Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Deep Learning for Reward Design to Improve Monte Carlo Tree Search in ATARI Games

Xiaoxiao Guo, Satinder Singh, Richard Lewis, Honglak Lee
University of Michigan, Ann Arbor
{guoxiao,baveja,rickl,honglak}@umich.edu

Abstract

Monte Carlo Tree Search (MCTS) methods have proven powerful in planning for sequential decision-making problems such as Go and video games, but their performance can be poor when the planning depth and sampling trajectories are limited or when the rewards are sparse. We present an adaptation of PGRD (policy-gradient for reward-design) for learning a reward-bonus function to improve UCT (a MCTS algorithm). Unlike previous applications of PGRD in which the space of reward-bonus functions was limited to linear functions of hand-coded state-action-features, we use PGRD with a multi-layer convolutional neural network to automatically learn features from raw perception as well as to adapt the non-linear reward-bonus function parameters. We also adopt a variance-reducing gradient method to improve PGRD's performance. The new method improves UCT's performance on multiple ATARI games compared to UCT without the reward bonus. Combining PGRD and Deep Learning in this way should make adapting rewards for MCTS algorithms far more widely and practically applicable than before.

1 Introduction

This paper offers a novel means of combining Deep Learning (DL; see Bengio 2009; Schmidhuber 2015 for surveys) and Reinforcement Learning (RL), with an application to ATARI games. There has been a flurry of recent work on combining DL and RL on ATARI games, including the seminal work using DL as a function approximator for Q-learning [Mnih *et al.*, 2015], the use of UCT-based planning to provide policy-training data for a DL function approximator [Guo *et al.*, 2014], the use of DL to learn transition-models of ATARI games to improve exploration in Q-learning [Oh *et al.*, 2015; Stadie *et al.*, 2015], and the use of DL as a parametric representation of policies to improve via policy-gradient approaches [Schulman *et al.*, 2015].

In contrast, the work presented here uses DL as a function approximator to learn reward-bonus functions from experience to mitigate computational limitations in UCT [Kocsis and Szepesvári, 2006], a Monte Carlo Tree Search (MCTS) algorithm. In large-scale sequential decision-making problems, UCT often suffers because of limits on the number of trajectories and planning depth required to keep the method computationally tractable. The key contribution of this paper is a new method for improving the performance of UCT planning in such challenging settings, exploiting the powerful feature-learning capabilities of DL.

Our work builds on PGRD (policy-gradient for reward-design; Sorg *et al.* 2010a), a method for learning reward-bonus functions for use in planning. Previous applications of PGRD have been limited in a few ways: 1) they have mostly been applied to small RL problems; 2) they have required careful hand-construction of features that define a reward-bonus function space to be searched over by PGRD; 3) they have limited the reward-bonus function space to be a linear function of hand-constructed features; and 4) they have high-variance in the gradient estimates (this issue was not apparent in earlier work because of the small size of the domains). In this paper, we address all of these limitations by developing PGRD-DL, a PGRD variant that automatically learns features over raw perception inputs via a multi-layer convolutional neural network (CNN), and uses a variance-reducing form of gradient. We show that PGRD-DL can improve performance of UCT on multiple ATARI games.

2 Background and Related Work

ALE as a challenging testbed for RL. The Arcade Learning Environment (ALE) includes an ATARI 2600 emulator and about 50 games [Bellemare *et al.*, 2013b]. These games present a challenging combination of perception and policy selection problems. All of the games have the same high-dimensional observation screen, a 2D array of 7-bit pixels, 160 pixels wide by 210 pixels high. The action space depends on the game but consists of at most 18 discrete actions. Building agents to learn to play ATARI games is challenging due to the high-dimensionality and partial observability of the perceptions, as well as the sparse and often highly-delayed nature of the rewards. There are a number of approaches to building ATARI game players, including some that do not use DL (e.g., Bellemare *et al.* 2012a, 2012b, 2013a, Lipovetzky *et al.* 2015). Among these are players that use UCT to plan with the ALE emulator as a model [Bellemare *et al.*, 2013b], an approach that we build on here.

Combining DL and RL. Deep Learning is a powerful set of techniques for learning feature representations from raw input data in a compositional hierarchy, where low-level features are learned to encode low-level statistical dependencies (e.g. edges in images), and higher-level features encode higher-order dependencies of the lower-level features (e.g. object parts; Lee *et al.* 2009). Combining DL and RL methods together thus offers an opportunity to address the high-dimensional perception, partial observability and sparse and delayed reward challenges in building ATARI game playing agents. Indeed, as summarized above, recent work has started to do just that, though research into combining neural networks and RL is older than the recent interest in ALE (e.g., Schmidhuber 1990; Tesauro 1994).

