Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks

Ramnandan Krishnamurthy[1]* NANDPARIKRISH@GMAIL.COM
Aravind Lakshminarayanan[1]* ARAVINDSRINIVAS@GMAIL.COM
Peeyush Kumar[2]* AGOOVI@GMAIL.COM
Balaraman Ravindran[1] RAVI@CSE.IITM.AC.IN
[1] RISE IIL, Indian Institute of Technology, Madras
[2] University of Washington

* Equal Contribution

Abstract

This paper introduces an automated skill acquisition framework in reinforcement learning which involves identifying a hierarchical description of the given task in terms of abstract states and extended actions between abstract states. Identifying such structures present in the task provides ways to simplify and speed up reinforcement learning learning algorithms. These structures also help to generalize such algorithms over multiple tasks without relearning policies from scratch. We use ideas from dynamical systems to find metastable regions in the state space and associate them with abstract states. The spectral clustering algorithm PCCA+ is used to identify suitable abstractions aligned to the underlying structure. Skills are defined in terms of the transitions between such abstract states. The connectivity information from PCCA+ is used to generate these skills or *options*. The skills are independent of the learning task and can be efficiently reused across a variety of tasks defined over a common state space. Another major advantage of the approach is that it does not need a prior model of the MDP and can work well even when the MDPs are constructed from sampled trajectories. Finally, we present our attempts to extend the automated skills acquisition framework to complex tasks such as learning to play video games where we use deep learning techniques for representation learning to aid our spatio-temporal abstraction framework.

Proceedings of the 33^{rd} *International Conference on Machine Learning*, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

1. Motivation and Introduction

The core idea of hierarchical reinforcement learning is to break down the reinforcement learning problem into subtasks through a hierarchy of abstractions. Typically, in the full reinforcement learning problem, the agent is assumed to be in one state of the Markov Decision Process at every time step. The agent then performs one of several possible primitive actions. Based on the agent's state at time t, and the action it takes from that state, the agent's state at time $t + 1$ is determined. For large problems, however, this can lead to too much granularity: when the agent has to decide on each and every primitive action at every granular state, it can often lose sight of the *bigger picture*. However, if a series of actions can be abstracted out as an abstract action, the agent can just remember the series of actions that was useful in getting it to a temporally distant useful state from the initial state. This is typically referred to as an *option* or a skill in the reinforcement learning literature. A good analogy is a human planning his movement for a traversal from current location A to a destination B. We identify intermediate destinations C_i to lead us from A to B when planning from A, instead of worrying about the exact mechanisms of immediate movement at A which are *abstracted over*. Options are a convenient way of formalising this abstraction. In keeping with the general philosophy of reinforcement learning, we want to build agents that can automatically discover options with no prior knowledge, purely by exploring the environment. Thus, our approach falls into the broad category of *automated discovery of skills*.

In order to exploit task structure, hierarchical decomposition introduces models defined by stand-alone policies (also known as temporally-extended actions, options, or skills) that can take multiple time steps to execute. Skills can exploit representing structure by representing subroutines that are executed multiple times during execution of a task. Such skills which are learnt in one task can be reused in a different task as long as it requires execution of the same subroutine. Options also make exploration more efficient by providing the decision maker with a high-level behaviour to look ahead to the completion of the corresponding subroutine.

Automated discovery of skills or options has been an active area of research and several approaches have been proposed for the same. The current methods could be broadly classified into sample trajectory based and partition based methods. Some of them are:

- Identifying bottlenecks in the state space, where the state space is partitioned into sets and the transitions between two sets of states that are rare can be seen as introducing bottleneck sets at the respective points of such rare transitions. Policies to reach such states are cached as options (McGovern & Barto, 2001).
- Using the structure present in a factored state representation to identify sequences of actions that cause what are otherwise infrequent changes in the state variables: these sequences are cached away as options (Hengst, 2004).
- Obtaining a graphical representation of an agent's interaction with its environment and using betweenness centrality measures to identify subtasks (Simsek & Barto, 2008).
- Using clustering methods (spectral or otherwise) to separate out different strongly connected components of the Markov Decision Process (MDP) and identifying *access-states* that connect different clusters (Menache et al., 2002).

While these methods have had varying amounts of success, they have certain deficiencies. Bottleneck based approaches don't have a natural way of identifying the part of the state space where options are applicable without external knowledge about the problem domain. Spectral methods need some form of regularization in order to prevent unequal splits that might lead to arbitrary splitting of the state space.

We present a framework that detects well-connected or meta stable regions of the state space from a MDP model estimated from trajectories. We use PCCA+, a spectral clustering algorithm from conformal dynamics (Weber et al., 2004) that not only partitions the MDP but also returns the connectivity information between the regions. We then propose a very effective way of *composing* options using the same framework to take us from one metastable region to another, giving us the policy for free. Once we have these options, we can use standard reinforcement learning algorithms to learn a policy over subtasks to solve the given task. Specifically, we show results using SMDP Q-learning on the 2-room domain. For our attempt at extending it to higher dimensional state space tasks such as Atari 2600 video games, we append the learnt options to the set of primitive actions using Intra-Option Value learning to learn a policy solving the given task. One major advantage of the approach is that we get the policy for the options for free while doing the partitioning by exploiting the membership functions returned by PCCA+. Our approach is able to learn reasonably good skills even with limited sampling which makes it useful in situations where exploration is limited by the environment costs. It also provides a way to refine the abstractions in an online fashion without explicitly reconstructing the entire MDP. More importantly, we extend it to the case where the state space is so large that exact modeling is not possible. In this case, we take inspiration from the recent work on forward prediction to learn the model (Oh et al., 2015) to use Deep Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM) to learn spatio-temporal representations of the state space. Using the learnt representation, we perform state-aggregation using clustering techniques and estimate our transitional model for the abstract space on these aggregated states.

We list the advantages of this approach below:

- Skills are acquired online, from sampled trajectories instead of requiring a prior model of the MDP.
- Instead of looking for bottleneck states, we look for well connected regions and hence, the discovered options are better aligned to the structure of the state space.
- The approach returns connectivity information between the metastable regions which can be used to construct an abstract graph of the state space, combining spatial and temporal information meaningfully.
- The clustering algorithm provides a fuzzy membership for every state in belonging to a particular metastable region, which provides a powerful way to compose options naturally.

We organize the rest of the paper as follows: We first explain the Option Generation Framework that we propose, by delving on the important aspects of the spectral clustering algorithm PCCA+ and how PCCA+ can be used to generate options. We show results on the 2-room domain for this framework. The next part of the paper focuses on our attempt to extend this framework for more complex tasks such as playing video games like Seaquest on the Atari 2600 domain with options. We explain the motivation and usage of deep networks, the clustering algorithms used for state-aggregation, followed by the model used and initial results. We then conclude with a discussion on the challenges in this approach and also comparison with other recent attempts at Hierarchical reinforcement learning for higher dimensional tasks.

