Iterative Policy-Space Expansion in Reinforcement Learning

Jan M. Lichtenberg
Department of Computer Science
University of Bath
Bath, United Kingdom
j.m.lichtenberg@bath.ac.uk

Özgür Şimşek
Department of Computer Science
University of Bath
Bath, United Kingdom
o.simsek@bath.ac.uk

Abstract

Humans and animals solve a difficult problem much more easily when they are presented with a sequence of problems that starts simple and slowly increases in difficulty. We explore this idea in the context of reinforcement learning. Rather than providing the agent with an externally provided curriculum of progressively more difficult tasks, the agent solves a single task utilizing a decreasingly constrained policy space. The algorithm we propose first learns to categorize features into positive and negative before gradually learning a more refined policy. Experimental results in Tetris demonstrate superior learning rate of our approach when compared to existing algorithms.

1 Introduction

In 1772, Benjamin Franklin received a plea for advice on a difficult career decision from his friend and fellow scientist Joseph Priestley [1]. In his reply, he described what later became known as *Franklin's rule* [2]:

> "My way is to divide half a sheet of paper by a line into two columns, writing over the one *Pro*, and over the other *Con*. [...] I put down under the different heads short hints of the different motives that at different times occur to me for or against the measure. When I have thus got them all together in one view, I endeavor to estimate their respective weights [...] If I judge some two reasons con equal to some three reasons pro, I strike out the five; and thus proceeding I find at length where the balance lies [and] come to a determination accordingly." (p. 878 in [1])

Almost 250 years later, pros-and-cons lists are still used extensively. What makes such a simple tool so effective? One aspect could be that the main problem of estimating the relative importance of arguments is facilitated by first solving the much simpler subproblem of deciding—for each feature individually—whether it is positively or negatively associated with the response.

We present a reinforcement learning algorithm that is similar to Franklin's rule in that the algorithm first learns, for each feature individually, whether it is positively or negatively associated with good decision outcomes. Building on these so-called *feature directions*, the algorithm then gradually learns a more refined policy.

The idea that a sequence of progressively more difficult tasks could accelerate learning has been exploited in animal training where it is called *shaping* [3–6]. Previous research has raised the question of whether learning machines could benefit from similar ideas. In robotics, learned dynamics from regions of easy solvability are reused in more difficult regions of the task environment [7].

Workshop on Biological and Artificial Reinforcement Learning at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

In *curriculum learning* [6, 8], neural networks are trained with progressively more noisy and less relevant training data. However, finding a good curriculum is a difficult problem and solutions are often task-specific (but see [9]).

The proposed algorithm does *not* require an external teacher who guides the learning agent with a carefully tailored curriculum of tasks with increasing difficulty. The task difficulty is instead regulated intrinsically along the following two dimensions. First, the agent initially learns weights naïvely (as in *naïve Bayes*), that is, without considering interdependencies among features. Eventually, weights are estimated jointly. Second, the agent learns in a decreasingly constrained policy space, which is consistent with the view of the cognitive mechanism of humans and great apes that initially has very low capacity but grows during development [5, 10, 11].

2 Background: Classification-based reinforcement learning with rollouts.

Preliminaries and Notation. We consider a discounted Markov decision process (MDP), defined by $(\mathcal{S}, \mathcal{A}, \mathcal{G}, r, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{G}(s,a) : \mathcal{S} \times \mathcal{A}(s) \rightarrow \mathcal{S} \times \mathbb{R}$ is a generative model of the environment used to sample a new state s' and reward r for a given state-action pair (s,a), $r : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in (0,1)$ is the discount factor. $\mathcal{A}(s)$ denotes the set of actions available in state s. The goal of reinforcement learning is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected cumulative reward of the agent. This article is concerned with learning linear policies of the form $\pi(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}}\, \boldsymbol{\beta}^T \boldsymbol{\phi}(s,a)$, where $\boldsymbol{\phi}(s,a) \in \mathbb{R}^p$ denote feature values that correspond to selecting action a in state s and $\boldsymbol{\beta} \in \mathbb{R}^p$ denotes the vector of feature weights to be estimated. Note that only relative differences between weights matter because the policy remains unchanged when all policy weights are multiplied with the same positive scalar.

