Evolutionary Feature Evaluation for Online Reinforcement Learning

Julian Bishop, Risto Miikkulainen
Department of Computer Science
The University of Texas at Austin
2317 Speedway, Stop D9500, Austin, TX, USA
{julian, risto}@cs.utexas.edu

***Abstract*—Most successful examples of Reinforcement Learning (RL) report the use of carefully designed features, that is, a representation of the problem state that facilitates effective learning. The best features cannot always be known in advance, creating the need to evaluate more features than will ultimately be chosen. This paper presents Temporal Difference Feature Evaluation (TDFE), a novel approach to the problem of feature evaluation in an online RL agent. TDFE combines value function learning by temporal difference methods with an evolutionary algorithm that searches the space of feature subsets, and outputs a ranking over all individual features. TDFE dynamically adjusts its ranking, avoids the sample complexity multiplier of many population-based approaches, and works with arbitrary feature representations. Online learning experiments are performed in the game of Connect Four, establishing (i) that the choice of features is critical, (ii) that TDFE can evaluate and rank all the available features online, and (iii) that the ranking can be used effectively as the basis of dynamic online feature selection.**

Keywords—Reinforcement Learning; online learning; Evolutionary Algorithms; feature selection; Connect Four.

I. INTRODUCTION

Reinforcement Learning (RL) [1] is a potentially powerful way to discover effective behavior in games. An RL *agent* attempts to change its policy in order to maximize its rate of *reward*. RL takes place *online* if the agent must learn from playing real games, and the quality of its performance in those games matters. Therefore an online agent must balance exploitation of what it has learned so far with effective exploration of alternative actions.

Typically, an RL agent perceives the state of the game as a one-dimensional array of real numbers called *features*. A *feature* is any intrinsic state-variable of the game, or any real-valued function taking some subset of the state-variables as its domain. An important challenge for RL is that of *feature selection*, in which some available features are ignored because they would harm the agent's performance, not help it. Feature selection can be further decomposed into two sub-problems: First, evaluate the available features for their potential utility and, second, apply some decision process based on the evaluations to pick which features will be ignored. This paper presents a novel approach to the first of these sub-problems, called *Temporal Difference Feature Evaluation* (TDFE). TDFE ranks features from best to worst, but does not specify a decision process for picking which features will be ignored. Although a decision process must be implemented in order to use TDFE as part of an agent, this paper does not advocate for any particular decision process or agent design. Instead, it demonstrates that feature evaluation matters, and that TDFE is an effective way to do it.

TDFE is intended for use in an online RL agent that uses features as inputs to a value function which it updates by temporal difference algorithms [1]. TDFE evaluates multiple feature subsets in parallel and uses an evolutionary algorithm to find combinations of features that work well together. Statistical estimates about each subset are combined to generate a ranking over the available features. TDFE is unique in that it combines three useful properties.

First, TDFE is scalable: The ranking induced includes *all* the features that are available to the agent, which is typically *more* features than the agent is actually using to learn its policy in the game. Moreover, increasing the number of available features does not increase the number of game-states that the agent needs to sample in order to compute the ranking. This property is desirable for online learning because the speed of improvement with respect to the number of games played matters. Although the cost per sampled state of computing the ranking does increase with the number of available features, the issue can be mitigated because TDFE is amenable to distribution across parallel hardware.

Second, the ranking induced by TDFE is dynamic: All the statistics needed to compute the ranking are updated incrementally after each sampled game-state, which allows the ranking to be adjusted at arbitrary time intervals. A dynamic ranking is desirable because the relative importance of the agent's available features can change over time. For example, as the agent learns to play better, it begins to encounter new game-states in which a previously irrelevant feature becomes useful. Alternatively, other players in the game could change their policies causing a similar effect. Or, if the agent attempts a search in the space of possible features, then some old features could be replaced with newly constructed ones which will then need to be included in the ranking.

Third, TDFE makes no assumptions about the underlying function representation of any features. For example, TDFE can evaluate a set of features containing a mixture of Neural Networks, Decision Trees, and Genetic Programs on a level playing field, and include them all in the same ranking. Representation-agnostic feature evaluation is desirable because the best features in different games may be based on different function representations.

Online learning experiments were performed in the game of Connect Four to test the benefit of TDFE. A simple RL agent's performance using TDFE was almost as good as when it was provided with the best possible choice of features in advance.

II. Related Work

Some recent RL algorithms have adapted techniques from Supervised Learning to the RL problem. LARS-TD [2] combines LARS (Least Angle Regression) and LSTD for finding the l_1-regularized fixed point value function. EGD (Equi-Gradient Descent) [3] adapted LARS for minimizing the Bellman residual for temporal difference learning. OMP (Orthogonal Matching Pursuit) is a greedy feature-selection algorithm for regression which uses the correlation between the residual and the candidate features to decide which feature to add next. OMP has been adapted to RL in OMP-BRM and OMP-TD [4].

Unlike TDFE, the applicability of the above methods to online learning is limited by two factors: First, they rely upon batch updates, meaning that the agent does not modify its policy until a sufficient number of game states have been sampled, which increases the amount of time during which the agent's policy is fixed. Second, these methods assume a set of features that does not change while the agent learns, which is particularly significant for the greedy algorithms because they offer no natural way to backtrack and revise earlier feature selection decisions.

Some methods attempt to obviate the problem of feature selection by constructing and/or tuning only a minimal set features and simply using all of them. Examples include Proto-Reinforcement Learning (PRL)[5], Bellman Error Basis Functions [6], and Adaptive Bases[7]. Unlike TDFE, these methods are tied to specific function representations for their features and hence are not representation-agnostic.