2. Option Generation Framework

We divide this section into two parts: We first explain the spectral clustering algorithm PCCA+ and motivate its usage for spatial abstraction. The second part discusses the option generation using PCCA+.

2.1. Spatial Abstraction using PCCA+

Given an algebraic representation of the graph representing a MDP we want to find suitable abstractions aligned to the underlying structure. We use a spectral clustering algorithm to do this. Central to the idea of spectral clustering is the graph Laplacian which is obtained from the similarity graph. There are many tight connections between the topological properties of graphs and the graph Laplacian matrices, which spectral clustering methods exploit to partition the data into clusters. However, although the spectra of the Laplacian preserves the structural properties of the graph, clustering data in the eigenspace of the Laplacian does not guarantee this. For example, k-means clustering (Ng et al., 2001) in the eigenspace of the Laplacian will only work if the clusters lie in disjoint convex sets of the underlying eigenspace. Partitioning the data into clusters by projecting onto the largest k-eigenvectors (Meila & Shi, 2001) does not preserve the topological properties of the data in the eigenspace of the Laplacian. For the task of spatial abstraction, the proposed framework requires a clustering approach that exploits the structural properties in the configurational space of objects as well as the spectral subspace, quite unlike earlier methods. Therefore, we take inspiration from the conformal dynamics literature, where (Weber et al., 2004) do a similar analysis to detect conformal states of a dynamical system. They propose a spectral clustering algorithm PCCA+, which is based on the the principles of Perron Cluster Analysis of the transition structure of the system. We extend their analysis to detect spatial abstractions in autonomous controlled dynamical systems.

In this approach, the spectra of the Laplacian $\mathcal{L}$ (derived from the adjacency matrix $\mathcal{S}$) is constructed and the best transformation of the spectra is found such that the transformed basis aligns itself with the clusters of data points in the eigenspace. A projection method described in (Weber et al., 2004) is used to find the membership of each of the states to a set of special points lying on the transformed basis, which are identified as vertices of a simplex in the $\mathbb{R}^k$ subspace (the Spectral Gap method is used to estimate the number of clusters k). For the first order perturbation, the simplex is just a linear transformation around the origin and to find the simplex vertices, one needs to find the k points which form a convex hull such that the deviation of all the points from this hull is minimized. This is achieved by finding the data point which is farthest located from the origin and iteratively identify data points which are located farthest from the hyperplane fit to the current set of vertices.

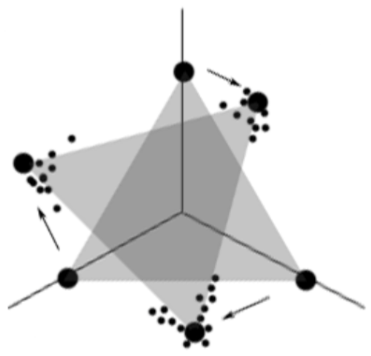

Figure 1. Simplex First order and Higher order Perturbation

Algorithm 1 PCCA+

```
1: Construct Laplacian $\mathcal{L}$
2: Compute $n$ (number of vertices) eigenvalues of $\mathcal{L}$ in descending order
3: Choose first $k$ eigenvalues for which $\frac{e_k - e_{k+1}}{1 - e_{k+1}} > t_c$ (Spectral Gap Threshold).
4: Compute the eigenvectors for corresponding eigenvalues $(e_1, e_2, \cdots, e_k)$ and stack them as column vectors in eigenvector matrix $\mathcal{Y}$.
5: Let's denote the rows of $\mathcal{Y}$ as $\mathcal{Y}(1), \mathcal{Y}(2), \cdots, \mathcal{Y}(N) \in \mathbb{R}^k$
6: Define $\pi(1)$ as that index, for which $||Y(\pi(1))||_2$ is maximal. Define $\gamma_1 = span\{Y(\pi(1))\}$
7: For $i = 2, \cdots, k$: Define $\pi_i$ as that index, for which the distance to the hyperplane $\gamma_{i-1}$, i.e., $||Y(\pi_i) - \gamma_{i-1}||_2$ is maximal. Define $\gamma_i = span\{Y(\pi_1), \cdots, Y(\pi_i)\}$. $||Y(\pi_i) - \gamma_{i-1}||_2 = ||Y(\pi_i) - \gamma_{i-1}^T((\gamma_{i-1}\gamma_{i-1}^T)^{-1}\gamma_{i-1})Y(\pi_i)^T)||$
```

The PCCA+ algorithm returns a membership function, χ, defining the degree of membership of each state s to an abstract state S_j. The connectivity information between two abstract states (S_i, S_j) is given by $(i, j)^{th}$ entry of $\chi^T L \chi$ while the diagonal entries provide relative connectivity information within a cluster. The connectivity information is utilized to learn decision policies across abstract states which is described in the next section. There is an intrinsic mechanism to return information about the goodness of clustering of states from the presence of sharp peaks (indicates good clustering) in the eigenvalue distribution.

2.2. Option generation from PCCA+

2.2.1. Option

Option is one of the formalisations used to represent an extended series of actions (Sutton & Barto, 1998). Formally, an option is a tuple $O = (I, \mu, \beta)$ where:

- I is the initiation set: a set of states from which the action can be activated.
- μ is a policy function where $\mu(s, a)$ represents the preference value given to action$a$ when in state s and following option O.
- β is the termination function: When an agent enters a state s while following option O, it terminates the option with probability $\beta(s)$.

When the agent is in state s where it can start an option, it can choose from all the options O for which $s \in I(O)$) and all primitive actions that can be taken in s. The choice is dictated by the policy guiding the agent. While executing an option, the agent follows μ corresponding to that option, moving from state to state with the option to terminate being decided by β. On termination, the agent can again pick from the set of options in the terminated state or one of the primitive actions, which is again decided by the policy.

2.2.2. Composing Options from PCCA+HRL

PCCA+ splits up our state into abstract states - thus, moving between different abstract states is an important operation that involves a sequence of primitive actions. We generate options that enable our agent to move from a state belonging to a particular abstract state, to any other abstract state, that is, the abstract tasks (options) are transitions between abstract states. The membership function χ returned by PCCA+ provides a very elegant method to compose options to realize these abstract tasks. Each state has a certain membership value to each abstract state. We can move to another abstract state by *simply following positive gradient of the membership value to those abstract states*. This will gradually move us through states that have a larger and larger membership to the abstract state, until we finally reach a state that belongs to that abstract state. In case of multiple exits or bottlenecks, PCCA+HRL is able to compose multiple options, each taking the agent to the respective exit.

