Boosting Nonparametric Policies*

Yang Yu Peng-Fei Hou Qing Da Qian Yu
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
{yuy, houpf, daq, qiany}@lamda.nju.edu.cn

ABSTRACT

Learning complex policies is a key step toward real-world applications of reinforcement learning. While boosting approaches have been widely applied in state-of-the-art supervised learning techniques to adaptively learn nonparametric functions, in reinforcement learning the boosting-style approaches have been little investigated. Only a few pieces of previous work explored this direction, however theoretical properties are still unclear and empirical performance is quite limited. In this paper, we propose the PolicyBoost method. It optimizes a finite-sample objective function, which leads to maximization of the expected total reward, by employing the GradientBoost approach. Experimental results verify the effectiveness as well as the robustness of PolicyBoost, even without feature engineering.

Categories and Subject Descriptors

H.4 [**Machine learning**]: Reinforcement learning

General Terms

Algorithms

Keywords

Policy gradient, Boosting, nonparametric model

1. INTRODUCTION

In reinforcement learning, an agent learns from trial-and-error feedback from its environment, and produces a state-to-action policy attempting to maximize its long-term total reward [34]. A good reinforcement learning algorithm would be able to generalize well from limited feedbacks, sharing similar principle with supervised learning algorithms. In supervised learning [24], an algorithm is given fixed training examples and expected to build a model that correctly predicts unseen instances. We may learn from successful supervised learning algorithms to design strong reinforcement learning algorithms.

*This research was supported by the NSFC (61375061, 61223003), Foundation for the Author of National Excellent Doctoral Dissertation of China (201451).

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), John Thangarajah, Karl Tuyls, Stacy Marsella, Catholijn Jonker (eds.), May 9–13, 2016, Singapore.*

A notable off-the-shelf supervised learning algorithm family is called *boosting*, with representatives such as AdaBoost [10], LogitBoost [11], GradientBoost [12], etc. Most of them share a common routine [22] that trains an additive combination of models via gradient decent in some function space. The success of boosting algorithms is supported by both theoretical guarantees and many successful real-world applications. Particularly, boosting approaches have a strong generalization ability and a good adaptivity to nonparametric and highly nonlinear models, which are also quite appealing properties in reinforcement learning.

Despite the powerful performance of boosting approaches verified in supervised learning, there are few previous studies investigating boosting-like approaches in reinforcement learning. As far as we are aware, the only existing work is the NPPG [17], by which a policy is formed from an *additive model* like the boosting approaches. It trains the model to directly maximize the total reward objective function via the gradient ascent in a function space. However, we found that directly maximizing the objective function on a small training trajectory set results in overfitting due to optimizing only the probability of these trajectories, which can be useful only when the policy is near optimal already. Similar to boosting approaches, the NPPG method has a strong learning ability, which makes the overfitting problem even more severe. Thus NPPG can only work on simple tasks.

In this paper, we firstly derive the finite-sample functional gradient, the core of boosting methods, for maximizing the expected total reward. We then show that the functional gradient method drives policy convergence, and fit the observed best actions. However, it may overfit the observed samples, we thus make a correction to the finite-sample objective to alleviate the overfitting problem. The proposed PolicyBoost optimizes the objective using GradientBoost. Moreover, for a better assessment of the gradient, it is quite useful to include some good trajectories in the training set. Thus unlike NPPG that throws away past samples, PolicyBoost maintains a pool of experienced good trajectories.

We conduct empirical studies of PolicyBoost on several domains. Experiment results reveal that NPPG suffers from the overfitting problem while PolicyBoost does not. Moreover, even using the raw feature representation of states and actions, PolicyBoost is shown to be not only effective in achieving good policies on the tested domains with raw state features, but also highly stable to its configuration parameters, which is desired in practical applications.

The remainder of the paper starts with a section introducing the related work. The PolicyBoost is then presented in

a section after, which is followed by the section of empirical studies. The final section suggests future directions.

2. BACKGROUND

2.1 Reinforcement Learning and Policy Gradient

In reinforcement learning, an autonomous agent is put in an environment. The environment involves a state space $\mathcal{X}$, an action space $\mathcal{A}$, a transition probability P such that $P(s'|s,a)$ gives the probability of being in state s' after taking action a on state s, and a reward function $r : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that gives a reward value for every step (i.e., a transition from a state to another). In this paper, we consider the state space being a vector space $\mathcal{X} = \mathbb{R}^n$ of n features. A policy π is a decision function that gives a probability distribution for actions, with $\pi(a|\boldsymbol{s})$ denoting the probability of choosing action a at state $\boldsymbol{s}$ by the policy. The agent seeks a policy maximizing its long-term total reward. The long-term total reward can be evaluated by the expected T-step reward $\rho(\pi) = \lim_{t\to\infty} E\{r_1 + r_2 + \cdots + r_t \mid \pi\}/t$, and the expected discounted reward, $\rho(\pi) = E\{\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid \pi\}$, where r_t is the reward at step t and γ is the discount factor.

Policy gradient methods [38, 18, 28, 25, 29] are a branch of reinforcement learning approaches, which derive a policy by directly maximizing the total reward via gradient ascent and can avoid the *policy degradation* problem of the value function estimation approaches [5]. Considering a finite execution of a policy, i.e., T-step horizon, all the trajectories with length T constitute the trajectory space $\mathcal{T}$. For each trajectory $\boldsymbol{\tau} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_T)$, let $R(\boldsymbol{\tau})$ be its trajectory-wise reward, for the T-step setting $R(\boldsymbol{\tau}) = \frac{1}{T-1}\sum_{i=1}^{T-1} r(\boldsymbol{s}_i, \boldsymbol{s}_{i+1})$ and for the discounted setting $R(\boldsymbol{\tau}) = \sum_{i=1}^{T-1} \gamma^{t-1} r(\boldsymbol{s}_i, \boldsymbol{s}_{i+1})$. The expected total reward is then equivalent to,

$$\rho(\pi) = \int_{\mathcal{T}} p^{\pi}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) \, \mathrm{d}\boldsymbol{\tau}, \tag{1}$$

where the probability of the trajectory $p^{\pi}(\boldsymbol{\tau})$ is

$$\begin{aligned} p^{\pi}(\boldsymbol{\tau}) &= p(\boldsymbol{s}_1^{\boldsymbol{\tau}}) \prod_{t=1}^{T-1} p^{\pi}(\boldsymbol{s}_{t+1}^{\boldsymbol{\tau}} \mid \boldsymbol{s}_t^{\boldsymbol{\tau}}) \\ &= p(\boldsymbol{s}_1^{\boldsymbol{\tau}}) \prod_{t=1}^{T-1} \int_{\mathcal{A}} p(\boldsymbol{s}_{t+1}^{\boldsymbol{\tau}} \mid \boldsymbol{s}_t^{\boldsymbol{\tau}}, a) \pi(a \mid \boldsymbol{s}_t) \, \mathrm{d}a. \end{aligned}$$