Policy iteration [12, 13] is a classic dynamic programming method that generates a sequence of monotonically improving policies $\pi_0, \ldots, \pi_k$, by alternating between two steps: estimating the value function of the current policy (*policy evaluation*) and computing a new improved policy based on the current value function (*policy improvement*). Large MDPs require the use of function approximation of policy and value function. The resulting algorithm is called approximate policy iteration (API).

Classification-based reinforcement learning with rollouts. Our work builds on a range of approximate policy iteration algorithms that cast the policy-improvement step as a classification problem [14–19]. A training instance of the classification data set is generated as follows, using *rollouts*. For a given state s, the value of an available action a is approximated by the cumulative sum of rewards obtained in a finite-length forward simulation of the environment, choosing action a in state s and following the current policy thereafter. The action that yields the highest cumulative reward (averaged across multiple rollouts for each action) becomes the class label for state s. A policy is trained to assign the correct class label to each state in the training set. The new policy is then used in the subsequent rollout. In each iteration, the rollout starting states are sampled from a large, pre-computed *rollout set*, which sometimes is generated by an existing expert policy [18].

Initial versions of classification-based API algorithms worked well in problems such as learning to balance a bicycle [14] or in planning domains [15]. *Classification-based modified policy iteration* (CBMPI, [18, 20]) was the first RL algorithm to achieve good results in the challenging domain of Tetris. CBMPI approximates both a policy and a state-value function. The state-value function improves the accuracy of rollout estimates but its estimation requires large training sets.

M-learning [19] does not use a pre-computed rollout set. Instead, rollouts are computed exclusively for the current state of the environment, meaning that only one training instance is added to the classification data set in every iteration. In earlier work [19], M-learning was given prior knowledge about feature directions, which helped to compensate for the smaller training data available to the classifier. The direction of a feature is the sign of the corresponding weight. This prior knowledge was used in two ways. First, feature directions were used to identify and filter out dominated [21] actions during rollouts. Second, multinomial logistic regression, which plays a central role in M-learning, was regularized using *shrinkage toward equal weights* (STEW) [19] regularization. The STEW penalty shrinks weights toward each other, resulting in an equal-weighting model [2, 22–24] in the limit of infinite regularization. Previous research has shown the surprising effectiveness of equal-weighting models when feature directions were known in advance [2, 24–26]. Using prior knowledge of feature

directions, M-learning was shown to learn strong Tetris policies, while using considerably fewer training samples than CBMPI [19].

When knowledge about feature directions is not available, the signs of feature weights are usually estimated implicitly as part of the general estimation procedure. We report results in Section 5 that show that the absence of this prior knowledge leads to considerably slower learning performance in M-learning. The magnitude of this effect surprised us, given that the isolated estimation of feature directions in supervised learning is a relatively easy task, as supported by experimental [27, 28] and theoretical [29] evidence. Motivated by this discrepancy, we decouple the estimation of directions from the remaining estimation procedure and explore a hierarchical approach that learns feature directions first, and then builds on these directions to learn weight magnitudes. This requires an algorithm to learn feature directions.

3 Learning feature directions (LFD) in reinforcement learning

We present a reinforcement learning algorithm, named LFD, that learns feature directions. The feature directions $d_i \in \{-1, 0, 1\},\ i = 1, \dots p$ are initialized to zero; they are said to be *undecided*. The agent navigates the environment using a rollout mechanism to select actions and keeps track of how often each feature is associated positively and negatively with selected actions. A feature is assigned a direction when the difference between positive and negative associations is deemed to be significant. The algorithm terminates when all feature directions have been decided. Pseudo-code for LFD is provided in Algorithm 1 in the Appendix. A more detailed description of the algorithm follows next.

