Modeling Skill Acquisition and Retention

Forrest J. Laine (forrest.laine@berkeley.edu)
Department of Electrical Engineering and Computer Sciences
Berkeley, CA 94720 USA

Abstract

This project attempts to model how humans maintain a set of low-level skills which can be used to plan and execute higher-level tasks. To investigate this, I developed a web game in which users are presented with a sequence of tasks to solve. By remembering the solutions to previously seen tasks, solutions are found more easily. By analyzing data collected on this task, I propose a simple linear model which is able to predict when it is advantageous to learn a new skill at the cost of forgetting an old one. This work hopes to inspire models for reinforcement learning agents which are capable of integrating past experience into decisions about skill acquisition and retention.

Keywords: Task hierarchy; Skill maintenance; Lifelong learning; Meta-reasoning;

Introduction

Reinforcement learning has recently demonstrated its effectiveness as a means for learning control policies for high-dimensional dynamical systems. Two examples include learning to play the Atari games using raw pixel data as input (Mnih et al.,), and learning to perform manipulation tasks with a robotic arm using camera data as inputs (Levine, Finn, Darrell, & Abbeel,). Methods like these have opened the door for learning-based approaches to robotic control, but have still only been shown to work on relatively toy tasks.

One major difficulty for robotic control tasks is the ability to learn tasks requiring long sequences of actions, where reward signals are sparse. To see why this is the case, consider learning a manipulation task for a 12-dimensional robotic arm, where each dimension could represent the angle or angular velocity of a particular joint. If we discretized each dimension into 10 bins (very coarse for most tasks of interest), at each point in time the system could exist in one of 10^{12} (one trillion) states. Assuming control frequency of 10Hz and system dynamics such that state transitions during a single control cycle are to one of 10 neighboring states, the total number of trajectories for this system is 10^{10t+12} where t is the duration of the trajectory in seconds. Brute-force search over trajectories is clearly intractable, even on tasks lasting only seconds. Learning-based methods perform better, but still fail once the length of the task being learned becomes such that the system is unable to find positive or negative feedback regarding its actions.

However, just as humans do not plan our days in terms of a sequence of muscle contractions lasting 100ms, control policies for autonomous systems need not operate on such a fine scale. One way to overcome this difficulty would to learn a sequence of higher-level actions, focusing only on a few key decisions instead of hundreds of relatively inconsequential micro-decisions. High-level actions could be thought of as skills or sub-policies which arise repeatedly in a domain. Thrun, Schwartz, et al. () argued that an optimal set of skills for a particular domain are those which minimize the description length of all policies arising in that domain. Solway et al. () showed that the same conclusion can be made using a Bayesian model selection approach, and showed that humans tend to decompose problems according to such a scheme. Daniel, vanHoof, Peters, and Neumann () demonstrated an approach to discovering continuous-space sub-policies while concurrently learning a high-level policy for the task at hand using the Options framework (Sutton, Precup, & Singh,).

Many others have come up with additional methods for sub-task discovery. However, many of these take a graph theoretical approach, such as identifying bottleneck states as targets for sub-tasks (Kazemitabar & Beigy,). These methods assume a discrete and known environment. In most robotic control domains, these assumptions do not hold. The nature of these domains suggest an approach more akin to that of Daniel et al., in which sub-tasks are identified from earlier problems and reused in the future. Since real systems have resource constraints such as limited memory, using a method such as this raises a question about how and which skills should be maintained as new tasks are encountered by the agent. This is the question this project aims to investigate.

In order to look deeper into this idea of skill maintenance, I formulate a simplified version of the problem as a meta-level Markov Decision Process. As will be seen, optimal solutions to this formulation can not easily be solved for due to uncertainty about future tasks. Instead of attempting to develop heuristics for the task from scratch, I instead look to humans to try and find patterns in the way they approach the problem, hoping any such patterns will lead to better heuristics, as well as possible insight into human behavior. To accomplish this, I have developed an interactive web game which collects game data from users as they play. This game was designed such that skills are clearly defined, yet difficult for users to maintain. Using the data collected, I fit and compare linear models of the skill maintenance problem using various feature representations, and remark on the quality of the models.

The following section of this paper will present the skill maintenance problem formulation. In the following section, the web game will be presented, highlighting connections to the problem formulation. Finally, the results section will present experiment data and discus the fitting of a linear policy.

