Self-Practice Imitation Learning from Weak Policy

Qing Da, Yang Yu(✉), and Zhi-Hua Zhou

National Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210046, China
{daq, yuy, zhouzh}@lamda.nju.edu.cn

Abstract. Imitation learning is an effective strategy to reinforcement learning, which avoids the delayed reward problem by learning from mentor-demonstrated trajectories. A limitation for imitation learning is that collecting sufficient qualified demonstrations is quite expensive. In this work, we study how an agent can automatically improve its performance from a weak policy, by automatically acquiring more demonstrations for learning. We propose the LEWE framework to sample tasks for the weak policy to execute, and then learn from the successful trajectories to achieve an improvement. As the sampling strategy is the key to the efficiency of LEWE, we further propose to incorporate active learning for the sampling strategy for LEWE. Experiments in a spatial positioning task show that LEWE with active learning can effectively and efficiently improve the weak policy and achieves a better performance than the comparing sampling approaches.

Keywords: Imitation learning · Active sampling

1 Introduction

In traditional reinforcement learning, an agent receives a reward after a sequential of actions, and tries to figure out the best actions according to the received rewards [1]. The high latency of the reward feedback forces the agent to search in a very large state space, which is of low efficiency and low effectiveness. Motivated by the teaching process in human society, imitation learning [2], alleviates this difficulty by introducing an expert mentor. The mentor provides demonstrations that successfully accomplish a specific task, and then the agent learns to follow the demonstrations that provide much more guiding information.

Imitation learning has drawn many attentions, and several approaches have been proposed (see [3] for a survey). These approaches roughly fall into two categories, inverse reinforcement learning and direct policy learning. In the first category, the aim is to learn a reward function, which associates actions with values, from the demonstration data instead of the delayed rewards, and the reward function is then used to derive a policy by reinforcement learning approaches [4,5]. In the second category, demonstration trajectories are used to directly

Z.-H. Zhou and F. Schwenker (Eds.): PSL 2013, LNAI 8183, pp. 9–20, 2013.
DOI: 10.1007/978-3-642-40705-5_2,

learn a mapping function from agent's state observations to action [6,7]. In this work, we focus on the second category for deriving policies directly.

Direct policy learning requires sufficient demonstrations from an expert mentor, so it has several limitations in real-world situations. First, it is usually quite expensive to collect demonstrations from expert mentors; even if we are free from the budget problem, expert mentors may also produce suboptimal demonstrations; and in real-world reinforcement learning problems, the state space can be extremely large so that it is hard to have enough demonstrations for a good learning.

In this work, in order to tackle the problem of insufficient demonstration, we investigate approaches that an agent can automatically acquire more demonstrations to improve itself. We propose the LEWE (LEarning from WEak policy) framework that automatically improves a *weak policy*. The main idea of LEWE is, for a few tasks, the weak policy can lead to successful trajectories, which can serve as good examples for learning an improved policy. Therefore, LEWE samples the task space and runs the weak policy on the sampled tasks in order to acquire successful trajectories. The sampling strategy is a key to the efficiency of LEWE. We therefore further propose three sampling strategies utilizing different information.

We experiment the proposed approaches in a spatial positioning task. Experiment results verify the effectiveness of LEWE by showing that LEWE significantly improves the weak policy with various configurations. Moreover, the results also show that the proposed active sampling leads to a better performance than other sampling strategies.

The rest of this paper starts with an introduction of the background, and then presents the LEWE framework, which is followed by the experiment results, and ends with a section of conclusion.

2 Background

A Markov Decision Process (MDP) can be represented by a 4-tuple (S, A, T, R), where S is a set of states, A is a set of actions, $T(s_j|s_i, a) : S \times A \times S \to \mathbb{R}$ is the transition probability of reaching state s_j from state s_i after executing a, and $R(s, a) : S \times A \to \mathbb{R}$ defines the immediate reward by executing a from state s_i. Given an MDP, the goal is to derive a policy π that maps from S to A maximizing a long term reward. Imitation learning derives a policy by learning from a set of successful demonstration trajectories $D = \{d_i = (s_{it}, a_{it})_{t=0}^{T_i}\}_{i=1}^{n}$ without using explicit reward functions.

One branch of imitation learning research focuses on forming the reward function, also named *inverse reinforcement learning* (IRL) [8]. This method assumes that there exists a latent reward function R of a given task and the actual intention of the demonstrations is to maximize the expected discounted reward over time. Based on this assumption, IRL learns a reward function from the trajectories data and then use conventional reinforcement learning algorithms to derive a policy, such as methods in [4,5,9].