Evolutionary algorithms are typically applied to feature evaluation as an offline method. That is, the learning task is used as a wrapper to provide the fitness function for evolving features and feature subsets. Examples include FS-NEAT [8], and variants of Genetic Programming [9]. These offline methods have been adapted to RL by using the performance of an RL agent as the wrapper. Examples include Pittsburgh-style Learning Classifier Systems [10], and agents that use Genetic Programming [11].

Unlike TDFE, the wrapper approaches are of limited use for *online* RL because they are based on a population of policies each of which needs to be independently evaluated. This requirement causes the overall sample complexity to be multiplied by the population size. NEAT+Q [12] attempts to mitigate this problem but must still evaluate a population of policies.

A Michigan-style Learning Classifier System [10] avoids the sample multiplier by being a complete RL agent in which feature evaluation is an integral part of the value function. In contrast, TDFE is not an agent, it is a method for performing feature evaluation as a separate module from the agent's value function.

III. Temporal Difference Feature Evaluation

Fig. 1 depicts the four-stage process that TDFE iterates over to evaluate features. Stages A-D are described in the next four sub-sections.

A. Feature Subset Population (FSP)

The benefit of any particular feature may depend upon which other features it is used in conjunction with. For example, consider a feature that only has predictive power in some subset of the agent's state space. Whether or not those states are even reached by the agent depends upon its policy, which in turn depends upon *all* the features it is using. Furthermore, temporal difference learning's ability to address the temporal credit assignment problem relies on features whose contributions can reduce the magnitude of the temporal difference error in sequences of states that lead to reward. Therefore some feature subsets may support one path to reward while other feature subsets support a different path. For these reasons, feature evaluation and selection in online RL are intertwined with the exploration-exploitation problem [1], and evaluating only a single feature subset is insufficient.

Initialization of the FSP is controlled by three system parameters. *PopSize* sets the number of subsets to be created, and *InitSubsetMean* and *InitSubsetSD* set the mean and standard deviation of a normal distribution that is sampled to choose the cardinality of each subset. Each subset is initialized by uniform randomly picking features without replacement from the available feature universe until the target cardinality is reached.

B. Feature Subset Evaluation

In TDFE, each feature subset in the FSP is connected to its own value function approximator, called a Feature Subset Evaluator (FSE). So if there are 100 feature subsets in the FSP, then there are 100 FSEs, and they are all updated in parallel along with the agent's value function. All the FSEs use the same kind of function approximator as the agent's value function, and they are updated using the same algorithm. The motivation for these similarities is to evaluate features in a way that is directly relevant to the agent. In this paper the agent is assumed to be using a linear value function with gradient descent weight updates guided by Q-Learning [1] and state-action pairs represented as after-states. Extending TDFE to more sophisticated RL agents and/or value function approximators is left for future work.

The FSEs do not participate in the agent's action selection (i.e. policy); their sole purpose is to evaluate features. Consequently, each feature subset is evaluated in the context of the same policy, i.e. the agent's policy, whatever that happens to be. Therefore the FSE must be updated by an off-

