Exploratory Gradient Boosting for Reinforcement Learning in Complex Domains

David Abel[‡] **Alekh Agarwal**[†] **Fernando Diaz**[†] **Akshay Krishnamurthy**[†] **Robert E. Schapire**[†]
[‡]Department of Computer Science, Brown University, Providence RI 02912
[†] Microsoft Research, New York NY 10011

Abstract

High-dimensional observations and complex real-world dynamics present major challenges in reinforcement learning for both function approximation and exploration. We address both of these challenges with two complementary techniques: First, we develop a gradient-boosting style, non-parametric function approximator for learning on Q-function residuals. And second, we propose an exploration strategy inspired by the principles of state abstraction and information acquisition under uncertainty. We demonstrate the empirical effectiveness of these techniques, first, as a preliminary check, on two standard tasks (Blackjack and n-Chain), and then on two much larger and more realistic tasks with high-dimensional observation spaces. Specifically, we introduce two benchmarks built within the game Minecraft where the observations are pixel arrays of the agent's visual field. A combination of our two algorithmic techniques performs competitively on the standard reinforcement-learning tasks while consistently and substantially outperforming baselines on the two tasks with high-dimensional observation spaces. The new function approximator, exploration strategy, and evaluation benchmarks are each of independent interest in the pursuit of reinforcement-learning methods that scale to real-world domains.

1 Introduction

Many real-world domains have very large state spaces and complex dynamics, requiring an agent to reason over extremely high-dimensional observations. For example, this is the case for the task in Figure 1, in which an agent must navigate to the highest location using only raw visual input. Developing efficient and effective algorithms for such environments is critically important across a variety of domains.

Even relatively straightforward tasks like the one above can cause existing approaches to flounder; for instance, simple linear function approximation cannot scale to visual input, while nonlinear function approximation, such as deep Q-learning [Mnih *et al.*, 2015], tends to use relatively simple exploration strategies.

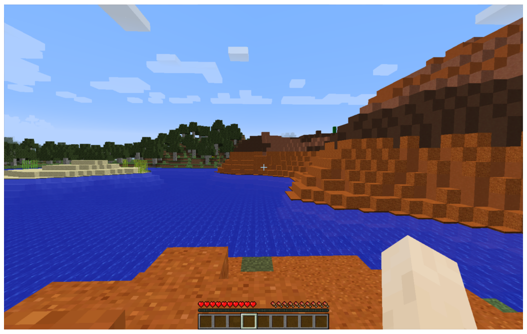

Figure 1: **Visual Hill Climbing**: The agent is rewarded for navigating to higher terrain while receiving raw visual input.

In this paper, we propose two techniques for scaling reinforcement learning to such domains: First, we present a novel non-parametric function approximation scheme based on *gradient boosting* [Friedman, 2001; Mason *et al.*, 2000], a method meant for i.i.d. data, here adapted to reinforcement learning. The approach seems to have several merits. Like the deep-learning based methods [Mnih *et al.*, 2015] which succeed by learning good function approximations, it builds on a powerful learning system. Unlike the deep-learning approaches, however, gradient boosting models are amenable to training and prediction on a single laptop as opposed to being reliant on GPUs. The model is naturally trained on residuals, which was recently shown to be helpful even in the deep learning literature [He *et al.*, 2015]. Furthermore, boosting has a rich theoretical foundation in supervised learning, the theory could plausibly be extended to reinforcement learning settings in future work.

As our second contribution, we give a complementary exploration tactic, inspired by the principle of *information acquisition under uncertainty* (IAUU), that improves over ε-uniform exploration by incentivizing novel action applications. With its extremely simple design and efficient use of data, we demonstrate how our new algorithm combining these techniques, called *Generalized Exploratory Q-learning* (GEQL), can be the backbone for an agent facing highly complex tasks with raw visual observations.

We empirically evaluate our techniques on two standard RL domains (Blackjack [Sutton and Barto, 1998] and n-

chain [Strens, 2000]) and on two much larger, more realistic tasks with high-dimensional observation spaces. Both of the latter tasks were built within the game Minecraft[1] where observations are pixel arrays of the agent's visual field as in Figure 1. The Minecraft experiments are made possible by a new Artificial Intelligence eXperimentation (AIX) platform, which we describe in detail below. We find that on the standard tasks, our technique performs competitively, while on the two large, high-dimensional Minecraft tasks, our method consistently and quite substantially outperforms the baseline.

2 Related Work

Because the literature on reinforcement learning is so vast, we focus only on the most related results, specifically, on function approximation and exploration strategies. For a more general introduction, see [Sutton and Barto, 1998].