Let $\tilde{a}$ denote the action chosen by the rollout procedure and let $\boldsymbol{\phi}(s, a_1), ..., \boldsymbol{\phi}(s, a_{|\mathcal{A}(s)|})$ denote the feature values of all actions available in state s. Furthermore, let sgn denote the mathematical sign function: $\mathrm{sgn}(x)$ is 1 if $x > 0$, 0 if $x = 0$, and -1 if $x < 0$. A training instance Δ_i for feature ϕ_i compares the feature values of the selected action $\tilde{a}$ to the feature values of all other actions. A training instance can be positive or negative and is defined as $\Delta_i = \mathrm{sgn}\left(\sum_{a \neq \tilde{a}} \mathrm{sgn}\left(\phi_i(s, \tilde{a}) - \phi_i(s, a)\right)\right)$. For example, a positive training instance means that feature ϕ_i was larger for the chosen action $\tilde{a}$ than for other actions more often than it was smaller.

Let n_i^+ denote the number of positive training instances and let n_i^- denote the number of negative training instances. A feature is assigned a direction only after the difference between n_i^+ and n_i^- is found to be statistically significant. We use a two-sided exact binomial test (for example, [30]) with null hypothesis that feature ϕ_i has no direction, that is, $H_0 \colon n_i^+/(n_i^+ + n_i^-) = 0.5$. If the resulting p-value is smaller than some pre-defined threshold α, the feature is assigned the direction $d_i = \mathrm{sgn}(n_i^+ - n_i^-)$.

The rollout policy utilizes features for which a direction has already been determined, while ignoring features with undecided directions. It is defined as $\pi_r(s) = \underset{a \in \mathcal{A}(s)}{\mathrm{argmax}}\ \boldsymbol{d}^T \boldsymbol{\phi}(s, a)$, where $\boldsymbol{d} = (d_1, \dots, d_p)$ is the vector of current directions. Ties are broken at random.

4 Iterative policy-space expansion (IPSE)

By combining LFD and M-learning with STEW regularization, we create a reinforcement learning algorithm that decouples the estimation of weight signs from the estimation of weight magnitudes. LFD is employed first, until all feature directions are learned. The algorithm then switches to M-learning, treating the learned directions as useful prior knowledge.

Under conditions defined further below, the combined algorithm learns in a monotonically expanding policy space. We call this algorithm *iterative policy-space expansion*, or IPSE. Note that building blocks other than LFD or M-learning could be used to create algorithms that learn in monotonically expanding policy-spaces (discussed in Section 6). To simplify notation, we will use the acronym IPSE to refer to the version that uses LFD and M-learning with STEW penalty in the remainder of this article.

The policy space of IPSE. We derive necessary conditions on the regularization strength of the STEW penalty (controlled by the parameter λ) that ensure that IPSE learns in monotonically expanding

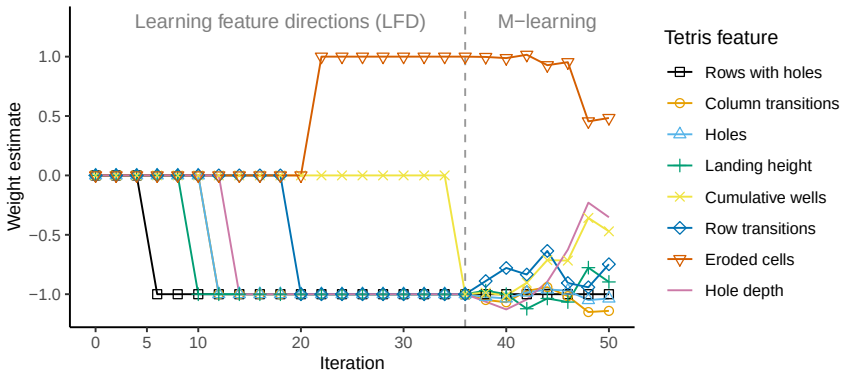

Figure 1: Policy weight trajectories of the IPSE algorithm in Tetris. The dashed vertical line signifies the transition from the LFD algorithm to M-learning with STEW penalty. Weights were rescaled such that the weight *rows with holes* always has an absolute value of 1.

