largest right eigenvector; the corresponding *left* eigenvector gives the stationary probabilities of the Markov chain, i.e. the probabilities of being in each state when the chain is run an infinite number of times. The eigenvectors we use are not related to this left eigenvector.

1982; Hyvärinen et al., 2001). Our algorithm does, however, need to store the total probabilities of being in each of the states in the memory. This algorithm is described in Table 1 as well.

Simulations

Simple grid-world

As an archetype of a reinforcement learning problem, consider a grid of states on a two-dimensional plane. At each time step the agent finds itself in one of these states, and can decide to move up, down, left or right. The agent then moves in the chosen direction, and finds itself in that new state at the next time point (unless it tried to move into the walls marking the boundaries of the grid, in which case the agent does not move at all). The size of the grid was 25×25.

There was first a learning period in which the agent randomly took actions in order to learn the structure of the world. There were a total of 2,500 trials; the goal state was randomly chosen, independently in each trial. The agent took 250 steps in each trial. Since the actions were random, the agent rarely found the goal, but the performance in this initial period provides a baseline against which we can measure the performance of the learning.

At the end of the learning phase, a two-dimensional representation was learned using the off-line version of our algorithm. The learned representation is shown in Fig. 2 a), b). The vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are shown in grey-scale: black is negative, and the lighter the grey, the larger the value of $\mathbf{v}_q(i)$, white being positive. Fig. 2 c) shows, for one randomly selected goal state, the value of the distance function as grey-scale One can see that the learned distance is computed in a meaningful way. The individual coordinates correspond quite well to the two coordinate axes of the world. They are slightly rotated version of the horizontal and vertical axis, but this has no significance, because such a rotation does not change the Euclidean distances.

In the test phase, we ran another 2500 trials, again with a maximum of 250 steps for each trial. Now, the action selection was done according to the learned representation. We computed the distances to the goal from each of the (typically four) states which are accessible from the present state. We then chose the next action so that the probability of going to the state was larger for those states with smaller distance from the goal. Specifically, we normalized the distances of the immediately accessible states to be between 0 and 1 by subtracting the smallest distance and dividing by the largest distance. Then, we computed the probabilities of going to those states according to the Boltzmann distribution $p(j|i) \propto \exp(-a_{ij}/T)$ where a_{ij} are the normalized distances, and T is a temperature parameter, chosen to equal 0.2 in our simulations.

To show that our method was effective in action selection, we computed the proportion of trials in which the agent was able to find the goal, see Fig. 5. As a baseline, we show how often the agent found the goal by moving randomly as in the learning phase. The success rate for

Off-line algorithm for learning representation

1. Initialize representational vectors $\mathbf{v}_q(i)$, $q = 1, \ldots, n$ and $i = 1, \ldots, k$ to random values, where n is the dimension of the representational space, and k is the number of states.
2. Initialize state transition counters of f_{ij}, $i, j = 1, \ldots, k$ to zero.
3. Repeat at each time step of learning phase
   (a) Take random action. Denote by i the state at previous time step, and by j the current state.
   (b) Update frequency counters: $f_{ij} \leftarrow f_{ij} + 1$
4. Compute transition probabilities: $p_{ij} \leftarrow f_{ij} / \sum_{j'} f_{ij'}$ for all $i, j = 1, \ldots, k$.
5. Repeat until convergence
   (a) Repeat for each $q = 1, \ldots, n$
      i. Power iteration in matrix multiplication formulation: $\mathbf{v}_q \leftarrow \mathbf{P}\,\mathbf{v}_q$
      ii. Subtract mean: $\mathbf{v}_q(h) \leftarrow \mathbf{v}_q(h) - \frac{1}{k} \sum_{h'} \mathbf{v}_q(h')$ for all $h = 1, \ldots, k$.
   (b) Orthogonalize vectors $\mathbf{v}_q$. (A number of methods is available for this operation (Golub and van Loan, 1996; Hyvärinen et al., 2001).)

On-line algorithm for learning representation

1. Initialize representational vectors $\mathbf{v}_q(i)$ as above.
2. Initialize state frequency counters of f_i, $i = 1, \ldots, k$ to zero.
3. Repeat at each time step
   (a) Take random action. Denote by i the state at previous time step, and by j the current state.
   (b) Update frequency counters: $f_i \leftarrow f_i + 1$
   (c) Compute state probabilities: $p_i \leftarrow f_i / \sum_{i'} f_{i'}$
   (d) Repeat for each $q = 1, \ldots, n$:
   - Main update step: $\mathbf{v}_q(i) \leftarrow \mathbf{v}_q(i) + \mu\, \mathbf{v}_q(j)/p_i$, where μ is a small step size constant.
   (e) Subtract means: $\mathbf{v}_q(h) \leftarrow \mathbf{v}_q(h) - \frac{1}{k} \sum_{h'} \mathbf{v}_q(h')$ for all q and h.
   (f) Orthogonalize vectors $\mathbf{v}_q$. (See (Hyvärinen et al., 2001) for on-line methods for orthogonalization.)