Function approximation is an important technique for scaling reinforcement-learning methods to complex domains. While linear function approximators are effective for many problems [Sutton, 1984], complex non-linear models for function approximation often demonstrate stronger performance on many challenging domains [Anderson, 1986; Tesauro, 1994]. Unlike recent approaches based on neural network architectures [Mnih *et al.*, 2015], we adopt gradient boosted regression trees [Friedman, 2001], a non-parametric class of regression models with competitive performance on supervised learning tasks. Although similar ensemble approaches to reinforcement learning have been applied in previous work [Marivate and Littman, 2013], these assume a fixed set of independently-trained agents rather than a boosting-style ensemble.

Our work introduces the interleaving of boosting iterations and data collection. By its iterative nature, our approximation resembles the offline, batch-style training of Fitted Q-Iteration [Ernst *et al.*, 2005] in which a Q-learner is iteratively fit to a fixed set of data. Our algorithm differs in that, at each iteration, the current Q-function approximation guides subsequent data collection, the results of which are used to drive the next update of the Q-function. This adaptive data collection strategy is critical, as the exploration problem is central to reinforcement learning, and, in experiments, our interleaved method significantly outperforms Fitted Q-Iteration.

Our other main algorithmic innovation is a new exploration strategy for reinforcement learning with function approximation. Our approach is similar to some work on state abstraction where the learning agent constructs and uses a compact model of the world [Dietterich, 2000; Li *et al.*, 2006]. An important difference is that our algorithm uses the compact model for exploration only, rather than for both exploration and policy learning. Consequently, model compression does not compromise the expressivity of our learning algorithm, which can still learn optimal behavior, in contrast with typical state-abstraction approaches.

A number of other works propose exploration tactics for RL with function approximation. For example, Oh et al. [2015] train a model to predict the future state from the current state and a proposed action, and then use similarity of the predicted state to a memory bank to inform exploration decisions. Another approach is to learn a dynamics model and to then use either optimistic estimates [Xie *et al.*, 2015] or uncertainty [Stadie *et al.*, 2015] in the model to provide exploration bonuses (see also [Guez *et al.*, 2012]). Lastly, there are some exploration strategies with theoretical guarantees for domains with certain metric structure [Kakade *et al.*, 2003], but this structure must be known a priori, and it is unclear how to construct such structure in general.

3 The GEQL Algorithm

In this section, we present our new model-free reinforcement-learning algorithm, Generalized Exploratory Q-Learning (GEQL), which includes two independent but complementary components: a new function-approximation scheme based on gradient boosting, and a new exploration tactic based on model compression.

3.1 The setting

We consider the standard discounted, model-free reinforcement-learning setting in which an agent interacts with an environment with the goal of accumulating high reward. At each time step t, the agent observes its state $s_t \in \mathcal{S}$, which might be represented by a high-dimensional vector, such as the raw visual input in Figure 1. The agent then selects an action $a_t \in \mathcal{A}$ whose execution modifies the state of the environment, typically by moving the agent. Finally, the agent receives some real-valued reward r_t. This process either repeats indefinitely or for a fixed number of actions. The agent's goal is to maximize its long-term discounted reward, $\sum_{t=1}^{\infty} \gamma^{t-1} r_t$, where $\gamma \in (0, 1)$ is a pre-specified discount factor.

This process is typically assumed to define a Markov decision process (MDP), meaning that (1) the next state reached s_{t+1} is a fixed stochastic function that depends only on the previous state s_t and the action a_t that was executed; and (2) that the reward r_t similarly depends only on s_t and a_t.

For simplicity, we assume in our development that the states are in fact fully observable. However, in many realistic settings, what the agent observes might not fully define the underlying state; in other words, the environment might only be a partially observable MDP. Nevertheless, in practice, it may often be reasonable to use observations as if they actually were unobserved states, especially when the observations are rich and informative. Alternatively for this purpose, we could use a recent past window of observations and actions, or even the entire past history.

