PolicyBoost: Functional Policy Gradient with Ranking-Based Reward Objective*

Yang Yu and **Qing Da**
National Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
yuy@lamda.nju.edu.cn, daq@lamda.nju.edu.cn

Abstract

Learning policies in nonlinear representations is an important step toward real-world applications of reinforcement learning in robotics. While functional representation has been widely applied in state-of-the-art supervised learning techniques (as known as boosting approaches) to adaptively learn nonlinear functions, in reinforcement learning the boosting-style approaches have been little investigated. Only a few pieces of work explored in this direction, which however may suffer from the *occurring-probability-pursuing* problem. In this paper, to alleviate the problem, we propose to employ a ranking-based objective function to guide the policy search in a function space, resulting in the PolicyBoost approach. Experiment results verify the effectiveness as well as the robustness of the PolicyBoost.

Introduction

A good reinforcement learning algorithm would be able to generalize well from limited feedbacks, sharing similar principle with supervised learning algorithms. In supervised learning, an algorithm is given a fixed training examples and expected to build a model that correctly predicts unseen instances. We may learn from successful supervised learning algorithms in designing strong reinforcement learning algorithms. A noticeable off-the-shelf supervised learning algorithm family is called *boosting*, with representatives as AdaBoost (Freund and Schapire 1997), LogitBoost (Friedman, Hastie, and Tibshirani 2000), GradientBoost (Friedman 2001), etc. Most of them share a common routine (Mason et al. 2000) that trains an additive combination of models via the gradient descent in some function space. The performance of booting 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 nonlinear models, which are also quite appealing 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 (Kersting and Driessens 2008), by which a policy is formed by learning an *additive model* as 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 could be the result of optimizing only the occurring probability of these trajectories, which can be useful only when the policy has been near optimal already. Inherited from boosting, the NPPG method has a strong learning ability, which makes the occurring-probability-pursuing problem even more severe.

In this paper, we propose the PolicyBoost approach that maximizes the *ranking-based objective function* to alleviate the occurring-probability-pursuing problem, which is optimized by the gradient ascent method in a function space. The new objective encourages a policy to rank the trajectories correctly according to their rewards, rather than encouraging maximizing the sum of rewards. Therefore, simply increasing the occurring probability of training trajectories no longer leads to a better objective value. 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 good trajectories that have been experienced. We perform empirical studies of PolicyBoost on several domains. Experiment results reveal that, firstly, NPPG can be injured by the occurring-probability-pursuing problem while PolicyBoost does not. Secondly, PolicyBoost is shown to be not only effective to achieve good policies on the tested domains, but also highly stable to its configuration parameters, which is an important property for practical applications.

Background

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

*This work was supported by the National Science Foundation of China (61375061, 61321491, 61223003) and the Jiangsu Science Foundation (BK2012303).

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 $\rho(\pi) = E\{\frac{1}{T}\sum_{t=1}^{T} r_t \mid \pi\}$, 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 (Williams 1992; Kimura 1995; Peshkin 2001; Ng and Jordan 2000; Peters 2010) 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 (Bartlett and Baxter 2001). 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 $\boldsymbol{\tau} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{T+1})$, let $R(\boldsymbol{\tau})$ be its trajectory-wise reward, for the T-step setting $R(\boldsymbol{\tau}) = \sum_{i=1}^{T} r(\boldsymbol{s}_i, \boldsymbol{s}_{i+1})/T$ and for the discounted setting $R(\boldsymbol{\tau}) = \sum_{i=1}^{T} \gamma^{t-1} r(\boldsymbol{s}_i, \boldsymbol{s}_{i+1})$. 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 $p^{\pi}(\boldsymbol{\tau}) = p(\boldsymbol{s}_1^{\boldsymbol{\tau}}) \prod_{t=1}^{T} p^{\pi}(\boldsymbol{s}_{t+1}^{\boldsymbol{\tau}} \mid \boldsymbol{s}_t^{\boldsymbol{\tau}})$
$= p(\boldsymbol{s}_1^{\boldsymbol{\tau}}) \prod_{t=1}^{T} \int_{\mathcal{A}} p(\boldsymbol{s}_{t+1}^{\boldsymbol{\tau}} \mid \boldsymbol{s}_t^{\boldsymbol{\tau}}, a) \pi(a \mid \boldsymbol{s}_t) \, \mathrm{d}a$.