policy spaces. The policy space at any given iteration is characterized by the values that the policy weight vector $\boldsymbol{\beta}$ can attain. Initially, IPSE learns directions using the LFD algorithm; the policy space is therefore constrained to be $\Pi_{\text{LFD}} = \{-1, 1\}^p$. During the M-learning phase, the policy space is a function of the regularization strength $\lambda > 0$. STEW-regularized multinomial logistic regression can be reformulated as a constrained optimization problem (similar to, for example, [31]) to see that the policy space is given by $\Pi_\lambda = \{\boldsymbol{\beta} \in \mathbb{R}^p | \sum_{i=1}^p (\beta_i - d_i)^2 \leq c(\lambda)\}$, where d_i are the directions estimated by LFD in the first phase of the algorithm, and $c(\lambda) \colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is a decreasing function of λ, for which the following holds true: $c(\lambda) \to 0$ for $\lambda \to \infty$; and $c(\lambda) \to 0$ for $\lambda \to \infty$. The policy space therefore is a hypersphere around the equal-weights solution that was found by the LFD algorithm. The size of that hypersphere is a decreasing function of the regularization strength λ. Let $\{\lambda_k\}_{k=1}^\infty$ denote a sequence of decreasing regularization strengths. It then follows that $\Pi_{\text{LFD}} \subset \Pi_{\lambda_k} \subset \Pi_{\lambda_{k+1}} \subset \mathbb{R}^p$, or in other words, the policy space is monotonically expanding.

Choice of λ. In practice (for example, in earlier work on M-learning [19]), the regularization strength is often chosen using cross validation. Here, we use a pre-defined schedule of decreasing regularization strengths in order to ensure a monotonically expanding policy space. We aim to find a schedule that satisfies the following two properties. First, the regularization strength should initially be high enough to ensure a smooth transition from LFD to M-learning. Second, the regularization strength should decrease rapidly enough so that the policy space is not overly constrained for too long. Both these properties are satisfied in the following example.

Example weight trajectories. Figure 1 shows policy weight trajectories of the IPSE algorithm obtained while learning to play Tetris (see Section 5 for a detailed description of the experimental setup). IPSE used $\lambda_k = 5/k$ in the k-th iteration of M-learning. In the iterations directly following the transition to M-learning (iteration 36 in this particular example), the estimated weights remained relatively close to the equal-weighting solution. Policy weight estimates then increasingly deviated from the equal-weights solution as the policy space expanded.

5 Experiments

We next present results from our experiments in Tetris. We used an experimental setup similar to the one that was used to demonstrate the fast learning rate of M-learning [19], with the important difference that in our experiments, feature directions (that is, weight signs) were *not* given in advance. Our primary objective is to examine whether IPSE benefits from learning weight signs and magnitudes sequentially rather than jointly, as is done by competing algorithms.

Tetris. Tetris can be formulated as a MDP, where the state consists of the board configuration and the identity of the falling tetrimino. Available actions are the possible placements of the tetrimino on the board. A reward of 1 is received for each cleared line. The game ends when a state allows no legal placement. We used a board size of 10×10 in all experiments. Eight features were used to describe a state-action pair: *landing height*, *number of eroded piece cells*, *row transitions*, *column transitions*,

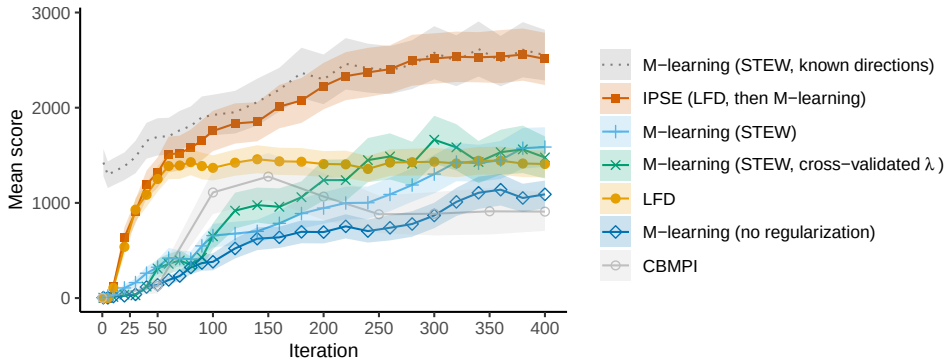

Figure 2: Quality of the policy learned as a function of the iterations of the algorithm. Each learning curve shows means across 100 replications of the algorithm. Quality of the policy is measured by the mean score obtained by the policy in 30 Tetris games.

number of holes, *number of board wells*, *hole depth*, and *number of rows with holes*. These features are from earlier work by Thiery and Scherrer [32], who describe them in detail.

