Feature Reinforcement Learning : State of the Art

Mayank Daswani and **Peter Sunehag** and **Marcus Hutter**
Research School of Computer Science
Australian National University,
Canberra, ACT, 0200, Australia.,
first.last@anu.edu.adu

Abstract

Feature reinforcement learning was introduced five years ago as a principled and practical approach to history-based learning. This paper examines the progress since its inception. We now have both model-based and model-free cost functions, most recently extended to the function approximation setting. Our current work is geared towards playing ATARI games using imitation learning, where we use Feature RL as a feature selection method for high-dimensional domains.

This paper is a brief summary of the progress so far in the Feature Reinforcement Learning framework (FRL) (Hutter 2009a), along with a small section on current research. FRL focuses on the general reinforcement learning problem where an agent interacts with an environment in cycles of action, observation-reward. The goal of the agent is to maximise an aggregation of the reward. The most traditional form of this general problem constrains the observations (and rewards) to be states which satisfy the Markov property, i.e. $P(o_t|o_{1:t-1}) = P(o_t|o_{t-1})$ and is called a Markov Decision Process (MDP) (Puterman 1994). A less constrained form is Partially Observable Markov Decision Processes (Kaelbling, Littman, and Cassandra 1998) when these observations are generated from some unobservable underlying Markov Decision Process.

Feature Reinforcement Learning (Hutter 2009a) is one way of dealing with the general RL problem, by reducing it to an MDP. It aims to construct a map from the history of an agent, which is its action-observation-reward cycles so far, to an MDP state. Traditional RL methods can then be used on the derived MDP to form a policy (a mapping from these states to actions). FRL fits in the category of a history-based approach. U-tree (McCallum 1996) is a different example of the history-based approach which uses a tree-based representation of the value function where nodes are split based on a local criterion. The cost in FRL is global, maps are accepted or rejected based on an evaluation of the whole map.

While the idea behind FRL is simple, there are several choices to be made. What space do we draw the maps from, and how do we pick the one that fits our data so far? In the best case, we'd like to choose a map ϕ from the space of all possible (computable) functions on histories, but this is intractable in practice and the choice of a smaller hypothesis class can encode useful knowledge and improve learning speed. We define a cost-function that ideally measures how well ϕ maps the process to an MDP. The problem of searching through the map class for the best map ϕ^* is addressed via a stochastic search method.

Taking a step back from the history-based learning problem, we can frame the general RL problem as trying to find a map from a very-high dimensional input space, namely that of all possible histories to a policy representation that allows us to perform well in the given environment. This policy representation is often in the form of a value function but it does not have to be. The model-based feature RL framework (Hutter 2009a; 2009b) tries to build an MDP space first, and then find a value function for that MDP. A model-free approach (Daswani, Sunehag, and Hutter 2013) goes straight for the value function representation without trying to build an MDP model. This approach easily extends to function approximation.

Note that this representation of a general RL problem as a problem in a very-high dimensional input space allows us to use feature RL in the traditional learning setting for feature selection in function approximation problems. Instead of features of the history, our features are now that of the MDP state. The cost function now selects for the smallest subset of features that can represent our model or the value-function. Our current work is on using the value-based cost both in the off-policy and on-policy setting to deal with domains within the scope of the Arcade Learning Environment (Bellemare et al. 2013).

The outline of this paper is as follows. Section 1 outlines some notation and relevant background, Section 2 deals with some related work, Section 3 looks at the Cost functions that have been examined in the FRL setting so far, and summarises the successes of the method. We conclude in Section 4.