Another branch alternatively focuses on learning a mapping function directly from demonstrated trajectories, also named *direct policy learning*. These methods directly learn a policy π from D by taking the state-action pairs (s_{it}, a_{it}) collected from demonstrations as training examples, using supervised learning algorithms, such as kNN [10], local weighted learning [11], decision trees [12], etc.

Active learning, also referred as *query learning* or *optimal experimental design* in the statistics literature, is a subfield of machine learning, which only queries the labels of useful instances so that it achieves a good performance with a small amount of examples [13]. Some popular approaches for active learning are [13,14].

3 The Proposed Approach

Our aim is to train an agent to accomplish a class of tasks. We assume without loss of generality that a task is parameterized by a vector $\boldsymbol{p} = [p_1, p_2, ..., p_C]^T$. The i-th parameter is constrained by the range L_i respectively, i.e., $p_i \in L_i = [a_i, b_i]$. Then $\mathcal{L} = L_1 \times L_2 \times \ldots \times L_C \in R^C$ is the *task space*, and C is the dimensionality of $\mathcal{L}$. A weak policy then can be defined as a policy which is only able to successfully accomplish tasks in $U \subseteq \mathcal{L}$ with a small volume ratio $\frac{|U|}{|\mathcal{L}|}$, in the given time. It is relatively easy to obtain a weak policy in practice, either from an insufficient imitation learning, or from hand-written rules. Our goal is to make an agent improve itself starting from a weak policy, such that it will be able to accomplish unseen tasks.

3.1 The LEWE Framework

We propose the *LEarning from WEak policy* (LEWE) framework that outlines the self-improve procedure for an agent, as shown in Algorithm 1.

A weak policy π_0, the task space $\mathcal{L}$, the maximum number of sampled tasks N and the maximum number of iterations in a run T are provided as the inputs of the framework. The framework consists of two phases, sampling and learning. In the sampling phase, LEWE first invokes *TaskSampling* to generate a task in line 3, and then executes the weak policy for the task at most T steps, in lines 4 to 10. In each step of executing a task, the agent queries an action a from the weak policy π_0 on the current state s in line 6, then the agent executes a and updates its state as in line 7, where *execute* returns the state of the agent after taking the action. A task is considered to be accomplished successfully if the goal state is reached (determined by the function *TaskFinished*) within T steps. If the task is accomplished successfully, LEWE records the trajectory as a successful demonstration, and at the same time, records the task parameters as a successful task or a failed task, in lines 11 to 17. In the learning phase, an improved policy π_* is learned from the recorded data D through direct policy learning. Note that, although one can easily take the improved policy as a weak policy and use LEWE to refine it again, we rather choose to stay studying the effectiveness of this simple framework.

Algorithm 1 The LEWE Framework

Input:
A weak policy π_0
Task space $\mathcal{L}$
The maximum number of sampled tasks N
The maximum number of iterations T

Output:
The learned policy π_*

$N_e \leftarrow 0$, $D \leftarrow \varnothing$, $P_{suc} \leftarrow \varnothing$, $P_{fail} \leftarrow \varnothing$
while $N_e < N$
  $\boldsymbol{p} = TaskSampling(\mathcal{L}, P_{suc}, P_{fail})$
  $s \leftarrow s_0$, $t \leftarrow 0$, $Demo \leftarrow \varnothing$
  while $!TaskFinished(\boldsymbol{p}, s)$ or $t < T$
    $a \leftarrow \pi_0(s)$
    $s \leftarrow execute(\boldsymbol{p}, s, a)$
    $Demo \leftarrow Demo \cup \{(s, a)\}$
    $t \leftarrow t + 1$
  end
  if $TaskFinished(\boldsymbol{p}, s)$
    $D \leftarrow D \cup \{Demo\}$
    $P_{suc} \leftarrow P_{suc} \cup \{\boldsymbol{p}\}$
  else
    $P_{fail} \leftarrow P_{fail} \cup \{\boldsymbol{p}\}$
  end
  $N_e \leftarrow N_e + 1$
end
$\pi_* \leftarrow directPolicyLearning(D)$

There are two important issues in order to achieve a successful application of LEWE framework. In LEWE framework, we expect that, by learning the successful demonstrations of a weak policy, an improved policy can be obtained via the generalization ability of the employed learning algorithm. Therefore, the generalization ability of the learning algorithm is important. The other issue is the sampling strategy that implements the *TaskSampling* function. We investigate three sampling strategies, *random sampling*, *evolutionary sampling* and *active sampling* in the follows.