Algorithms. We compared IPSE to the LFD algorithm, CBMPI, and four versions of M-learning. The M-learning versions differed in regularization behavior and prior knowledge available. One version did not make use of regularization at all. The other three versions used STEW regularization. Among the three regularizing versions, one used cross-validation to estimate λ (as in [19]), while the other two used a schedule as described in the previous section. Among the latter two versions, one was given knowledge about feature directions obtained from the weights of the BCTS policy [32].

All M-learning versions, IPSE, and LFD used the same rollout parameters. These algorithms computed $M = 10$ rollouts of length $T = 10$ for each action (compare to Algorithm 2 in the Appendix). Given that the number of actions is always smaller than 34, the maximum number of calls to the generative model of Tetris for one iteration of the algorithm was at most $34TM = 3400$. We used a per-iteration budget of 170,000 calls for CBMPI. The Appendix contains further implementation details.

Results. Learning curves are shown in Figure 2. M-learning with given feature directions represents an upper baseline. This is the dotted line in the figure. Among algorithms that were not given prior knowledge about feature directions, IPSE showed the highest learning rate and learned the best policies overall. Furthermore, it rapidly approached the ceiling performance obtained with known feature directions. All other algorithms learned more slowly. At 400 iterations, there was a large performance gap between IPSE and all other algorithms.

The hypothesis that IPSE benefits from learning directions independently was supported by the strong performance of the standalone LFD algorithm at the beginning of the learning curve. This indicates that IPSE could fruitfully use the naïve direction estimates as a stable basis for later learning.

6 Discussion and future work

Our experimental results show that reinforcement learning algorithms can benefit from learning in a policy space that initially is strongly constrained but expands during the learning process. Similar to how people structure their thoughts by categorizing arguments into *pro* and *contra*, learning feature directions has proven to be a useful building block for learning more complex policies.

An interesting direction for future work is to extend the approach presented in this paper to reinforcement learning with non-linear function approximators such as neural networks. These algorithms estimate a much larger number of feature weights, which requires more data, and thus makes a hierarchical approach potentially even more promising. However, it is unclear whether the notion of feature directions is useful for certain network architectures such as convolutional neural networks.

References

[1] Benjamin Franklin. *Writings*. Library of America, 1987.

[2] Gerd Gigerenzer, Peter M. Todd, and the ABC Research Group. *Simple heuristics that make us smart*. Oxford University Press, USA, 1999.

[3] Burrhus F. Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958.

[4] Gail B. Peterson. A day of great illumination: BF Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior*, 82(3):317–328, 2004.

[5] Kai A. Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3): 380–394, 2009.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48, 2009.

[7] Terence D. Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Transactions on Robotics and Automation*, 10(3):323–333, 1994.

[8] Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[9] Alex Graves, Marc G. Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1311–1320, 2017.

[10] Stacey L. Brown, Joby Joseph, and Mark Stopfer. Encoding a temporally structured stimulus with a temporally structured neural representation. *Nature Neuroscience*, 8(11):1568, 2005.

[11] Earl K. Miller and Jonathan D. Cohen. An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience*, 24(1):167–202, 2001.

[12] Ronald A. Howard. *Dynamic programming and markov processes.* MIT Press, Cambridge, MA, 1960.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[14] Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning*, pages 424–431, 2003.

[15] Alan Fern, SungWook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems 16*, pages 847–854, 2004.

[16] Lihong Li, Vadim Bulitko, and Russell Greiner. Focus of attention in reinforcement learning. *Journal of Universal Computer Science*, 13(9):1246–1269, 2007.

[17] Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Analysis of classification-based policy iteration algorithms. *Journal of Machine Learning Research*, 17(1):583–612, 2016.

