Rollout Allocation Strategies for Classification-based Policy Iteration

Victor Gabillon VICTOR.GABILLON@INRIA.FR
Alessandro Lazaric ALESSANDRO.LAZARIC@INRIA.FR
Mohammad Ghavamzadeh MOHAMMAD.GHAVAMZADEH@INRIA.FR
INRIA Lille - Nord Europe, Team SequeL, 40 avenue Halley, 59650 Villeneuve d'Ascq, France

Abstract

Classification-based policy iteration algorithms are variations of policy iteration that do not use any kind of value function representation. The main idea is **1)** to replace the usual value function learning step with rollout estimates of the value function over a finite number of states, called the rollout set, and the actions in the action space, and **2)** to cast the policy improvement step as a classification problem. The choice of rollout allocation strategies over states and actions has significant impact on the performance and computation time of this class of algorithms. In this paper, we present new strategies to allocate the available budget (number of rollouts) at each iteration of the algorithm over states and actions. Our empirical results indicate that for a fixed budget, using the proposed strategies improves the accuracy of the training set over the existing methods.

1. Introduction

Classification-based policy iteration algorithms were proposed as an alternative to standard policy iteration (Lagoudakis & Parr, 2003b; Fern et al., 2004). These methods have been theoretically analyzed (Fern et al., 2006; Lazaric et al., 2010) and successfully applied to benchmark problems (Lagoudakis & Parr, 2003b; Fern et al., 2004). They replace the value function learning step with rollout estimates of the value function Q^π over a finite number of states $\mathcal{D}_R = \{x_i\}_{i=1}^N$, called the *rollout set*, and all the actions in the action space, and cast the policy improvement step as a *classification* problem. The rollouts aim to identify the greedy actions (w.r.t. the current policy) at the states in the rollout set in order to form a training set for the policy improvement classifier. As a result, the strategies used to allocate the available rollouts over states and actions have significant impact on the performance and computation time of these algorithms. To the best of our knowledge, there has been only one attempt to explicitly address this issue and to present strategies other than uniform allocation over the states and actions (Dimitrakakis & Lagoudakis, 2008). However, the proposed strategies are only over states and allocation over actions is simply uniform.

In this paper, we present new strategies to allocate the available budget (number of rollouts) at each iteration of the algorithm over states and actions. We empirically evaluate the performance of the proposed rollout allocation strategies and compare them with the existing methods in two widely-used reinforcement learning (RL) problems: mountain car and inverted pendulum.

2. Preliminaries

A discounted Markov decision process $\mathcal{M}$ is a tuple $\langle \mathcal{X}, \mathcal{A}, r, p, \gamma \rangle$, where the state space $\mathcal{X}$ is a bounded closed subset of a Euclidean space $\mathbb{R}^d$, the set of actions $\mathcal{A}$ is finite ($|\mathcal{A}| = M < \infty$), the reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is uniformly bounded by $R_{\max}$, the transition model $p(\cdot|x,a)$ is a distribution over $\mathcal{X}$, and $\gamma \in (0,1)$ is a discount factor. We use π to denote a deterministic policy $\pi : \mathcal{X} \to \mathcal{A}$. The action-value function for a policy π, Q^π, is the solution to the system $Q^\pi(x,a) = r(x,a) + \gamma \int_{\mathcal{X}} p(dy|x,a) Q^\pi(y,\pi(y))$. The optimal action-value function Q^* is defined as $Q^*(x,a) = r(x,a) + \gamma \int_{\mathcal{X}} p(dy|x,a) \max_{a' \in \mathcal{A}} Q^*(y,a')$. A deterministic policy π is *greedy* w.r.t an action-value function Q, if $\pi(x) \in \arg\max_{a \in \mathcal{A}} Q(x,a), \forall x \in \mathcal{X}$. Greedy policies are important because any greedy policy w.r.t. Q^* is optimal. We define the greedy policy operator $\mathcal{G}$ as $(\mathcal{G}\pi)(x) = \arg\max_{a \in \mathcal{A}} Q^\pi(x,a)$.[1]

3. Rollout Classification-based Policy Iteration

A template for this class of algorithms is shown in Figure 1. The algorithm begins with an arbitrary policy π_0 in the policy space Π. At each iteration k, given policy π_k, a new policy π_{k+1} is computed as the best

[1] The tie among the actions maximizing $Q^\pi(x,a)$ is broken in an arbitrary but consistent manner.