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 (Orate et al. 1996), which estimates the gradient by sampling some neighbors of the current parameter vector. The REINFORCE (Williams 1992) 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, $\nabla_{\theta_k} \rho_S(\pi_{\theta_k}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta_k} \log p^{\pi_{\theta_k}}(\boldsymbol{\tau}_i) R(\boldsymbol{\tau})$
$= \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_k} \log p^{\pi_{\theta_k}}(\boldsymbol{s}_{t+1}^{\boldsymbol{\tau}_i} \mid \boldsymbol{s}_t^{\boldsymbol{\tau}_i}) R(\boldsymbol{\tau})$.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 (Greensmith, Bartlett, and Baxter 2004). In the actor-critic framework, policy gradient has also been employed. For example, considering the stationary distribution of states, the gradient can be (Sutton et al. 2000)

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

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. (Peters and Schaal 2008; Bhatnagar et al. 2009)).

Non-Parameteric Policy Gradient

The non-parametric policy gradient (NPPG) method (Kersting and Driessens 2008) is, to the best of our knowledge, the only previous boosting-like approach that uses additive function models to represent policies.

A policy $\pi(a|\boldsymbol{s})$ is represented as a function g of some potential function $\Psi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, i.e., $\pi_{\Psi}(a|\boldsymbol{s}) = g(\Psi(\boldsymbol{s}, a))$. 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, $\pi_{\Psi}(a|\boldsymbol{s}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\Psi(\boldsymbol{s})-a)^2}{\sigma^2}\right)$.

NPPG employs the gradient of Eq.(2) directly, except that the gradient is with respect to the potential function (i.e., functional gradient),

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

Then the potential function is updated as $\Psi_{k+1} = \Psi_k + \eta_k \nabla_{\Psi} \rho(\pi_{\Psi_k})$, which results an additive combination of base models, i.e., $\Psi_K = \sum_{k=1}^{K} \eta_k \nabla_{\Psi} \rho(\pi_{\Psi_k})$ for K iterations.

Different with the gradient in parameter spaces, the gradient in a function space $\nabla_{\Psi} \rho(\pi_{\Psi_k})$ is also a function but can not be explicitly expressed. Then the point-wise estimation is used (Friedman 2001) to approximate the gradient function via regression learning algorithms. Given a set of state-action samples (which can be extract from the trajectories), the value of the gradient function 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_k by regression learning from this set. Now the update rule is by $\Psi_{k+1} = \Psi_k + \eta_k h_k$. Note this step is a standard supervised regression task, and thus many well-established learning algorithms with strong generalization ability can be used here.

The Proposed Approach

The Occurring-Probability-Pursuing Problem

The executing policy is often different from the exploration policy due to the introduced randomization for collecting perturbated samples. Since Eq.(1) can be rewritten as $\rho(\pi) = \int_{\boldsymbol{\tau} \in \mathcal{T}} q(\boldsymbol{\tau}) \frac{p^{\pi}(\boldsymbol{\tau})}{q(\boldsymbol{\tau})} R(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau}$ for any exploration distribution $q(\boldsymbol{\tau}) > 0$, the unbiased estimator of the gradient on a sample S of trajectories from the distribution q is

$$\nabla \rho_S(\pi) = \frac{1}{m} \sum\nolimits_{i=1}^{m} \frac{p^{\pi}(\boldsymbol{\tau}_i)}{q(\boldsymbol{\tau}_i)} \nabla \log p^{\pi}(\boldsymbol{\tau}_i) R(\boldsymbol{\tau}_i),$$