Specifically, consider that we have n states $s_1, ..., s_n$ and k abstract states (equal to the number of PCCA+ clusters), $S_1, ..., S_k$. Typically, $n \gg k$. Let χ_{ij} denote the membership of state s_i to the abstract state S_j. s_i is said to *belong* to abstract state S_a, where $a = \text{argmax}_j \chi_{ij}$. Thus, any option between abstract states S_i and S_j will start at a state in S_i and end at a state in S_j. We generate an option for every pair of connected abstract states (we determine if they are connected using the connectivity information as described above). For an option from S_i to S_j:

- The initiation set I represents states that belong to S_i.
- The **option policy** $\mu(s, a)$ that takes the agent from abstract state S_i to S_j is a stochastic gradient function given by:

  $$\mu(s, a) = \text{max}\left(\alpha(s)(m_{S_j S_i} - m_{S_j}(s)), 0\right),$$

  with

  $$m_{S_j S_i} = (\sum_{s'} P(s, a, s')m_{S_j}(s')) - m_{S_j}(s) \; \forall s \in S_i$$

  $\alpha(s)$ is a normalization constant to ensure $\mu \in [0, 1]$.
- Finally, **termination condition** β is a probability function which assigns the probability of termination of the current option at state s. It can also be viewed as the probability of a state s being a decision epoch given the current option being executed. For an option taking an agent from abstract state S_i to S_j, we define β as follows:

  $$\beta(s) = min(\frac{log(m_{S_i}(s))}{log(m_{S_j}(s))}, 1) \; \forall s \in S_i$$

  Such a termination condition provides a smooth peaking function across transitions.

2.2.3. Online Agent

The previous sections describe what to do given the model of the MDP. However, we want our agent to work in an online fashion. Here, we estimate our transition matrix from sampled trajectories and feed these into PCCA+ to obtain the abstract states. We generate options and augment our agent with them. The agent then goes from state to state choosing the available options and primitive actions, updating its value function estimates using any appropriate reinforcement learning technique like SMDP Q-learning. It continuously updates the counts of the transition matrix, and does PCCA+ again to obtain the new χ matrix and generate new options. This is repetitively performed till convergence.

However, in practice, it is too expensive to perform PCCA+ after every episode. Thus, we run the agent using a simple reinforcement learning algorithm like Q-learning for a fixed number of episodes, until it learns a decent policy. We then freeze this policy and run the agent for several more episodes, keeping track of the transitions between states to obtain an estimated model of our MDP and a transition matrix. We then run PCCA+ on this *bootstrapped* transition matrix to generate options and learn using SMDP Q-learning (with ϵ greedy exploration) for a large number of episodes. The transition matrix is updated based on the experiences in the SMDP Q-learning phase to run PCCA+ again. This cycle can be repeated and our agent keeps improving over time.

We evaluate our PCCA+HRL approach on the 2-room domain and the Taxi domain below, with a brief description of the domains.

2.2.4. 2-Room Domain

This is a very simple domain where one can still acquire skills. The domain consists of 2 rooms of unequal sizes where the agent has to start from the first room and reach a particular goal state in the second room (Figure 2). Typical skill acquired would be to reach the doorway from any state in the first room and to navigate from this doorway to the intended goal state (2 abstract tasks and 3 abstract states). This is indeed what we observe while using the PCCA+HRL framework (Figure 3), after 3 episodes of the trajectory of length 3000 each. We find 3 abstract states where one abstract state corresponds to the lone goal state itself. Figure 4 compares the average return for different methods while solving the same task. We plot the average return with respect to the number of epochs of decision used, comparing it with different methods. Our approach identifies 2 options as compared to 12 by LCut and 10 by Random Options with primitive actions method with consistently higher average return than the other two. From these results, it is evident that our framework is able to learn the correct abstractions for solving the task. That it constructs the options to realize the transitions between these abstract states is clear, since it arises naturally from the PCCA+ connectivity, as explained.

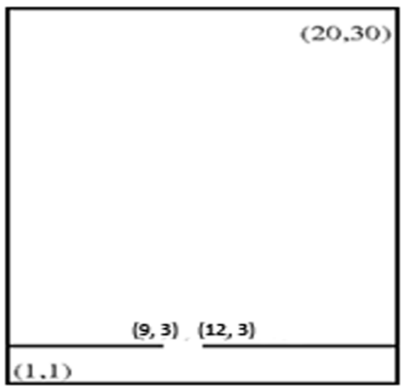

Figure 2. 2 Room Domain

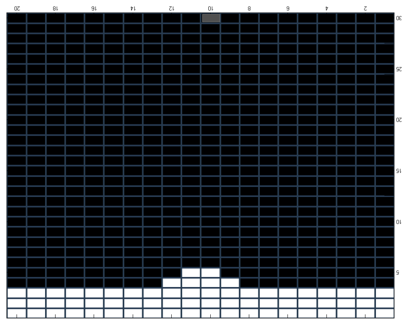

Figure 3. Abstractions in 2 Room Domain

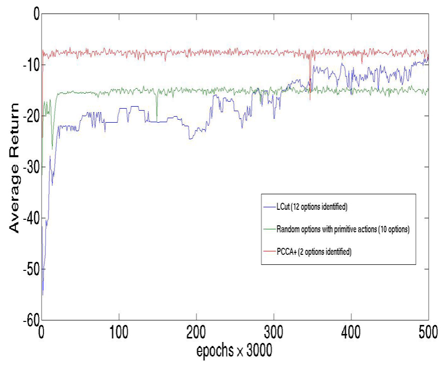

Figure 4. Average Return for 2 Room Domain

2.2.5. Model Estimation

We now propose an online method for efficiently finding Spatio-Temporal abstractions while the agent is following another strategy. The method is inspired from the UCT framework which is a Monte-Carlo search algorithm based on rollouts. A rollout-based algorithm builds its look-ahead tree by repeatedly sampling episodes from the initial state. The tree is built by adding the information gathered during an episode to it in an incremental manner. We use the UCT algorithm because it is more effective than the vanilla Monte-Carlo planning where the actions are sampled uniformly, while UCT does a selective sampling of actions.