Several policy gradient methods have been proposed, and most of them consider the policy as a linear function with parameter vector $\boldsymbol{\theta}$, i.e., $\pi(a|\boldsymbol{s}) = g(\boldsymbol{\theta}^{\top}\boldsymbol{s})$ for some probability function g. Then a general expression of the gradient of the expected total reward w.r.t. $\boldsymbol{\theta}$ is $\nabla_{\boldsymbol{\theta}}\rho(\pi_{\boldsymbol{\theta}}) = \int_{\boldsymbol{\tau}\in\mathcal{T}} \nabla_{\boldsymbol{\theta}} p^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$. Once the gradient has been estimated, the parameter can be updated by the gradient ascent $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \nabla_{\boldsymbol{\theta}}\rho(\pi_{\boldsymbol{\theta}_t})$ with a learning rate η_t.

A straightforward way to estimate the gradient is via the finite difference [27], which estimates the gradient by sampling some neighbors of the current parameter vector. The REINFORCE [38] uses log-likelihood ratio trick $\nabla_{\boldsymbol{\theta}} p^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = p^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log p^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})$, such that the gradient of the objective $\nabla_{\boldsymbol{\theta}}\rho(\pi_{\boldsymbol{\theta}})$ can be estimated on a sample of trajectories $S = \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_m\}$ as that, for the k-th dimension,

$$\begin{aligned} \nabla_{\theta_k}\rho_S(\pi_{\theta_k}) &= \frac{1}{m}\sum_{i=1}^{m} \nabla_{\theta_k} \log p^{\pi_{\theta_k}}(\boldsymbol{\tau}_i) R(\boldsymbol{\tau}) \\ &= \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{T-1} \nabla_{\theta_k} \log p^{\pi_{\theta_k}}(\boldsymbol{s}_{t+1}^{\boldsymbol{\tau}_i} \mid \boldsymbol{s}_t^{\boldsymbol{\tau}_i}) R(\boldsymbol{\tau}). \end{aligned}$$

Moreover, it usually plugs a constant b into the reward as $(R(\boldsymbol{\tau}) - b)$, aiming to minimize the variance of the gradient for stability [15]. In the actor-critic framework, policy gradient has also been employed. For example, considering the stationary distribution of states, the gradient can be [35]

$$\nabla_{\boldsymbol{\theta}}\rho(\pi_{\boldsymbol{\theta}}) = \int_{\mathcal{X}} d^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) \sum_{a\in\mathcal{A}} Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, a) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|\boldsymbol{s}) \mathrm{d}\boldsymbol{s}, \tag{2}$$

where $d^{\pi}(\boldsymbol{s})$ is the probability of state s under the stationary distribution, and $Q^{\pi}(\boldsymbol{s}, a)$ is the state-action value. More variants are proposed in different aspects and settings (e.g. [4, 14, 30, 6]).

2.2 Supervised Learning and Boosting

Boosting [33] is a family of algorithms for supervised learning. In supervised learning, we consider an input space $\mathcal{X}$, an output space $\mathcal{Y}$ (e.g. $\mathcal{Y} = \{-1, +1\}$), an underlying distribution $\mathcal{D}$ over $\mathcal{X}$ and an underlying ground truth function $f^* : \mathcal{X} \to \mathcal{Y}$. We are given only a finite set of samples $S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ drawn i.i.d. from $\mathcal{D}$, together with their labels $y_i = f^*(\boldsymbol{x}_i)$ assigned by the underlying ground truth function f^*. With the given samples, a supervised learning algorithm searches for a proper hypothesis $h : \mathcal{X} \to \mathcal{Y}$ in a hypothesis space $\mathcal{H}$. The properness of the hypothesis can be commonly measured by the generalization zero-one error, i.e., $\epsilon_{\mathcal{D}}(h) = E_{\boldsymbol{x}\in\mathcal{D}}[\boldsymbol{I}(h(\boldsymbol{x}) \neq f^*(\boldsymbol{x}))]$.

An interesting question on the supervised learning is that whether it is possible to improve a learning algorithm, which is slightly better than random guess to have a strong generalization ability on learnable tasks [16], and boosting was the constructive positive answer to this question [32]. which is a convex spanning of a base hypothesis space $\mathcal{H}_{base}$ that can be the hypothesis space of a base learning algorithm. To search in $\mathcal{H}$, most boosting algorithms employ some kind of gradient descent methods to minimize some surrogate losses of the zero-one error on the training set. Despite the different implementations, they share a common procedure described as follows. First, a base learning algorithm is involved to produce the initial hypothesis h_1 from the training set, and let the initial combined hypothesis be $H_1 = h_1$. Then an iteration is repeated $T - 1$ times. In iteration $t = 2, \ldots, T$, it calculates the gradient value for H_{t-1} point-wisely on every training example, such that a new training set is formed. After that the base learning algorithm is invoked to train h_t from the new training set, and a coefficient α_t is searched to minimize the loss of the combination $H_t = H_{t-1} + \alpha_t h_t$. After all the iterations, a hypothesis $H_T = \sum_{i=1}^{T} \alpha_i h_i$ is obtained, which is called an *additive model*.

Different boosting algorithms mainly differ in the employed gradient methods and surrogate loss functions. They easily produce highly nonlinear models with small generalization error, and thus are among the state-of-the-art learning approaches and have been widely applied (e.g. [37, 21]).

2.3 NPPG Method

The NPPG (non-parametric policy gradient) method [17] is, to the best of our knowledge, the only previous boosting approach for reinforcement learning, following the GradientBoost [12].