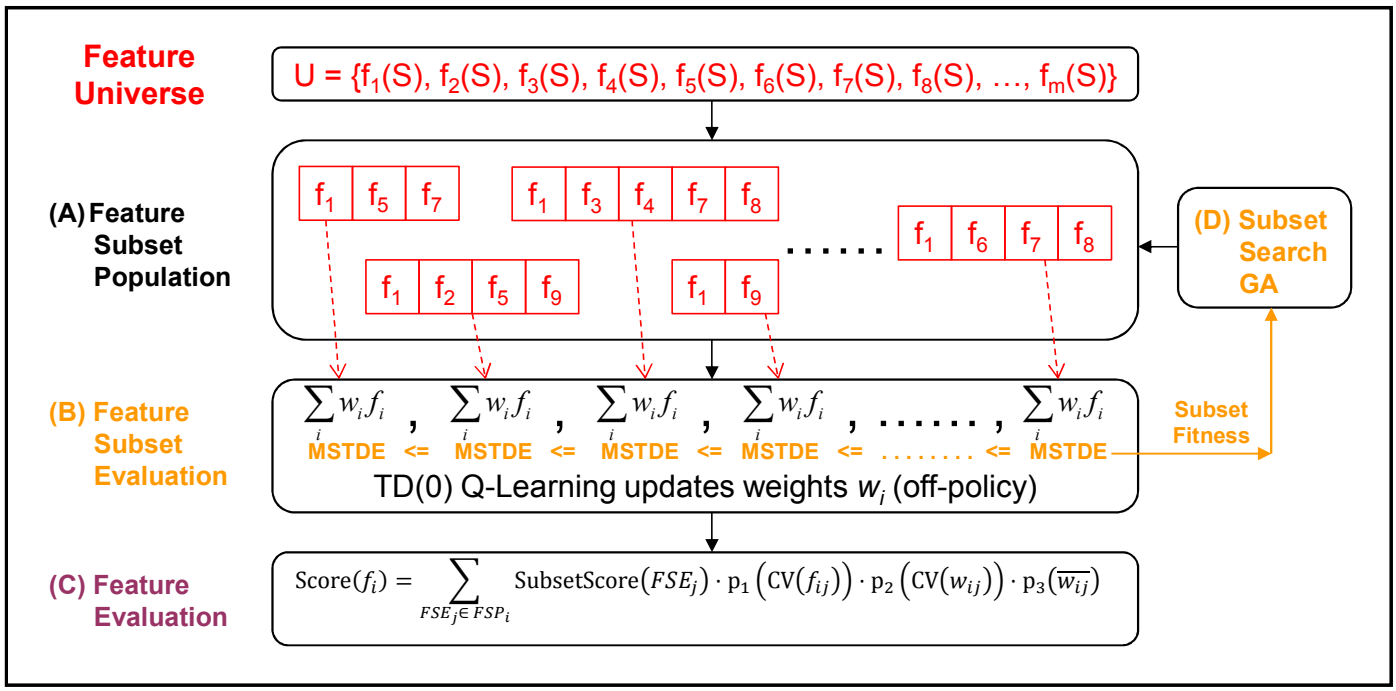


Fig. 1. TDFE. (A) Consider multiple different feature subsets in parallel. (B) Evaluate each feature subset in the setting of the learning algorithm used by the agent, updating statistical estimates incrementally. (C) Combine all statistical estimates into a single feature-quality score and rank all features (FSP_i is the set of feature subsets that include f_i and $\overline{w_{ij}}$ is the mean weight for f_i in FSE_j). (D) Search the space of feature subsets to find better combinations of features for the next iteration.

policy RL algorithm [1]. That is, an algorithm that can (in theory) learn the optimal value function while the agent continues to follow a sub-optimal policy. Although this limits the choice of algorithm, the benefit is that the number of states the agent needs to experience in order to update the FSEs is independent of the number of FSEs. By distributing FSEs across parallel hardware, the number of features that TDFE can evaluate simultaneously can be further scaled up.

For each after-state s selected by the agent's policy, the weight updates reduce the difference between an FSE's value estimate and a target value. The target value depends on information from the next time step. Hence the term temporal difference error, or TDE:

$$\begin{aligned}\mathrm{TDE}(s) &= \mathrm{TargetValue}(s) - \mathrm{CurrentEstimate}(s) \\ &= \mathrm{reward} + \gamma \cdot \max_{s'} Q(s') - Q(s)\ . \end{aligned} \tag{1}$$

The extent to which TDE can be minimized is dependent upon the quality of the features f_i . Hence a measure of the magnitude of the TDE can be interpreted as a measure of the quality of the feature subset in the context of the policy that induced the agent's state trajectory. In TDFE the measure chosen is the mean squared TDE:

$$\mathrm{MSTDE} = \frac{1}{t_2 - t_1 + 1} \sum_{t=t_1}^{t_2} \mathrm{TDE}(s_t)^2\ , \tag{2}$$

where t_1 and t_2 denote the time interval of interest.

TDFE uses the heuristic that within each FSE, the more important features will tend to have weights of greater magnitude and stability. The stability of a signal can be measured by its coefficient of variation which is the ratio between its standard deviation and mean:

$$\mathrm{CV}(X) = \frac{\sigma_X}{\bar{X}}\ . \tag{3}$$

All the statistics computed within each FSE are either means or can be calculated from means. Means are tracked incrementally using the exponentially decaying recency-weighted average update rule [1]. Recency-weighting in the means is important because they all change with time as the weights in the FSE are updated under learning.

C. *Feature Evaluation*

The formula defined in Fig. 1(C) is used to assign a score to every feature. For feature f_i, the summation includes one term for each subset that includes f_i. Each term is the product of a subset-score and three penalty functions, p_1, p_2 and p_3. The subset-score is defined as:

$$\mathrm{SubsetScore}(FSE) = \sqrt{|FSP| - \mathrm{Rank}(FSE)}\ , \tag{4}$$

where Rank*(FSE)* is zero for the best FSE (lowest MSTDE) and the cardinality of the FSP for the worst. The square-root is chosen only because it is a slow-growing function. The three penalty functions (equations 5-7) are used to normalize into [0, 1] the penalties for features having too constant values, unstable weights, and smaller weights, respectively:

$$\mathrm{p}_1(x) = \begin{cases} |x|, & |x| < 0.5 \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

$$\mathrm{p}_2(x) = \mathrm{MAX}(0, 1 - |x|) \tag{6}$$

$$\mathrm{p}_3(x) = \mathrm{MIN}(1, |x| / RewardRange)\ , \tag{7}$$

where *RewardRange* is the difference between the most extreme rewards the agent has experienced. All the features are then sorted by their scores, and the resulting ranking is the output of TDFE.

D. Subset Search GA

Algorithm 1 implements one generation of evolution, and is run at a regular interval set by the system parameter *GamesPerGeneration*. TDFE evolves subsets of features, not value functions or policies. Each newly created subset always gets a new FSE with weights initialized to zero. In short, TDFE uses Darwinian evolution, not Lamarckian. The fitness of a feature subset is the MSTDE of its FSE, which measures how self-consistently and accurately its value function makes predictions about long-term future reward given the agent's policy. The benefit of the GA is to find combinations of features that work well together, which improves the quality of TDFE's feature ranking in stage (C).

TDFE employs a steady-state population model, i.e. only a small fraction of the population are replaced in each generation. Consequently the population contains individuals of different ages: Some FSEs have undergone more generations of weight optimization than others, and a fair comparison of their MSTDE is difficult. Algorithm 1 resolves this in line 4 by applying a fixed survivor rate within each age group. In increasing order of age, the sizes of the age groups form a simple geometric series with the survivor rate as the common ratio. The final age group combines all ages from which the expected number of survivors is less than one, and forms a natural hall of generation champions. This scheme delays the need to distinguish between the best subsets until their MSTDE estimates are more refined, and allows evolution to remove the worst subsets meanwhile.