which is, at the same time, the exact gradient of the objective

$$\rho_S(\pi) = \frac{1}{m} \sum_{i=1}^{m} \frac{p^{\pi}(\boldsymbol{\tau}_i)}{q(\boldsymbol{\tau}_i)} R(\boldsymbol{\tau}_i) = Z_{S,\pi} \frac{1}{m} \sum_{i=1}^{m} \frac{\frac{p^{\pi}(\boldsymbol{\tau}_i)}{Z_{S,\pi}}}{q(\boldsymbol{\tau}_i)} R(\boldsymbol{\tau}_i), \tag{3}$$

where $Z_{S,\pi} = \sum_{i=1}^{m} p^{\pi}(\boldsymbol{\tau}_i)$ is the normalization factor. Note that $Z_{S,\pi}$ is commonly much smaller than 1, particularly when the whole trajectory space $\mathcal{T}$ is quite large and the sample S is small.

Eq.(3) is actually the objective on a finite sample set. As the transformation in Eq.(3), this objective can have two components: one is the normalization factor $Z_{S,\pi}$, and the other component $\frac{1}{m}\sum_{i=1}^{m}\frac{1}{q(\boldsymbol{\tau}_i)}\frac{p^\pi(\boldsymbol{\tau}_i)}{Z_{S,\pi}}R(\boldsymbol{\tau})$ is the weighted correlation between the normalized probability $\frac{p^\pi(\boldsymbol{\tau}_i)}{Z_{S,\pi}}$ and the reward $R(\boldsymbol{\tau})$. When increasing the objective Eq.(3), it is possible that only $Z_{S,\pi}$, the occurring probability of the sampled data, is optimized. Consider a simple example, where the rewards are all positive, $p^\pi(\boldsymbol{\tau}) = c_1$ for all $\boldsymbol{\tau} \in S$ and $p^\pi(\boldsymbol{\tau}) = c_2$ for all the rest $\boldsymbol{\tau}$. If the updated policy π' simply increases the probability of every trajectory in S as $p^{\pi'}(\boldsymbol{\tau}) = c_1 + c$ and decreases the probability of the rest trajectories by a constant value, we will get that $\rho_S(\pi') = (1 + \frac{c}{c_1})\rho_S(\pi)$. This looks like a progress, but the policy π' only increases the occurring probability of the training set S. For the ultimate goal ρ it is easy to see that $\rho(\pi') < \rho(\pi)$ as long as that the average reward in S is below the average reward over all trajectories (i.e., $\frac{1}{m}\sum_{\tau\in S} R(\tau) < \frac{1}{|\mathcal{T}|-m}\sum_{\tau\in\mathcal{T}\setminus S} R(\tau)$ in discrete space version).

The occurring-probability-pursuing problem could be even more severe for boosting-style approaches, which employs additive function models and thus can produce quite flexible models to fit the occurring probability.

The Ranking-based Objective Function

To deal with the problem, it is ideal to focus on maximizing the correlation between the normalized probability $\frac{p^\pi(\boldsymbol{\tau}_i)}{Z_{S,\pi}}$ and the reward $R(\boldsymbol{\tau})$, while pushing the normalization factor towards the value from the optimal policy. This can be described by the objective function, for some constant λ,

$$\rho_S^*(\pi) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{q(\boldsymbol{\tau}_i)}\frac{p^\pi(\boldsymbol{\tau}_i)}{Z_{S,\pi}}R(\boldsymbol{\tau}_i) - \lambda|Z_{S,\pi^*} - Z_{S,\pi}|,$$

which will equal Eq.(1) when summing up over the trajectory distribution, similar to that of Eq.(3). However, this objective function is obviously infeasible, since that we cannot know Z_{S,π^*} and it is hard to optimize the probability under the normalization constraint.