3.2 Boosting-based Q-function approximation

Our approach is based on Q-learning, a standard RL technique. Recall that the optimal $Q^\star$ function is defined, on state-action pair (s, a), to be the expected discounted reward of a trajectory that begins at state s, with a the first action taken, and all subsequent actions chosen optimally (to maximize the expected discounted reward). Under general conditions, this function satisfies, for all s, a,

$$Q^\star(s, a) = \mathrm{E}[r + \gamma \cdot \max_{a'} Q^\star(s', a')] \tag{1}$$

[1]https://minecraft.net/

Algorithm 1 GEQL

```
INPUT: Number of episodes $T$, discount factor $\gamma$,
state-collapsing function $\phi$, learning rate schedule $\alpha_t$.
OUTPUT: Policy $\pi$.

1: $\hat{Q}(s,a) = 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$.
2: for $t = 1, \dots, T$ do
3:     Set $s_1$ to start state
4:     for $i = 1, \dots, E$ do
5:         Choose $a_i$ using $\hat{Q}$ and IAUU exploration strategy
6:         Execute $a_i$, observe $r_i$, transition to $s_{i+1}$.
7:     end for
8:     Set $h$ to minimize (2) over regressors in $\mathcal{H}$
9:     $\hat{Q} \leftarrow \hat{Q} + \alpha_t h$
10: end for
11: return $\pi_{\hat{Q}}$ where $\pi_{\hat{Q}}(s) = \arg\max_a \hat{Q}(s,a)$
```

where r is the (random) reward and s' is the (random) next state reached when action a is executed from state s. Like other function-approximation schemes, ours constructs a function $\hat{Q}$ that approximates $Q^\star$ by attempting to fit Eq. (1) on observed data.

As shown in Algorithm 1, we build the function $\hat{Q}$ iteratively in a series of episodes using a gradient-boosting approach. In each episode, we first use $\hat{Q}$ to guide the behavior of the agent, mainly choosing actions that seem most beneficial according to $\hat{Q}$, but occasionally taking exploration steps in a way that will be described shortly. In this way, the agent observes a series of state-action-reward tuples (s_i, a_i, r_i). (For simplicity, we suppose each episode has fixed length E, but the method can easily handle variable length episodes.)

The next step is to use these observations to improve $\hat{Q}$. Specifically, the algorithm fits a regressor h to the residuals between the current approximation and the observed one-step look-ahead, as is standard for boosting methods. That is, $h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is chosen to (approximately) minimize

$$\sum_{i=1}^{E-1} [h(s_i, a_i) + \hat{Q}(s_i, a_i) - (r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'))]^2 \quad (2)$$

over functions h in some class $\mathcal{H}$ of *weak regressors*. As a typical example, the weak regressors might be chosen to be regression trees, which are highly flexible, effective, and efficiently trainable [Mohan *et al.*, 2011]. Once h has been computed, it is added to the function approximator $\hat{Q}$, thus also updating how actions will be selected on future episodes.

This function-approximation scheme has several important advantages over existing approaches. First, by using a non-linear base model, such as regression trees, the agent is able to learn a complex, non-parametric approximation to the $Q^\star$ function, which is crucial for settings with high-dimensional observations. At the same time, the training procedure is computationally efficient as updates occur in batches and trajectories can be discarded after each update. Finally, by interleaving data collection using the learned policy induced by $\hat{Q}$ with residual regression on the data so collected, we intuitively improve the quality and informativeness of the dataset, thus enabling the agent to more effectively and accurately approximate the optimal $Q^\star$ function.

3.3 The IAUU exploration strategy

The second novel component of the algorithm is an exploration technique that borrows ideas from the state-abstraction literature. The technique uses a *state-collapsing function* ϕ which maps each state s to one of m clusters, where m is relatively small. In GEQL, this function is trained by clustering a large dataset of states, for instance, using the k-means algorithm (as in all our experiments), and associating each state s with the nearest cluster center. Ideally, an optimality-preserving compression would be used, such as those characterized by Li *et al.*, [2006], but such abstractions are not always easily computable.

The main idea of our technique is to keep track of how often each action has been taken from states in each of the clusters, and to choose exploratory steps that encourage the choice of actions that were taken less often in the current cluster. Thus, on each episode, for each of the m clusters c, and for each action a, we maintain a count $M(c, a)$ of how often action a was taken from states s in cluster c, i.e., for which $\phi(s) = c$. For each state s, this table induces a Gibbs distribution over actions defined by $p(a|s) \propto \exp\{-\rho M(\phi(s), a)\}$ where $\rho > 0$ is a temperature parameter that controls the uniformity of this distribution. In concert with the current function approximator $\hat{Q}$, we use this distribution to define a randomized choice of actions at each step of the episode. Specifically, when in the current state s, with probability $1 - \varepsilon$, we choose to act greedily, selecting the action a that maximizes $\hat{Q}(s, a)$; and with probability ε, we take an exploration step, sampling a from $p(a|s)$. We call this exploration strategy *Information Acquisition Under Uncertainty* or IAUU.