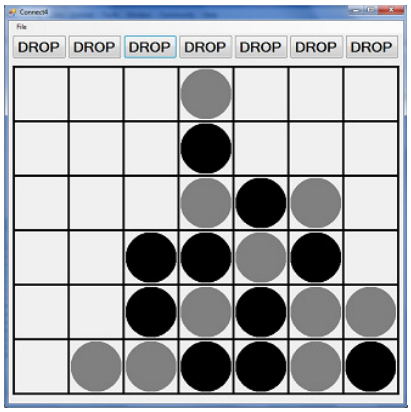
Algorithm 1: Makes the next generation of subsets: fixed survival rate within age groups, tournament selection (size 2) of parents from survivors, edit-walk crossover and mutation.

```
 1. Evolve():
 2.   FSP.SORT_BY(fse.age, fse.MSTDE)
 3.   for each fse in FSP:
 4.     if fse.ELITE_IN_AGE_GROUP(survivorRate):
 5.       .age += 1
 6.     else:
 7.       .age = 0
 8.   Survivors = [FSP where .age > 0]
 9.   Children = [FSP where .age == 0]
10.
11.   for i = 1 to |Children| - 1 step 2:
12.     p1 = SELECT_PARENT_FROM(Survivors)
13.     p2 = SELECT_PARENT_FROM(Survivors)
14.     if RANDOM_REAL(0, 1) < probCrossover:
15.       Child1Subset = EDIT_WALK(p1, p2)
16.       Child2Subset = EDIT_WALK(p2, p1)
17.     else:
18.       Child1Subset = EDIT_WALK(p1, nil)
19.       Child2Subset = EDIT_WALK(p2, nil)
20.     Children[i - 1].initialize(Child1Subset)
21.     Children[i].initialize(Child2Subset)
22.   if i == |Children|:
23.     pLast = SELECT_PARENT_FROM(Survivors)
24.     LastSubset = EDIT_WALK(pLast, nil)
25.     Children[i - 1].initialize(LastSubset)
```

Recombination and mutation are both implemented by the EDIT_WALK function, which performs a random walk in the space of subsets with a sequence of edit operations. Starting at the first parent, each edit is either the removal or addition (equally likely) of a feature chosen at random. The system parameters *EditsMean* and *EditsSD* set the mean and standard deviation of a normal distribution that is sampled to choose the number of edits. If a second parent is specified, it serves as an attractor for the random walk by restricting the cuts and adds to those that reduce the edit distance between the two parents. EDIT_WALK has no bias towards subsets of any particular cardinality.

IV. Experimental Domain: Connect Four

Connect Four has been a fun and interesting real world game since before personal computers were invented. Fig. 2 outlines the rules of the game. Connect Four is suitable for testing TDFE for several reasons: First, the game is difficult enough to be an open problem for online RL, and also for human learners. Second, the state-space is small enough (~4.5e12 [13]) that is has been solved by exhaustive methods [14], and therefore it is possible to measure the quality of an agent's learned policy on an absolute scale between random and optimal play. Third, Connect Four strategy has been thoroughly studied [15], and many board features are known to be important for strong play. Such features should be highly ranked by any good system of feature evaluation.



Two players take turns dropping their colored counter into one of seven columns on the board. Dropped counters fall down the column as if under gravity. The goal is to make four-in-a-row on a horizontal, vertical or diagonal line. In the game shown the second player to move has won with a diagonal . A draw occurs if neither player makes 4-in-a-row before the board becomes full.

Fig. 2. The Connect Four implementation developed for this project.

A. Features

The most basic set of features is an enumeration of the contents of the 42 board cells. With respect to Fig. 2, a 1, -1 or 0 denote black, gray or empty, respectively. Such a naive *board-vector* considers each board cell in isolation, yet success in the game depends upon exploiting patterns over adjacent board cells. Therefore there is no reason to expect the board-vector to facilitate learning a good policy by itself. In addition to the board vector, 26 hand-coded features were implemented as a best effort to generalize over board positions that are equivalently good.

The hand-coded features are similar to those suggested by Allen [15] and are listed in Table I where the following terminology applies: A *threat* for a player is an empty cell that could be part of a four-in-a-row for that player. A level

n-threat is a threat for which *n* members of the four-in-a-row are already present. The *perimeter* is all the cells in which the next counter could be placed. An *on-perimeter threat* is a threat in one of the perimeter cells. An *off-perimeter threat* is a threat not in one of the perimeter cells. A player *controls a column* if it has the lowest off-perimeter 3-threat. An *adjacent threat* is two vertically adjacent cells that are both threats for the same player. An *adjacent threat score* is the sum of the threat levels in an adjacent threat. A *trap* is an adjacent threat score of six. A *playable trap* is a trap in a column controlled by the player who has the trap.

TABLE I. HAND-CODED FEATURES, COMPUTED FOR BOTH PLAYERS.

Feature Description
Number of on-perimeter 1-threats
Number of on-perimeter 2-threats
Number of on-perimeter 3-threats
Number of off-perimeter 1-threats, each weighted by 1/(2^(*h*-1))*
Number of off-perimeter 2-threats, each weighted by 1/(2^(*h*-1))*
Number of off-perimeter 3-threats (max 1 per cell), weighted by 1/(2^(*h*-1))
Highest adjacent threat score on the board
Sum over all adjacent threats of 1/(4^(6-adjacent threat score))
Maximum over all playable traps of 1-(*h*/8)*
Number of controlled columns
Win-next-move flag: 1 if player is guaranteed to win next turn, 0 otherwise
Four-in-a-row flag: 1 if player has four-in-a-row, 0 otherwise.
Win-forced-by-turn flag: binary flag indicating that if no new 3-threats are created then opponent will be forced by turn taking to give player a win.