UCT. UCT is widely used for planning in large scale sequential decision games. It takes three parameters: the number of trajectories, the maximum-depth, and an exploration parameter. In general, the larger the trajectory and depth parameters, the better the performance, though at the expense of increased computation at each action selection. UCT computes a score for each possible action a in state-depth pair (s, d) as the sum of two terms: an exploitation term that is the Monte Carlo average of the discounted sum of rewards obtained from experiences with state-depth pair (s, d) in the previous $k - 1$ trajectories, and an exploration term that encourages sampling infrequently taken actions.[1] UCT selects the action to simulate in order to extend the trajectory greedily with respect to the summed score. Once the number of sampled trajectories reaches the parameter value, UCT returns the exploitation term for each action at the root node as its estimate of the utility of taking that action in the current state of the game.

Although they have been successful in many applications, UCT and more generally MCTS methods also have limitations: their performance suffers when the number of trajectories and planning depth are limited relative to the size of the domain, or when the rewards are sparse, which creates a kind of needle-in-a-haystack problem for search.

Mitigating UCT limitations. Agent designers often build in heuristics for overcoming these UCT drawbacks. As one example, in the case of limited planning depth, the classical Leaf-Evaluation Heuristic (LEH) adds a heuristic value to the estimated return at the leaf states. However, the value to compensate for the missing subtree below the leaf is often not known in practice. Many other approaches improve UCT by mechanisms for generalization of returns and visit statistics across states: e.g., the Transposition Tables of Childs *et al.* (2008), the Rapid Action Value Estimation of Gelly and Silver (2011), the local homomorphisms of Jiang *et al.* (2014), the state abstractions of Hostetler *et al.* (2014), and the local manifolds of Srinivasan *et al.* (2015).

In this paper, we explore learning a reward-bonus function to mitigate the computational bounds on UCT. Through deep-learned features of state, our method also provides a generalization mechanism that allows UCT to exploit useful knowledge gained in prior planning episodes.

Reward design, optimal rewards, and PGRD. Singh *et al.* (2010) proposed a framework of optimal rewards which allows the use of a reward function *internal* to the agent that is potentially different from the *objective* (or task-specifying) reward function. They showed that good choices of internal reward functions can mitigate agent limitations.[2] Sorg *et al.* (2010b) proposed PGRD as a specific algorithm that exploits the insight that for planning agents a reward function implicitly specifies a policy, and adapts policy gradient methods to search for good rewards for UCT-based agents. As discussed in the Introduction, PGRD has limitations that we address next through the introduction of PGRD-DL.

3 PGRD-DL: Policy-Gradient for Reward Design with Deep Learning

Understanding PGRD-DL requires understanding three major components. First, PGRD-DL uses UCT differently than usual in that it employs an internal reward function that is the sum of a reward-bonus and the usual objective reward (in ATARI games, the change in score). Second, there is a CNN-based parameterization of the reward-bonus function. Finally, there is a gradient procedure to train the CNN-parameters. We describe each in turn.

UCT with internal rewards. As described in the Related Work section above, in extending a trajectory during planning, UCT computes a score that combines a UCB-based *exploration term* that encourages sampling infrequently sampled actions with an *exploitation term* computed from the trajectories sampled thus far. A full H-length trajectory is a sequence of state-action pairs: $s_0 a_0 s_1 a_1 ... s_{H-1} a_{H-1}$. UCT estimates the exploitation-term value of a state, action, depth tuple (s, a, d) as the average return obtained after experiencing the tuple (non-recursive form):

$$Q(s,a,d) = \sum_{i=1}^{N} \frac{I_i(s,a,d)}{n(s,a,d)} \sum_{h=d}^{H-1} \gamma^{h-d} R(s_h^i, a_h^i) \quad (1)$$

where N is the number of trajectories sampled, γ is the discount factor, $n(s, a, d)$ is the number of times tuple (s, a, d) has been sampled, $I_i(s, a, d)$ is 1 if (s, a, d) is in the i^{th} trajectory and 0 otherwise, s_h^i is the h^{th} state in the i^{th} trajectory and a_h^i is the h^{th} action in the i^{th} trajectory.

The difference between the standard use of UCT and its use in PGRD-DL is in the choice of the reward function in

[1] Specifically, the exploration term is $c\sqrt{\log n(s,d)/n(s,a,d)}$ where $n(s, d)$ and $n(s, a, d)$ are the number of visits to state-depth pair (s, d), and of action a in state-depth pair (s, d) respectively in the previous $k - 1$ trajectories, and c is the exploration parameter.