Input: policy space Π, state distribution ρ, total budget (available # of rollouts) per iteration L
Initialize: Let $\pi_0 \in \Pi$ be an arbitrary policy
for $k = 0, 1, 2, \ldots$ **do**
  Construct a rollout set $\mathcal{D}_R = \{x_i\}_{i=1}^{N},\ x_i \overset{\text{iid}}{\sim} \rho$
  Set $\mathcal{D}_T = \emptyset$, $c(x,a) = 0$ for any $x \in \mathcal{D}_R$ and $a \in \mathcal{A}$
  for $l = 1, 2, \ldots, L$ **do**
    $x =$ Select-State$(\mathcal{D}_R, \ldots)$
    $a =$ Select-Action$(x, \mathcal{A}, \ldots)$
    $c(x,a) = c(x,a) + 1$
    Perform a rollout according to π_k and return $R^{\pi_k}_{c(x,a)}(x,a) = r(x,a) + \sum_{t \geq 1} \gamma^t r\big(x^t, \pi_k(x^t)\big)$, with $x^t \sim p\big(\cdot\,|x^{t-1}, \pi_k(x^{t-1})\big),\ x^1 \sim p(\cdot|x,a)$
    $\widehat{Q}^{\pi_k}(x,a) = \frac{1}{c(x,a)} \sum_{j=1}^{c(x,a)} R^{\pi_k}_j(x,a)$
  end for
  for all states $x \in \mathcal{D}_R$ **do**
    if a dominating action $\hat{a}^*$ exists in x **then**
      $\mathcal{D}_T = \mathcal{D}_T \cup \{(x, \hat{a}^*), +\}$
      $\mathcal{D}_T = \mathcal{D}_T \cup \{(x,a), -\},\ \forall a \neq \hat{a}^*$
    end if
  end for
  $\pi_{k+1} =$ Classifier$(\mathcal{D}_T)$
end for

Figure 1. A template for rollout classification-based policy iteration algorithms.

approximation of $\mathcal{G}\pi_k$, by solving a classification problem. At the beginning of each iteration, a rollout set $\mathcal{D}_R$ is constructed by sampling N states from a distribution ρ. While the number of rollouts used at the current iteration is less than the total budget L, i.e., the available number of rollouts per iteration, a state in $\mathcal{D}_R$ and an action in $\mathcal{A}$ are selected according to *Select-State(·)* and *Select-Action(·)* procedures. After the total budget is spent, a training set $\mathcal{D}_T$ is formed using the estimated action-value function as follows: for each $x \in \mathcal{D}_R$, if the estimated value $\widehat{Q}^{\pi}(x, \hat{a}^*)$ is greater than the estimated value of all other actions (with *high confidence*), the state-action pair $(x, \hat{a}^*)$ is added to the training set with a positive label. In this case, (x,a) for the rest of the actions are labelled negative and added to the training set. Finally, a classifier is trained using $\mathcal{D}_T$ and generates a new policy π_{k+1} as an approximation to $\mathcal{G}\pi_k$. The classifier minimizes the number of misclassifications in $\mathcal{D}_T$, i.e.,

$$\pi_{k+1} = \arg\min_{\pi \in \Pi} \frac{1}{|\mathcal{D}_T|} \sum_{i=1}^{|\mathcal{D}_T|} \mathbb{I}\left\{\hat{a}^*(x_i) \neq \pi(x_i)\right\}. \qquad (1)$$

The main bottleneck of these algorithms is the computational cost of the rollouts needed to reach a good level of accuracy in selecting the greedy action at the rollout states. Previous experimental studies suggest that the main source of (computational) loss is at the states in which **i)** the difference between the action-values of the greedy and sub-greedy actions is small and **ii)** a greedy action can be identified without exhausting all the rollouts allocated to it. In this paper, we present new strategies for rollout allocation over states and actions in order to deal with these issues.