Preliminaries

Agent-Environment Framework. The notation and framework is taken from (Hutter 2009a; 2004). An $Agent$ acts in an Environment Env by choosing from actions $a \in \mathcal{A}$. It receives observations $o \in \mathcal{O}$ and real-valued rewards $r \in \mathcal{R}$ where $\mathcal{A}$,$\mathcal{O}$ and $\mathcal{R}$ are all finite. This observation-reward-action sequence happens in cycles indexed by $t = 1,2,3,...$. We use $x_{1:n}$ throughout to represent the sequence $x_1...x_n$. The space of histories is $\mathcal{H} := (\mathcal{O}\times\mathcal{R}\times\mathcal{A})^* \times\mathcal{O}\times\mathcal{R}$. The history at time t is given by $h_t = o_1 r_1 a_1 ... o_{t-1} r_{t-1} a_{t-1} o_t r_t$. The agent (or policy) is then formally a (stochastic) function $Agent : \mathcal{H} \rightsquigarrow \mathcal{A}$ where $Agent(h_t) := a_t$. Similarly, the environment can be viewed as a (stochastic) function of the history, $Env : \mathcal{H}\times\mathcal{A} \rightsquigarrow \mathcal{O}\times\mathcal{R}$, where $Env(h_{t-1}, a_{t-1}) := o_t r_t$.

Markov Decision Process (MDP). If $Pr(o_t r_t | h_{t-1}, a_{t-1}) = Pr(o_t r_t | o_{t-1} a_{t-1})$, the environment is said to be a discrete MDP (Puterman 1994). In this case, the observations form the state space of the MDP. Formally an MDP is a tuple $\langle\mathcal{S},\mathcal{A},\mathcal{T},R\rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions and $R : \mathcal{S}\times\mathcal{A} \rightsquigarrow \mathcal{R}$ is the (possibly stochastic) reward function which gives the (real-valued) reward gained by the agent after taking action a in state s. $\mathcal{T} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [0,1]$ is the *state-transition function*. The agent's goal is to maximise its expected future discounted reward, where a geometric discount function with rate γ is used. In an episodic setting, the discounted sum is truncated at the end of an episode. The value of a state-action pair according to a stationary policy is given by $Q^\pi(s,a) = E^\pi[R_t | s_t = s, a_t = a]$ where $R_t = \sum_{k=0}^{t_{end}} \gamma^k r_{t+k+1}$ is the *return* and t_{end} indicates the end of the episode containing time t. We want to find the optimal action-value function Q^* such that $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, since then the greedy policy $\pi^*(s) = \mathrm{argmax}_a Q^*(s,a)$ is the optimal policy.

Arcade Learning Environment (ALE). The reinforcement learning community has lacked a set of general environments that can be used for testing new algorithms in a robust manner. (Veness et al. 2011) introduced a set of small challenging problems, but several algorithms can no longer be differentiated based on them. The recently introduced ALE (Bellemare et al. 2013) attempts to address this big gap in the field by utilising games made for the ATARI 2600 as a testbed for reinforcement learning algorithms. The environments in this setting are games made for humans which can be relatively complex, but due to the space/processing limits of the ATARI console, still feasible for current RL techniques. The ALE consists of an interface to Stella which is an open-source Atari 2600 games emulator. The interface provides access to the screen pixel matrix and the internal state representation of the ATARI games themselves.

Related Work

MC-AIXI-CTW. The Monte Carlo (MC) AIXI Context Tree Weighting (CTW) algorithm (Veness et al. 2011) is an approximation of the theoretically optimal universal agent AIXI (Hutter 2004). It uses a mixture over all suffix trees with the weights based on their code lengths. It uses the Krichevsky-Trofimov estimator to estimate the probabilities of symbols occurring in each context of the tree and by using properties of CTW (Wilems, Shtarkov, and Tjalkens 1995) it calculates the probability of a history sequence in a computationally efficient way. The (action-conditional) CTW tree maintains a model of the world. A Monte-Carlo Tree Search (MCTS) algorithm UCT (Kocsis and Szepesvári 2006) is used to determine which action to take next, by using the current mixture of suffix trees as a generative model. The drawback is that the policy representation is not explicit, hence we need to call UCT whenever we want to choose an action, which makes MC-AIXI-CTW computationally expensive.