This strategy shares some of the benefits of the work on state abstraction without suffering from the drawbacks. The main advantage is that the state-collapsing function promotes applying infrequently-taken actions, thereby encouraging the agent to visit new regions of the state space. However, in contrast with the state-abstraction literature, the method remains robust to misspecification of the state-collapsing function since we use it only for exploration so that it does not compromise the agent's ability to learn optimal behavior. Finally, IAUU exploration adds minimal computational overhead as the space needed is $O(m|\mathcal{A}|)$ and the additional running time per action remains $O(|\mathcal{A}|)$.

The IAUU exploration tactic is not attached to the function approximation scheme, so it can also be used with Q-learning in the tabular setting, i.e., when $\hat{Q}$ is maintained explicitly as a table. In this case, there are a number of modifications to the exploration strategy under which the convergence properties of Q-learning can be retained. One option is to modify the exploration distribution $p(a|s)$ to mix in a vanishingly small amount of the uniform distribution. Another option is to only count exploration steps when updating the state-visitation table M. In both cases, each action is taken from each state infinitely often, and this property suffices to ensure convergence of Q-learning in the tabular setting.

4 Experiments on Standard Benchmarks

This section details our evaluation on the two standard reinforcement learning benchmarks of Blackjack and n-Chain.

4.1 Blackjack

We conducted experiments on the Blackjack domain, implemented exactly as defined in Section 5.1 of Sutton and Barto [1998]. In this experiment, we test the hypothesis that GEQL will improve over standard baselines even in fairly simple domains without high-dimensional visual input. In particular, Blackjack is fully observable, has low noise, low dimension, short episodes, and a small action space.

Algorithms and Parameters: For GEQL, we used depth-2 regression trees as the *weak regressors*, fit using Python's *scikit-learn* package [Buitinck *et al.*, 2013]. To test and isolate the effectiveness of our incremental boosting approach (henceforth `booster`), we compared GEQL against three baseline function approximators:

1. Q-learning with a linear approximator (`linear`)
2. Q-learning with a Batch Boosted Regression approximator, similar to Fitted Q-Iteration (`batchboost`)
3. Q-learning with a Batch Random Forest Regression approximator (`forest`).

Each approximator used the same set of features for all experiments. The two batch-based regression approaches were trained after every 50 episodes. Similar to GEQL, we set the depth of the regression trees for each of the batch approaches to two. The `batchboost` and `forest` approximators used the same number of total trees as GEQL, but were trained in batch. That is, if we run for 500 episodes, then each tree based approach gets a total of 500 trees. For `batchboost` and `forest`, all the 500 trees are retrained on 50 episodes worth of data, while `booster` adds a new tree every episode.

We ran all function approximators with ϵ-uniform exploration and with IAUU exploration. Across all experiments, we set $\epsilon_0 = 0.4$, $\alpha_0 = 0.15$, $\gamma = 0.95$, and decayed so that, during episode t, $\epsilon_t = \epsilon_0/(1 + 0.04t)$, and $\alpha_t = \alpha_0/(1 + 0.04t)$. The state clustering function used for IAUU was learned for each individual task by randomly sampling states via a random policy.

Results: Figure 2 shows results from 100 trials run for 500 episodes each. The results indicate that during these 500 episodes, very minimal learning occurred for the linear approximator, while the gradient-boosting approximator was able to learn to play far more effectively (yielding a statistically significant improvement in performance). The two batch approximators demonstrated some learning, though the gradient-boosting approximator far outperformed both of them. However, the exploration tactic had a negligible effect in this domain, likely due to the small action space (two actions), and short episodes (typically between one and two actions per episode). The brevity of each episode may also explain why the linear approximator learned so little in this many episodes.

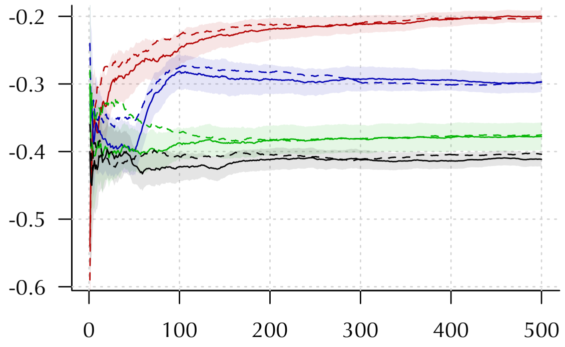

Figure 2: **Blackjack**: Running average per-episode reward with error band denoting 2 standard errors for IAUU runs (**Legend:** red: `booster`, blue:`forest`, green: `batchboost`, black: `linear`; solid: IAUU, dashed: uniform).