4. Rollout Allocation Strategies

In the following we consider a slightly simplified version of the algorithm in Figure 1, in which no high confidence test is used and all the rollout states are included in the training set. This way we can focus on the effectiveness of the allocation strategies in generating an accurate training set when a fixed budget L is available. The objective of maximizing the accuracy of the training set is consistent with the problem solved by the classifier. In fact, the classifier tries to fit to the training samples by minimizing the empirical average number of mistakes (Eq. 1). Thus, inaccuracy in training set may lead the classifier to approximate a policy which is far from the greedy one.

4.1. Strategies over Actions

While the global budget L is known and fixed in advance, the number of rollouts available at each rollout state might be different depending on the state allocation strategy. On one hand, the objective is to reduce as much as possible the number of rollouts needed to achieve a fixed accuracy at each state. For this scenario, one can consider using Hoeffding or Bernstein races strategies. On the other hand, the budget L is uniformly divided over states and the objective of the strategy over actions is to allocate the L/N rollouts available in each state so as to maximize the accuracy of the estimated greedy actions. For this scenario, we describe below a successive rejects algorithm.

The successive rejects (*SR*) strategy was proposed by Audibert et al. (2010) in the bandit setting for the identification of the best arm. The budget $L(x) = L/N$ is the only input of this method and is split into $M-1$ phases. At the end of each phase the action with the smallest estimated value $\hat{Q}(x,a)$ is discarded from the action set and the rollouts in the next phase are allocated uniformly on the remaining actions. When all the $L(x)$ rollouts have been allocated, the action (among the two remaining ones) with the highest estimated action-value is chosen as the greedy action in x. More formally, the phases are designed so that the number of rollouts allocated to the actions are:

$$P(x, a^m) = \left\lceil \frac{1}{\overline{\log}(M)} \frac{L(x) - M}{M + 1 - m} \right\rceil \qquad \forall m \in 1, \ldots, M-1,$$

where $\overline{\log}(M) = 0.5 + \sum_{m=2}^{M} 1/m$ and a^m is the action discarded at the end of the m-th phase. Compared to uniform, this strategy samples more the most promising arms in order to discriminate them more accurately. It is possible to prove that the SR strategy optimally minimizes the probability of mistakes in the identification of the greedy action (up to a log factor).

4.2. Strategies over States

Previous work (Fern et al., 2006; Lazaric et al., 2010) showed that a mistake in selecting the greedy action should be weighted by the regret incurred by the wrong action. According to this observation, we define a regret-based (RB) strategy over states that allocates rollouts to states at which the current estimation of the greedy action might be wrong, and thus, lead to a big regret. In a state x we define the *regret* as the difference between the action-values of the true and estimated greedy actions

$$\rho_l(x) = Q(x, a^*(x)) - Q(x, \hat{a}_l^*(x)) \,, \tag{2}$$

where $\hat{a}_l^*(x)$ is the estimated greedy action in state x after l rollouts. Unfortunately, the regret ρ is not available since the action-value function is unknown. Nonetheless, similar to Dimitrakakis & Lagoudakis (2008), we use UCB-like confidence intervals on the action-values to build an over-estimation of the regret. In particular, we define a high-probability upper bound on the true regret as

$$\hat{\rho}_l(x) = \max_{a \neq \hat{a}_l^*(x)} \left(\hat{Q}(x,a) + \sqrt{\frac{2 \log c(x)}{c(x,a)}} \right) - \left(\hat{Q}(x, \hat{a}_l^*(x)) - \sqrt{\frac{2 \log c(x)}{c(x, \hat{a}_l^*(x))}} \right) \,, \tag{3}$$

where $c(x) = \sum_{a \in \mathcal{A}} c(x,a)$. Given the estimated regret, the next rollout is allocated to the state with the highest regret. It is worth noting that the RB strategy is completely independent from the strategy used to allocate rollouts over actions in a selected state.

5. Experiments

In this section, due to the lack of space, we only report the empirical evaluation of action allocation strategies (allocation over states is uniform). The experiments are made in two standard RL problems: mountain car and inverted pendulum. There are only three possible actions in these domains. To better show the difference between the action allocation strategies, we artificially increase the number of actions using a method suggested in *RL competition 2009*. We map extra actions to the three original actions independently from the state. We will describe the mapping used for each problem in Sections 5.1 and 5.2.