* *h* is the height of a threat or trap above the perimeter

B. Performance Measure

Three fixed-policy agents are important for defining the performance measure that appears on the *y*-axis of all the learning curves presented in this paper:

1) Random: always selects a move uniformly-randomly from among the legal moves.

2) Simpleton: If a move to make four-in-a-row and win is present, the agent selects it. If no four-in-a-row move is present but a move is present to block the opponent from making four-in-a-row on its next turn, the agent selects it. Otherwise the agent selects a move uniformly-randomly, excluding any move that immediately presents the opponent with a move to make four-in-a-row on its next turn.

3) Genius: Always selects an optimal move. An optimal move will lead to a game outcome (win, lose or draw) at least as good as any other move under the assumption of optimal play by both players until the end of the game. Ties between winning moves are broken uniformly randomly. Ties between drawing or losing moves are broken using a variant of the Expectimax algorithm [16], by computing the probability of a Simpleton opponent making a sub-optimal move thereafter. Using the Simpleton as an opponent model results in selecting moves that maximize the sub-optimal proportion of the opponent's responses, which is a sensible strategy in the absence of the true opponent model.

A log containing nine million solved board positions was generated by recording games between two augmented Genius agents. An augmented Genius is forced to move as a Random player with probability *p*. Twelve values for *p* were used to create twelve Player1 Geniuses and twelve Player2 Geniuses: 0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55. Three thousand games were recorded for each of the 144 possible pairings of a Player1 with a Player2, making a total of 432,000 games in the log, comprised of about six million distinct board positions distributed over nine million played positions. The duplicate positions are desirable because they naturally attribute more weight to board positions that occur more often. For each board position, the legal moves were ranked by a Genius agent.

To evaluate a *candidate* Connect Four policy using the log, the following procedure is followed: For each applicable position in the log (Player-1 or Player-2), the candidate is asked for its move, and the probability that a Random agent would have made a better move is computed from the move rankings. The average of these probabilities over all applicable positions in the log is denoted Pr(*CandidateWTR*), where *WTR* stands for Worse-Than-Random. Two moves are equal in rank with probability 0.3. Hence Pr(*RandomWTR*) ≅ 0.35, and any candidate such that Pr(*CandidateWTR*) > 0.35 is truly worse than a Random player. Consequently the value of Pr(*CandidateWTR*) is rescaled into a metric called WTR:

$$\mathrm{WTR}(Candidate) = \frac{\Pr(RandomWTR) - \Pr(CandidateWTR)}{\Pr(RandomWTR)}. \qquad (8)$$

By definition, WTR(*Random*) = 0, and WTR(*Genius*) = 1 which is the best possible score. Even a highly random player such as a Simpleton receives a very consistent WTR score: 0.1624 ± 0.00011 as Player-1, 0.1829 ± 0.00016 as Player-2 (99% confidence intervals).

V. EXPERIMENTS

Experiments with TDFE require a decision process for picking which features will be ignored based on the TDFE feature ranking, and an online RL agent that uses the remaining features as inputs to its value function. These details are presented next, followed by the results obtained in two sets experiments: one with TDFE disabled as a baseline, and one with TDFE enabled.

A. RL Agents

A canonical RL agent is needed to serve as a platform for testing TDFE. The agent chosen is a Q-Learning agent, with after-states representing state-action pairs, a linear value-function approximator trained by gradient descent weight updates, and ε-greedy action selection [1]. This formulation of an RL agent is one of the oldest and best understood and it has some limited convergence guarantees. Game features are normalized into the range [-1, 1] and used as inputs to the agent's value function. Fig. 3 shows how the Canonical RL agent (green box) was combined with TDFE.

The Feature Selection module in Fig. 3 resets which features will be ignored at the start of each new game based on the dynamic feature ranking from TDFE: Features are selected greedily from the feature ranking. The initial number of features used by the agent is based on the average cardinality of the evolving feature subsets. In addition, a pool of candidate features is selected by continuing greedily down the feature ranking until a cost limit is reached. As the agent

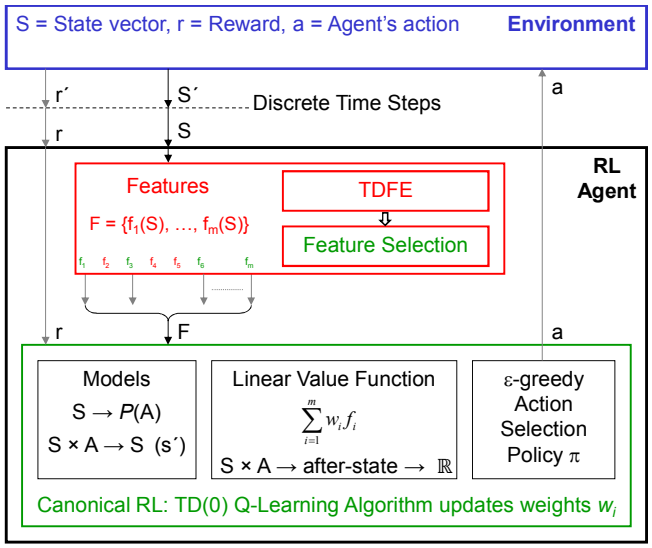


Fig. 3. System diagram for the online RL agent comprised of the Canonical Agent, a Feature Selection module, and TDFE. The models are used by the agent to obtain a list of legal moves in the current state, and to compute the after-state of any action. In this paper the Environment is Connect Four. On each discrete time-step in the game, the agent has access to the current state 'S' and a reward signal 'r'. The agent executes its chosen action 'a'. When it is the agent's turn to act again, it sees the next state and reward, S' and r'.