In the UCT approach, in state s, at depth d, the action that maximizes $Q_t(s,a,d) + c_{N_{s,d}(t),N_{s,a,d}(t)}$ is selected, where $Q_t(s,a,d)$ is the estimated value of action a in state s at depth d and time t, $N_{s,d}(t)$ is the number of times state s has been visited up to time t at depth d and $N_{s,a,d}(t)$ is the number of times action a was selected when state s has been visited, up to time t at depth d, $c_{t,s}$ has the form $c_{t,s} = 2C_p\sqrt{\frac{ln(t)}{s}}$, where C_p is an empirical constant. A variant of this search method is used in PCCA+HRL. Since we are composing option policies rather than learning them, we replace $Q(s,a)$ with the stochastic gradient function $\mu(s,a)$, where the particular μ corresponding to the greedy option chosen from the option value function. Hence the search criteria becomes max(argmax$_a \alpha(s)(\sum_{s'} P(s,a,s')m_{s_j}(s') - m_{s_j}(s), 0)(d) + c_{N_{s,d)}(t), N_{s,a,d}(t)}$. To include the underlying reward structure, we modify the local adjacency matrix D as $D_{posterior} = D_{prior}(s,s') + \sum_a \phi^a_{ss'} e^{-v|R^{ss'}_a|}$ where $R^{ss'}_a$ is the reward received while v is the regularization constant and $\phi^a_{ss'}$ is the number of times the transition occurred from s to s' with action a which ensures the adjacency function has very low value at spike points, returns a value for 1 for zero rewards and allows for easy tuning of relative weights by change the parameter v.

Algorithm 2 PCCA+HRL

```
$Q \Longrightarrow ActionValueFunction$
 1: Observe initial state $s_o$
 2: Initialize $Q$ arbitrarily
 3: Initialize $T$ transition matrix
 4: U={}
 5: for e=1 to maximum number of episodes do
 6:   Initialize Membership function $\chi$, Simplex Vertex $Y$=$PCCA+(\tau)$
 7:   Find all pairs of connected abstract states $C_k = (S_j, S_k)$ from the non-zero entries in $\chi^T T \chi$
 8:   $O'_k = O_k \; \forall k$
 9:   $\forall C_k$ construct $O_k = I_k, \mu_k, \beta_k$
10:   match $O_k \; \forall k$ (New set of Options) with previous sets of options $O'_k \; \forall k$
11:   find $k$ for which $s_o \in C_k(1)$
12:   $i = k; s_i = s_0$
13:   while not the end of episode do
14:     $O_i \Leftarrow \left\{ \begin{array}{l} \text{argmax}_O Q(s_i, O) \text{ w.p } 1-\epsilon \\ \text{Random option beginning from } s_i \text{w.p } \epsilon \end{array} \right\}$
15:     Update $Q(O_i, s_i)$ using any action value function learning method
16:     Sample action according to $\alpha(\mu_i(s) + c_{N_{s,d}(t), N_{s,a,d}(t)})$ and follow until termination. return termination state $s_t$
17:     $\phi^a_{ss'} = \phi^a_{ss'} + \delta^a_{ss'}$ where $\delta$-function =1 if action $a$ takes from state $s$ to $s'$
18:     $R^a_{ss'}$ = reward returned while taking action $a$ in state $s$ taking the system to state $s'$
19:     $U(s,a,s') = U(s,a,s') + \phi^a_{ss'} e^{-v|R^a_{ss'}|}$
20:     $D(s,s') = \sum_a U(s,a,s')$
21:     $P(s,a,s') = \frac{U(s,a,s')}{\sum_{s'} U(s,a,s')}$
22:     $T(s,s') = \frac{D(s,s')}{\sum_s D(s,s')}$
23:     $s_i = s_t$
24:   end while
25: end for
```

We evaluate the proposed approach for the Taxi Domain, shown in the next section with a detailed description of the domain task and our results in comparison to other algorithms.

2.2.6. Taxi Domain

(Dietterich, 2000) created the taxi task (Figure 5) to demonstrate MAXQ hierarchical reinforcement learning. The taxi problem can be formulated as an episodic MDP with the 3 state variables: the location of the taxi (values 1-25), the passenger location including in the taxi (values 1-5, 5 means in the taxi) and the destination location (values 1-4). Figure 5 shows a 5-by-5 grid world inhabited by a taxi agent. There are four specially designated locations in this world, marked as R(ed), B(lue), G(reen), and Y(ellow). The taxi problem is episodic. In each episode, the taxi starts in a randomly chosen square. There is a passenger at one of the four locations (chosen randomly), and that passenger wishes to be transported to one of the four locations (also chosen randomly). The taxi must go to the passengers location, pick up the passenger, go to the destination location, and put down the passenger there. The episode ends when the passenger is deposited at the destination location. There are six primitive actions in this domain: (a) four navigation actions that move the taxi one square North, South, East, or West, (b) a Pickup action, and (c) a Putdown action. There is a reward of -1 for each action and an additional reward of $+20$ for successfully delivering the passenger. There is a reward of -10 if the taxi attempts to execute the Putdown or Pickup actions illegally. This task has a simple hierarchical structure in which there are two main sub-tasks: Get the passenger and Deliver the passenger. Each of these subtasks in turn involves the subtask of navigating to one of the four locations and then performing a Pickup or Putdown action. The temporal abstraction is obvious, wherein, for example, the process of navigating to the passengers location and picking up the passenger is a temporally extended action that can take different numbers of steps to complete depending on the distance to the target. The top level policy (get passenger; deliver passenger) can be expressed very easily if these temporal abstractions can be employed. Reusing policies is critical in this domain. For example, if the system could learn how to solve the navigation subtask once, then the solution could be shared by both the *Get the passenger* and *Deliver the passenger* subtasks. We use the episodic format for task solving in the Taxi Domain proposed by (Dietterich, 2000), where in the beginning of each episode, the taxi, the passenger position and the intended destination are randomly set.

Figure 6 shows the plot of average return for PCCA+HRL compared to Lcut and Random Options generated with primitive actions. The PCCA+HRL identified 20 options while the LCut method identified 27. Among the 20 discovered by PCCA+HRL are the different pick-up and drop-off options corresponding to the different destinations. Our approach consistently performed significantly better than the random options method. In the case of the LCut approach, our approach does significantly better for smaller number of epochs. For a larger number of epochs, the LCut method performs marginally better but the difference is not significant.

2.2.7. NOTE:

The PCCA+HRL updates the transition matrix periodically and learns options. With every sampled trajectory, the structure of the transition matrix changes which in turn changes the spatial abstractions identified. In order to define the SMDP update rule, we still need a mechanism to match the previous options to new ones. This can be done easily by mapping the vertices of the simplex returned by PCCA+. Let $\widetilde{Y_1}$ and $\widetilde{Y_2}$ be the eigenvector matrices returned by PCCA+ in iterations 1 and 2 respectively. The similarity metric between simplex vertex i in iteration 1 and vertex j in iteration 2 is given by $Sim_{12}(i,j)$ where $\kappa_{12} = \widetilde{Y_1}\widetilde{Y_2}^{-1}$. Using κ_{12}, we assign vertex i of simplex 1 to vertex j of simplex 2 using the Munkres Algorithm (a.k.a Hungarian Method) where the match weighting between i and j is $\kappa_{12}(i,j)$.