Predictive Representations of State. PSRs are an alternative approach that also attempt to find sufficient statistics from the history to represent some notion of state. A test is a sequence of possible future observations. In a PSR, the dynamics of a system are represented by probability distributions over tests (τ) given histories. In theory, having a complete (infinite) matrix of the probabilities of all tests conditioned on all possible histories would completely define the system we are trying to model. Obviously this is infeasible, but one can aim to find a set of sufficient tests of which all other tests are (linear) projections. Learning these core tests is a difficult problem, but there has recently been much progress, first through transformed PSRs (Rosencrantz, Gordon, and Thrun 2004; Boots, Siddiqi, and Gordon 2010) which are based on spectral-learning algorithms and most recently compressed versions of these TPSRs which use ideas from compressed sensing (Hamilton, Fard, and Pineau 2013).

Cost functions

ΦMDP. Feature RL (Hutter 2009a) is a framework that extracts features from the history that are useful in predicting future consequences of actions. It finds a map $\phi : \mathcal{H} \to \mathcal{S}$ such that the state at any time step $s_t = \phi(h_t)$ is approximately a sufficient statistic of the history. It selects the map ϕ from a class of such maps, minimising a cost function by using a stochastic search method. We will discuss the various cost functions used in the following subsections.

State-reward prediction cost

The first proposed cost was inspired by the minimum description length principle (Rissanen 1978). The cost is the sum of the code lengths of state and reward sequences given actions. This cost is used within a global stochastic search technique (for example simulated annealing (Liu 2008)) to find the optimal map. The standard cost is defined by

$$\mathrm{Cost}(\phi|h_n) := CL(s_{1:n}|a_{1:n}) + CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi)$$

where $CL(s_{1:n}|a_{1:n})$ is the code length of the state sequence given the action sequence, etc (Hutter 2009a). Cost is well-motivated since it balances between coding states and coding rewards. A state space that is too large results in poor learning and a long state coding, while a state space that is too small can obscure structure in the reward sequence resulting in a long code for the rewards. The consistency of this cost criterion was proven by (Sunehag and Hutter 2010). CTΦMDP (Nguyen, Sunehag, and Hutter 2011) uses the above cost with simulated annealing over the space of context trees to find the best map. It then finds the best policy via approximate value iteration.

CTMRL. The Context Tree Maximising (CTM) for Reinforcement Learning (RL) algorithm (Nguyen, Sunehag, and

Hutter 2012) uses the CTM approach to sequence prediction that analytically finds the context-tree model by the minimum description length principle (Rissanen 1978). The sequence prediction setting is adapted for RL by predicting the state-reward sequence conditioned on the actions. For large domains such as Pocman, the CTMRL approach binarises the percept (observation, reward) space, and additionally adds an "unseen" context, whose action value is initialised based on the value of the first subsequent seen state. Due to the high space requirements of the CTM method in such environments the algorithm discards all the context tree maximisers (CTMs) at the start of every learning loop. New CTMs must then be created from only the history gained in the previous loop. This results in a significant loss in data efficiency. Function approximation is a better alternative to deal cleanly with large observation spaces.

Feature Dynamic Bayesian Networks. One can use factored MDPs (or Dynamic Bayesian Networks) to represent the model more compactly (Hutter 2009b). This introduces more complexity to the problem, since we must learn the best structure of the DBN for each ϕ, and then evaluate its cost. For a fixed ϕ the above cost reduces to the Bayesian Information Criterion (BIC). Finding the best structure for a DBN that minimises the BIC is NP-complete. However, introducing additional constraints on the DBN can allow this to be pseudo-polynomial in practice.

Value-based cost : Off-policy TD Cost

The value-based cost was introduced in (Daswani, Sunehag, and Hutter 2013). For each ϕ, a Q-table based on the state space given by ϕ is defined. We denote this Q-table by $Q_\phi : \mathcal{H}\times\mathcal{A}\to\mathbb{R}$ and it is of the form $Q(\phi(h),a)$. We use the squared pathwise Q-learning error to find a suitable map $\phi : \mathcal{H}\to\mathcal{S}$ by selecting ϕ to minimise the following cost,

$$\text{Cost}_{QL}(\phi)=\min_{Q_\phi}\frac{1}{2}\sum_{t=1}^{n}(\Delta_t^{Q_\phi})^2+Reg(\phi)$$

where $\Delta_t^{Q_\phi}=r_{t+1}+\gamma\max_a Q_\phi(h_{t+1},a)-Q_\phi(h_t,a_t)$.