In NPPG, A policy $\pi(\boldsymbol{s}, a)$ is represented as $g(\Psi(\boldsymbol{s}, a))$ with some potential function Ψ. For discrete action spaces, g can be the Gibbs Sampling function (i.e., the logistic regression function), $\pi_\Psi(a|\boldsymbol{s}) = \frac{\exp(\Psi(\boldsymbol{s},a))}{\sum_{a'}\exp(\Psi(\boldsymbol{s},a'))}$, and for continuous action spaces, g can be the Gaussian function with parameter σ, $\pi_\Psi(a|\boldsymbol{s}) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\Psi(\boldsymbol{s})-a)^2}{\sigma^2}\right)$. The potential function Ψ is an additive model $\Psi = \sum_{t=1}^{T} h_t$, where the component function h_t is to be trained iteratively. NPPG employs the gradient of Eq.(2) directly, except that the gradient is with respect to the potential function,

$$\nabla_\Psi \rho(\pi_\Psi) = \int_{\mathcal{X}} d^{\pi_\Psi}(\boldsymbol{s}) \sum_{a\in\mathcal{A}} Q^{\pi_\Psi}(\boldsymbol{s}, a)\nabla_\Psi \pi_\Psi(a|\boldsymbol{s})\mathrm{d}\boldsymbol{s}.$$

Given the current potential function $\Psi_t = \sum_{i=1}^{t} h_t$, the function can be updated as

$$\Psi_{t+1} = \Psi_t + \eta_t \nabla_\Psi \rho(\pi_{\Psi_t}).$$

However, different with the gradient of linear vectors, the gradient in a function space $\nabla_\Psi \rho(\pi_{\Psi_t})$ is also a function but can not be explicitly expressed. We can only know the gradient value on the samples. Then the point-wise estimation [12] is used to approximate the gradient function via regression learning algorithms. Given a set of state-action samples (which can be extract from the trajectories), the gradient value on each sample (state s and action a) is calculated as $grad(\boldsymbol{s}, a) = Q^\pi(\boldsymbol{s}, a)\nabla_{\Psi(\boldsymbol{s},a)}\pi_\Psi(a|\boldsymbol{s})$. It then constructs a set of examples with features $(\boldsymbol{s}, a)$ and label $grad(\boldsymbol{s}, a)$, and derives a model h_t by regression learning from this set. Now the update rule is by

$$\Psi_{t+1} = \Psi_t + \eta_t h_t.$$

Note this step is a standard supervised regression task, and thus many well-established learning algorithms with strong generalization ability can be used here, which results an adaptively nonlinear model.

3. POLICYBOOST

3.1 Functional Gradient

Following REINFORCE [38], on a sample of m trajectories S, the unbiased gradient of the expected total reward is $\nabla\rho_S(\pi) = \frac{1}{m}\sum_{i=1}^{m}\nabla \log p^\pi(\tau_i)R(\tau_i)$. Considering the same action functions of NPPG, a policy is formed from a potential function Ψ. For a state-action pair $(\boldsymbol{s}, a)$ in a trajectory $\boldsymbol{\tau}$ with the next state s', the *functional gradient* with respect to $\Psi(\boldsymbol{s}, a)$ is

$$\begin{aligned}
&\nabla_{\Psi(\boldsymbol{s},a)}\rho(\pi_\Psi) = \frac{1}{m}R(\boldsymbol{\tau})\nabla_{\Psi(\boldsymbol{s},a)}\log p^{\pi_\Psi}(\boldsymbol{\tau})\\
&= \frac{1}{m}\frac{p(\boldsymbol{s}'|\boldsymbol{s}, a)}{p^{\pi_\Psi}(\boldsymbol{s}'|\boldsymbol{s})}R(\boldsymbol{\tau})\nabla_{\Psi(\boldsymbol{s},a)}\pi_\Psi(a \mid \boldsymbol{s})\\
&= \frac{1}{m}\frac{p(\boldsymbol{s}'|\boldsymbol{s}, a)}{\sum_{t=1}^{n} p(\boldsymbol{s}'|\boldsymbol{s}, a_t)\pi(\boldsymbol{s}, a_t)}R(\boldsymbol{\tau})\nabla_{\Psi(\boldsymbol{s},a)}\pi_\Psi(a \mid \boldsymbol{s}).
\end{aligned}$$

Then for discrete action space, we have

$$\nabla_{\Psi(\boldsymbol{s},a)}\pi(a \mid \boldsymbol{s}) = \pi_\Psi(a \mid \boldsymbol{s})(1 - \pi_\Psi(a \mid \boldsymbol{s})) \tag{3}$$

and for continuous action space,

$$\nabla_{\Psi(\boldsymbol{s},a)}\pi(a \mid \boldsymbol{s}) = 2\pi_\Psi(a \mid \boldsymbol{s})(a - \Psi(\boldsymbol{s}))/\sigma^2. \tag{4}$$

Since the functional gradient results in a function, of which the value can only be calculated on observed state-action pairs, we need to train a least square model h_t to fit the gradient value on the samples, and update the potential function as $\Psi_{t+1} = \Psi_t + \eta_t h_t$ with a small positive constant η. This results in the update of the policy.

3.2 On-Sample Convergence

To disclose how the functional gradient leads the policy, we consider discrete actions, i.e., $\pi_\Psi(a|\boldsymbol{s}) = \frac{\exp(\Psi(\boldsymbol{s},a))}{\sum_{a'}\exp(\Psi(\boldsymbol{s},a'))}$, and study its convergence on the training samples.

Let Ψ_0 be a constant function (e.g. always outputs 0), and recall $\Psi_{t+1} = \Psi_t + \eta\nabla_\Psi\rho_S(\pi_\Psi)$. For simplicity, when the state $\boldsymbol{s}$ is clear, we make some notations: let $\Psi_{t,k}$ be $\Psi_t(\boldsymbol{s}, a_k)$, let $\alpha_k^t = \pi_{\Psi_t}(a_k|\boldsymbol{s})$ for the action a_k, $\beta_{kj} = p(\boldsymbol{s}_j|\boldsymbol{s}, a_k)$, $\gamma_j = p^{\pi_\Psi}(\boldsymbol{s}_j|\boldsymbol{s})$ and $c_{kj} = \sum_{i=1}^{m} 1_{(\boldsymbol{s}_j\in\boldsymbol{\tau}_i)}\beta_{kj}R(\boldsymbol{\tau}_i)$ where $1_{expression}$ is the indicator function that is 1 when *expression* is true and 0 otherwise. Denote k^* the index of the *observed best action* of the state $\boldsymbol{s}$, such that $\forall k \neq k^*\ \forall j : c_{k^*j} \geq c_{kj}$.