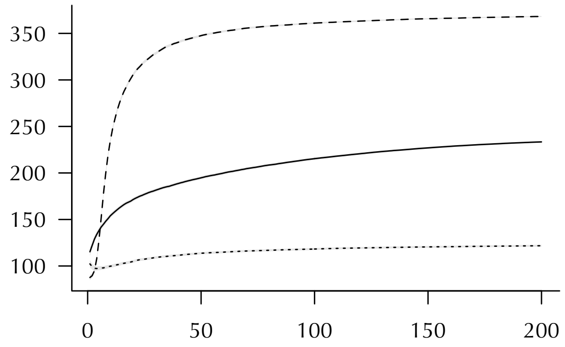

Figure 3: n-**Chain**: Running average per-episode reward (**Legend:** IAUU: solid, RMAX: dashed, Q-learning: dotted)

4.2 n-Chain

We conducted experiments on the n-Chain domain from Strens [2000]. In this domain, there are n states numbered $0, \ldots, n-1$, each with two actions available: applying the *forward* action in state i advances the agent to state $i+1$, while the *return* action moves the agent to state 0. Applications of the *forward* action provide zero reward in all states except transitions to state $n-1$, which provides reward 100. Applications of the *return* action receive reward 2 for all transitions. Both actions are also stochastic, so that with probability 0.2, they have the opposite effect. This task poses a challenging exploration problem since the agent must avoid greedily taking the *return* action to learn optimal behavior by exploring all the way to the last state in the chain. We used $n = 5$, which is a typical setting for this task.

Algorithms: As this is a tabular problem, we evaluated only tabular methods. We used a tabular Q-learner with uniform exploration, IAUU exploration, and the RMAX algorithm [Brafman and Tennenholtz, 2003], which is a model based algorithm with strong sample complexity guarantees.

Results: Figure 3 displays results from 5000 trials. Unsurprisingly, RMAX significantly outperformed the two tabular Q-learning strategies used here since it is designed to seek out all state-action applications helping it to quickly discover the high reward at the end of the chain. Q-Learning with ϵ-uniform exploration, on the other hand, will only discover the high reward in the last state of the chain with exponentially small probability since the agent will favor the greedy action and must repeatedly explore to advance down the chain. The IAUU exploration is between these two extremes; it generalizes to extremely large state spaces unlike RMAX, but provides more effective exploration than ϵ-uniform.

5 Experiments on Visual Domains

This section describes our empirical evaluation on two highly challenging problems in which the agent must reason over raw RGB images. We used the Minecraft game platform to provide environments for the two tasks. Minecraft is a 3D blocks world in which the player can place blocks, destroy blocks, craft objects, and navigate the terrain. The size of the underlying state space grows exponentially with the number of blocks allowed in the world, which is typically on the order of millions, making planning and learning infeasible for tabular approaches. Moreover, there are day and night cycles and weather cycles that dramatically alter the visual world, as well as animals that roam the world, and underwater environments. The size of the state space and these complex visual elements pose significant challenges to learning from raw visual inputs. Minecraft has previously been used in planning research [Abel *et al.*, 2015], and has been advocated as a general artificial intelligence experimentation platform [Aluru *et al.*, 2015].

Our experimental results here are enabled by a recently developed Artificial Intelligence eXperimentation (AIX) platform, designed for experimentation within Minecraft. AIX provides a flexible and easy-to-use API to the Minecraft engine that allows for full control of an agent, including action execution and perception, as well as precise design of the Minecraft world the agent operates in (i.e. specific block placement, day and night cycles, etc.). For the Visual Grid World task, we hand crafted a grid world inspired by the classical reinforcement learning task [Russell and Norvig, 1995], while the Visual Hill Climbing task was built by Minecraft's random world generator. With AIX running, Minecraft runs around 30 frames per second, though our agents only execute around 2 actions per second.

5.1 Implementation Details

Visual System: In a rich 3D world such as Minecraft, reinforcement learning agents require additional preprocessing of raw RGB observation. We employ classical computer vision techniques for preprocessing of the raw visual images from the Minecraft game. The input to the visual pipeline is a 320×240 image of the Minecraft agent's view, with all distracting UI elements (such as the toolbar) removed.

We use data from a five-minute random exploration of the Minecraft world (in which the agent takes a random action every time step). Specifically, for every 20th frame received from the game engine, the agent performs SURF key-point detection [Bay *et al.*, 2008], and stores the set of key points in a dataset. After five minutes, the agent performs k-means clustering on the space of all key points to reduce the dimensionality of the key-point space of interest. This training is done offline before experiments are conducted. The same visual system is used for all algorithms. The system is trained separately for each task.