Problem Statement

The problem of maintaining and acquiring skills in general is not clearly defined. In an attempt to be more rigorous, I

here consider a simplified version of the problem. In this formulation, I consider problems in which there exist a series of high-level tasks comprised of clearly separated low-level tasks. Each low-level task has an associated optimal solution which can be executed deterministically. Solutions to low-level tasks can be learned, with a known expected cost to learn. When a low-level task is encountered and the solution to the task is known, it is executed perfectly. When the solution is unknown, the task is learned, and upon completion, the set of known skills is potentially updated by some transition function. Known skills are maintained for the duration of the current and all future high-level tasks unless otherwise dictated by the knowledge transition function.

The high-level decision problem at each point becomes the problem of choosing which sub-tasks to execute, assuming a choice exists, such as to maximize the reward on the current and future high-level tasks. Since the knowledge transition function dictates the updating of skills, the difficulty in this problem arises from the potential negative effect updating skills has on future tasks. Thus, when considering which sub-tasks to execute in a given high-level task, an intelligent agent might leverage information about past experiences to hypothesize about which skills will yield high future reward.

This problem is still poorly defined, as optimizing for unknown future events is impossible. However, considering a world in which each time-point corresponds to solving a single high-level task, this problem we are optimizing over looks very similar to that which is solved in reinforcement learning, in which a policy is optimized to maximize the expected sum of discounted rewards in an environment with unknown transition dynamics. Here, the transition dynamics between high-level tasks is also unknown. However, unlike in the reinforcement learning domain, we are unable to repeatedly execute and refine our policy. By formulating this problem as a standard MDP, I hope to gain insight as to how humans approximate value functions on-line without repeated experience.

This problem can be represented as a meta-level MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma\}$. Here, $\mathcal{S} = \mathcal{L}\ x\ \mathcal{B}^K$ where $\mathcal{L}$ is the set of all high-level tasks, $\mathcal{B}$ is the set of all sub-tasks, and $\mathcal{B}^K$ is the space of all possible configurations of a *policy bank* of size K. Each high-level task in $\mathcal{L}$ is considered to be comprised of disjoint sub-tasks in the set $\mathcal{B}$. $\mathcal{A}$ then represents the set of all possible sequences of low-level tasks which can accomplish a high-level task. Given this formulation, a subscript $l \in \mathcal{L}$ is used to represent the set of available actions $\mathcal{A}_l \subseteq \mathcal{A}$ and the set of possible states $\mathcal{S}_l \subseteq \mathcal{S}$ for a given high-level task l. $\mathcal{R}$ is a reward function $\mathcal{S}\ x\ \mathcal{A} \rightarrow \mathbb{R}$ which is the cost of executing a path of low-level tasks $a \in \mathcal{A}_l$ given $s \in \mathcal{S}_l$. γ represents the discounting factor for future rewards, and finally $\mathcal{T}$ is the transition function $\mathcal{S}\ x\ \mathcal{A} \rightarrow \mathcal{S}$, which updates the skills known and presents a new high-level task upon completion of a previous task.

The objective for this problem is then to find a policy π : $\mathcal{S}\ x\ \mathcal{A} \rightarrow [0,1]$ that optimizes the following:

$$\max\ \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t\ r(s_t,a_t)\right] \tag{1}$$

Where $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t,a_t)$

Although $\mathbb{E}[r(s_0,a_0)]$ is known, expected future rewards are unknown. If optimal solutions to this problem were given, we could create a function approximation of the optimal policy by fitting a linear model Z to predict path selection given a featurized state representation. This linear model can then represent, in some sense, an approximation to the Q-function for the MDP on which demonstrations were given. Traditionally $Q(s,a)$ represents the expected sum of discounted rewards starting in state s and taking action a. Given the true Q-function, the optimal policy $\pi(a|s) \propto Q(s,a)$. Although the targets I propose regressing to are simply one-hot encodings of observed policies and *not* the true Q-values, an approximation to the optimal policy can be constructed in a similar fashion $\tilde{\pi}(a|s) \propto Z(s,a)$. By comparing feature representations of the models that minimize prediction error can give insight into how predictions about how prior experience can be encoded such as to account for potential future tasks. In addition, analyzing human demonstrations may give insight into the computational model humans use for such tasks.