The functional gradient of total reward on S at a state-action pair $(\boldsymbol{s}, a_k)$ can be rewritten as

$$\begin{aligned}
&\nabla_{\Psi_{t,k}}\rho_S(\pi_\Psi)\\
&= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{l} 1_{(\boldsymbol{s}_j\in\boldsymbol{\tau}_i)}\frac{p(\boldsymbol{s}_j|\boldsymbol{s}, a_k)}{p^{\pi_\Psi}(\boldsymbol{s}_j|s)}R(\boldsymbol{\tau}_i)\nabla_{\Psi_{t,k}}\pi_\Psi(a_k|\boldsymbol{s})\\
&= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{l} 1_{(s_j\in\boldsymbol{\tau}_i)}\frac{\beta_{kj}R(\boldsymbol{\tau}_i)}{\gamma_j}\alpha_k(1-\alpha_k)\\
&= \frac{1}{m}\alpha_k(1-\alpha_k)\sum_{j=1}^{l}\sum_{i=1}^{m} 1(s_j \in \boldsymbol{\tau}_i)\frac{\beta_{kj}R(\boldsymbol{\tau}_i)}{\gamma_j}\\
&= \frac{1}{m}\alpha_k(1-\alpha_k)\sum_{j=1}^{l}\frac{c_{kj}}{\gamma_j}
\end{aligned}$$

We prove below that functional gradient converges to the observed best action. Denote $\delta = \min_{k\neq k^*}\sum_{j=1}^{l} c_{k^*j} - c_{kj}$ be the reward margin, which will effect the convergence rate.

Lemma 1
For an observed state $\boldsymbol{s}$, let a_{k^*} be the observed best action, it holds that

$$\nabla_{\Psi_{t,k^*}}\rho_S(\pi_{\Psi_t}) - \nabla_{\Psi_{t,k}}\rho_S(\pi_{\Psi_t}) \geq \frac{1}{m}\alpha_{k^*}^t(1-\alpha_{k^*}^t)\delta.$$

Proof. We first need to prove $\alpha_{k^*}^t \geq \alpha_k^t$ for all t and $k \neq k^*$. The proof is by induction. When $t = 0$, since Ψ_{0,a_k} is a constant for all k, $\alpha_{k^*}^0 = \alpha_k^0$ for all k.

Then inductively assume that $\alpha_{k^*}^t \geq \alpha_k^t$ for all $k \neq k^*$. From the inductive assumption we have that, for all $k \neq k^*$, $\Psi_{t,k^*} \geq \Psi_{t,k}$ since $\alpha_k^t = \frac{\exp(\Psi_k^t)}{\sum_{i=1}^{n}\exp(\Psi_i^t)}$, and $\alpha_{k^*}^t(1-\alpha_{k^*}^t) \geq \alpha_k^t(1-\alpha_k^t)$ since $\sum_{k=1}^{n}\alpha_k^t = 1$. Therefore, we have that

$$\begin{aligned}
&\Psi_{t+1,k^*} - \Psi_{t+1,k}\\
&\geq \eta\nabla_{\Psi_{t,k^*}}\rho_S(\pi_{\Psi_t}) - \eta\nabla_{\Psi_{t,k}}\rho_S(\pi_{\Psi_t})\\
&\geq \frac{\eta}{m}\left(\alpha_{k^*}^t(1-\alpha_{k^*}^t)\sum_{j=1}^{l}\frac{c_{k^*j}}{\gamma_j} - \alpha_k^t(1-\alpha_k^t)\sum_{j=1}^{l}\frac{c_{kj}}{\gamma_j}\right)\\
&\geq 0,
\end{aligned}$$

and consequently, $\alpha_{k^*}^{t+1} \geq \alpha_k^{t+1}$. This induction proves that $\forall t\ \forall k \neq k^* : \alpha_{k^*}^t \geq \alpha_k^t$.

We then calculate the difference of the gradient.

$$\begin{aligned}
&\nabla_{\Psi_{t,k^*}} \rho_S(\pi_{\Psi_t}) - \nabla_{\Psi_{t,k}} \rho_S(\pi_{\Psi_t}) \\
&= \frac{1}{m}\alpha_{k^*}^t(1-\alpha_{k^*}^t)\sum_{j=1}^{l}\frac{c_{k^*j}}{\gamma_j} - \frac{1}{m}\alpha_k^t(1-\alpha_k^t)\sum_{j=1}^{l}\frac{c_{kj}}{\gamma_j} \\
&\geq \frac{1}{m}\alpha_{k^*}^t(1-\alpha_{k^*}^t)\sum_{j=1}^{l}\frac{c_{k^*j}-c_{kj}}{\gamma_j} \\
&\geq \frac{1}{m}\alpha_{k^*}^t(1-\alpha_{k^*}^t)\delta,
\end{aligned}$$

where the first inequality is by $\forall k \neq k^* : \alpha_{k^*} \geq \alpha_k$ and $\sum_{k=1}^n \alpha_k = 1$, the last inequality is by $\gamma_j \in (0,1]$. ■

Theorem 1
For any $\varepsilon \in (0, \frac{1}{n}]$, let a_{k^*} be the observed best action, functional gradient achieves $\alpha_{k^*}^t \geq 1-\varepsilon$ in iterations

$$t \leq \lceil \frac{m \ln n(1-\varepsilon)}{\eta\varepsilon(1-\varepsilon)\delta} \rceil.$$

Proof. By Lemma 1, we have, for any $k \neq k^*$,

$$\begin{aligned}
&\Psi_{t,k^*} - \Psi_{t,k} \\
&= \Psi_{t-1,k^*} - \Psi_{t-1,k} \\
&\quad + \eta\Big(\nabla_{\Psi(\boldsymbol{s},a_{k^*})}\rho_S(\pi_\Psi) - \nabla_{\Psi(\boldsymbol{s},a_k)}\rho_S(\pi_\Psi)\Big) \\
&\geq \Psi_{t-1,k^*} - \Psi_{t-1,k} + \eta\frac{1}{m}\alpha_{k^*}^{t-1}(1-\alpha_{k^*}^{t-1})\delta \\
&\geq \ldots \geq \Psi_{0,k^*} - \Psi_{0,k} + \sum_{i=0}^{t-1}\eta\frac{1}{m}\alpha_{k^*}^i(1-\alpha_{k^*}^i)\delta \\
&= \sum_{i=0}^{t-1}\eta\frac{1}{m}\alpha_{k^*}^i(1-\alpha_{k^*}^i)\delta.
\end{aligned}$$

Assume $\forall i < t : \alpha_{k^*}^i < 1-\varepsilon$, otherwise we have already found a policy satisfying the theorem. Thus

$$\Psi_{t,k^*} - \Psi_{t,k} \geq \sum_{i=0}^{t-1}\eta\frac{1}{m}\varepsilon(1-\varepsilon)\delta.$$

When $t = \frac{m\ln n(1-\varepsilon)}{\eta\varepsilon(1-\varepsilon)\delta}$, $\Psi_{t,k^*} - \Psi_{t,k} \geq \ln n(1-\varepsilon)$ and thus

$$\alpha_{k^*}^t = \frac{1}{\sum_{i=1}^n \exp(\Psi_{t,k} - \Psi_{t,k^*})} \geq 1-\varepsilon$$

Therefore, $\alpha_{k^*}^t \geq 1-\varepsilon$ is achieved in $t \leq \lceil \frac{m\ln n(1-\varepsilon)}{\eta\varepsilon(1-\varepsilon)\delta} \rceil$ iterations. ■

The theorem states that the functional policy gradient can converge to the observed best action very quickly.