[2] Others have developed methods for the design of rewards under alternate settings. Potential-based Reward Shaping [Ng *et al.*, 1999; Asmuth *et al.*, 2008] offers a space of reward functions with the property that an optimal policy for the original reward function remains an optimal policy for each reward function in the space. This allows the designer to heuristically pick a reward function for other properties, such as impact on learning speed. In some cases, reward functions are simply unknown, in which case inverse-RL [Ng and Russell, 2000] has been used to infer reward functions from expert behavior. Yet others have explored the use of queries to a human expert to elicit preferences [Chajewska *et al.*, 2000].

Equation 1. To emphasize this difference we use the notation R^O to denote the usual *objective* reward function, and R^I to denote the new *internal* reward function. Specifically, we let

$$R^I(s,a;\theta) = \text{CNN}(s,a;\theta) + R^O(s,a) \tag{2}$$

where the reward-bonus that is added to the objective reward in computing the internal reward is represented via a multi-layered convolution neural network, or CNN, mapping from state-action pairs to a scalar; θ denotes the CNN's parameters, and thus the reward-bonus parameters. To denote the use of internal rewards in Equation 1 and to emphasize its dependence on the parameters θ, we will hereafter denote the Q-value function as $Q^I(\cdot,\cdot,\cdot;\theta)$. Note that the reward bonus in Equation 2 is distinct from (and does not replace) the exploration bonus used by UCT during planning.

CNN parameterization of reward-bonuses. PGRD-DL is capable of using any kind of feed-forward neural network to represent the reward bonus functions. The Experiment Setup section below defines the specific convolution network used in this work.

Gradient procedure to update reward-bonus parameters. When UCT finishes generating the specified number of trajectories (when planning is complete), the greedy action is

$$a = \arg\max_b Q^I(s,b,0;\theta) \tag{3}$$

where the action values of the current state are $Q^I(s,.,0;\theta)$. To allow for gradient calculations during training of the reward-bonus parameters, the UCT agent executes actions according to a softmax distribution given the estimated action values (the temperature parameter is omitted):

$$\mu(a|s;\theta) = \frac{\exp Q^I(s,a,0;\theta)}{\sum_b \exp Q^I(s,b,0;\theta)}, \tag{4}$$

where μ denotes the UCT agent's policy.

Even though internal rewards are used to determine the UCT policy, only the task-specifying objective reward R^O is used to determine how well the internal reward function is doing. In other words, the performance of UCT with the internal reward function is measured over the experience sequence that consists of the actual executed actions and visited states: h_T= $s_0a_0s_1a_1...s_{T-1}a_{T-1}$ in terms of objective-return $u(.)$:

$$u(h_T) = \sum_{t=0}^{T-1} R^O(s_t,a_t) \tag{5}$$

where s_t and a_t denote the actual state and action at time t. Here we assume that all tasks are episodic, and the maximum length of an experience sequence is T. The expected objective-return is a function of the reward bonus function parameters θ because the policy μ depends on θ, and in turn the policy determines the distribution over state-action sequences experienced.

The PGRD-DL objective in optimizing reward bonus parameters is maximizing the expected objective-return of UCT:

$$\theta^* = \arg\max_\theta U(\theta) = \arg\max_\theta \mathbb{E}\{u(h_T)|\theta\}. \tag{6}$$

The central insight of PGRD was to consider the reward-bonus parameters as policy parameters and apply stochastic gradient ascent to maximize the expected return. Previous applications of PGRD used linear functions of hand-coded state-action-features as reward-bonus functions. In this paper, we first applied PGRD with a multi-layer convolutional neural network to automatically learn features from raw perception as well as to adapt the non-linear reward-bonus parameters. However, empirical results showed that the original PGRD could cause the CNN parameters to diverge and cause degenerate performance due to large variance in the policy gradient estimation. We therefore adapted a variance-reduction policy gradient method GARB (GPOMDP with Average Reward Baseline; Weaver and Tao (2001)) to solve this drawback of the original PGRD. GARB optimizes the reward-bonus parameters to maximize the expected objective-return as follows:

$$\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}\{u(h_T)|\theta\} \tag{7}$$

$$= \mathbb{E}\left\{u(h_T)\sum_{t=0}^{T-1}\frac{\nabla_\theta \mu(a_t|s_t;\theta)}{\mu(a_t|s_t;\theta)}\right\} \tag{8}$$