Table 1: The off-line and on-line versions of our learning algorithm. The subtraction of the mean of each vector is equivalent to discarding the degenerate eigenvector with eigenvalue 1.

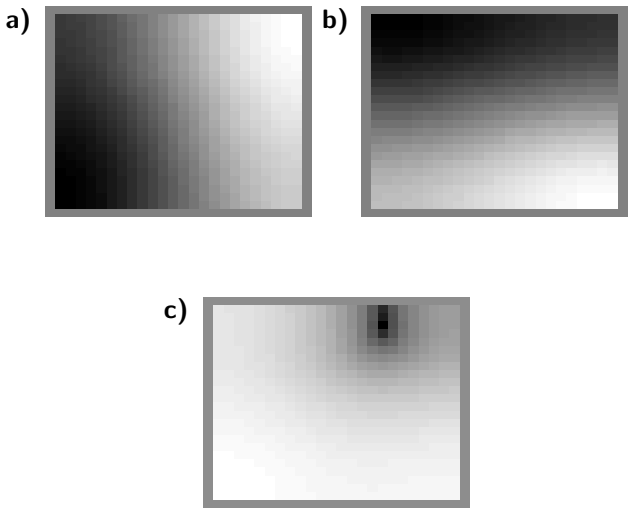

Figure 2: Simulation of a world where the states form a simple two-dimensional grid surrounded by a wall. Our algorithm is able to learn that underlying structure. a) First coordinate in the representation, shown as grey-scale. b) Second coordinate in the representation. c) The distance function from a random goal state, shown as grey-scale.

the case 100% ("no obstacles"). In the baseline trial, the success rates were around 15%. Thus, learning the representation was very efficient in action selection.

Grid world with obstacles

In the next set of simulations, obstacles are added: These are pieces of wall which again prevent movement into or through them. Figure 3 shows the results with such obstacles, which are shown in medium grey in Fig. 3 a)-d). Now, we learned three coordinate axes instead of two to account for the increase in the complexity of the world, shown in Fig. 3 a), b), c). Again, the learned distance, shown for a randomly chosen goal state, is shown in Fig. 3 d). The learned coordinates show that the fundamental thing the system learned is that the dead ends in the "maze" are far way from the rest. This is logical because it takes many steps to get anywhere from those dead ends.

The improvement in action selection is again shown in Fig. 5: success rate was 93% after learning whereas it was around 15% without learning (random case).

Tower of Hanoi

Finally, we used our method in a rather different kind of world: the Tower of Hanoi with 3 disks and 3 pegs. Due to the simplicity of the problem, the number of steps in each trial was reduced to 12. The learned representation is shown in Fig. 4. Action selection was tested by assigning completely random initial and goal states: success rate was 99.8% after learning this representation, and 25% without learning (Figure 5).

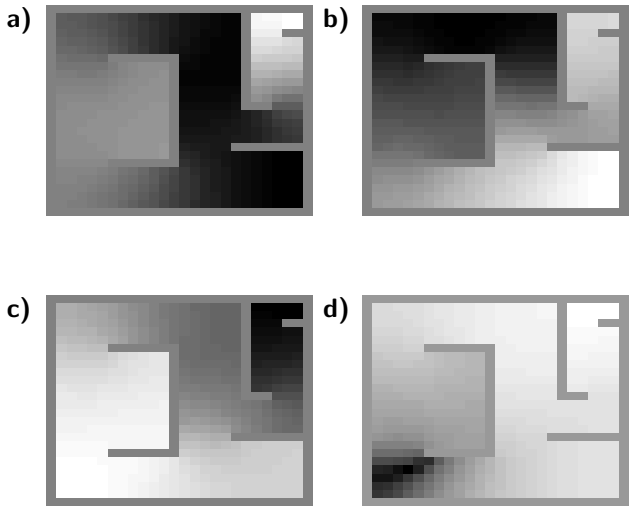

Figure 3: Simulation of a world in which there are obstacle (inaccessible) states among the states forming the grid. Our algorithm is able to learn that underlying structure. a-c) The three coordinates in the representation, shown as grey-scale. Obstacles are shown in grey as well. d) The distance function from a random goal state, shown as grey-scale.