3.3 Avoid Overfitting

When fitting the gradient function on samples, state-of-the-art regression techniques can be employed to fit the gradient quite well. However, as disclosed previously, the gradient converges towards the observed best actions, which however may not be the real best actions.

When linear models are employed, this problem is commonly addressed by the *variance reduction* trick [15]: for each dimension, an optimal bias to the reward can be calculated. The bias has no effect over the trajectory space, but will reduce the variance of the gradient estimation over limited trajectories. However, this trick cannot straightforwardly apply to functional gradient, since the dimensions are not directly manipulatable.

We instead consider this problem from the aspect of boosting. There are several possible ways to avoid overfitting: reduce the model capacity, and do not fit the samples too well. The model capacity of a boosting algorithm is related to the number of iteration [10], thus we shall use a small, say 1, number of iterations. To less fit the samples, we prefer the model fit only the trajectories with better reward. To do this, note that Theorem 1 says that the best observed actions with $c_{k^*j} > 0$ will be fitted. By the definition of c_{ij} it is easy to turn it to be negative by setting the corresponding trajectories to have negative rewards. Therefore, we use the centralized rewards:

$$\rho'(\pi) = \int_{\mathcal{T}} p(\boldsymbol{\tau}|\pi)\big(R(\boldsymbol{\tau}) - \bar{R}\big)\mathrm{d}\boldsymbol{\tau}, \tag{5}$$

where $\bar{R} = \int_{\mathcal{T}} \frac{1}{\int_{\mathcal{T}}\mathrm{d}\boldsymbol{\tau}} R(\boldsymbol{\tau})$. Similarly with the variance reduction trick, it is easy to verify that maximizing Eq.(1) is equivalent with maximizing Eq.(5) since $\int_{\mathcal{T}} p(\boldsymbol{\tau}|\pi)\mathrm{d}\boldsymbol{\tau}$=1. The gradient on a sample S with m trajectories is

$$\nabla\rho'_S(\pi) = \frac{1}{m}\sum_{i=1}^{m}\nabla\log p^\pi(\boldsymbol{\tau}_i)(R(\boldsymbol{\tau}_i) - \bar{R}_S), \tag{6}$$

where $\bar{R}_S = \sum_{i=1}^m \frac{1}{p^\pi(\boldsymbol{\tau}_i)z_S} R(\boldsymbol{\tau}_i)$ with $z_S = \sum_{i=1}^m \frac{1}{p^\pi(\boldsymbol{\tau}_i)}$. Therefore, to use this gradient, we only need to shift the reward of the sampled trajectories by $\bar{R}_S$, and keep anything else unchanged.

Algorithm 1 PolicyBoost

Input:
T: Number of iterations
ϵ: Probability for ϵ-greedy (for discrete action)
σ: Gaussian width (for continues action)
m: Sample size
(b, u): Parameters for memory pool size
$\{\eta_t\}_{t=1}^T$: Learning rate
$\mathcal{L}$: Base regression learner

Procedure:
π: The learned policy
1: $Pool_{best} = \emptyset, Pool_{uniform} = \emptyset$
2: $\Psi_0(\boldsymbol{s}, a) = 1, \forall(\boldsymbol{s}, a) \in S \times A$ (for discrete action), or $\Psi_0(s) = 0, \forall s \in S$ (for continues action)
3: **for** $t = 1$ to T **do**
4: Let S_t be m trajectories sampled by executing π_{t-1}:
$\pi_{t-1}(a|\boldsymbol{s}) = e^{\Psi_{t-1}(\boldsymbol{s},a)} / \sum_b e^{\Psi_{t-1}(\boldsymbol{s},b)}$
with ϵ-greedy for discrete action, or
$\pi_{t-1}(a|\boldsymbol{s}) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\Psi_{t-1}(\boldsymbol{s})-a)^2}{\sigma^2}\right)$
with a width of σ for continues action
5: Generate functional gradient examples D_t as
$D_t = \{((\boldsymbol{s}, a), \nabla_{\Psi(\boldsymbol{s},a)}\rho'_\tau(\pi_\Psi))\}$
for discrete action, or for continues action as
$D_t = \{(\boldsymbol{s}, \nabla_{\Psi(\boldsymbol{s},a)}\tilde{\rho}'_\tau(\pi_\Psi))\}$
for all $(\boldsymbol{s}, a)$ in all $\tau \in S_t \cup Pool_{best} \cup Pool_{uniform}$
6: Train a regression model h_t using $\mathcal{L}$ from D_t
7: $\Psi_t = \Psi_{t-1} + \eta_t h_t$
8: Update $Pool_{best}$ and $Pool_{uniform}$ with S_t
9: **end for**
10: **return** π_T

3.4 The PolicyBoost Algorithm

We then propose an *actor-only* algorithm, the PolicyBoost, as in Algorithm 1. The details are explained in the following.

To better estimate the gradient, instead of through away all the past sampled trajectories, PolicyBoost employs two pools $Pool_{best}$ and $Pool_{uniform}$ to record some past samples: the former one keeps some best-so-far trajectories, and the latter one contains randomly selected trajectories. They are initialized in line 1. Line 2 initializes the potential function. Then PolicyBoost performs T iterations to update the policy. Given a potential function Ψ, the policy is formed by the Gibbs Sampling function for discrete action spaces and the Gaussian distribution for continuous action spaces, which is the same as that for NPPG. Line 4 uses the policy to sample m trajectories, stored in S_t. In line 5, from the union of all sets, a training data is constructed for learning an approximation of the gradient function in line 6, and the potential function is then updated in line 7 with the learning rate η_t. Line 8 updates the pools.

For the generation and approximation of the gradient function, through the point-wise gradient [12], we calculate the gradient function of Eq.(6) with Eq.(3) or Eq.(4) at every observed state-action pair. Then a data set D_t is constructed in line 5. This data set expresses the gradient values at some state-action points. A regression algorithm is then used to learn a function h_t from the data set to approximate the gradient function, as in line 6.