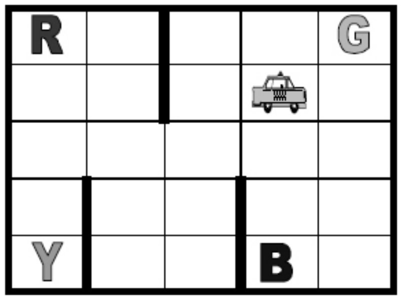

Figure 5. Taxi world domain

3. Playing video games with options

In the previous section, the effectiveness of the PCCA+HRL framework was shown on problems like the 2-room domain and the taxi domain. Even though these problems require abstraction in the policy, they have a considerably small state space. The PCCA+HRL framework must be extended to large complex problems with much higher dimensional state spaces and many more options which are complex in nature. We delve into the difficulties faced in complex tasks such as learning to play video games due to high dimensionality of the state space and (or) complexity of the task. We have

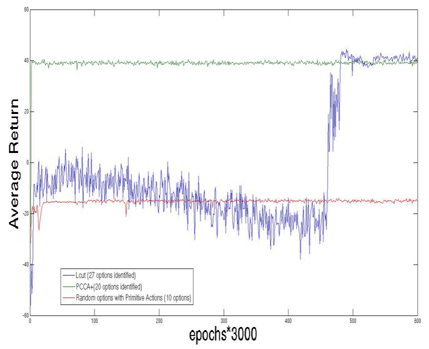

Figure 6. Average Return for Taxi Domain

conducted experiments on Infinite Mario and the Atari 2600 game, Seaquest to test our proposed model. The two games differ in the level at which the emulator provides the game state to the agent - Mario provides a higher-level representation which tells the agent the type of object present at a particular grid cell while Atari games such as Seaquest provide a pixel-space representation to the agent which makes object detection an added task for the agent. We present the model for the latter since it is more general with respect to planning based on perception. The approach to this would be to come up with a scheme for aggregating states, and learn a transitional model in these aggregated states and run our partitioning on them to discover metastable regions. To perform an aggregation of the state space that would be suitable for spatio-temporal abstraction, we would not only want to aggregate based on the visual features but also contextual information like previous states and actions in the trajectory leading to the current state. This way, we implicitly capture the model based information into our lower dimensional representation of the state space which will be used for a spatio-temporal state-space aggregation.

3.1. Sampling trajectories

Spatial abstraction using PCCA+ is essentially breaking down the MDP into *metastable* states which are densely connected within and have sparse connections to other metastable states. These densely connected regions have low mixing times and hence particles stay in same region of state space for long periods of time without external stimulus (an idea originating from conformal dynamics theory). The important assumption here is that particles are performing random walks. For hard tasks such as playing video games, we cannot sample trajectories where the agent is performing random walks since it would very rarely cross the initial region of the state space due to enemies and obstacles. We need to cover different subspaces of the state space, which is all the more relevant in complex video games like Mario and Seaquest, where the scene also changes as the game progresses. Therefore, we need a better policy than a random player to sample trajectories. To be more precise, we need guided exploration to help the agent cross enemies and obstacles and at the same time, not get restricted to visiting particular parts of the state space if the guiding policy is executed greedily. Thus, we need a reasonably good policy executed with a sufficient exploration rate.

Our target task is to learn to play an Atari 2600 game Seaquest well. For fast online play to sample good trajectories, we need a partially trained Deep Q-Network that performs reasonably well on the task. Therefore, we train a Deep Q-network (with primitive actions) as explained in the paper (Mnih et al., 2015) upto 40 epochs. The Deep Q-network (DQN) combines the compositional representation capabilities of convolutional neural networks with the robustness of a Q learning setup, predicting the action-value $Q(s, a)$ for each primitive action a for a state s. We sample trajectories from a partially trained DQN (50 million frames corresponding to 40 epochs) with exploration factor ϵ annealed from 0.5 to 0.3.

Once options are generated based on the initial set of sampled trajectories (from a primitive action based DQN), the agent begins executing options which would lead to unexplored (yet) regions of the state space. Hence, we need to sample trajectories periodically as the agent trains using options to capture the changes in the structure of the transition matrix, and hence, improve our option learning.

Ideally, we would want an end-to-end system without a partially trained Deep Q Network being used as an initial expert to guide our framework for trajectory sampling. However, it might take a large number of cycles of repeated trajectory sampling and option composing through PCCA+ to get the agent to learn reasonable options due to the reason that it might take us a long time to cross initial obstacles and get into new regions of the state space. For example, in Seaquest, filling up oxygen after capturing 6 humans gives plenty of points and this is a skill that has to be learnt. However, only if the agent proceeds long enough in the game to be able to evade the enemies till then and capture 6 humans, it can even get to a stage where the option execution is possible. However, we still make sure we don't use a very well trained expert trained for 200 epochs as in typical DQN implementations in (Mnih et al., 2015), but only a partially trained network up to 40 epochs. We leave it for future work to explore how long it takes to learn from a completely random policy through multiple cycles of exploration and option learning, for a the ultimate goal of realizing an end-to-end skill acquisition agent. As mentioned in the PCCA+ section, it is expensive to perform PCCA+ after every episode, and thus we update the transition matrix periodically and modify the transition matrix to capture the changes in the spatial abstraction by matching previous options to the new ones using the Hungarian Algorithm.

3.2. Representation Learning

The original state space in Atari 2600 domain is $210 \times 160 \times 3$ with each cell having 256 discrete possibilities, and the image having 3 color channels. Aggregation of states in this space is practically infeasible. We therefore need to learn a lower dimensional representation of the state space to enable conventional clustering algorithms to perform our state space aggregation. The important idea is to encode spatio-temporal information in the lower dimensional space, to help our PCCA+HRL framework learn the abstractions on the task aligned to the spatio-temporal aspects of the game. Therefore, a conventional RGB channel autoencoder reconstructing the image, with each treated as a static entity does not make sense. We need to aggregate states based on the history leading to the state in addition to the visual features of the state. Thus, we need to encode history in the state space representation to be learnt. The history must include not only the past states but also the actions taken at each, which led to the current. In essence, we need to encode the trajectory information leading to a state in its state space representation. This can also be motivated by the fact that Atari 2600 games have a deep partial observability nature in the sense that just looking at the current frame is insufficient to decide the optimal action. In the case of Seaquest, it is important to know whether the bullet is traveling towards the agent or away or the previous direction of movement of the agent when evading enemies, etc. Thus, we need to aggregate the states based on the context to make sure the options are learnt by relating different concepts (like evading enemies or filling oxygen) rather than just visual features.

We took inspiration from the work on action-conditional video modelling problem (Oh et al., 2015). In this work, the authors propose and evaluate architectures to generate future frames given the sequence of frames observed conditioned on the agent's action. Their motivation to do this is to help improve exploration policies, where the agent can predict the next frame conditioned on the action to be explored and prioritize on choosing actions whose predicted next states have features very different to regions of state space already encountered. We borrowed the idea of learning to predict the next sequence conditioned on our action so as to learn a good representation of the states conditioned on the game world model.