Therefore, we propose to employ the following ranking-based objective function, which focuses on the correlation between the executing probabilities and the rewards,

$$\tilde{\rho}(\pi) = \int_{\mathcal{T}}\int_{\mathcal{T}}\big(p(\boldsymbol{\tau}|\pi) - p(\boldsymbol{\tau}'|\pi)\big)\big(R(\boldsymbol{\tau}) - R(\boldsymbol{\tau}')\big)\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\tau}'. \quad (4)$$

It is easy to verify that maximizing Eq.(4) is equivalent to maximizing Eq.(1). But the new objective can be safer. When over a finite set S sampled from the distribution $q(\boldsymbol{\tau})$, the objective is to maximize

$$\tilde{\rho}_S(\pi) = \frac{1}{m^2}\sum_{\boldsymbol{\tau},\boldsymbol{\tau}'\in S}\big(\frac{p^\pi(\boldsymbol{\tau})}{q(\boldsymbol{\tau})} - \frac{p^\pi(\boldsymbol{\tau}')}{q(\boldsymbol{\tau}')}\big)\big(R(\boldsymbol{\tau}) - R(\boldsymbol{\tau}')\big), \quad (5)$$

by which the policies π and π' with $\forall\boldsymbol{\tau} \in S : p^{\pi'}(\boldsymbol{\tau}) = p^\pi(\boldsymbol{\tau}) + c$ are equally good due to the subtraction.

We can expand the sum and obtain an equivalent but simpler objective as that

$$\tilde{\rho}_S(\pi) = \frac{2}{m}\sum\nolimits_{\boldsymbol{\tau}\in S}\frac{p^\pi(\boldsymbol{\tau})}{q(\boldsymbol{\tau})}\Big(R(\boldsymbol{\tau}) - \frac{1}{m}\sum\nolimits_{\boldsymbol{\tau}'\in S}R(\boldsymbol{\tau}')\Big).$$

It reveals that the optimization of the new objective is the same as before except the rewards of the training set are centralized. The new objective encourages increasing the probability of the trajectories with above-average rewards and decreasing the probability of the rest. Note that the reward centralization and the variance reduction trick (Greensmith, Bartlett, and Baxter 2004) have very different derivations, as well as very different values.

Moreover, the objective of Eq.(4) tells that an optimal policy should rank the trajectories with the largest reward above all the others. Therefore, to have a good evaluation of the gradient towards the optimal policy via Eq.(5), it is helpful to include trajectories with large rewards in the training set S. Otherwise, when S contains only low reward trajectories, the policy that perfectly ranks these trajectories may not rank the large reward trajectories well. Motivated by this understanding, we propose to record the best past trajectories and put them in the training set. Note that this is different with the reweighting of past trajectories in PoWER (Kober and Peters 2011), the trajectories are kept not according to their probability in the current sampling distribution.

Algorithm 1 PolicyBoost

Input:

K: 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
$\{\eta_k\}_{k=1}^K$: Learning rate
$\mathcal{L}$: Base regression learner

Output:

π: The learned policy

$Pool_{best} = \emptyset, Pool_{uniform} = \emptyset$
$\Psi_0(\boldsymbol{s}, a) = 1, \forall(\boldsymbol{s}, a) \in S \times A$ (for discrete action), or $\Psi_0(s) = 0, \forall s \in S$ (for continues action)
for $k = 1$ to K **do**
  Let S_k be m trajectories sampled by executing π_{k-1}:
    $\pi_{k-1}(a|\boldsymbol{s}) = e^{\Psi_{k-1}(\boldsymbol{s},a)}/\sum_b e^{\Psi_{k-1}(\boldsymbol{s},b)}$
  with ϵ-greedy for discrete action, or
    $\pi_{k-1}(a|\boldsymbol{s}) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\Psi_{k-1}(\boldsymbol{s})-a)^2}{\sigma^2}\right)$
  with a width of σ for continues action
  Generate functional gradient examples D_k as
    $D_k = \{((\boldsymbol{s}, a), \nabla_{\Psi(\boldsymbol{s},a)}\tilde{\rho}_\tau(\pi_\Psi))\}$
  for discrete action, or for continues action as
    $D_k = \{(\boldsymbol{s}, \nabla_{\Psi(\boldsymbol{s},a)}\tilde{\rho}_\tau(\pi_\Psi))\}$
  for all $(\boldsymbol{s}, a)$ in all $\tau \in S_t \cup Pool_{best} \cup Pool_{uniform}$
  Train a regression model h_k using $\mathcal{L}$ from D_k
  $\Psi_k = \Psi_{k-1} + \eta_k h_k$
  Update $Pool_{best}$ and $Pool_{uniform}$ with S_k
end for
return π_K

The PolicyBoost Algorithm

In order to verify the effective of the new objective function, we propose an *actor-only* algorithm, the PolicyBoost as in Algorithm 1. The details are explained in the follows. The codes can be found from http://cs.nju.edu.cn/yuy.

The 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 the 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 (Friedman 2001), we can calculate the gradient function of Eq.(5) at a state-action pair $(\boldsymbol{s}_k, a_k)$ in a trajectory τ as

$$\nabla_{\Psi(\boldsymbol{s}_k,a_k)}\tilde{\rho}_\tau(\pi_\Psi)=\frac{2}{n}\frac{p^{\pi_\Psi}(\boldsymbol{\tau})}{q(\boldsymbol{\tau})}\hat{R}(\boldsymbol{\tau})\nabla_{\Psi(\boldsymbol{s}_k,a_k)}\log p^{\pi_\Psi}(\boldsymbol{\tau})$$

$$=\frac{2}{n}\frac{p^{\pi_\Psi}(\boldsymbol{\tau})}{q(\boldsymbol{\tau})}\hat{R}(\boldsymbol{\tau})\nabla_{\Psi(\boldsymbol{s}_k,a_k)}\log p^{\pi_\Psi}(\boldsymbol{s}^{\boldsymbol{\tau}}_{k+1}|\boldsymbol{s}^{\boldsymbol{\tau}}_k)$$

$$=\frac{2}{n}\frac{p^{\pi_\Psi}(\boldsymbol{\tau})}{q(\boldsymbol{\tau})}\frac{p(\boldsymbol{s}^{\boldsymbol{\tau}}_{k+1}|\boldsymbol{s}^{\boldsymbol{\tau}}_k,a_k)}{p^{\pi_\Psi}(\boldsymbol{s}^{\boldsymbol{\tau}}_{k+1}|\boldsymbol{s}^{\boldsymbol{\tau}}_k)}\hat{R}(\boldsymbol{\tau})\nabla_{\Psi(\boldsymbol{s}_k,a_k)}\pi_\Psi(a_k|\boldsymbol{s}_k),$$

where $\hat{R}$ is the centralized reward among the training trajectories, and the gradient of the policy is, for discrete action space,

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

and for continuous action space,

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

Then we calculate the gradient value at every state-action pair in every available trajectory, and construct a data set D_t 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.

Note that the gradient requires to calculate the probability of sampling probability $q(\boldsymbol{\tau})$. In a model-based environment, where the transition probability can be accurately estimated, we can do the calculation, while in a model-free environment, we can simply let $q(\boldsymbol{\tau}) = p^{\pi_\Psi}(\boldsymbol{\tau})$, and the fraction is aways one.

Experiments

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 of the agent 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 from $\{(x,\dot{x})|x \in [-0.75,-0.25], \dot{x} \in [-0.025, 0.025]\}$. 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 with 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 correspond to torque to the joint between the first and second link of $-1, 0, 1$ respectively. The goal of the agent is to let the tip above the goal line from an initial state from $\{(\theta_1, \theta_2,), \dot{\theta_1}, \dot{\theta_2}|\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2} \in [-0.5, 0.5]\}$. 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 (Sutton and Barto 1998).