Method

To investigate this problem, I developed a web game to which solutions can be thought of in the formulation above. The game, described in more detail below, was hosted on my web page and was written in javascript. During play, game data was transferred to a Google Sheet using the Sheets API and an embedded Google Apps Script. Users played one of four random instances of the game, each having a slightly different sequence of levels. All levels were fixed before being released for play and were not modified in anyway to adjust to user data. The users playing the game for this data set were EECS graduate students recruited from the email list serve eecs-grads-misc.

Skill Blocks: The Game

The game [1] asked users to complete a series of tasks, composed of 7 or 8 levels, each of which required the user to get from a starting position to a goal position. An example layout of a level can be seen in Figure 1. Levels are comprised of a sequence of blocks, where aside from the start and goal blocks, each block requires the user to enter a code comprised of 3-5 symbols to move to the next block. Symbols are one of the four arrow keys (left, up, down, right). The length of each code is indicated by the number printed in the center of each block.

Blocks are color-coded, and all blocks of the same color share the same code, remaining constant across levels. Therefore, each color of block represents a skill which transfers

[1] Web game can be played here: `https://people.eecs.berkeley.edu/~forrest.laine/skills/index.html`

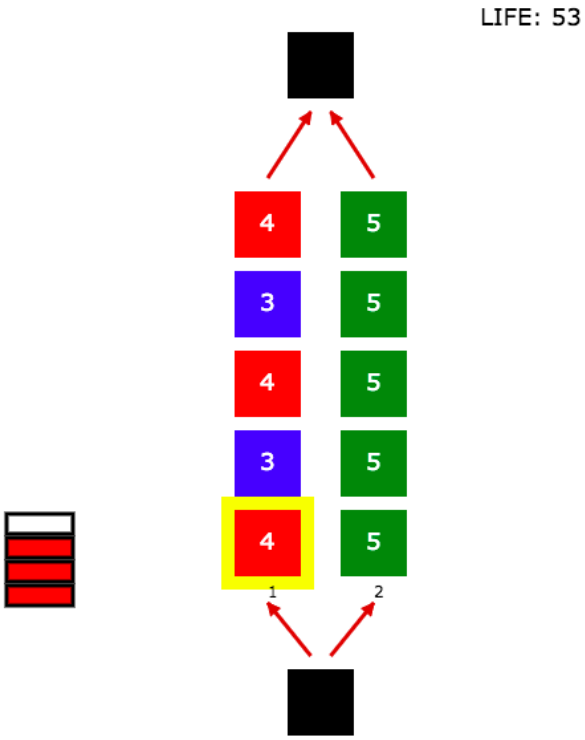

Figure 1: Example level from web game. The user's location is indicated by the yellow box. The progress of the current skill is indicated by the progress bar on the left, with three of four keys successfully entered.

to future tasks. The length of the codes and organization of levels was designed such that remembering skill codes is not easy, and therefore common to forget previously known skills. As codes are unknown when entering skill blocks for the first time, the user must experiment with keys to determine the code.

Progress towards determining the code is displayed within a progress bar appearing next to the active skill block. When a correct key is pressed, one portion of the progress bar will fill up. When an incorrect key is pressed, progress will be reset and the user must start over. Each keystroke used causes the loss of one unit of Life, and the game is terminated when all Life is depleted.

Data Collection

Game information collected from each user consisted of the following items:

- Level set (which game version was being played)
- Codes associated with each color skill block (codes are generated randomly for each user)
- Path chosen by user on each level
- Count of total keystrokes on each level

As previously mentioned, there were four level sets which users were randomly assigned to play. The sequence of levels in each of these sets can be see in Figures 2, 4, 6 and 7. Data was collected approximately 24 hours after public release. 104 user responses were recorded in total. Of these, 34 did not complete more than 2 levels and were deemed to not understand the game. These user's data entries were removed from the pool.

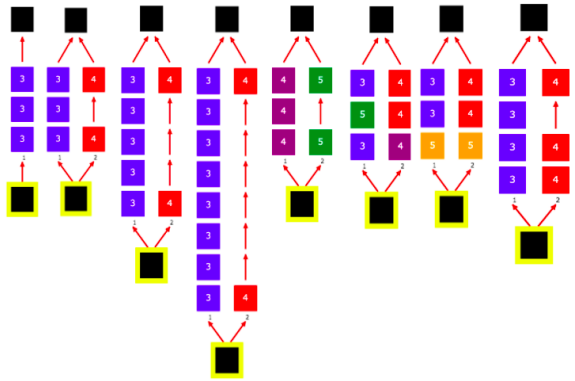

Figure 2: Game sequence for level set 'A'. Levels progress from left to right.