3.5 Incoporating Demonstrations

Interestingly, PolicyBoost can make use of demonstrated trajectories that could be provided by experts. When demonstrated trajectories are available, we can put them into the $Pool_{best}$ set at the initialization step. If the demonstration data is provided with no rewards, we can either assign a high enough estimated reward, or access the environment and evaluate the demonstrations when it is possible. When the demonstrations are put into the $Pool_{best}$ set, PolicyBoost will automatically utilize them to better estimate the gradient, and learn better policies. Note that, unlike in inverse reinforcement learning [26], PolicyBoost does not require optimal demonstrations. When PolicyBoost generates trajectories better than the demonstrations, it will update the $Pool_{best}$ to wipe them out, so that its performance will not be limited by the quality of the demonstrations.

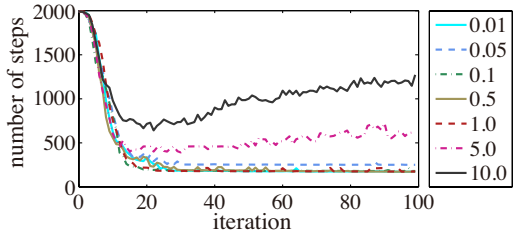
4. EXPERIMENTS

4.1 Parameter Sensitivity

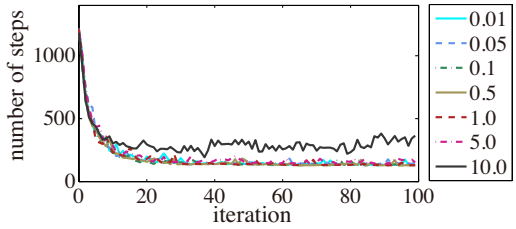
We employ two typical domains, the *Mountain Car* and the *Acrobot*, to examine the parameter sensitivity.

In the *Mountain Car* task, a state is a two dimensional vector, and each dimension is a continuous variable: the horizontal position x and the velocity $\dot{x}$, which are restricted to the ranges $[-1.2, 0.6]$ and $[-0.07, 0.07]$ respectively. Note that we do not discretize the state space. The agent has three actions: driving left, driving right, and not to use the engine at all. The goal of the agent is to reach the mountain top, i.e., $x > 0.5$, from an initial state in $\{(x, \dot{x})|x \in [-0.75, -0.25], \dot{x} \in [-0.025, 0.025]\}$. We run each episode at a maximal horizon of 2000 steps. The car receives a reward of -1 before reaching the goal and 1 for reaching the goal. In the *Acrobot* task, a state consists of four dimensions. Each is a continuous variable: two joint positions θ_1, θ_2 and two joint velocities $\dot{\theta_1}$, $\dot{\theta_2}$. There are three actions corresponding to torque to the joint between the first and second link of $-1, 0, 1$ respectively. The goal of the agent is to maintain the tip above the goal line from an initial state in $\{(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2})|\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2} \in [-0.5, 0.5]\}$. We run each episode at a maximal horizon of 2000 steps. It receives a reward of -1 before reaching the goal line and 1 for reaching the goal. The dynamics of the two domains and other details can be found in [34]. Note that we use the raw state features here, but not any better feature encoding.

In the implementation of PolicyBoost, the base regression learner used in all policies is regression decision trees [7] as implemented in WEKA [39], and we set $m = 50$. The variable parameters include the learning rate, the decision tree depths, and the size of the pools. The performance with

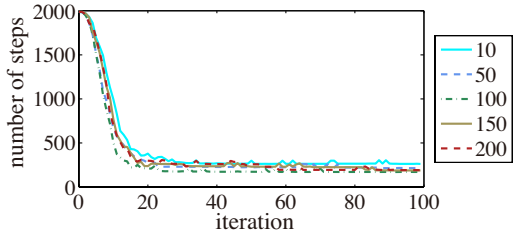


(a) on Mountain Car

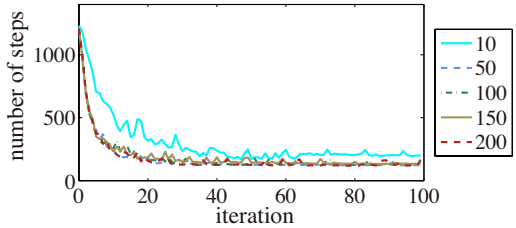


(b) on Acrobot

Figure 1: Performance of PolicyBoost with different learning rates

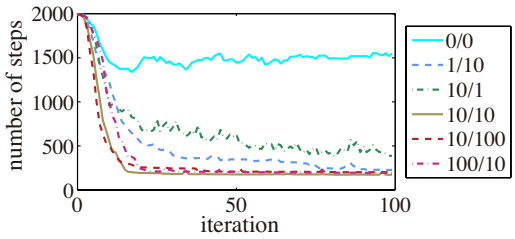


(a) on Mountain Car

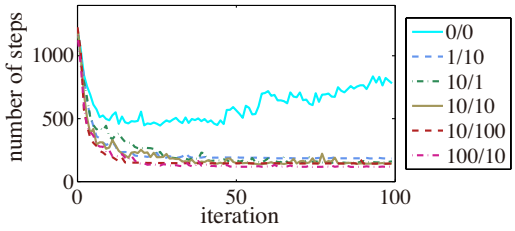


(b) on Acrobot

Figure 2: Performance of PolicyBoost with different tree depths



(a) on Mountain Car



(b) on Acrobot

Figure 3: Performance of PolicyBoost with different pool sizes (best/uniform)

different learning parameters are shown in Figures 1 to 3.

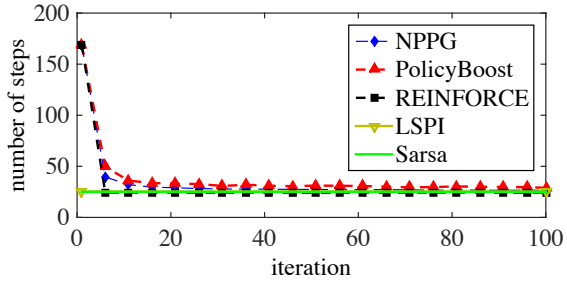
It can be observed that, first from Figure 1, except for too large learning rates (i.e., 5 and 10), PolicyBoost policies does not have a significant different performance. We thus suggest that a fixed learning rate of 1 or 0.1 is good enough in practice. Second, from Figure 2, the tree depth does not show a significant difference except for the smallest depth 10. For it's time consuming to employ deep trees, 100 could be sufficient. On the pool size from Figure 3, we observe that the performance is quite bad with zero pool size, while other pool sizes lead to close performances. We suggest "10/10" in practice. Overall, these comparisons show that PolicyBoost is not quite sensitive to its parameters, and some moderate parameters are suggested.