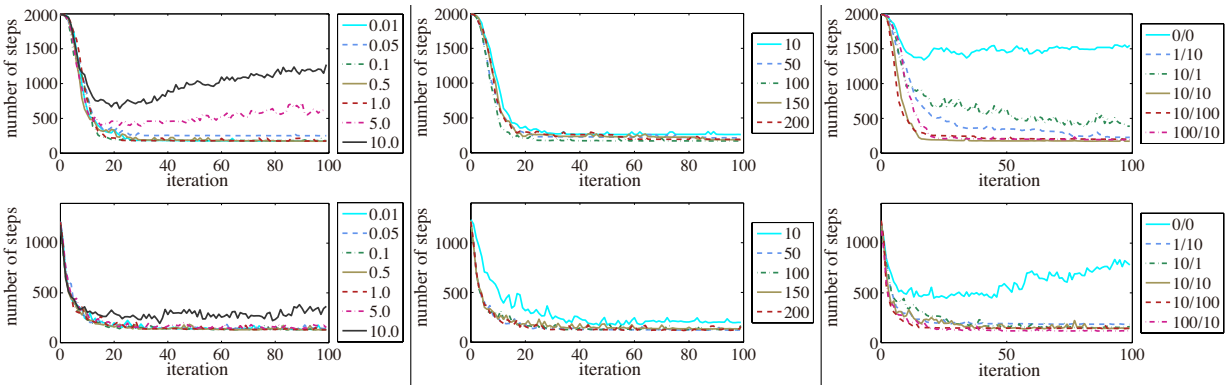
In the implementation of the PolicyBoost, the base regression learner used in all policies is regression decision trees (Breiman, Friedman, and Stone 1984) as implemented in WEKA (Witten and Frank 2005), and we set $m = 50$. The variable parameters include the learning rate, the decision tree depths, and the size of the pools. The performance of with different learning rates are shown in Figure 1.

It can be observed that, first, except with 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, the tree depth does not show a significant difference except for the smallest depth 10. Consider that it would be time consuming to employ deep trees, 100 could be sufficient. On the pool size, we observe that it is quite different whether the pool is used, since 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 from the observations.

Comparison with NPPG

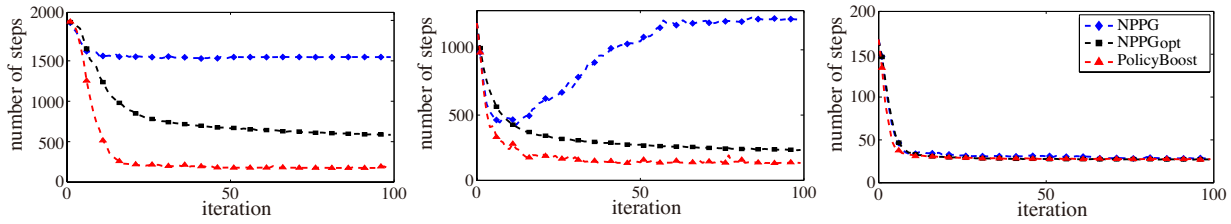
In the following we compare the PolicyBoost with a fixed configuration with NPPG. For PolicyBoost, we set $\eta = 1$, $\epsilon = 0$, tree depth$= 100$, and the pool size is 10 for the both pools. For NPPG, we make two versions, the one named "NPPG" uses the same fixed configuration as PolicyBoost, and the other named "NPPG$_{\text{opt}}$" is with carefully tuned parameters: $\eta = 0.1$ and $\epsilon = 0.2$ for Mountain Car, and $\eta = 0.08$ and $\epsilon = 0.1$ for Acrobot. For this comparison, we use an extra domain Corridor World (Kersting and Driessens 2008), which is a very simple task that an agent from any random position $x \in [4, 6]$ in a one dimensional corridor $[0, 10]$ need to reach one of the exits at both ends (0 and 10). For Corridor World, NPPG$_{\text{opt}}$ uses $\eta = 0.05$ and $\epsilon = 0.1$.