3.2 Random Sampling

Random sampling simply selects a task $\boldsymbol{p}$ from the task space $\mathcal{L}$ uniformly at random. This strategy serves as the baseline approach, and is denoted as $TaskSampling_r$ function. A drawback of random sampling is that it is high likely to sample tasks that are too hard for a weak policy, which will waste a lot of time and resource.

3.3 Evolutionary Sampling

Evolutionary sampling is an improvement over random sampling on the basis of a simple observation: a weak policy will be more likely to succeed on tasks similar to the succeeded tasks in history rather than on totally strange tasks. Evolutionary sampling uses the mutation operator of evolutionary strategy algorithms [15] to generate new tasks by perturbating the succeeded tasks. Instead of sampling a task from $\mathcal{L}$ uniformly, with probability ρ, evolutionary sampling selects a succeeded task, and generates a new task by perturbating this task as following

$$p_i \leftarrow q_i + \mathcal{N}(0, \epsilon_i^2), i = 1, 2, ..., C \tag{1}$$

where $\mathcal{N}(0, \epsilon_i^2)$ is a normal distribution with mean zero and standard deviation ϵ_i. With the remaining $1 - \rho$ probability, it does the random sampling. The probability ρ can be regarded as a trade-off between the possibility of success of the new task and the overall exploration in the task space.

3.4 Active Sampling

The evolutionary sampling, however, may not try best to exploit the whole task space, moreover, is only aware of the succeeded tasks but not the failed tasks. Ideally, we want to sample a task that

1. will result a successful trajectory with confidence.
2. is dissimilar to the past tasks as much as possible.
3. can produce the states dissimilar to the past collected states as much as possible.

The first property is with the same idea of evolutionary sampling. The second one is based on the assumption that different tasks tend to produce different states. The other one directly seeks such a task by estimating the distribution of possible states for a given task.

Based on this idea, we propose an active sampling strategy incorporating the active learning [14] idea. For an unseen task, we employ an SVM model to estimate the confidence that the weak policy will be successful, a Gaussian Mixture Model to estimate the distance to the explored area in task space, and the likelihood that unseen states will be visited. The implementation details of these three components are introduced as the follows.

To estimate the probability that a task can be successfully executed by the weak policy, we train a classifier to distinguish the succeeded tasks from the failed tasks. We combine the records P_{suc} and P_{fail} to get a training set $P_{train} = (\boldsymbol{p}_j, y_j)_{j=1}^n$, where n is the total number of recorded tasks, and y_j is the label of task $\boldsymbol{p}_j$ and is assigned to 1 for $\boldsymbol{p}_j \in P_{suc}$ and -1 otherwise. Then a standard SVM classifier $f(\boldsymbol{p}) = \mathrm{w}^T \Phi(\boldsymbol{p})$ is trained from P_{train}, where Φ is a function mapping $\boldsymbol{p}$ to a high-dimension space that appears implicitly in the kernel function $K(\boldsymbol{p}_1, \boldsymbol{p}_2) = \langle \Phi(\boldsymbol{p}_1), \Phi(\boldsymbol{p}_2) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product. For any unseen task $\boldsymbol{p}$, the higher $f(\boldsymbol{p})$ is, the higher probability that task $\boldsymbol{p}$ can be successfully executed by the weak policy.

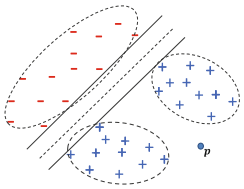


Fig. 1. An illustrative example of combining a SVM classifier and a GMM density estimator in task space

To estimate how far a task is from the past tasks, we use Gaussian Mixture Models (GMM) to estimate the visited area in task space. We approximate a distribution of task $\boldsymbol{p}$ belonging to the area we have explored with a GMM as

$$\Pr(\boldsymbol{p}|\tau_k, \mu_k, \Sigma_k) = \sum_{k=1}^{K} \tau_k \mathcal{N}(\boldsymbol{p}, \mu_k, \Sigma_k) \tag{2}$$