4.2 Comparisons

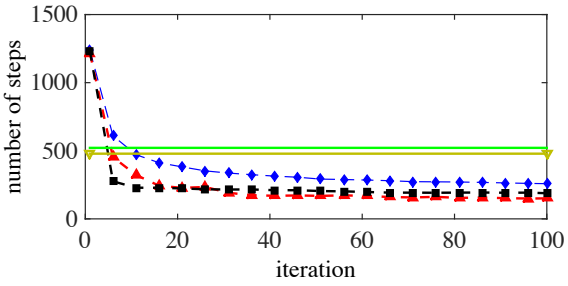
We compare the PolicyBoost, using default parameters and raw features of the states and actions, with fine-tuned NPPG, REINFORCE, Sarsa(λ) and LSPI using linear function approximators [31, 13].

For PolicyBoost, we use a fixed setting for all domains: $\eta = 1$, $\epsilon = 0$, tree depth= 100, and the pool size is 10 for both pools. For NPPG, the parameters are carefully tuned: $\eta = 0.1$ and $\epsilon = 0.2$ for Mountain Car, and $\eta = 0.08$ and $\epsilon = 0.1$ for Acrobot. To compare with a linear model using policy gradient methods, we further employ REINFORCE with same parameters: $\eta = 0.1$,$\epsilon = 0$, while the tabular representation [31] is used for REINFORCE. Sarsa(λ) is used as reference of non-policy gradient method. Sarsa(λ) uses the Fourier features [20], and its parameters are set to be: γ=1.0, λ=0.9, $\epsilon = 0.01$ with Fourier bases of order 3. The learning rate is adaptive as in [9]. LSPI is a batch reinforcement learning algorithm and its parameters are set to be: γ=1.0, $\epsilon = 0.01$ with Fourier bases of order 3.

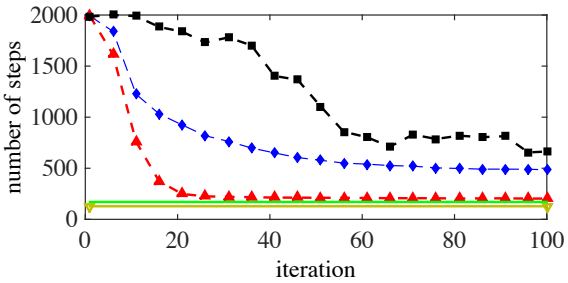
For this group of comparisons, we use an extra domain, Corridor World [17], which is a very simple task where an agent from any random position $x \in [4, 6]$ in a one dimensional corridor $[0, 10]$ needs to reach one of the exits at both ends (0 and 10). For Corridor World, NPPG_{opt} uses $\eta = 0.05$ and $\epsilon = 0.1$.



(a) Corridor World



(b) Acrobot



(c) Mountain Car

Figure 4: Performance of different policies at each iteration. The fewer steps the better. Plots (b) and (c) share the legend with plot (a).

The comparison results are presented in Figure 4, where the lines of Sarsa(λ) and LSPI are their performance. Despite that PolicyBoost uses the raw features and the fixed configuration, it produces policies with a clear trend of convergence to the near optimal solution. NPPG works the same as PolicyBoost only on Corridor World, because this task is quite simple so that the overfitting issue is not important. On Acrobot and Mountain Car, NPPG can gain some improvement only after carefully turning the parameters, but is still much worse than PolicyBoost. REINFORCE performs quite well on Corridor World, but the performance degrades badly on Mountain Car. We can conclude that PolicyBoost is more effective and robust than NPPG and REINFORCE.

To verify whether the overfitting problem does exists that encumbers NPPG, we calculated the normalized expected rewards of NPPG and PolicyBoost, i.e.,

$$\sum_{\boldsymbol{\tau} \in S} R(\boldsymbol{\tau}) \cdot p^{\pi}(\boldsymbol{\tau}) / \sum_{\boldsymbol{\tau}'} p^{\pi}(\boldsymbol{\tau}'),$$

using the Mountain Car task. The result is shown in Figure 5. It is clear that PolicyBoost using the proposed objective

function indeed learns to rank higher the trajectories with larger rewards, while NPPG fails to do that. This observation confirms that PolicyBoost can well handle the occurring probability overfitting problem, which blocks NPPG.

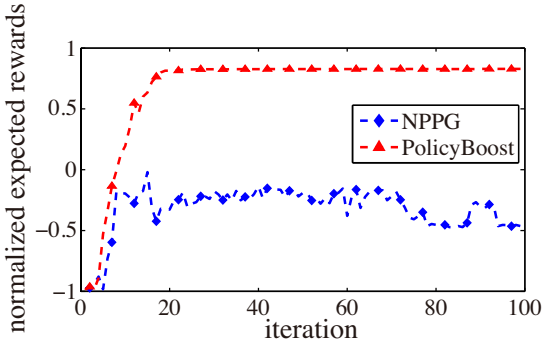


Figure 5: Normalized rewards of NPPG and PolicyBoost on the Mountain Car task.

4.3 Application to Helicopter Control

We use a simulator [36] that models one of the Stanford autonomous helicopters in the flight regime close to hover. The model was learned from data collected using "frequency sweeps" (which systematically vary each of the helicopter's four controls) around the hover flight regime by [3]. This is a challenge task that the agent needs to manipulate 4 continuous control inputs simultaneously based on a 12-dimensional state space. Previously evolutionary algorithm based method [19], programming method [2], and learning from demonstrations [8, 1] have been examined on this task.

We set the maximal horizon to 6000 steps. We regard the four continuous control inputs as independent variables and set $\delta = 0.05$ for each input. So we will learn four policy functions simultaneously each for an input, which does not need to change the structure of the PolicyBoost or NPPG. The PolicyBoost with the fixed configuration is tested. For NPPG we did not find any parameter that leads to any improvement, so the best performance is reported, as in Figure 6. For Sarsa(λ), even if we set the Fourier basis be order 3, the dimensions of the representation for value-function approximator is more than 500,000, leading us to decrease order 3 to 2. LSPI requires too large memory and is not included in the comparison. For REINFORCE, tabular representation would also cause such problem, so we have to use original features. NPPG is not able to find the right gradient direction to improve the hovering steps, while PolicyBoost constantly improves the policy, without any parameter tuning. This observation confirms the advantage of PolicyBoost in real-world applications.