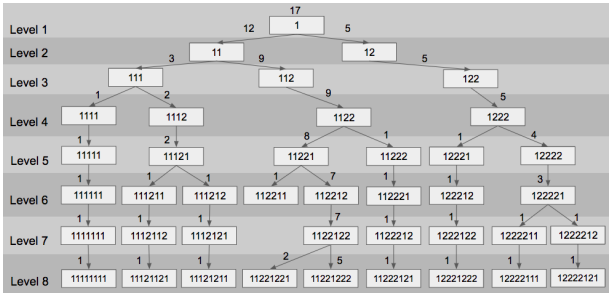

Figure 3: User path distribution for level set 'A'. Each layer of the tree corresponds to a level. The string of numbers inside each block represents the choice of paths made by users up to and including the current level. The numbers above each block represents the number of users who chose that sequence of paths.

Data Analysis

Upon collection, data was analyzed qualitatively to gain intuition as to how users made decisions. These insights would lead to potential feature representations for fitting our linear model of the Q-function. To make discussion of some of the results more clear, flowcharts depicting path selection amongst users are provided for level sets 'A' and 'B', shown in Figures 3 and 5 respectively. These charts show how many users made which sequence of path selections on the corresponding version of the game.

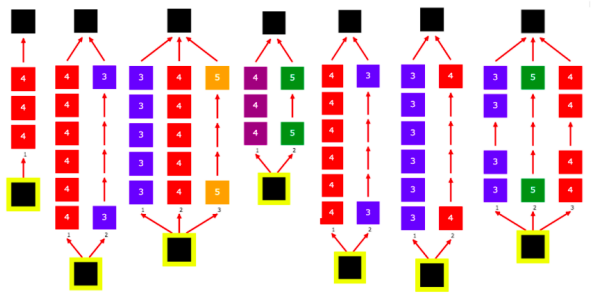

Figure 4: Game sequence for level set 'B'. Levels progress from left to right.

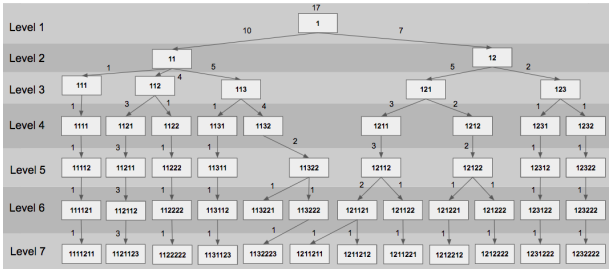

Figure 5: User path distribution for level set 'B'. Each layer of the tree corresponds to a level. The string of numbers inside each block represents the choice of paths made by users up to and including the current level. The numbers above each block represents the number of users who chose that sequence of paths.

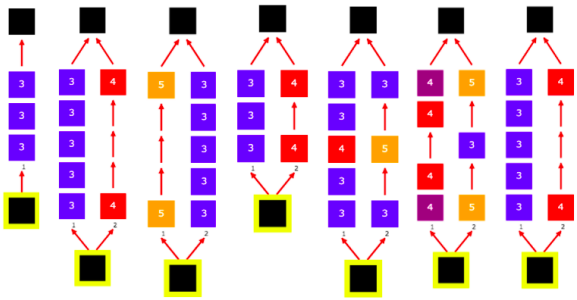

Figure 6: Game sequence for level set 'C'. Levels progress from left to right.

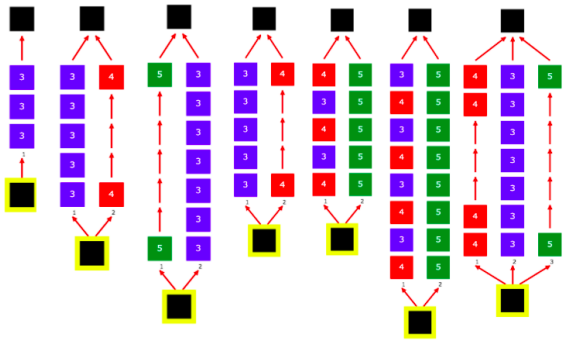

Figure 7: Game sequence for level set 'D'. Levels progress from left to right.