The comparison results are presented in Figure 2. On all three tasks, even if PolicyBoost uses the same fixed configuration, it produces policies with a clear trend of convergence



(a) On learning rate (b) On tree depth (c) On pool size

Figure 1: Performance of PolicyBoost with different parameters. The top row is on Mountain Car; the bottom row is on Acrobot. The first column shows different learning rates, the second column shows different maximal depth of the regression decision tree, and the third column shows different pool sizes (best trajectory pool size/uniform trajectory pool size)



(a) Mountain Car (b) Acrobot (c) Corridor World

Figure 2: Performance of different policies at each iteration. The fewer steps the better. Plots (a) and (b) share the legend with plot (c).

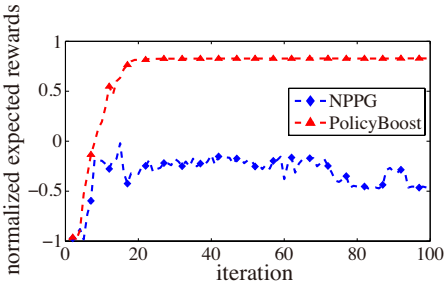


Figure 3: Normalized rewards of NPPG and PolicyBoost on the Mountain Car task.

to the near optimal solution. NPPG works the same as PolicyBoost only on Corridor World, possibly because this task is quite simple so that the occurring-probability-pursuing issue is not important. On Mountain Car and Acrobot, NPPG with the fixed configuration does not seem to converge at all. Only after the parameter tuning, NPPG_{opt} can gain some improvement, but is still worse than PolicyBoost.

To verify whether the occurring-probability-pursuing problem do exists, 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 3. 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 occurring-probability-pursuing problem, which injures NPPG.

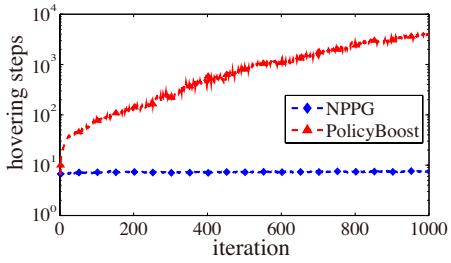


Figure 4: Performance of different policies on the Helicopter Hovering task. The more steps the better.

Application to Helicopter Control

We use a simulator (Tanner and White 2009) 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 (Abbeel, Ganapathi, and Ng 2005). This is a challenge task that the agent needs to manipulate 4 continuous control inputs simultaneously based on a 12-dimensional state space. Previously this task was considered as a hard challenge and methods were tested(Abbeel et al. 2007; Koppejan and Whiteson 2011).

We set the maximal horizon 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 a good parameter, so the best performance is reported, as in Figure 4. NPPG does not able to find the right gradient direction to improve the hovering steps, while PolicyBoost constantly improve the policy, without any parameter turning.

Conclusion

In this paper, we propose the ranking-based objective function to avoid the occurring probability occurring-probability-pursuing problem in NPPG, and derive the PolicyBoost method which searches in an additive function space for a policy that maximizes the ranking-based objective. Empirical studies on several domains show that 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. The future work will focus on designing more appropriate mechanism for trajectory reuse than the current pool based solution.

References

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

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

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

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

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

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

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

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

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

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

Kimura, H. 1995. Reinforcement learning by stochastic hill climbing on discounted reward. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, 295–303.

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

Koppejan, R., and Whiteson, S. 2011. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary Intelligence* 4(4):219–241.

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

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

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

Peshkin, L. 2001. *Reinforcement learning by policy search*. Ph.D. Dissertation, MIT.

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

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

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

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

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

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

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