[18] Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of Tetris. *Journal of Machine Learning Research*, 16:1629–1676, 2015.

[19] Jan M. Lichtenberg and Özgür Şimşek. Regularization in directable environments with application to Tetris. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3953–3962, 2019.

[20] Victor Gabillon, Mohammad Ghavamzadeh, and Bruno Scherrer. Approximate dynamic programming finally performs well in the game of Tetris. In *Advances in Neural Information Processing Systems 26*, pages 1754–1762, 2013.

[21] Özgür Şimşek, Simón Algorta, and Amit Kothiyal. Why most decisions are easy in Tetris—And perhaps in other sequential decision problems, as well. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1757–1765, 2016.

[22] Robyn M. Dawes and Bernard Corrigan. Linear models in decision making. *Psychological Bulletin*, 81(2): 95–106, 1974.

[23] Hillel J. Einhorn and Robin M. Hogarth. Unit weighting schemes for decision making. *Organizational Behavior and Human Performance*, 13(2):171–192, 1975.

[24] Victor DeMiguel, Lorenzo Garlappi, and Raman Uppal. Optimal versus naive diversification: How inefficient is the 1/N portfolio strategy? *Review of Financial Studies*, 22(5):1915–1953, 2009.

[25] Howard Wainer. Estimating coefficients in linear models: It don't make no nevermind. *Psychological Bulletin*, 83(2):213–217, 1976.

[26] Jan M. Lichtenberg and Özgür Şimşek. Simple regression models. In *Imperfect Decision Makers: Admitting Real-World Rationality*, pages 13–25, 2017.

[27] Konstantinos V. Katsikopoulos, Lael J. Schooler, and Ralph Hertwig. The robust beauty of ordinary information. *Psychological Review*, 117(4):1259, 2010.

[28] Jason Dana and Robyn M. Dawes. The superiority of simple alternatives to regression for social science predictions. *Journal of Educational and Behavioral Statistics*, 29(3):317–331, 2004.

[29] Özgür Şimşek and Marcus Buckmann. Learning from small samples: An analysis of simple decision heuristics. In *Advances in Neural Information Processing Systems 28*, pages 3141–3149, 2015.

[30] David C. Howell. *Statistical methods for psychology*. Cengage Learning, 2009.

[31] Wessel N. van Wieringen. Lecture notes on ridge regression. *arXiv preprint arXiv:1509.09169*, 2015.

[32] Christophe Thiery and Bruno Scherrer. Building controllers for Tetris. *Icga Journal*, 32(1):3–11, 2009.

[33] Simón Algorta and Özgür Şimşek. The game of Tetris in machine learning. *arXiv preprint arXiv:1905.01652*, 2019.

Appendix

A Additional implementation details

An overview of machine learning solutions to Tetris can be found in Algorta and Şimşek [33]. We used implementations of Tetris and the M-learning algorithm from Lichtenberg and Şimşek [19].

M-learning. Multinomial logistic regression in iteration k used the most recent $n(k)$ training samples, where $n(k) = min(100, \lfloor \frac{k}{2} \rfloor + 2)$.

LFD. We used alternative rollout policy that uses the rollout policy $\pi_r(s)$ (described in Section 3) unless an immediate reward greater than zero for at least one action is possible, in which case the action that promises the highest reward is selected. We found this alternative rollout policy to have a positive effect on the learning rate in all of our experiments.

CBMPI. The CBMPI results reported by Scherrer et al. [18] used a per-iteration budget of 8,000,000 calls to the generative model of Tetris. In comparison, the total budget (after 400 iterations) we used for the other algorithms was 1,360,000. In order to compare the algorithms meaningfully, we experimented with CBMPI using budgets in the same range as M-learning. We present results with CBMPI using a per-iteration budget of 170,000.