One key aspect of the results that needed checking was to what extent users were able to remember previously executed skills. Before analyzing the data, I predicted most users would have a hard time remembering old skills after learning new skills. This behavior was indeed demonstrated throughout the data. One example of this is in level 6 of level set 'B'. In this level, assuming both red and blue skills are known, taking path 2 (red) saves 8 units of life. However, although all users began by learning the red skill on level 1, 1/3 of all users chose to take the blue path on level 6. Numerous other examples of this can be seen by examining the flowcharts in further detail.

The second observation of note is, as would be expected, the users did not always act according to the greedy policy (i.e. maximize reward on current task only). Examples of this can be seen in level set 'A' where 30% of users took the disadvantageous red path on level 2, or in level set 'B' where 41% of users took the disadvantageous orange path on level 3. More concretely, as seen in Table 1, the average human score on the final level exceeds the expected greedy score on level set 'A', but is worse on level sets 'B', 'C' and 'D'. This is to be expected, as the progression of the level sets 'B', 'C', and 'D' favor a greedy policy, where as set 'A' favors skill exploration. Nonetheless, it is interesting noting that humans do not necessarily take the greedy policy when available. It is important to take this comparison with a grain of salt, a the greedy policy considered here is expected to act optimally on each task, remembering perfectly all previous tasks executed. This of course fails to account for the inevitable errors that humans make, or the possibly limited memory resources available to an agent.

Table 1: Comparing average final life remaining (LR) of human and greedy policies.

Level Set	Average Human LR	Expected Greedy LR
1	72.21	67.5
2	52.71	84
3	57.4	71
4	51.75	60

These basic, qualitative observations demonstrate that, although potentially not aware of it, the users were updating their set of skills as they learned new ones, as if constrained by finite memory resources. Satisfied with this knowledge, I generated feature representations described below to try and capture how this process was happening.

Model Fitting

In order to fit a linear model to approximate the Q-function, a suitable feature representation needed to be extracted from the raw data entries. Since path selection occurred on each level, user data was split according to level, where features attempted to capture user history and comparison to alternate available paths. The features which were calculated are presented below:

Features

- Remaining life (RL): The remaining life before beginning this path.
- Path variance (PV): The variance in the cost to execute the path given current skills. Variance of the cost of learning an unknown skill is $5/24K(K+1)(2K+1)$ where K is the length of the skill's code.
- Path standard deviation (PSD): The standard deviation in the cost to execute the path given current skills.
- Known path length (KPL): The length of executing this path assuming all skills are known.
- Unknown path variance (UPV): The variance in the cost to execute the path assuming no skills are known.
- Unknown path standard deviation (UPSD): The standard deviation in the cost to execute the path assuming no skills are known.
- Unknown path length (UPL): The expected length of executing this path assuming no skills are known. Expected cost of learning an unknown skill is $K(1+3/4(K+1))$ where K is the length of the skill's code.
- Expected cost disadvantage (ECD): The expected immediate disadvantage of taking this path compared to the optimal greedy path.
- Expected cost disadvantage with hindsight (ECDH): The expected immediate disadvantage of taking this path compared to the optimal greedy path, after subtracting 'hindsight costs' from expected cost to execute this path. 'Hindsight costs' refer to all previous cost that could have been saved if the unknown skills in this path were known at that time.
- Ideal cost disadvantage (ICD): The expected immediate disadvantage of taking this path compared to the optimal greedy path, IF this path could be executed with all skills known.
- Complexity (C): The sum of the empirical entropy of all unique sequences in the path. For example, the code 'left, left, left' has a lower empirical entropy (0) than 'left, right, up' ($log_2(3)$).
- Complexity disadvantage (CD): The difference in complexity between this path and the path with lowest complexity.