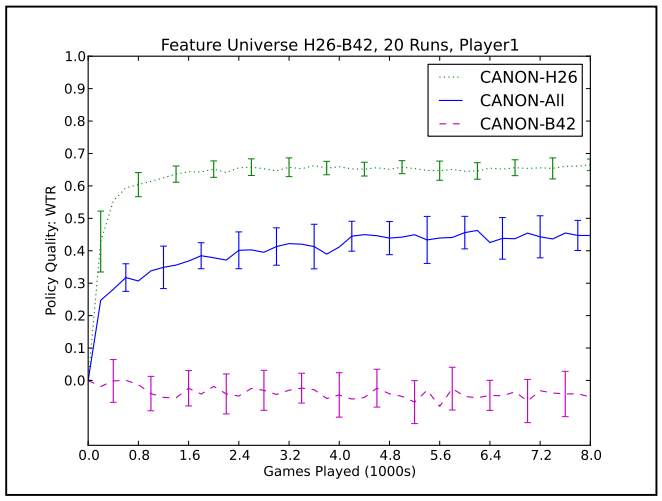


Fig. 4. Performance comparison of the canonical RL agent using three different Connect-4 feature sets: (B42) 42 board-vector features, (H26) 26 hand-coded features, and (All) the union of B42 and H26. Given only the hand-coded features, the policy learned is human-competitive. Given only the board-vector, the policy learned is worse than a random player. The union of all features results in worse learned policy than the hand-coded features alone.

learns, estimates are made of the Pearson correlation coefficient between the agent's temporal difference error and each candidate feature's potential contribution to that error (which is similar to OMP-BRM [4]). If the most correlated candidate feature satisfies a selection condition it is added to the agent's value function and all correlation estimates are re-estimated from scratch in the context of the new value function until the selection condition is satisfied again. If the MSTDE of the agent's value function exceeds that of any FSE then all features that were added by correlation are thrown out and the agent's feature selection is reset greedily with respect to the feature ranking only. Over multiple generations of evolution the average cardinality becomes a better indicator of how many features to use. However, that is a slow process, so the correlation-based selection mechanism is useful on shorter timescales.

B. *Baseline Experiments*

As a baseline for experiments with TDFE, the canonical agent's ability to learn a good Connect Four policy online was tested with TDFE and feature selection both disabled, i.e. the agent used all available features unconditionally. The same experiment was repeated three times varying only the set of features given to the agent: first with the *board-vector* only, second with the *hand-coded* features only, and third with the *union* of the board-vector and hand-coded features.

All three experiments involve two canonical RL agents, one as Player-1 and a separate one as Player-2. The two agents learned while playing 8000 games against each other. Each game was treated as a separate learning episode with zero reward except in the terminal states which gave +100 for a win, -100 for a loss, and 0 for a draw. The discount factor γ was set to 1 because the problem is episodic. The exploration rate ε was set to 0.1. The learning rate α was set to $1/n$, where n is the number of features. This makes the maximum rate by which the value function's output can change independent of n. The agents were saved every 200 games, and every saved agent was evaluated to determine its WTR score. This setup was run 20 times for each feature set.

The average learning curves for Player-1 are shown in Fig. 4 (results for Player-2 are similar and omitted). Each data-point and its accompanying error bars show the mean and standard deviation of WTR(Player-1). The top curve, using the 26 hand-coded features, demonstrates that the canonical RL agent is sufficient for effective learning in Connect Four. After 8000 games, both agents (Player-1 and Player-2) are consistently playing at a human-competitive level. That is, experienced adult human Connect-4 players have been defeated by these agents and/or reported substantial effort required to beat them.

The bottom curve in Fig. 4 demonstrates that the canonical RL agent cannot learn a good policy at all when using only the board-vector as its feature set. Moreover, it appears that learning from the board-vector alone actually misleads the agent, causing its policy to be slightly worse than that of a random player. This is unsurprising because each board cell generalizes over states of unrelated value.

The middle curve in Fig. 4 shows that the approach of adding good features to a set of bad ones, while better than having no good features at all, is worse than being selective about which features to use and avoiding the bad features.

C. *TDFE Experiments*

The baseline experiment in which the agent was given the union of all H26-B42 features was repeated with TDFE and feature selection enabled. An equivalent experiment was also performed in each of two additional feature universes, first with TDFE and feature selection disabled, and then enabled.