Thus, $u(h_T)\sum_{t=0}^{T-1}\frac{\nabla_\theta \mu(a_t|s_t;\theta)}{\mu(a_t|s_t;\theta)}$ is an unbiased estimator of the objective-return gradient. GARB computes an eligibility trace vector $\mathbf{e}$ to keep track of the gradient vector $\mathbf{g}$ at time t:

$$\mathbf{e_{t+1}} = \beta\mathbf{e_t} + \frac{\nabla_\theta \mu(a_t|s_t;\theta)}{\mu(a_t|s_t;\theta)} \tag{9}$$

$$\mathbf{g}_{t+1} = \mathbf{g}_t + (r_t - b)\mathbf{e_{t+1}} \tag{10}$$

where $\beta \in [0,1)$ is a parameter controlling the bias-variance trade-off of the gradient estimation, $r_t = R^O(s_t,a_t)$ is the immediate objective-reward at time t, b is a reward baseline and it equals the running average of r_t, and $\mathbf{g_T}$ is the gradient estimate vector. We use the gradient vector to update the reward parameters θ when a terminal state is reached; at the end of the j^{th} episode, $\mathbf{g_T}$ is used to update θ using an existing stochastic gradient based method, ADAM [Kingma and Ba, 2015], as described below.

Since the reward-bonus function is represented as a feed-forward network, back-propagation can compute the gradient of the reward-bonus function parameters, i.e. $\frac{\nabla_\theta \mu(a_t|s_t;\theta)}{\mu(a_t|s_t;\theta)}$, efficiently. In order to apply BackProp, we need to compute the derivative of policy μ with respect to the reward $R^I(s_h^i,a_h^i;\theta)$ in the i^{th} sampling trajectory at depth h in UCT planning:

$$\delta_t^{r(i,h)} = \frac{1}{\mu(a_t|s_t)}\frac{\partial \mu(a_t|s_t)}{\partial R^I(s_h^i,a_h^i)} \tag{11}$$

$$= \frac{1}{\mu(a_t|s_t)}\sum_b \frac{\partial \mu(a_t|s_t)}{\partial Q^I(s_t,b,0)}\frac{\partial Q^I(s_t,b,0)}{\partial R^I(s_h^i,a_h^i)} \tag{12}$$

$$= \sum_b (I(a_t=b) - \mu(b|s_t))\frac{I_i(s_t,b,0)}{n(s_t,b,0)}\gamma^h \tag{13}$$

where $I(a_t=b)=1$ if a_t equals b and 0 otherwise. Thus the derivative of any parameter θ_k in the reward parameters can be represented as:

$$\delta_t^{\theta_k} = \sum_{i,h}\delta_t^{r(i,h)}\frac{\partial R^I(s_h^i,a_h^i)}{\partial \theta_k} \tag{14}$$

where $\frac{\partial R^I(s_h^i, a_h^i)}{\partial \theta_k}$ is determined by (s_h^i, a_h^i) and the CNN, and can be computed efficiently using standard BackProp.

What does the PGRD-DL learn? We emphasize that the only thing that is learned from experience during repeated application of the PGRD-DL planning procedure is the reward-bonus function. All the other aspects of the PGRD-DL procedure remain fixed throughout learning.

4 Experimental Setup: Evaluating PGRD-DL on ATARI Games

We evaluated PGRD-DL on 25 ATARI games (Table 1). All the ATARI games involve controlling a game agent in a 2-D space, but otherwise have very different dynamics.

UCT objective reward and planning parameters. As is standard in RL work on ATARI games, we take the objective reward for each state to be the difference in the game score between that state and the previous state. We rescale the objective reward: assigning +1 for positive rewards, and -1 for negative rewards. A game-over state or life-losing state is considered a terminal state in UCT planning and PGRD training. Evaluation trajectories only consider game-over states as terminal states.

All UCT baseline agents in our experiments sample 100 trajectories of depth 100 frames[3]. The UCB-parameter is 0.1 and the discount factor $\gamma = 0.99$. Following Mnih *et al.* (2015) we use a simple frame-skipping technique to save computations: the agent selects actions on every 4^{th} frame instead of every frame, and the last action is repeated on skipped frames. We did not apply PGRD to ATARI games in which UCT already achieves the highest possible score, such as Pong and Boxing.

Screen image preprocessing. The last four game screen images are used as input for the CNN. The 4 frames are stacked in channels. The game screen images (210×160) are downsampled to 84×84 pixels and gray-scaled. Each image is further preprocessed by pixel-wise mean removal. The pixel-wise mean is calculated over ten game trajectories of a uniformly-random policy.