We closely follow their deep neural network architecture, which we refer to as *TemporalRepLearner* that performs the task of an encoder. This involves an LSTM layer after multiple stacked convolutional layers to capture temporal information in the latent space representation. In order to capture the effect of the agent's action on the objects in the game screen, there are multiplicative (element-wise) interactions between the LSTM's hidden representation and the action variable. These are used to predict the next frame in the high-level latent representation. The high-level latent representation of the next frame is then decoded to the original image space through a stack of deconvolution layers which together make up the decoder. The predicted image of the next frame is compared with the original next frame image and the mean squared error between the two is back-propagated across time to facilitate the learning. Figure 7 shows the architecture of the LSTM Convolution-Deconvolution Network used in (Oh et al., 2015).

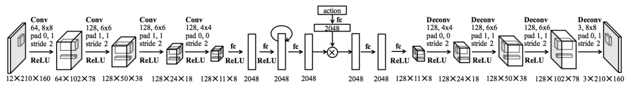
Conv 64, 8x8 pad 0, 1 stride 2 ReLU
Conv 128, 6x6 pad 1, 1 stride 2 ReLU
Conv 128, 6x6 pad 1, 1 stride 2 ReLU
Conv 128, 4x4 pad 0, 0 stride 2 ReLU
fc ReLU
fc ReLU
fc
action
fc
2048
fc
fc ReLU
Deconv 128, 4x4 pad 0, 0 stride 2 ReLU
Deconv 128, 6x6 pad 1, 1 stride 2 ReLU
Deconv 128, 6x6 pad 1, 1 stride 2 ReLU
Deconv 3, 8x8 pad 0, 1 stride 2
12×210×160
64×102×78
128×50×38
128×24×18
128×11×8
2048
2048
2048
2048
2048
128×11×8
128×24×18
128×50×38
128×102×78
3×210×160

Figure 7. Architecture of *TemporalRepLearner*

Since the encoding layers involve both convolution layers as well as an RNN, we refer to it as a recurrent encoding. The input to *TemporalRepLearner* is a frame of the game for each time-step. There are 4 convolutional layers with rectifier nonlinearity applied after each, followed by a dense layer consisting of 2048 hidden units. The LSTM layer with 2048 hidden units is unrolled through last 11 frames. Thus, there are 2048 factors in the action-conditional transformation. This is followed by a rotation to get the next frame's lower dimensional representation. A fully connected layer is then used to construct a 3D feature map from which the 4 deconvolution layers in the decoder finally output a pixel space prediction for the next frame.

The recurrent network is trained using frames partitioned into training and test sets from the trajectories sampled using the trained Deep Q-network. We use the same splits as (Oh et al., 2015). The 2048-dimensional hidden state of the LSTM layer (after the action conditioning layer) is used as the lower dimensional representation for the state space in Seaquest. This representation ensures spatio-temporal understanding of the frames in the game. We have also transformed the original high dimensional space at the pixel level ($210 \times 160 \times 3$) to a lower dimension (2048) representation. This will now enable clustering algorithms to run on the lower dimensional space and help us in state aggregation for the PCCA+HRL framework to work on.

3.3. State Aggregation

In this section, we discuss how to perform the state aggregation on top of the discovered representation from the deep network. As pointed out earlier, in higher dimensional problems, state aggregation is necessary to capture a workable transition matrix for the PCCA+HRL framework to discover metastable regions and options. This is equivalent to discretizing the exponential state space for a finite dimensional transition matrix to be used for PCCA+. Note that the deep network was used only to acquire a representation for state aggregation to work in the first place. We need to aggregate on top of the discovered representation to discover metastable regions connecting different regions of the R^{2048} space.

K-Means algorithm, a well-known clustering algorithm has an average complexity of $O(n * k * d * i)$ where n is the number of data points, k is the number of cluster centroids, d is the dimensionality of the space and i is the number of iterations where we begin with a new random initialization of the cluster centroids. This presents a scalability problem for the video game setting - the learnt representation still has a high dimensionality (2048) and the number of data points is large as well, since they come from all frames encountered in the sampled trajectories of the game. At the same time, we can't afford to encode the original image space in a very low dimensional representation of say 10 or 50 dimensions in the LSTM network. The features discovered might too complex and few and whether reconstruction is possible has to be investigated. Therefore, we decided to work with the same 2048 dimensions as used in (Oh et al., 2015). Future work will investigate if we can afford to learn the action conditional representation with lower dimensions like 100 or so with a low test Mean Squared Error (MSE). For the current model, 2048 dimensions is still high and it is very common that for any two data points, there exist at least two dimensions along which the points are far apart from each other. Distance functions that use all the input dimensions are thus not effective and we need feature selection prior to clustering.

We utilize *mini-batch* based K-Means to handle the issue of scalability with respect to number of data points.The main idea proposed in (Sculley, 2010) is to use randomly sampled batches (of a fixed small batch-size) at each iteration. Each *mini-batch* updates the cluster centroids using a convex combination of existing centroids and the mean of the mini-batch data points. With respect to curse of dimensionality, global feature selection techniques are not apt for discovering clusters that exist in different subspaces. Each dimension could possess information that is relevant to a particular cluster which makes global pruning of features inefficient. In order to capture the correlations among the features while clustering, feature selection must be performed locally. (Parsons et al., 2004) propose a soft feature selection procedure in which features are assigned local weights based on correlation among data points in each dimension. We use this to assign to each cluster a weight vector for the dimensions based on the correlation of points in that cluster represented by them.

3.3.1. Note

We selected mini-batch K-Means with local feature selection to address the issues of huge number of datapoints and high dimensionality. However, we do not claim that mini-batch K-Means is the best approach to do state space aggregation on top of the action conditional network's representation. We leave it for future work to experiment with other state aggregation (clustering) techniques and evaluate the effectiveness of the PCCA+HRL framework for each state.

3.4. Model Learning

Post state-aggregation, we closely follow the PCCA+HRL framework that has already been described to identify the metastable regions of the state space and options which connect them that naturally arise from the PCCA+ connectivity information. Sampled trajectories of game play (in terms of aggregated micro-states) are used to populate a count matrix which keeps track of the number of transitions seen for every (s, a, s') tuple. That is, if at a particular time step, an agent is at state s, takes action a and moves to state s', we increment the element corresponding to (s, a, s') in the matrix. Note that the states here are not the original states (at pixel level), but the aggregated micro-states found after Deep Representation Learning & K-Means clustering.