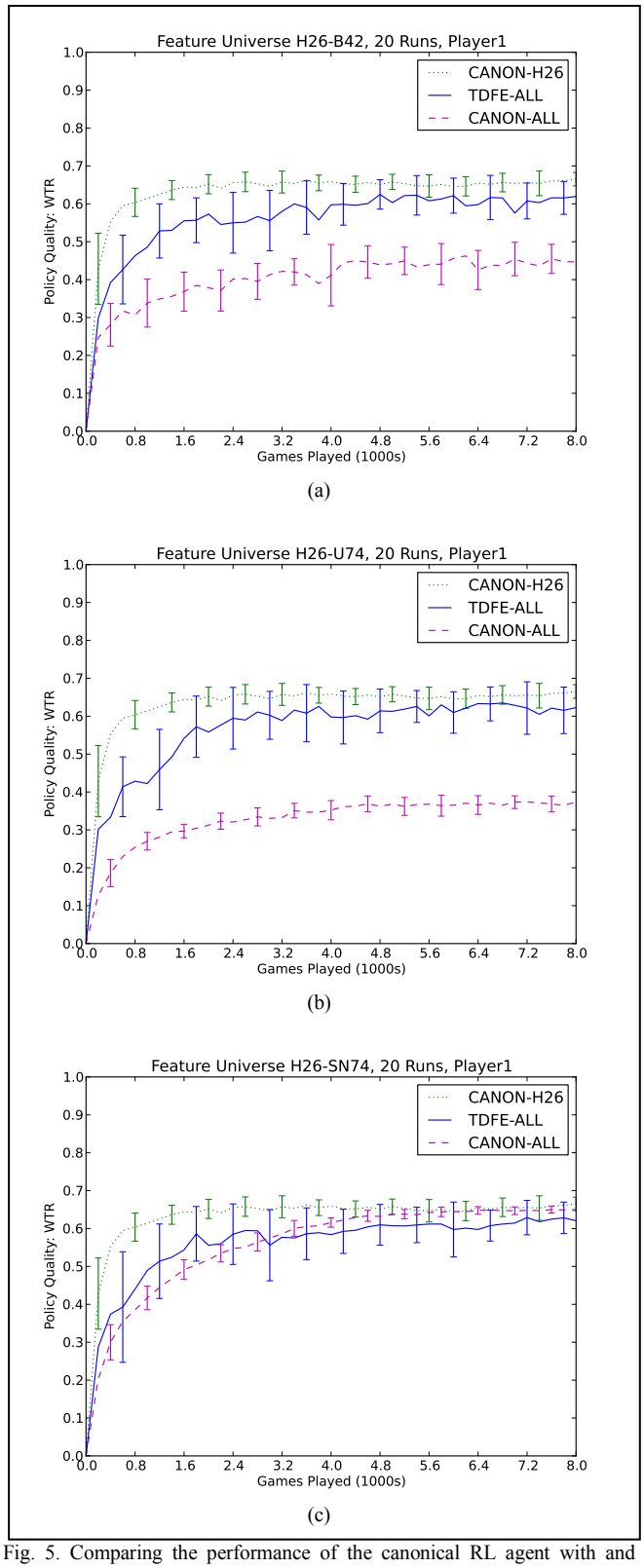


Fig. 5. Comparing the performance of the canonical RL agent with and without TDFE and Feature Selection enabled in three different feature sets. All three feature sets include the 26 hand-coded features (H26), and the learning curve of the canonical agent using only H26 is repeated in each sub-plot as a referent. H26-B42 (a) is the same as in the baseline experiments. H26-U74 (b) includes 74 uniformly distributed random variables. H26-SN74 (c) includes 74 standard normally distributed random variables.

In both additional feature universes, the 42 board-vector features were replaced with 74 independent identically distributed random variables, introduced as features that are pure noise, i.e. bearing no relation to the state of the game. In H26-U74, each random variable samples a uniform distribution on the interval [-1, 1]. In H26-SN74, each random variable samples a standard normal distribution. In all runs, the TDFE system parameters were set as follows: *PopSize*=100, *GamesPerGeneration*=100, *SurvivorRate*=0.8, *InitSubsetMean*=3.25, *InitSubsetSD*=1, *ProbCrossover*=0.5, *EditsMean*=10, *EditsSD*=3,.

The average learning curves for Player-1 are shown in Fig. 5 (again the Player-2 curves are similar). All three sub-plots include the canonical agent's performance when using only the hand-coded features (H26) as a referent for the best performance that the TDFE agent could hope to achieve.

Fig. 5(a) shows that the agent's performance in H26-B42 is significantly better with TDFE and feature selection enabled. In that case the agent's average final performance is 6.9% worse than the level associated with selecting H26 exclusively. However, its average best performance is within 1.9% of the H26 level, indicating that the TDFE agent peaks close to the H26 level but is less stable. These differences are statistically significant at the 0.05 level.

Fig. 5(b) shows that the agent's performance in H26-U74 is significantly better with TDFE and feature selection enabled. The difference between the TDFE agent and the H26 level is not statistically significant.

Fig. 5(c) shows that the agent's performance in H26-SN74 is no better with TDFE and feature selection enabled. Interestingly, there is no statistically significant difference between the TDFE agent in H26-U74 and H26-SN74, showing that it is insensitive to the distribution of noise. However, the change from uniform to normal random variables greatly improves the canonical agent's baseline performance, such that the only negative impact of the 74 normal variables is to slow down learning. This result is explained by feature normalization, but is useful for showing the effect of TDFE on the agent's performance in a feature universe that has relatively little need for feature selection.

VI. Discussion and Future Work

The experiments in Section V establish that the performance of online RL in Connect Four depends upon the available features. Given a well-engineered set of features based upon established heuristics, even a very basic RL agent can bootstrap itself up from an initially random policy to a human-competitive level guided only by a reward signal that is zero everywhere except states having 4-in-a-row. In contrast, given only the vector of board cells, the same canonical agent is unable to improve and actually performs worse than randomly due to the misleading nature of those features. These bad features are not irrelevant: They are a perfect noiseless encoding of the game-state, and indeed are the domain of all functions found to be good features. Despite their relevance, they significantly harm the agent's performance even when all the good features are present. The effect of irrelevant noisy features on the canonical agent

depends upon the distribution of the noise: Uniformly distributed random variables significantly harm the agent's learned policy, but normally distributed random variables only delay learning.