Convolutional network architecture. The same network architecture is used for all games. The network consists of 3 hidden layers. The first layer convolves 16, 8×8 filters with stride 4. The second hidden layer convolves 32, 4×4 filters with stride 2. The third hidden layer is a full-connected layer with 256 units. Each hidden layer is followed by a rectifier nonlinearity. The output layer has one unit per action.

PGRD-DL learning parameters. After computing the gradients of CNN parameters, we use ADAM to optimize the parameters of the CNN. ADAM is an adaptive stochastic gradient optimization method to train deep neural networks [Kingma and Ba, 2015]. We use the default hyper-parameters of ADAM[4]. We set $\beta = 0.99$ in GARB. Thus the only remaining hyper-parameter is the learning rate for ADAM, which we selected from a candidate set $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ by identifying the rate that produced the greatest sum of performance improvements after training 1000 games on Ms. Pacman and Q*Bert. The selected learning rate of 10^{-4} served as the initial learning rate for PGRD-DL. The learning rate was then lowered during learning by dividing it by 2 every 1000 games.

[3]Normally UCT does planning for every visited state. However, for some states in ATARI games the next state and reward is the same no matter which action is chosen (for example, when the Q*Bert agent is falling) and so UCT planning is a waste of computation. In such states our agents do not plan but instead choose a random action.

5 Experimental Results

We allowed PGRD-DL to adapt the bonus reward function for at most 5000 games or at most 1 million steps, whichever condition was satisfied first. For each game, the final learned reward bonus function was then evaluated in UCT play (using the same depth and trajectory parameters specified above) on 20 "evaluation" games, during which the reward function parameters were held fixed and UCT selected actions greedily according to the action value estimate of root nodes. Note that even though UCT chose actions according to action values greedily, there was still stochasticity in UCT's behavior because of the randomized tree expansion procedure of UCT.

5.1 Learned reward bonuses improve UCT

Table 1 shows the performance of UCT using the objective game-score-based reward (R^O column) and UCT with the learned reward bonus (R^I column). The values show the mean scores over 20 evaluation games and the numbers in parentheses are the standard errors of these means.

The results show that PGRD-DL learns reward bonus functions that improve UCT significantly on 18 out of 25 games. (all except Assault, BankHeist, BeamRider, Berzerk, Centipede, UpNDown and VideoPinball). The mean scores of UCT with learned rewards are significantly higher than UCT using the game-score-based reward for these 18 games. The R^I/R^O column displays the ratio of the mean scores in column R^I to those in column R^O, and so a ratio over 1 implies improvement. These ratio values are plotted as a blue (dashed) curve in Figure 1; the learned rewards improve UCT by a ratio of more than 10 on 4 games (StarGunner, DemonAttack, Q*Bert and Breakout), and a ratio between 2 and 10 on 7 games (Amidar, Alien, Asterix, Seaquest, RoadRunner, MsPacman and BattleZone).

5.2 Comparison considering computational cost

The previous results establish that planning using the learned internal reward bonus can improve the performance of UCT on many ATARI games. However there is some computational overhead incurred in using the deep network to compute the internal reward bonuses (300 ms and 200 ms for UCT with R^I and R^O respectively). Is it worth spending the additional computational resources to compute the internal reward, as opposed to simply planning more deeply or

[4]A discount factor of 0.9 to compute the accumulated discount sum of the first moment vector and 0.999 for the second order moment vector.

Table 1: Performance comparison of different UCT planners. The R^O columns denote UCT agents planning with objective rewards: R^O is depth 100 with 100 trajectories, R^O*(deeper)* is depth 200 with 100 trajectories, and R^O*(wider)* is depth 100 with 200 trajectories. The R^I column shows UCT's performance with internal rewards learned by PGRD-DL, with depth 100 and 100 trajectories. The table entries are mean scores over 20 evaluation games, and in parentheses are the standard errors of these means. The last two columns show the performance ratio of R^I compared to R^O and *max(R^O(deeper),R^O(wider))* (denoted max(R^O(deeper,wider)) in the last column of the table). The R^I, R^O(deeper), and R^O(wider) agents take approximately equal time per decision.