This is similar to the objective function in Regularised Least Squares Fitted Q-iteration (Farahmand et al. 2008), with differences in regularisation.

This off-policy cost is designed to select features that can represent the optimal policy in an iterative manner based on any behaviour policy that might also be changing. It can be extended easily to the function approximation setting by approximating $Q(\phi(h_t),a_t)$ by $\xi(h_t,a_t)^T w$ where $\xi:\mathcal{H}\times\mathcal{A}\to\mathbb{R}^k$ for some $k\in\mathbb{R}$.

Current work : On-policy Monte-Carlo cost, imitation learning. Another scenario is imitation learning, where one has existing trajectories sampled from a good policy and one wants to select features that can accurately represent the value function of that policy. Then one can continue to improve by a traditional algorithm like SARSA or Q-learning.

The imitation problem can be reduced to a supervised learning problem, with the training samples being ($\phi(h_t,a_t)$, return). Our current work involves learning from an expert trajectory in an ATARI 2600 game provided by an MCTS algorithm such as UCT, but trying to find the best possible policy representation given our current feature class. Using only the training samples from the UCT trajectory is not enough to build an estimate of the value function, since a perfect UCT agent does not tell us anything about potential mistakes. We instead force UCT to make suboptimal choices with some probability, which gives us a richer trajectory to learn form.

Doing feature RL in this supervised form allows us to learn a good explicit policy representation that can be used to play the game without access to the game simulator. Ideally we want the best policy representable by our class of features. However, given a perfect regression, the quality of the policy we learn depends on the ability of the feature set to represent the UCT policy. Currently we use ϵ-SVMs with LIBSVM (Chang and Lin 2011) to form a regression model on the training set.

Part of our motivation for this approach was the large gap between the planning (UCT) and learning (SARSA with linear function approximation) results reported in (Bellemare et al. 2013), the first thorough comparison for the ALE. If the problem is with the feature representation, then more sophisticated algorithms are unlikely to provide much more benefit. However, our preliminary results suggest that the features themselves might not be the problem. We use SVMs with a Gaussian kernel over the features and achieve good prediction accuracy on the training set. However, it should be noted that prediction accuracy on the training set is not necessarily indicative of good performance. Additionally, a linear function approximation over these features, as used in SARSA, is less expressive.

UCT can be used as an oracle. It provides both the correct action to take in any given state and also an estimate of the return for each action in that state. The usual imitation learning setting is to train a classifier based on an oracle providing a correct action. We instead perform a regression based on the return estimates provided by UCT. The agent is then defined by acting greedily according to the resulting Q-function, which can then be used for further training using traditional RL methods.

Model-based value-based cost function. There is another (so far unexamined) cost function that can be used, namely the one that is model-based but optimises the above value-based cost. It comes with the computation cost of estimating the model and determining the value function.

Advantages and Disadvantages

Model-based cost vs. model-free cost. In a model-based approach a sufficiently explorative policy is enough to learn about all policies. Once we have a model, planning techniques such as value-iteration and Monte Carlo Tree Search can be used. Model-based methods can be more data efficient, although less computationally efficient on a per-time-step basis than model-free methods. The model-free cost is easier to scale via function approximation. Scalability in the model-based cost is in theory possible via factored MDPs (Hutter 2009b), but in practice structure learning is a hard problem and although potentially pseudo-polynomial, having this in the inner loop of the algorithm is still computationally problematic.

Value-based cost vs. sequence prediction cost. The value-based cost is more discriminative than the sequence prediction one (it cares directly about predicting the value function). It can also flexibly be used for both model-based and model-free purposes. The sequence prediction based cost, which belongs purely in the model-based setting, can be led astray by needing to predict features that are irrelevant to the value. This could result in maps that are much larger than needed. The distinction is similar to the difference between generative versus discriminative learning.