TDFE was presented as a novel way to evaluate and rank all available features online, i.e. while the agent is learning in the actual game. The canonical agent was modified to dynamically adjust its feature selection at the start of each game based upon the current TDFE rankings. The resulting performance is close to that of using only the best available features and unaffected by whether the other available features are relevant yet harmful, irrelevant harmful noise, or irrelevant yet mostly harmless noise. In all cases, the good features were in the minority. An inspection of the TDFE feature-ranking (not shown) revealed that the hand-coded features for detecting winning moves and blocking the opponent's winning moves are always in the top-three ranked features by the end of every run. Although this evidence is anecdotal, it is encouraging that such domain-critical features are reliably identified by TDFE without the use of a domain-specific heuristic.

The primary line of future work is for the TDFE feature ranking to guide an evolutionary search over the space of possible features as part of an online RL agent. TDFE is representation-agnostic with respect to features, so any evolvable function representations can be used side-by-side. TDFE's feature ranking is dynamic, so newly constructed features can be naturally included. TDFE does not treat the game as a wrapper for evaluating multiple policies, and so it will avoid the sample complexity multiplier associated with most evolutionary methods.

This paper has not attempted to show that TDFE is the only or even the best way to do feature evaluation in online RL. However, none of the other methods surveyed in Section II have *all* of the desirable properties explained in Section I. Those properties make TDFE of unique interest as a platform for this project's future work.

VII. Conclusion

This paper has shown that performing feature evaluation as part of an online RL agent matters, and that it can be performed in a way that is scalable, sample efficient, dynamic, and agnostic with respect to any feature's underlying function representation. The resulting method is called Temporal Difference Feature Evaluation or TDFE, and will serve as a stepping stone to future work on automatic feature construction in an online RL agent.

Acknowledgment

Thanks to James D. Allen and John Tromp for providing a C++ program that functions as a Connect-4 oracle capable of solving the game from any board position. Thanks to Matthew Hausknecht for multiple supporting references.

References

[1] R. S. Sutton, and A. G. Barto, *Introduction to Reinforcement Learning*: MIT Press, 1998.

[2] J. Z. Kolter, and A. Y. Ng, "Regularization and feature selection in least-squares temporal difference learning," in Proceedings of the 26th international Conference on Machine Learning (ICML'09), 2009, pp. 521-528.

[3] M. Loth, M. Davy, and P. Preux, "Sparse temporal difference learning using LASSO," in Proceedings of IEEE international Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007, pp. 352-359.

[4] C. Painter-Wakefield, and R. Parr, "Greedy Algorithms for Sparse Reinforcement Learning," in Proceedings of the Twenty-Ningth International Conference on Machine Learning (ICML-2012), 2012.

[5] S. Mahadevan, "Proto-value functions: developmental reinforcement learning," in Proceedings of the 22nd International Conference on Machine learning (ICML'05), Bonn, Germany, 2005, pp. 553-560.

[6] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing feature generation for value-function approximation," in Proceedings of the 24th international conference on Machine learning, Corvalis, Oregon, 2007, pp. 737-744.

[7] D. Di Castro, and S. Mannor, "Adaptive bases for reinforcement learning," in Proceedings of the 2010 European conference Machine Learning and Knowledge Discovery in Databases, 2010, pp. 312-327.

[8] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl, "Automatic Feature Selection via Neuroevolution," in Genetic and Evolutionary Computation Conference, 2005.

[9] M. G. Smith, and L. Bull, "Genetic programming with a genetic algorithm for feature construction and selection," *Genetic Programming and Evolvable Machines,* vol. 6, no. 3, pp. 265-281, 2005.

[10] J. Holland, L. Booker, M. Colombetti, M. Dorigo, D. Goldberg, S. Forrest, R. Riolo, R. Smith, P. Lanzi, W. Stolzmann, and S. Wilson, "What Is a Learning Classifier System?," *Learning Classifier Systems, Lecture Notes in Computer Science*, Lecture Notes in Computer Science P. Lanzi, W. Stolzmann and S. Wilson, eds., pp. 3-32: Springer Berlin / Heidelberg, 2000.

[11] S. Girgin, and P. Preux, "Feature discovery in reinforcement learning using genetic programming," in Proc. 11th European Conference on Genetic Programming (EUROGP), 2008, pp. 218-229.

[12] S. Whiteson, and P. Stone, "Evolutionary Function Approximation for Reinforcement Learning," *Journal of Machine Learning Research,* vol. 7, pp. 877-917, 2006.

[13] S. Edelkamp, and P. Kissmann, "Symbolic Classification of General Two-Player Games," *KI 2008: Advances in Artificial Intelligence*, Lecture Notes in Computer Science A. Dengel, K. Berns, T. Breuel, F. Bomarius and T. Roth-Berghofer, eds., pp. 185-192: Springer Berlin Heidelberg, 2008.

[14] V. Allis, "A knowledge-based approach of connect-four," *The game is solved: White wins. Master's thesis, Faculty of Mathematics and Computer Science, Free University, Amsterdam*, 1988.

[15] J. Allen, *The Complete Book of Connect 4*: Sterling, 2011.

[16] S. Russell, and P. Norvig, *Artificial Intelligence: A Modern Approach*: Prentice Hall, 2003.