	Mean Game Score (standard error)				*Mean Game Score Ratios*	
Game	R^O	R^I	R^O*(deeper)*	R^O *(wider)*	R^I / R^O	R^I / *max(R^O(deeper,wider))*
Alien	2246 (139)	**12614** (1477)	2906 (387)	1795 (218)	5.62	4.34
Amidar	152 (13)	**1122** (139)	204 (20)	144 (16)	7.39	5.50
Assault	1477 (36)	1490 (32)	1495 (42)	**1550** (59)	1.01	0.96
Asterix	11700 (3938)	60353 (19902)	**99728** (16)	77211 (10377)	5.16	0.61
BankHeist	226 (13)	248 (13)	262 (17)	**284** (19)	1.10	0.88
BattleZone	8550 (879)	**17450** (1501)	13800 (1419)	8450 (1274)	2.04	1.26
BeamRider	2907 (322)	2794 (232)	**2940** (537)	2526 (333)	0.96	0.95
Berzerk	467 (25)	460 (26)	**506** (48)	458 (35)	0.99	0.91
Breakout	48 (14)	**746** (24)	516 (38)	79 (30)	15.47	1.45
Carnival	3824 (240)	**5610** (678)	3827 (173)	3553 (218)	1.47	1.47
Centipede	**4450** (236)	3987 (185)	2771 (231)	4076 (325)	0.90	0.98
DemonAttack	5696 (3316)	**121472** (201)	72968 (13590)	67166 (11604)	21.32	1.66
MsPacman	4928 (513)	**10312** (781)	6259 (927)	4967 (606)	2.09	1.65
Phoenix	5833 (205)	**6972** (371)	5931 (370)	6052 (330)	1.20	1.15
Pooyan	11110 (856)	**20164** (1015)	13583 (1327)	13106 (1605)	1.81	1.48
Q*Bert	2706 (409)	**47599** (2407)	6444 (1020)	4456 (688)	17.59	7.39
RiverRaid	3406 (149)	**5238** (335)	4165 (306)	4254 (308)	1.54	1.23
RoadRunner	8520 (3330)	**32795** (4405)	12950 (3619)	7217 (2758)	3.85	2.53
Robotank	2 (0.26)	3 (0.38)	**6** (0.84)	1 (0.33)	1.66	0.47
Seaquest	422 (19)	**2023** (251)	608 (41)	518 (45)	4.79	3.33
SpaceInvaders	1488 (114)	1824 (88)	**2154** (142)	1516 (166)	1.23	0.85
StarGunner	21050 (1507)	**826785** (3865)	33000 (4428)	22755 (1294)	39.28	25.05
UpNDown	127515 (10628)	103351 (5802)	109083 (9949)	**144410** (38760)	0.81	0.72
VideoPinball	702639 (17190)	736454 (23411)	**845280** (88556)	779624 (90868)	1.05	0.87
WizardOfWor	140475 (7058)	**198495** (225)	152886 (7439)	149957 (7153)	1.41	1.30

broadly with the objective reward? We consider now the performance of two additional baselines that use additional computation to improve UCT without reward bonuses. The first baseline, *deeper UCT*, plans to a depth of 200 frames instead of 100 frames (the number of trajectories is still 100 trajectories). The second baseline, *wider UCT*, samples 200 trajectories to depth 100 (the depth is still 100 frames). Both deeper UCT and wider UCT use about the same computational overhead as the UCT agent with reward bonuses that samples 100 trajectories to depth 100; the time per decision of deeper or wider UCT is slightly greater than UCT with reward bonuses.

The mean scores of the deeper UCT and wider UCT are summarized in Table 1. We take the higher of the mean scores of the deeper and wider UCTs as a useful assessment of performance obtainable using the computational overhead of reward bonuses for better planning, and compare it to the performance of UCT using the learned internal reward R^I. The last column in Table 1 displays the ratio of the mean scores in column R^I to the higher of the wider UCT and deeper UCT scores, and this ratio appears as the red line in Figure 1. Among the 18 games in which reward bonuses improve UCT, reward bonuses outperform even the better of deeper or wider UCT agents in 15 games. These results show that the additional computational resources required to compute the reward bonuses may be better spent in this way than using those resources for more extensive planning.

5.3 The nature of the learned reward-bonuses

What kinds of state and action discriminations does the reward bonus function learn? We consider now a simple summary visualization of the reward bonuses over time, as well as specific examples from the games Ms. Pacman and Q*Bert. The key conclusion from these analyses is that PGRD-DL learns useful game-specific features of state that help UCT planning, in part by mitigating the challenge of delayed reward.