Linear Model Selection

Two aspects of the model had to be chosen. Aside from choosing feature representations, it was necessary to model the knowledge transition function, or in other words, which skills were remembered for how long. This was important to model since many of the feature representations depended on which skills were known and which were unknown when entering a path. For example, when calculating the expected cost of a path, the first instance of all unknown skills in that path contribute cost equal to the expected cost of learning that skill, and then all subsequent instances of those skills, as well as all instances of known skills, contribute cost equal to their code length. Since it is hard to pinpoint exactly when users forgot a skill and what happened in the events leading up to that point, I chose to model the skill knowledge base of each user as a queue of size D (I avoid using Q here to avoid confusion with the Q-function). After execution of each high-level task, all distinctly colored skill-blocks from that path would be pushed into the queue. If more than D skills existed in the queue, the oldest skills in the queue were pushed out. This model captures the leaky nature of memory. Note that since in this game there were 5 distinct skills, setting D to 5 was equivalent to modeling perfect memory. To evaluate the different values of D, a linear predictor of whether or not a path was executed was fit using all 13 of the above features for values of D ranging from 1 to 5. Results can be seen in Table 2

Table 2: Mean square error of path prediction for different size 'skill queues'. Although changes are relatively minute, a skill memory of 1 leads to the lowest modeling error.

Queue Size	MSE
1	0.20190
2	0.20591
3	0.20307
4	0.20269
5	0.20269

Using a queue of size 1 produced the lowest modeling error for this set of data. Using this value, I computed the mean square error using various combinations of the features listed above. Results of this comparison are shown in Table 3, where features are referred to using the abbreviations provided next to their full name and description in the list above. Unsurprisingly, using all available features resulted in the lowest mean squared error. However, results that I found interesting included how incorporating hindsight cost significantly increased error. My prediction was that as hindsight error accumulated, users would be more inclined to skill exploration. I was also surprised by the fact that using only the expected cost disadvantage feature (with a bias) was able to achieve a MSE so close to the minimum given by all features.

This, along with the relatively large MSE in general, it is

evident that the features outlined here do not provide much more advantage over using a simple greedy approach in terms of modeling human behavior. Thus, work remains to determine feature representations that better capture human behavior.

It is also important to consider potential causes of error in this model. The obvious ones are that this experiment as set up does not accurately capture human intent in the problem outlined, or that humans are simply bad at this task.

Table 3: Mean square error of path prediction for different combinations of features, using a skill queue size of 1. Unsurprisingly, using all the features results in the lowest modeling error.

Feature Representation	MSE
RL, ECD, CD	0.20662539
RL, ECDH, CD	0.24260309
RL, ECD, ICD, CD	0.20703982
RL, PV, EPL, ECD, ICD, CD	0.20638206
RL, C, PSD, EPL, UPL, KPL	0.22074289
ECD	0.20726595
ECDH	0.24689894
ALL	0.20190364

Conclusion

In this project, I presented a simple version of the skill maintenance problem as a meta-level MDP. I created an interactive web game representing said MDP which allowed me to track human performance. Using the user data, I created a set of features of the MDP state and history, and by assuming human actions were optimal, fit a linear policy. By analyzing the change in mean-square-error for various versions of the linear model, I was able to reach the boring conclusion that the feature representations I used did not capture the motivation for deviating from the greedy policy.

I find this conclusion to be quite unsatisfying. However, now that the overhead of designing and implementing the web experiment is done, I hope to continue to flesh out this work. The main things I wish to improve upon are creating a more general formulation of the skill maintenance problem, the design of better levels which capture more interesting behavior, the collection of much more data, and most importantly, the investigation into better representations of past experience. In addition, I plan to publish the code for the web game as well as the data collection scripts so that anyone can use the game for experiments of their own.

References

Daniel, C., vanHoof, H., Peters, J., Neumann, G. (2016). Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, *104*(2), 337–357. Retrieved from `http://dx.doi.org/10.1007/s10994-016-5580-x` doi: 10.1007/s10994-016-5580-x

Kazemitabar, S. J., Beigy, H. (2008). Automatic discovery of subgoals in reinforcement learning using strongly connected components. In *International conference on neural information processing* (pp. 829–834).

Levine, S., Finn, C., Darrell, T., Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, *17*(39), 1–40.

Lieder, F., Plunkett, D., Hamrick, J. B., Russell, S. J., Hay, N., Griffiths, T. (2014). Algorithm selection by rational metareasoning as a model of human strategy selection. In *Advances in neural information processing systems* (pp. 2870–2878).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Solway, A., Diuk, C., Córdova, N., Yee, D., Barto, A. G., Niv, Y., Botvinick, M. M. (2014). Optimal behavioral hierarchy. *PLOS Comput Biol*, *10*(8), e1003779.

Sutton, R. S., Precup, D., Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1), 181–211.

Thrun, S., Schwartz, A., et al. (1995). Finding structure in reinforcement learning. *Advances in neural information processing systems*, 385–392.