Successes of FRL

FRL has so far been successful on a wide-variety of toy problems, and the work is progressing towards scaling upwards. CTΦMDP (Nguyen, Sunehag, and Hutter 2011) showed competitive results to the best performing MC-AIXI-CTW agent on various small toy domains (Tiger, Cheese Maze, etc). CTMRL (Nguyen, Sunehag, and Hutter 2012) allowed some scaling to larger domains to compete on POCMAN albeit with several caveats. LSTΦMDP (Daswani, Sunehag, and Hutter 2012) extended the map space to deal with long-term dependencies using looping suffix trees, and solved problems such as T-Maze, which were not viable for standard context-tree based approaches. hQL (Daswani, Sunehag, and Hutter 2013) used the value-based cost to scale up easily via function approximation, and outperformed MC-AIXI-CTW on the POCMAN domain. We are now working on much larger environments such as the ALE.

Conclusion

We reviewed the progress of feature reinforcement learning over the five years since its introduction. Several approaches based on different cost functions have been introduced, and we discussed the advantages and disadvantages of each. We have during these five years seen an increase in complexity for the domains successfully attacked, with Atari games as the next big challenge.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47:253–279.

Boots, B.; Siddiqi, S. M.; and Gordon, G. J. 2010. Closing the Learning-Planning Loop with Predictive State Representations. In *Proceedings of Robotics: Science and Systems*.

Chang, C.-C., and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Daswani, M.; Sunehag, P.; and Hutter, M. 2012. Feature Reinforcement Learning using Looping Suffix Trees. *JMLR Workshop and Conference Proceedings : EWRL 2012* 24:11–24.

Daswani, M.; Sunehag, P.; and Hutter, M. 2013. Q-learning for history-based reinforcement learning. In *Asian Conference on Machine Learning*, 213–228.

Farahmand, A.; Ghavamzadeh, M.; Szepesvári, C.; and Mannor, S. 2008. Regularized Fitted Q-Iteration: Application to Planning. In *Recent Advances in Reinforcement Learning*, volume 5323 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 55–68.

Hamilton, W. L.; Fard, M. M.; and Pineau, J. 2013. Modelling Sparse Dynamical Systems with Compressed Predictive State Representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, 178–186. JMLR Workshop and Conference Proceedings.

Hutter, M. 2004. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Berlin: Springer.

Hutter, M. 2009a. Feature Reinforcement Learning: Part I: Unstructured MDPs. *Journal of Artificial General Intelligence* 1:3–24.

Hutter, M. 2009b. Feature Dynamic Bayesian Networks. In *Proc. 2nd Conf. on Artificial General Intelligence (AGI'09)*, volume 8, 67–73. Atlantis Press.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *In* The 17^{th} European Conference on Machine Learning, 99–134.

Liu, J. S. 2008. *Monte Carlo Strategies in Scientific Computing*. Springer, corrected edition.

McCallum, A. K. 1996. Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 315–324. MIT Press.

Nguyen, P.; Sunehag, P.; and Hutter, M. 2011. Feature reinforcement learning in practice. In *Proc. 9th European Workshop on Reinforcement Learning (EWRL-9)*, volume 7188 of *LNAI*, 66–77. Springer.

Nguyen, P. M.; Sunehag, P.; and Hutter, M. 2012. Context Tree Maximizing. In Hoffmann, J., and Selman, B., eds., *AAAI*. AAAI Press.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc.

Rissanen, J. J. 1978. Modeling by Shortest Data Description. *Automatica* 14(5):465–471.

Rosencrantz, M.; Gordon, G.; and Thrun, S. 2004. Learning low dimensional predictive representations. In *Proceedings of the twenty-first international conference on Machine learning*, 88. ACM.

Sunehag, P., and Hutter, M. 2010. Consistency of Feature Markov Processes. In *Proc. 21st International Conf. on Algorithmic Learning Theory (ALT'10)*, volume 6331 of *LNAI*, 360–374. Canberra: Springer, Berlin.

Veness, J.; Ng, K. S.; Hutter, M.; Uther, W.; and Silver, D. 2011. A Monte Carlo AIXI Approximation. *Journal of Artificial Intelligence Research* 40:95–142.

Wilems, F.; Shtarkov, Y.; and Tjalkens, T. 1995. The context tree weighting method: Basic properties. In *IEEE Transactions on Information Theory*.