As described earlier, we incorporate the reward structure of the environment into the transition matrix: the idea is that our spatial abstractions should *degenerate* in places where there is a spike in the reward distribution so that our agent interprets a state of high reward (as goal states typically are) as a different abstract state, and naturally composes options that lead the agent to that state. The update equations when we see a transition from (aggregated) state s to state s' on taking action a and obtaining reward r are given by:

$$\phi(s, a, s') = \phi(s, a, s') + 1$$
$$V(s, a, s') = V(s, a, s') + \phi(s, a, s')e^{-\nu|r|}$$

where ν is a regularization parameter

Now, with the transition count matrix V, we estimate the following matrices:

$$D(s, s') = \sum_a V(s, a, s')$$
$$T(s, s') = \frac{D(s, s')}{\sum_{s'} D(s, s')}$$
$$P(s, a, s') = \frac{V(s, a, s')}{\sum_{s'} V(s, a, s')}$$

The state-to-state transition matrix T is supplied as input to the spectral clustering algorithm PCCA+. Options are generated as explained in section 2.2. The probability matrix P is utilized in defining the option policy explained in section 2.2 (used for calculating $\mu(s, a)$ for each option).

Once the PCCA+ identifies the abstract states and the abstract tasks naturally arise from the connectivity information, we have the options ready for our reinforcement learning algorithm to work. However, there is a challenge here, since our reinforcement learning algorithm has to work online in the image space for deciding between primitive actions and applicable options at every state. Therefore, we need to adapt the DQN set up with experience replay used in (Mnih et al., 2015) to include our options framework. We also need to change Q learning to handle updates to the options as well. For this, we resort to Deep Intra-Option Q learning, which is explained in the next section.

3.5. Intra-Option Value Learning

In this section, we describe our reinforcement learning attempts after having derived the options from PCCA+. We experiment with two different setups. In the first setup, we use the *TemporalRepLearner* (which uses convolutional and recurrent layers to encode spatio-temporal history) till the layer we use for our representation in the K-Means clustering. The network is frozen till this depth and is followed by a Multi Layer Perceptron with one fully connected hidden layer and sigmoid activation function, with the output linear layer having as many units as the number of primitive actions and options. The Q-learning updates take place only on the MLP layers. In the other setup, we use the representation learnt from a DQN with the DQN network from (Mnih et al., 2015) frozen till the pre-output layer, and the output layer containing as many units as the number of primitive actions and options with the last rotation matrix alone updated. We want to empirically confirm our hypothesis that the first setup with extra spatio-temporal information is better for learning options than just using the DQN representation. We verify that it is indeed the case as seen in Figure 8, with an additional graph coming from the scores that a normal DQN with only primitive actions would acquire at the same number of training epochs.

We retain the extra features that were used in the DQN by (Mnih et al., 2015) such as experience replay with uniform sampling and usage of a separate target network that is frozen and updated periodically while the current network is learnt. The cost function of the above network is designed such that the parameters θ_i at iteration i are adjusted according to the Intra-Option Q learning rule (Sutton et al., 1998). For a single transition sampled from the experience replay, we may need to apply multiple updates to the network. In order to do this in minimal time, we designed a dictionary in order to map a (micro-state, primitive action) pair to the set of options which are consistent with executing the primitive action at that micro-state. This dictionary is populated once before training begins.

When the agent is at play, primitive actions are used at the emulator level even when an option is being executed. Thus, when transitions are dumped into the experience replay, they consist of only primitive actions. When a transition $(s, a, s^{'}, r)$ is sampled from the experience replay, we compute the micro-state assignment of s - say c_s, lookup the dictionary using the key (c_s, a) to find the set of all options O which are consistent with this primitive action. For the nodes corresponding to each option $o \in O$ in the output layer along with the node for the primitive action a, we apply the Intra-Option Q-value update.

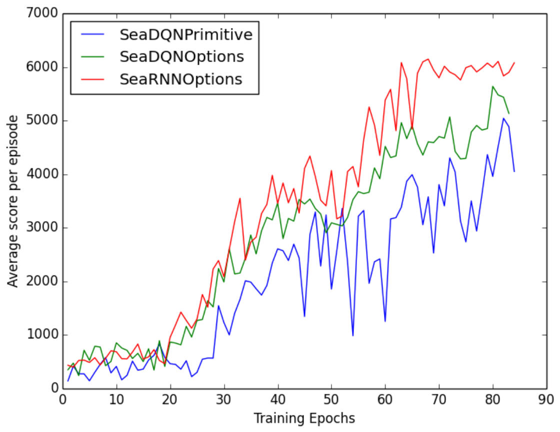

Figure 8. Comparing performance of SeaDQNPrimitive, SeaDQNOptions and SeaRNNOptions

Figure 8 plots the average score per episode against training epochs. *SeaDQNPrimitive* corresponds to a DQN trained on Seaquest with only primitive actions while *SeaDQNOptions* corresponds to a modified DQN which is trained with the output layer having both primitive actions and the options generated by our approach. *SeaRNNOptions* corresponds to the model described in section above where a latent representation is learnt using *TemporalRepLearner*. Intra-option Q-learning is used in both *SeaDQNOptions* and *SeaRNNOptions*. We see that *SeaRNNOptions* is the best model over the course of the training epochs for which results are shown. Figure 9 helps to appreciate the semantics of an example option generated in Seaquest. We plot the agent's membership in the abstract state corresponding to the particular option for a randomly chosen episode's trajectory. The skill learnt here corresponds to resurfacing to replenish oxygen in Seaquest.

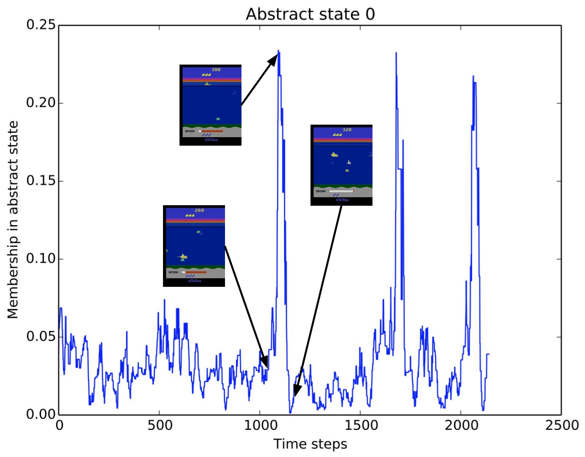

Figure 9. Visualization of an option in terms of membership value in corresponding abstract state in Seaquest which corresponds to resurfacing to replenish oxygen

4. Discussion

In this paper, we have presented a new framework for automated skill acquisition in reinforcement learning that discovers hierarchical structures over a given task in terms of abstract states and abstract tasks (options or skills) connecting different abstract states. The important advantage of our model is that the options come for free from the PCCA+ connectivity information. We also emphasize that our model does not require a prior model of the MDP and can sample trajectories and learn in an online fashion, refining the abstractions over time based on the current estimate of the transition matrix. The skills learnt through this framework are general and can be reused across tasks sharing a common state space. This is really useful in the transfer learning context. We also wish to highlight another important aspect of our work in light of recent attempts at Hierarchical Reinforcement Learning (Kulkarni et al., 2016). Our model discovers the subgoals and the options to realize the subgoals with no explicit need to have special intrinsic rewards or a list of skills or subgoals already engineered or listed for the task.