During the task, for each new frame the agent receives, it partitions the frame into a 3×3 grid. For each partition, the agent finds the key point that is most similar to each of the key-point centers, and computes its distance. This distance is used as a feature for that cell. The ultimate feature vector is the concatenation of each partition's key-point distances (so if $k = 10$, there will be 10 features per 3×3 partitions for a total of 90 features plus a bias term).

Occupancy Grid: Since the RGB image available to the vision system is based on the agent's first-person perspective (see Figure 1), the agent's immediate surroundings are partially occluded. As immediate surroundings are crucial to decision making, we augment the agent with a $4 \times 3 \times 3$ *occupancy grid* of the cells the agent is touching. This occupancy grid contains a 1 if the corresponding cell adjacent to the agent contains a block that is solid (e.g. dirt, grass, stone, etc.) or water, and a 0 otherwise. These binary features, along with the key point distances from the vision system, comprise the state feature vector available to the agent.

State-Collapsing Function: For the state-collapsing function ϕ (see Section 3.3), we train another k-means instance that maps a given state object to a lower dimension. We let the Minecraft agent explore for another five minutes, saving every 20th frame. For each saved frame, the agent computes the feature vector as described above and concatenates the occupancy grid of the agent's surrounding cells, storing these vectors in a data set. After five minutes, the agent performs k-means on the data set of features to reduce the feature space to a lower dimension. The training is also done offline before experiments are conducted, and all IAUU algorithms use the same state-collapsing function for each task.

During learning, IAUU agents evaluate the state-collapsing function by mapping the current state's feature vector to the nearest cluster center in this k-means instance.

5.2 Visual Grid World

The first task we consider is the *Visual Grid World* task. Here, the environment consists of a 6×6 grid. The agent always starts at $(0, 0)$ and must navigate to $(5, 5)$ using only movements North, East, South, and West. There is no rotation (i.e. the agent is always facing North). At each state, the agent observes the raw bitmap image of the agent's view (Figure 4), which is preprocessed using the vision system and augmented with the occupancy grid. The reward function is the negation of the agent's Euclidean distance away from the goal. For example, if the agent is a distance of 5 from the goal, the agent receives -5 reward. All transitions are deterministic. The optimal policy will achieve a reward of roughly -31 for directly proceeding to the goal.

Algorithms and Parameters: As in Blackjack, we used four function-approximation schemes (`linear`, `batchboost`, `forest`, and our interleaved gradient-boosting approach denoted as `booster`), and two exploration strategies (ϵ-uniform and IAUU). Regression problems are solved after each episode for the linear and gradient approximators using data from only the most recent episode. The `batchboost` and `forest` approximators are completely retrained every five episodes, using only the most recent five episodes of data. As before, the depth and number

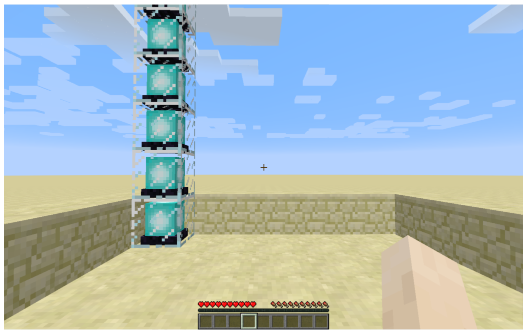

Figure 4: **Visual Grid World**: The agent is rewarded for navigating to the blue pillar while receiving raw visual input.

of trees in all tree-based methods is set to two and the total number of episodes (100). Other parameters are set as in the Blackjack experiment.

Results: Figure 5 shows results from five trials for 100 episodes, where each episode was at most 40 seconds. Results demonstrate that the gradient-boosting approximator led to dramatically faster learning on this task than the linear approximator (statistically significant at the 0.05 level). `booster` also outperforms the two batch approximators with similar statistical significance, apart from the `batchboost` baseline with uniform exploration (which still underperforms, though insignificantly). While the combination of gradient boosting and IAUU exploration has the best average performance, the improvement over gradient boosting with ϵ-uniform is not statistically significant. Due to the speed of the Minecraft engine, scaling these experiments is challenging. Nevertheless, these results clearly show that GEQL (`booster` with IAUU exploration) is a major improvement over all previous baselines for this challenging task.