5.1. Mountain Car

The mountain car problem is to drive a car to the top of a one-dimensional hill. The car is not powerful enough to accelerate directly up the hill, thus, it must learn to oscillate back and forth to build up enough inertia. There are three possible actions: forward $+1$, reverse -1, and stay 0. The reward is -1 on all time steps until the car reaches its goal at the top of the hill, which ends the episode with a reward 0. The discount factor is set to 0.99. We use the formulation described in Dimitrakakis & Lagoudakis (2008) with uniform noise in $[-0.2, 0.2]$ added to the actions. We increase the number of actions by mapping one to forward, one to reverse, and the rest to the stay action.

5.2. Inverted Pendulum

The problem is to balance a pendulum at the upright position by applying force to the cart it is attached to. There are three possible actions: left (-50N), right ($+50$N), and stay (0N), with uniform noise in $[-15, 15]$ added to them. The reward is 0 as long as the pendulum is above the horizontal line. The episode ends with reward -1 when the pendulum goes below the horizontal line. The discount factor is set to 0.95. We use the formulation described in Lagoudakis & Parr (2003a). When we increase the number of actions, we map one to right, one to left, and define the rest as stochastic actions that with equal probabilities behave as one of the three possible actions.

5.3. Evaluation

Following the setting of Dimitrakakis & Lagoudakis (2008), we use a multi-layer perceptron with 10 hidden units as the classifier in our experiments. To train this classifier, we run stochastic gradient descent with a learning rate of 0.5 for 200 iterations. We noticed that 25 iterations used by Dimitrakakis & Lagoudakis (2008) is not enough for training the classifier in all cases. In Figures 2 and 3, we compare the performance of two rollout allocation strategies over actions, uniform and successive rejects, for different values of the available budget L, the number of actions M, and the number of states in the rollout set N. In these figures, the horizontal axis is L/MN, the number of rollouts that the uniform strategy allocates to each state-action pair. The results are averaged over 1000 runs.

We first compare these two strategies in terms of the accuracy of the training sets they generate. We use the two strategies to find the greedy actions w.r.t. to a fixed deterministic policy that takes action forward if the velocity is positive and backward otherwise, for 400 states in the mountain car problem. The top-left panel in Figure 3 shows the difference between the percentage of mistakes made by these two strategies. Figure 2 and the remaining panels in Figure 3 show the difference between the performance of the policies learned after 5 iterations by both strategies in mountain car and inverted pendulum, respectively. In mountain car, the performance of a policy is evaluated in terms of the number of steps to goal with a maximum of 300 steps. In inverted pendulum, the performance of a policy is evaluated in terms of the number of steps that it keeps the pendulum balanced with a maximum of 3000 steps.

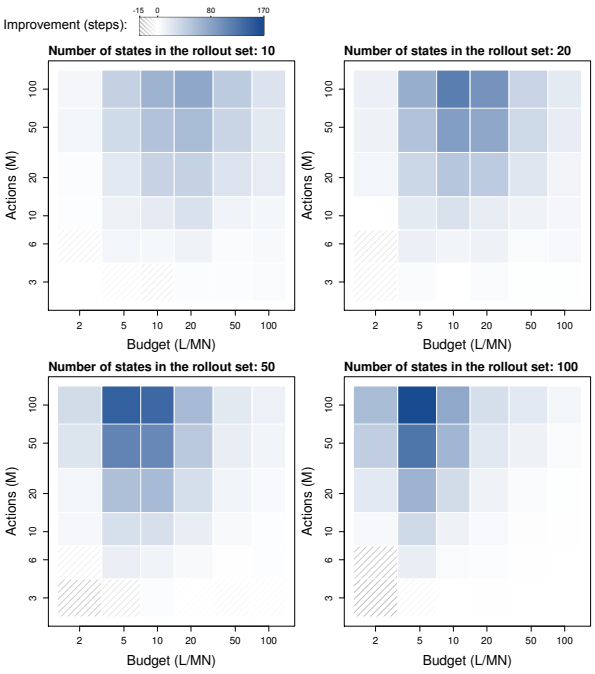

Figure 2. The performance of the learned policy in mountain car.