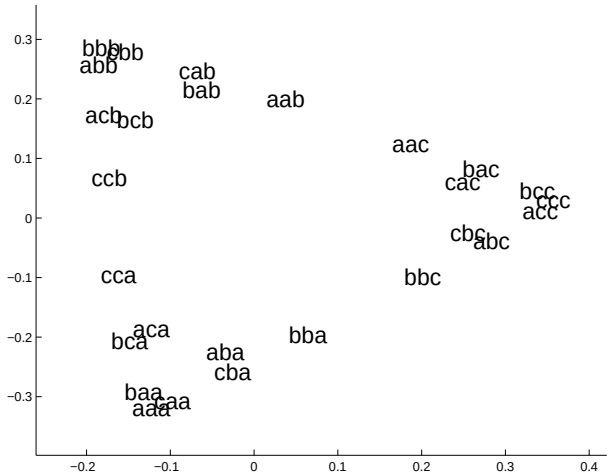

Figure 4: The two-dimensional representation learned in the Tower of Hanoi world. The pegs are denoted by the letters a, b, c and the states are represented as triplets of such letters, so that the first letter tells the location of the first disk and so on. The states form a triangle whose corners (aaa,bbb, and ccc) correspond to states where all the disks are in the same peg.

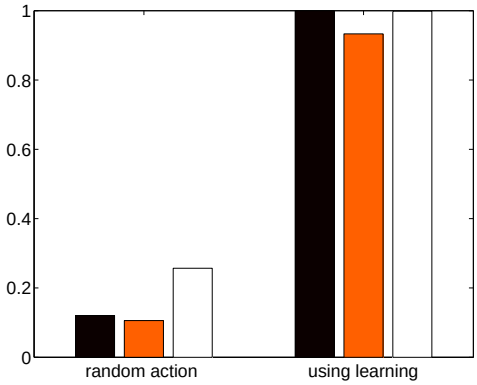

Figure 5: Success rates of finding the goal using the learned representation and the baseline of random actions, in the case of the basic grid world with no obstacles (black bar on the left), the grid with obstacles (red or grey bar on the middle), and the Tower of Hanoi domain (white bar on the right)

Discussion

Our method was originally inspired by the successor representation (Dayan, 1993). However, here we emphasize the low dimensionality of the representation, whereas the successor representation does not reduce the dimension at all, which may be computationally very demanding. Another closely related method was proposed by (Engel and Mannor, 2001). Instead of PCA, they used a more complicated learning method. However, the their goal and philosophy were very similar to ours.

Our learning principle could be interpreted as trying to find a representation which changes as slowly as possible. In a random representation (where each state is a random point is the n-dimensional space), the point that represents the agent's present state jumps randomly from one place to another when the agent takes actions. In contrast, in the representation learned by our method, the point moves slowly, since two points which are accessible from each other tend to be close to each other in the representation. Thus, our method is closely related to models which try to maximize temporal coherence or stability of a representation (Földiák, 1991; Hurri and Hyvärinen, 2003).

Here, we considered the theoretical setting widely used in machine learning where the world is given as a set of discrete states and transitions between them. In other words, we assumed that the perceptual system classifies the state into a discrete set efficiently and unambiguosly. Future work will address how this method could be adapted to the case where the information about the state comes in the form of high-dimensional sensory input. Related work can be found in robotics (Thrun, 2002) and in computational neuroscience (Trullier and Meyer, 2000), but these are usually very different from our approach in that they are constrained to navigation, or 2D environments, whereas we consider completely general problem domains.

To conclude, we proposed a computational model for learning a dimensional representation of a discrete-state world, with the goal of facilitating action selection. The basic principle is to do a kind of principal component analysis on the matrix of transition probabilities between the states. Then, action selection and planning can become quite simple. Similar to model-based reinforcement learning, or planning, adaptation to changing goals is straightforward and re-learning is not needed. However, the exponential explosion of computation, which is inherent in planning, is avoided. Thus, our method seems to combine some of the advantages of model-based and model-free reinforcement learning.

References

Atkins, J. E., Boman, E. G., and Hendrickson, B. (1998). A spectral algorithm for seriation and the consecutive ones problem. *SIAM J. on Computing*, 28(1):297–310.

Daw, N. D., Niv, Y., and Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8:1704–1711.

Dayan, P. (1993). Improving generalisation for temporal difference learning: The successor representation. *Neural Computation*, 5:613–624.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience*. MIT Press.

Engel, Y. and Mannor, S. (2001). Learning embedded maps of markov processes. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 138–145.

Földiák, P. (1991). Learning invariance from transformation sequences. *Neural Computation*, 3:194–200.

Golub, G. and van Loan, C. (1996). *Matrix Computations*. The Johns Hopkins University Press, 3rd edition.

Hurri, J. and Hyvärinen, A. (2003). Simple-cell-like receptive fields maximize temporal coherence in natural video. *Neural Computation*, 15(3):663–691.

Hyvärinen, A., Karhunen, J., and Oja, E. (2001). *Independent Component Analysis*. Wiley Interscience.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *J. of Mathematical Biology*, 15:267–273.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Thrun, S. (2002). Robotic mapping: A survey. In Lakemeyer, G. and Nebel, B., editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann.

Trullier, O. and Meyer, J.-A. (2000). Animat navigation using a cognitive graph. *Biological Cybernetics*, 83:271–285.