Since we know the reward of the optimal policy in this case, we also checked the reward for the policy learned by `booster` at the end of 100 episodes. We found that the average rewards of `booster` with IAUU and uniform explorations were -34.22 and -82.77 respectively. Note that the optimal policy gets a reward close to -31, while the best baseline of `batchboost` with uniform exploration picks up a reward around -119.5 in this task. Hence, we can conclude that GEQL learns a significantly better policy than other baselines on the task.

5.3 Visual Hill Climbing

The second task, which we call *Visual Hill Climbing*, is especially difficult. At its core, it is a variant of non-convex optimization in a 3D environment. As with Visual Grid World, the state is the preprocessed raw RGB bitmap image of the agent's view along with the occupancy grid. The agent must climb the highest hill it can find by navigating a highly complex 3D world that includes animals, lakes, rivers, trees, caverns, clouds, pits, and a variety of different terrain types. An example snapshot of the agent's task is pictured in Figure 1. This is an especially challenging exploration problem as the agent's view is restricted to a finite horizon, and the agent can only see in one direction at a time. Large hills may also be partially occluded by trees, animals, the agent's own arm, and other hills. Scaling some hills involves steps with jumps larger than agent can make in one action, meaning the agent cannot scale the hill even if it gets there.

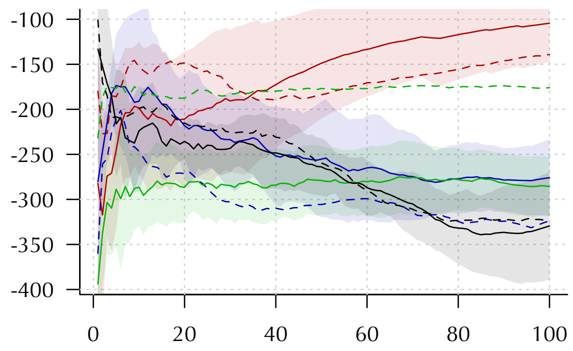

Figure 5: **Visual Grid World**: Running average per-episode reward with error bands on IAUU versions, denoting 2 standard errors (**Legend:** red: `booster`, blue:`forest`, green: `batchboost`, black: `linear`; solid: IAUU, dashed: uniform).

The agent may move forward in the direction it is facing, turn left 90 degrees, turn right 90 degrees, jump up two units, or perform a combination that jumps up two units and then moves forward. The agent receives $+N$ reward for increasing its elevation by N units, $-N$ reward for decreasing its elevation by N units, and a small reward proportional to its current height relative to "sea level". So, if the agent's initial elevation is 0, then if the agent reaches an elevation of 10, it will receive an additional $\frac{10}{1000}$ reward. All transitions are deterministic, but the state is only partially observable, so repeated application of an action at a particular observation may result in dramatically different future observations.

Algorithms and Parameters: We used the same algorithms and parameter settings as for the Visual Grid World.

Results: Figure 6 displays results from ten trials for 100 episodes, where each episode was exactly 40 seconds. Again, results indicate that the gradient booster is able to learn a far better policy than any of the other approximators, and that the IAUU exploration tactic helps. Indeed, only for the gradient booster does non-negligible learning occur; given how complex this domain is, it is extremely promising that we can learn a reasonable policy at all (i.e. that the agent is able to learn a policy that leads to positive reward, suggesting that the agent is climbing a substantial amount).

To visualize the performance of our agent better, we plot the elevation profile of the policy learned by `booster` with IAUU exploration over time in Figure 7. We notice that while the agent barely increases its elevation in the initial quarter of the episodes, it is reliably reaching much better altitudes in the last quarter indicating that it does identify the key to succeeding at this task.

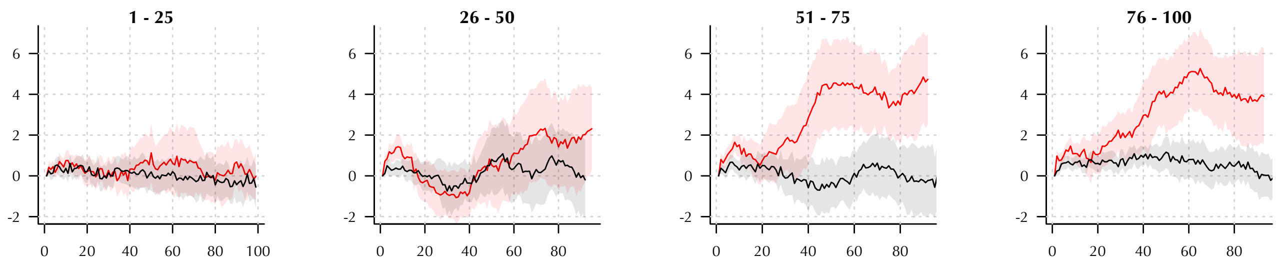

Figure 7: **Visual Hill Climbing Elevation Profile:** Elevation throughout an episode averaged over the episodes indicated in the range (i.e. the first, second, third, and fourth 25 episodes) for `booster` (red) and `forest` (black), both with IAUU exploration. All elevations are with respect to the starting elevation. The elevation of an effective agent should go up.