Visualizing the dynamically changing reward bonus across states experienced in game play. Consider first how the learned reward bonus for each action changes as a function of state. Figure 2 shows the varying learned reward bonus values for each of the five actions in Q*Bert for the states experienced during one game play. The action with the highest (and lowest) reward bonus changes many times over the course of the game. The relatively fine-grained temporal dy-

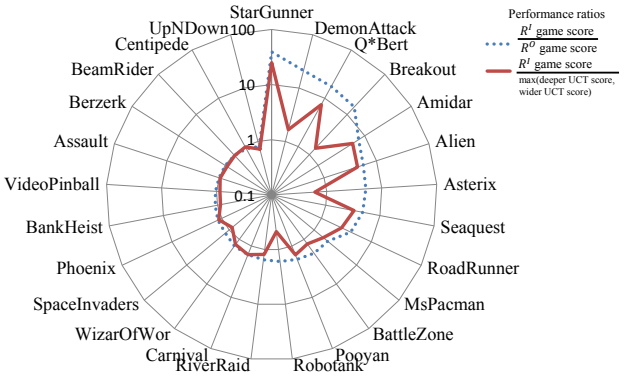


Figure 1: Performance comparison summary. The blue (dashed) curve shows the ratio of the mean game score obtained by UCT planning with the PGRD-DL-adapted internal reward R^I, and the mean game score obtained by UCT planning with the objective reward R^O. The red (solid) curve shows the ratio of the mean game score obtained by UCT with R^I, and the mean game score of UCT planning with R^O with either deeper or more (wider) trajectories (whichever yields the higher score). UCT with the internal reward bonus outperforms the baseline if the ratio value lies outside the circle with radius 1. Games are sorted according to R^I/R^O.

namics of the reward bonuses throughout the game, and especially the change in relative ordering of the actions, provides support for the claim that the learned reward makes game-specific state discriminations—it is not simply unconditionally increasing or decreasing rewards for particular actions, which would have resulted in mostly flat lines across time in Figure 2. We now consider specific examples of the state discriminations learned.

Examples of learned reward bonuses that capture delayed reward consequences. In the game Ms. Pacman, there are many states in which it is important to choose a movement direction that avoids subsequent encounters with enemies (and loss of a "Pacman life"). These choices may not yield differences in immediate reward and are thus examples of delayed

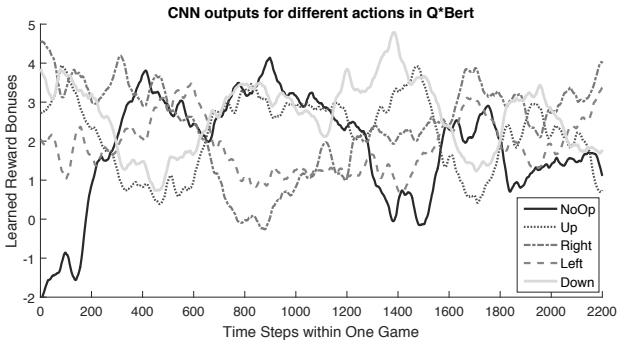
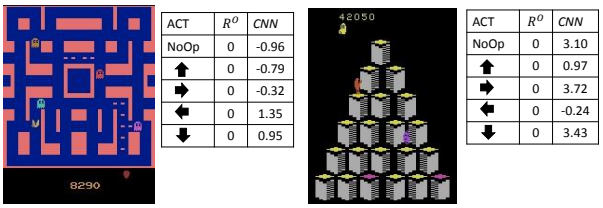


Figure 2: The learned reward bonuses for each of the five actions in Q*Bert for the states experienced during one game play. It is visually clear that different actions have the largest reward-bonus in different states.



ACT	R^O	CNN
NoOp	0	-0.96
↑	0	-0.79
→	0	-0.32
←	0	1.35
↓	0	0.95

ACT	R^O	CNN
NoOp	0	3.10
↑	0	0.97
→	0	3.72
←	0	-0.24
↓	0	3.43

Figure 3: Examples of how the reward bonus function represented by the CNN learns to encourage actions that avoid delayed bad outcomes. *Left*: A state in Ms. Pacman where the agent will encounter an enemy if it continues moving up. The learned reward bonus (under "CNN" in the small table) gives positive reward for actions taking the agent away and negative reward for actions that maintain course; the objective game score does not change (and so R^O is zero). *Right*: A state in Q*Bert where the agent could fall off the pyramid if it moves left and so left is given a negative bonus and other actions are given positive bonuses. The objective reward R^O is zero and indeed continues to be zero as the agent falls.

reward consequences. Similarly, in Q*Bert, falling from the pyramid is a bad outcome but the falling takes many times steps before the "life" is lost and the episode ends. These consequences could in principle be taken into account by UCT planning with sufficient trajectories and depth. But we have discovered through observing the game play and examining specific bonus rewards that PGRD-DL learns reward bonuses that encourage action choices avoiding future enemy contact in Ms. Pacman and falling in Q*Bert (see Figure 3; the figure caption provides detailed descriptions.) The key lesson here is that PGRD-DL is learning useful and interesting game-specific state discriminations for a reward bonus function that mitigates the problem of delayed objective reward.