B Pseudo-code

Algorithm 1 Learning feature directions (LFD)

Output:
$\boldsymbol{d} \in \{-1, 0, 1\}^p$ *// feature directions, initialized to* $\mathbf{0}$
Input:
$\alpha \in (0, 1)$ *// significance threshold*
$\pi_r(s, \boldsymbol{d}) : \mathcal{S} \times \{-1, 0, 1\}^p \to \mathcal{A}$ *// rollout policy using current feature directions*

$s \leftarrow$ state sampled from initial state distribution
while not all directions are learned **do**
  for all $a \in \mathcal{A}(s)$ **do**
    $\hat{U}(s, a) \leftarrow \textsc{Rollout}(s, a, \pi_r(s, \boldsymbol{d}))$
  end for
  $\tilde{a} \leftarrow \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \hat{U}(s, a)$
  Take action $\tilde{a}$ and observe new state s'
  if s' is not terminal **then**
    for all $i = 1, \ldots, p$ **do**
      $\Delta_i = \operatorname{sgn}\left(\sum_{a \neq \tilde{a}} \operatorname{sgn}\left(\phi_i(s, \tilde{a}) - \phi_i(s, a)\right)\right)$
      $n_i^+ \leftarrow n_i^+ + \max(\Delta_i, 0)$
      $n_i^- \leftarrow n_i^- - \min(\Delta_i, 0)$
      p-val $\leftarrow$ test $H_0 \colon n_i^+ / (n_i^+ + n_i^-) = 0.5$
      if *p-val* $< \alpha$ **then**
        $d_i \leftarrow \begin{cases} 1 & \text{if } n_i^+ > n_i^- \\ -1 & \text{otherwise} \end{cases}$
      end if
    end for
    $s \leftarrow s'$
  else
    // reset episode
    $s \leftarrow$ state sampled from initial state distribution
  end if
end while

Algorithm 2 $\textsc{Rollout}(s, a, \pi_r)$

Output:
$\hat{U} \in \mathbb{R}$, *estimated value of taking action* a *in* s
Input:
$s \in \mathcal{S}$ *// rollout starting state*
$a \in \mathcal{A}(s)$ *// action to be evaluated*
$\pi_r(s) : \mathcal{S} \to \mathcal{A}$ *// rollout policy*
$M \in \mathbb{N}$ *// number of rollouts*
$T \in \mathbb{N}$ *// rollout length*
$\gamma \in [0, 1]$ *// discount factor*
$\mathcal{G}(s, a) : \mathcal{S} \times \mathcal{A}(s) \to \mathcal{S} \times \mathbb{R}$ *// generative model*

for all $j = 1, \ldots, M$ **do**
  $(s', r) \leftarrow \mathcal{G}(s, a)$
  $\hat{U}_j \leftarrow r$
  $s \leftarrow s'$
  for all $t = 1, \ldots, T - 1$ **do**
    $(s', r) \leftarrow \mathcal{G}(s, \pi_r(s))$
    $\hat{U}_j \leftarrow \hat{U}_j + \gamma^t r$
    $s \leftarrow s'$
  end for
end for
return $\hat{U} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \hat{U}_j$

Algorithm 3 Online reinforcement learning with rollouts (general form)

Input:
Π *// policy space*
Ω *// space of data sets produced by a rollout procedure*
LEARN: $\Omega \to \Pi$, where *// learning procedure*
$\mathcal{D} = \emptyset$ *// data structure to store choice data*
Output:
$\pi \in \Pi$ *// policy, initialized to uniform random policy*

$s \leftarrow$ state sampled from initial state distribution
while not all directions are learned **do**
  for all $a \in \mathcal{A}(s)$ **do**
    $\hat{U}(s, a) \leftarrow$ ROLLOUT(s, a, π)
  end for
  $\tilde{a} \leftarrow \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \hat{U}(s, a)$
  Take action $\tilde{a}$ and observe new state s'
  if s' is not terminal **then**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\{\tilde{a}, \boldsymbol{\phi}(s, a_1), \boldsymbol{\phi}(s, a_2), ..., \boldsymbol{\phi}(s, a_{|\mathcal{A}(s)|})\}\}$ *// append choice set to $\mathcal{D}$*
    $\pi \leftarrow$ LEARN$(\mathcal{D})$
    $s \leftarrow s'$
  else
    $s \leftarrow$ state sampled from initial state distribution *// reset episode*
  end if
end while