4.4 Incorporating Demonstrations

Finally, we investigate PolicyBoost with demonstrations using the Ms. Pac-Man Game[1] with random ghosts on the first level. The state is described by the directions of the ghosts and directions of the pills with respect to Ms. Pac-Man. Again, PolicyBoost uses the fixed configuration, and NPPG uses well turned parameters. For this game, we

[1]We use the simulator from http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html

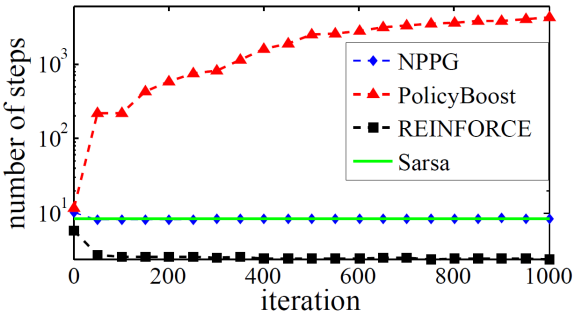


Figure 6: Performance of different policies on the Helicopter Hovering task. The more steps the better.

record a starter-level human player's actions as the demonstration data, which are not optimal demonstrations. We use 10 such demonstration trajectories in PolicyBoost, which is denoted as PolicyBoost$_{\text{demo}}$.

Figure 7 shows the performance of different approaches. It can be observed that PolicyBoost converges better than NPPG. It is clear that, as the demonstration data is utilized, PolicyBoost$_{\text{demo}}$ does not only improves the policy much faster than PolicyBoost alone, but also converges to a better performance. The experiment results show that PolicyBoost can well make use of demonstrations to significantly improve its performance.

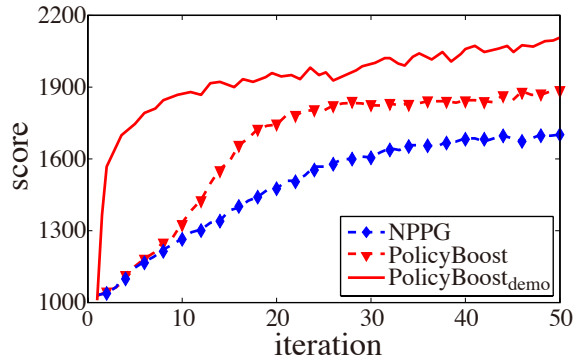


Figure 7: Performance of different policies on the Ms. Pac-Man Game task. The higher score the better.

5. CONCLUSION

In this paper, to combine powerful boosting technique with policy gradient ascent for reinforcement learning problems, we analyzed the functional gradient method and proposed the PolicyBoost method, which searches in an additive function space for a policy that maximizes a corrected total rewards function. Empirical studies on several domains show that, even without feature engineering, PolicyBoost is not only effective to achieve good policies, but is also highly stable to its configuration parameters. Moreover, experiments on the Helicopter Hovering task verifies its applicability in real-world problems, and on the Ms. Pac-Man Game task shows its ability to incorporate non-optimal demonstrations. The future work will focus on designing more

appropriate mechanisms for trajectory reuse than the current pool-based solution, such as integrating the importance weighting technique with the boosting framework [23].

REFERENCES

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS)*, pages 1–8. MIT Press, 2007.

[3] P. Abbeel, V. Ganapathi, and A. Y. Ng. Learning vehicular dynamics, with application to modeling helicopters. In B. S. Yair Weiss and J. Platt, editors, *Advances in Neural Information Processing Systems 18 (NIPS)*, pages 1–8. MIT Press, 2005.

[4] D. A. Aberdeen. *Policy-gradient algorithms for partially observable Markov decision processes*. PhD thesis, Australian National University, 2003.

[5] P. L. Bartlett and J. Baxter. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[6] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[7] L. Breiman, J. Friedman, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[8] A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine learning (ICML)*, pages 144–151, Helsinki, Finland, 2008. ACM.

[9] W. Dabney and A. G. Barto. Adaptive step-size for online temporal difference learning. In *AAAI*, 2012.

[10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[12] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[13] A. Geramifard, R. H. Klein, C. Dann, W. Dabney, and J. P. How. RLPy: The Reinforcement Learning Library for Education and Research, 2013.

[14] M. Ghavamzadeh and Y. Engel. Bayesian actor-critic algorithms. In *Proceedings of the 24th International Conference on Machine Learning*, pages 297–304, Corvallis, OR, 2007.

[15] E. Greensmith, P. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530, 2004.

[16] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 433–444, Seattle, WA, 1989.

[17] K. Kersting and K. Driessens. Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 456–463, Helsinki, Finland, July 2008.

[18] H. Kimura. Reinforcement learning by stochastic hill climbing on discounted reward. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pages 295–303, Tahoe City, CA.

[19] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84(1):171–203, 2011.

[20] G. Konidaris, S. Osentoski, and P. S. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011.

[21] H.-Y. Lo, K.-W. Chang, S.-T. Chen, T.-H. Chiang, and C.-S. Ferng. An ensemble of three classifiers for kdd cup 2009: Expanded linear model, heterogeneous boosting, and selective naive Bayes. *The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009) Challenges in Machine Learning, Volume 3*, page 53, 2009.

[22] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 512–518. The MIT Press, 2000.

[23] T. Matsubara, T. Morimura, and J. Morimoto. Adaptive step-size policy gradients with average reward metric. In *Proceedings of the 2nd Asian Conference on Machine Learning*, pages 285–298, Tokyo, Japan, 2010.

[24] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, 1997.

[25] A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[26] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 663–670, Stanford, CA, 2000.

[27] E. Orate, S. Idelsohn, O. C. Zienkiewicz, and R. L. Taylor. A finite point method in computational mechanics. Application to convectioe transport and fluid flow. *International Journal for Numerical Methods in Engineering*, 39:3839–3866, 1996.

[28] L. Peshkin. *Reinforcement learning by policy search*. PhD thesis, MIT, 2001.

[29] J. Peters. Policy gradient methods. *Scholarpedia*, 5(10):3698, 2010.

[30] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

[31] N. Roy and J. How. A tutorial on linear function approximators for dynamic programming and reinforcement learning. 2013.

[32] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[33] R. E. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. MIT Press, Cambridge, MA, 2012.

[34] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, 1998.

[35] R. S. Sutton, D. McAllester, S. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 1057–1063. 2000.

[36] B. Tanner and A. White. RL-Glue: Language-independent software for reinforcement-learning experiments. *The Journal of Machine Learning Research*, 10:2133–2136, 2009.

[37] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[38] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[39] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.