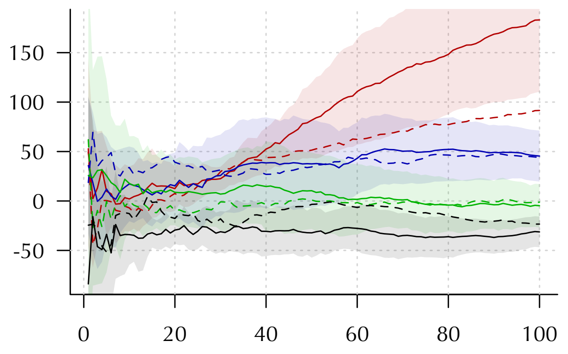

Figure 6: **Visual Hill Climbing**: Running average per-episode reward with error bands on IAUU versions, denoting 2 standard errors (**Legend:** red: `booster`, blue:`forest`, green: `batchboost`, black: `linear`; solid: IAUU, dashed: uniform).

6 Discussion

In this paper, we have described novel approaches to function approximation as well as exploration in reinforcement learning, and evaluated it on challenging tasks implemented within Minecraft. The encouraging performance of our methods suggests several other exciting avenues for future research. Empirically, the performance of gradient boosting coupled with its favorable computational properties appears very promising, and it would be interesting to compare with more computationally-intensive deep-learning based approaches in future work. Extending the existing theory of gradient boosting from supervised to reinforcement learning is also a natural question.

In terms of exploration, while IAUU certainly improves over ϵ-uniform, it is still limited by the state-collapsing function used, and can be suboptimal if the least-frequent action also happens to be bad. It remains challenging to find better alternatives that are tractable for reinforcement learning with real-time decisions and high-dimensional observations.

Finally, Minecraft provides an attractive framework to develop visual versions of standard RL tasks. We show two examples here, but the opportunity to translate other tasks that stress and highlight various learning abilities of an agent, as well as understand the gulf to human performance is a very exciting proposition for future work.

Acknowledgement: We would like to thank Katja Hoffman, Matthew Johnson, David Bignell, Tim Hutton, and other members of the AIX platform development team, without whose support this work would not be possible.

References

[Abel *et al.*, 2015] David Abel, David Ellis Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2015.

[Aluru *et al.*, 2015] Krishna Aluru, Stefanie Tellex, John Oberlin, and James MacGlashan. Minecraft as an experimental world for ai in robotics. In *AAAI Fall Symposium*, 2015.

[Anderson, 1986] Charles William Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts Amherst, 1986.

[Bay *et al.*, 2008] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 2008.

[Brafman and Tennenholtz, 2003] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2003.

[Buitinck *et al.*, 2013] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 2000.

[Ernst *et al.*, 2005] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. In *Journal of Machine Learning Research*, 2005.

[Friedman, 2001] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

[Guez *et al.*, 2012] Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[Kakade *et al.*, 2003] Sham Kakade, Michael Kearns, and John Langford. Exploration in metric state spaces. In *International Conference on Machine Learning*, 2003.

[Li *et al.*, 2006] Lihong Li, Thomas J Walsh, and Michael L. Littman. Towards a Unified Theory of State Abstraction for MDPs. *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006.

[Marivate and Littman, 2013] Vukosi Marivate and Michael Littman. An ensemble of linearly combined reinforcement-learning agents. In *AAAI Conference Late-Breaking Papers*, 2013.

[Mason *et al.*, 2000] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*, 2000.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[Mohan *et al.*, 2011] Ananth Mohan, Zheng Chen, and Kilian Q Weinberger. Web-search ranking with initialized gradient boosted regression trees. In *Yahoo! Learning to Rank Challenge*, 2011.

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

[Russell and Norvig, 1995] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice-Hall, Englewood Cliffs, 1995.

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

[Strens, 2000] Malcolm Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[Sutton, 1984] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[Tesauro, 1994] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 1994.

[Xie *et al.*, 2015] Christopher Xie, Sachin Patil, Teodor Moldovan, Sergey Levine, and Pieter Abbeel. Model-based reinforcement learning with parametrized physical models and optimism-driven exploration. *arXiv:1509.06824*, 2015.