However, we still have issues to address in the current framework, which we list down below:

- Our method needs to scale better. The deep network training is too slow since every decision making involves identifying the metastable region the current state belongs to, caching the options corresponding to that region and then performing a Q learning update. The recurrent encoding is very deep and the forward pass thus introduces a latency in online performance as well as training.

- We need to avoid the usage of an initial partially trained DQN to make the model end-to-end.
- We need to explore more efficient clustering strategies for the state-aggregation component.
- Our framework will still suffer in the case of sparse reward problems like Montezeuma's Revenge. Even a DQN wouldn't help for being an initial policy to give us sample trajectories. We probably require human played trajectories for Montezeuma's Revenge to observe good regions of the state space for our sample trajectories. However, we believe this (with human trajectories sampled for model learning) is still a better framework compared to explicitly giving the agent a list of skills as done in recent work by (Kulkarni et al., 2016).
- Currently, we use the same network to decide between options and actions. However, deciding between options must not depend on lower dimensional feature detectors, but rather must be based on more high level filters. Therefore, we need to experiment with different networks for the actions and options as done by (Kulkarni et al., 2016).

Our future work will focus on refining each piece of the model and making it more robust with experiments on more domains that require hierarchical reasoning in high dimensional state space like other Atari 2600 games like Montezeuma's Revenge. We wish to explore different clustering algorithms and also the *TemporalRepLearner* network with smaller number of units in the hidden representation. We also want to perform more iterations of the PCCA+ in the Deep Learning setup. The results shown in the paper are for one cycle, where we obtained sampled trajectories from a partially trained DQN, found a lower dimensional representation, aggregated states and learnt options using PCCA+. Using the learnt options, we train using Intra Option Q Learning an online agent that can decide when to use these options and actions optimally. However, we need to experiment further with this online agent being used as the next agent to sample trajectories with, for the next round of PCCA+ to discover better options and further improve the performance of the agent. We also wish to explore other reinforcement learning methods than Intra Option Q Learning to learn the value functions for options and actions with possibly different networks. In addition to this, we would like to study the issues related to training stability and speed in learning the policies. There is also a need to show experiments on how the abstractions learnt can be reused across tasks in the same state space in Atari 2600 by synthesizing tasks on the same state space with different subgoals and goal. Successful results on other domains can add to the empirical evidence for the robustness of the model. We experimented with uniform random sampling experience reply in this paper. It would also be a good direction to see how much the PCCA+HRL framework benefits from the more recent attempts at improving experience replays like the Prioritized Experience Replay proposed by (Schaul et al., 2016).

5. Acknowledgement

We thank the reviewers of the Abstraction in Reinforcement Learning ICML 2016 workshop for their useful comments regarding different components of our framework. The DQN implementation we use is: `https://github.com/spragunr/deep_q_rl` and is based on Theano (Bergstra et al., 2010). We built the hierarchical PCCA framework on top of the DQN codebase.

References

Amato, Christopher and Shani, Guy. High-level reinforcement learning in strategy games. *9th International Conference on Autonomous Agents and Multi-Agent Systems*, 1:75–82, 2010.

Bakker, Bram. Reinforcement learning with long short-term memory. *NIPS*, 2002.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, pp. 253–279, June 2013.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Dietterich, Thomas G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, pp. 227–303, 2000.

Hengst, Bernhard. Model approximation for HEXQ hierarchical reinforcement learning. In *Machine Learning: ECML 2004, 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004, Proceedings*, pp. 144–155, 2004. doi: 10.1007/978-3-540-30115-8_16. URL `http://dx.doi.org/10.1007/978-3-540-30115-8_16`.

Kulkarni, Tejas D., Narasimhan, Karthik R., Saeedi, Ardavan, and Tenenbaum, Joshua B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Arxiv*, 2016.

Langley, P. Crafting papers on machine learning. In Langley, Pat (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lin, Long-Ji. Reinforcement learning for robots using neural networks. *Technical Report, DTIC Document*, 1993.

McGovern, Amy and Barto, Andrew G. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pp. 361–368, 2001.

Meila, Marina and Shi, Jianbo. A random walks view of spectral segmentation. 2001.

Menache, Ishai, Mannor, Shie, and Shimkin, Nahum. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002, 13th European Conference on Machine Learning, Helsinki, Finland, August 19-23, 2002, Proceedings*, pp. 295–306, 2002. doi: 10.1007/3-540-36755-1_25. URL http://dx.doi.org/10.1007/3-540-36755-1_25.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

Ng, Andrew Y., Jordan, Michael I., and Weiss, Yair. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pp. 849–856. MIT Press, 2001.

Oh, Junhyuk, Guo, Xiaoxiao, Lee, Honglak, Lewis, Richard L, and Singh, Satinder. Action-Conditional Video Prediction using Deep Networks in Atari Games. In Cortes, C, Lawrence, N D, Lee, D D, Sugiyama, M, Garnett, R, and Garnett, R (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2845–2853. Curran Associates, Inc., 2015.

Parsons, Lance, Haque, Ehtesham, and Liu, Huan. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *ICLR*, 2016.

Sculley, D. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pp. 1177–1178, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772862. URL http://doi.acm.org/10.1145/1772690.1772862.

Silver, David, Sutton, Richard, and Muller, Martin. Reinforcement learning of local shape in the game of go. *IJCAI*, 7:1053–1058, 2007.

Simsek, Özgür and Barto, Andrew G. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pp. 1497–1504, 2008.

Sutton, Richard S. and Barto, Andrew G. Introduction to reinforcement learning. *MIT Press*, 1998.

Sutton, Richard S., Precup, Doina, and Singh, Satinder P. Intra-option learning about temporally abstract actions. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 556–564, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1558605568. URL http://portal.acm.org/citation.cfm?id=657453.

Vafadost, Mostafa. Temporal abstraction in monte carlo tree search. *Masters thesis, Department of Computer Science, University of Alberta*, 2013.

Watkins, Christopher J. C. H. and Dayan, Peter. Technical note: Q-learning. *Mach. Learn.*, 8(3-4):279–292, May 1992.

Weber, Marcus, Rungsarityotin, Wasinee, and Schliep, Alexander. Perron cluster analysis and its connection to graph partitioning for noisy data. Technical Report 04-39, ZIB, Takustr.7, 14195 Berlin, 2004.