6 Conclusions

In this paper we introduced a novel approach to combining Deep Learning and Reinforcement Learning by using the former to learn good reward-bonus functions from experience to improve the performance of UCT on ATARI games. Relative to the state-of-the art in the use of PGRD for reward design, we also provided the first example of automatically learning features of raw perception for use in the reward-bonus function, the first use of nonlinearly parameterized reward-bonus functions with PGRD, and provided empirical results on the most challenging domain of application of PGRD thus far. Our adaptation of PGRD uses a variance-reducing gradient procedure to stabilize the gradient calculations in the multi-layer CNN. Our empirical results showed that PGRD-DL learns reward-bonus functions that can significantly improve the performance of UCT, and furthermore that the learned reward-bonus functions can mitigate the computational limitations of UCT in interesting ways. While our empirical results were limited to ATARI games, PGRD-DL is fairly general and we expect it to generalize it to other types of domains. Combining more sophisticated DL architectures, e.g., LSTM [Hochreiter and Schmidhuber, 1997], with RL in learning reward-bonus functions remains future work.

Acknowledgments. This work was supported by NSF grant IIS-1526059. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsor.

References

[Asmuth *et al.*, 2008] John Asmuth, Michael L Littman, and Robert Zinkov. Potential-based shaping in model-based reinforcement learning. In *23rd national conference on Artificial intelligence*, 2008.

[Bellemare *et al.*, 2012a] Marc Bellemare, Joel Veness, and Michael Bowling. Sketch-based linear value function approximation. In *Advances in Neural Information Processing Systems*, 2012.

[Bellemare *et al.*, 2012b] Marc G Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using ATARI 2600 games. In *26th AAAI Conference on Artificial Intelligence*, 2012.

[Bellemare *et al.*, 2013a] Marc Bellemare, Joel Veness, and Michael Bowling. Bayesian learning of recursively factored environments. In *30th International Conference on Machine Learning*, 2013.

[Bellemare *et al.*, 2013b] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

[Bengio, 2009] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2009.

[Chajewska *et al.*, 2000] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *17th National Conference on Artificial Intelligence*, 2000.

[Childs *et al.*, 2008] Benjamin E Childs, James H Brodeur, and Levente Kocsis. Transpositions and move groups in Monte Carlo tree search. In *IEEE Symposium On Computational Intelligence and Games.*, 2008.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 2011.

[Guo *et al.*, 2014] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang. Deep learning for real-time ATARI game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*, 2014.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[Hostetler *et al.*, 2014] Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in Monte Carlo tree search. In *28th AAAI Conference on Artificial Intelligence*, 2014.

[Jiang *et al.*, 2014] Nan Jiang, Satinder Singh, and Richard Lewis. Improving UCT planning via approximate homomorphisms. In *International Conference on Autonomous agents and multi-agent systems*, 2014.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, 2015.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *15th European Conference on Machine Learning*. 2006.

[Lee *et al.*, 2009] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *26th Annual International Conference on Machine Learning*, 2009.

[Lipovetzky *et al.*, 2015] Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical planning with simulators: results on the atari video games. In *International Joint Conference on Artificial Intelligence*, 2015.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, and et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[Ng and Russell, 2000] Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *17th International Conference on Machine Learning*, 2000.

[Ng *et al.*, 1999] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *16th International Conference on Machine Learning*, 1999.

[Oh *et al.*, 2015] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in ATARI games. In *Advances in Neural Information Processing Systems*, 2015.

[Schmidhuber, 1990] Jürgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Neural Networks.*, 1990.

[Schmidhuber, 2015] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 2015.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. In *32nd International Conference on Machine Leanring*, 2015.

[Singh *et al.*, 2010] Satinder Singh, Richard Lewis, Andrew Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2010.

[Sorg *et al.*, 2010a] Jonathan Sorg, Richard L Lewis, and Satinder Singh. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems*, 2010.

[Sorg *et al.*, 2010b] Jonathan Sorg, Satinder Singh, and Richard L Lewis. Internal rewards mitigate agent boundedness. In *27th international conference on machine learning*, 2010.

[Srinivasan *et al.*, 2015] Sriram Srinivasan, Erik Talvitie, and Michael Bowling. Improving exploration in UCT using local manifolds. In *26th Conference on Artificial Intelligence*, 2015.

[Stadie *et al.*, 2015] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[Tesauro, 1994] Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 1994.

[Weaver and Tao, 2001] Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *17th conference on Uncertainty in Artificial Intelligence*, 2001.