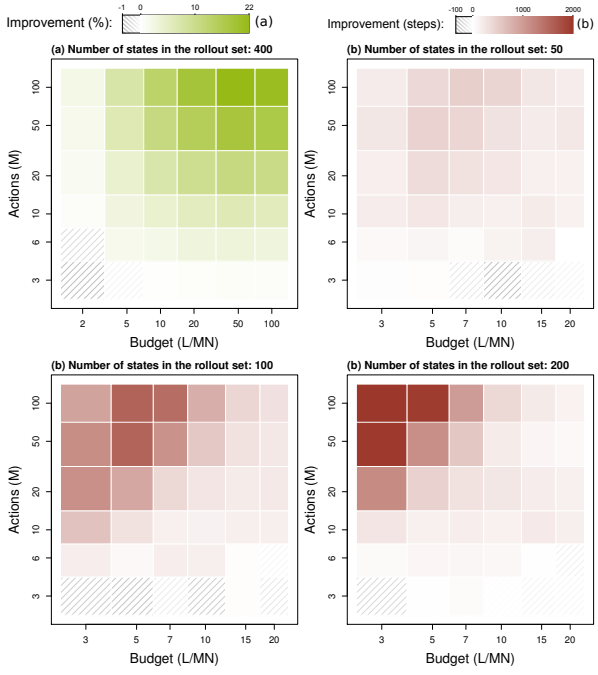

Figure 3. The accuracy of the training set in mountain car (top-left) and the performance of the learned policy in inverted pendulum (top-right and bottom).

The results in the top-left panel of Figure 3 indicate that using the SR allocation strategy improves the accuracy of the training set used by the classifier at each iteration. All the other graphs show that these improvements effectively propagate through iterations finally leading to a better performance for the final policy learned by the algorithm. In particular, we report results for different combinations of parameters L, M, and N. In both mountain car and inverted pendulum extreme values of the parameters make the problem either too hard or too easy to solve for both strategies, which achieve almost the same performance. For instance, consider the top-left graph of Figure 2 when $N = 10$, $M = 100$, and $L/MN = 2$. In this case not enough rollouts are available and both *SR* and *Uniform* generate poor training sets. Nonetheless, as soon as the budget increases *SR* effectively takes advantage of a better allocation of the rollouts and finally obtains policies with a better performance than *Uniform* by up to 70 steps. When the budget becomes even bigger, the *Uniform* allocation strategy has enough rollouts to achieve an accuracy similar to *SR* thus leading again to a similar performance. Another interesting effect that can be noticed from the graphs is that the difference in performance between the two strategies increases with the number of actions. In fact, the new actions are replica of the sub-greedy actions, thus making the identification of the greedy action even more difficult for the *Uniform* strategy. On the other hand, *SR* is more and more effective in discarding sub-greedy actions and identifying the greedy one. Finally, we notice that increasing the size of the rollout set makes the difference bigger although we expect it to stop increasing when very large rollout sets are used. In fact, the low complexity of the classifier prevent it from overfitting, thus increasing even more the size of the rollout set could make *Uniform* perform as *SR* even if the second generates more accurate training sets. The same results hold for the inverted pendulum. The negative values in the graphs are mostly due to noise.

References

Audibert, J.-Y., Bubeck, S., and Munos, R. Best arm identification in multi-armed bandits. In *COLT*, 2010.

Dimitrakakis, C. and Lagoudakis, M. Rollout sampling approximate policy iteration. *Machine Learning Journal*, 72(3):157–171, 2008.

Fern, A., Yoon, S., and Givan, R. Approximate policy iteration with a policy language bias. In *NIPS*, 2004.

Fern, A., Yoon, S., and Givan, R. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25:85–118, 2006.

Lagoudakis, M. and Parr, R. Least-squares policy iteration. *JMLR*, 4:1107–1149, 2003a.

Lagoudakis, M. and Parr, R. Reinforcement learning as classification: Leveraging modern classifiers. In *ICML*, pp. 424–431, 2003b.

Lazaric, A., Ghavamzadeh, M., and Munos, R. Analysis of a classification-based policy iteration algorithm. In *ICML*, 2010.