where K is number of components of this model, and $\{\tau_k, \mu_k, \Sigma_k\}_{k=1}^{K}$ are model parameters to be estimated. We estimate these parameters through the Expectation Maximization (EM) algorithm from data $P_{suc} \cup P_{fail}$. For any unknown task $\boldsymbol{p}$, the lower $\Pr(\boldsymbol{p})$ is, the higher probability that the surrounding of task $\boldsymbol{p}$ has not been explored yet. We give an illustration of above discussion in Fig. 1. By the SVM, we separate the succeeded and failed tasks, so that we can estimate the task $\boldsymbol{p}$ has a high probability to be successfully executed by the weak policy, since it is far from the failed tasks. By the GMM, we can estimate that the task $\boldsymbol{p}$ is not in the explored area of the task space. Further, to estimate how far all possible states produced by a task from the past collected states, we firstly build the relationship between a task and all the states it produces. For task $\boldsymbol{p}$, all states $\{s_i\}_{\boldsymbol{p}}$ produced by this task are assumed to be generated from a linear model

$$s^T = \boldsymbol{p}^T B + \boldsymbol{\epsilon}^T \tag{3}$$

here $B \in R^{C \times d}$ is a coefficient matrix, C is the dimensionality of task space and d is that of state space. $\boldsymbol{\epsilon}$ is a noise sampled from $\mathcal{N}(0, \Sigma_{\boldsymbol{\epsilon}})$.

Denote M as the number of states collected so far, and $\hat{S} = [s_1, s_2, ..., s_M]^T \in R^{M \times d}$ are all the historical states, and $\hat{P} = [p_1, p_2, ..., p_M]^T \in R^{M \times C}$ are the corresponding tasks that produced the states. Then the likelihood function can be represented as

$$p(\hat{S}|\hat{P}, B, \Sigma_{\boldsymbol{\epsilon}}) \propto |\Sigma_{\boldsymbol{\epsilon}}|^{-\frac{M}{2}} exp(-\frac{1}{2} tr((\hat{S} - \hat{P}B)^T \Sigma_{\boldsymbol{\epsilon}}^{-1} (\hat{S}^T - \hat{P}B))) \tag{4}$$

Since we don't have any prior knowledge of the distribution of the parameter, we apply the classical frequentist least squares solution to estimate B using moore-penrose pseudoinverse

$$B = (\hat{P}^T \hat{P})^{-1} \hat{P}^T \hat{S} \tag{5}$$

So for any unknown task $\boldsymbol{p}$, the probability that a history state s_i can be produced by $\boldsymbol{p}$ is $p(s_i|\boldsymbol{p})$, we can now estimate the improvement of the diversity in state space that the task $\boldsymbol{p}$ brings by calculating the probability of historical states being produced by $\boldsymbol{p}$

$$h(\boldsymbol{p}) = \prod_{i=1}^{M} p(s_i|\boldsymbol{p}) \tag{6}$$

For any unknown task $\boldsymbol{p}$, the lower $h(\boldsymbol{p})$ is, the higher probability that task $\boldsymbol{p}$ can produce more unobserved states.

Considering all three components, active sampling strategy uses the following objective function

$$g(\boldsymbol{p}) = -\mathrm{w}^T \Phi(\boldsymbol{p}) + c \sum_{k=1}^{K} \tau_k \mathcal{N}(\boldsymbol{p}, \mu_k, \Sigma_k) + \lambda \prod_{i=1}^{M} p(s_i|\boldsymbol{p}) \tag{7}$$

where c and λ are the trade-off coefficient. To find a task $\boldsymbol{p}^*$ that minimizes the objective function, we first employ the random sampling to generate a pool of tasks $P_{candidate}$, then we find $\boldsymbol{p}^*$ from the pool that

$$\boldsymbol{p}^* = \underset{\boldsymbol{p} \in P_{candidate}}{\arg\min} \; g(\boldsymbol{p}) \tag{8}$$

4 Experiment

We investigate three questions by experiments:

1. Can LEWE improve a weak policy?
2. Is active sampling better than the other two sampling strategies?
3. Is the generalization ability of the direct policy learning algorithm important to the performance of LEWE?

4.1 Spatial Positioning Task

We use a *positioning with heading* problem studied in [16] to validate our algorithms. This task consists of attaining a 2D planar target position with a target heading (x_g, y_g, θ_g) from the origin $(0, 0, 0)$, as shown in Fig. 2.

For this problem, the task parameters are the target position and the target heading, i.e., (x_g, y_g, θ_g). The corresponding task space is $\mathcal{L} = L_1 \times L_2 \times L_3 = [3m, 8m] \times [3m, 8m] \times [0, \frac{2\pi}{3} rad]$. For this class of tasks, some tasks are easy to perform (for example, task $(0, 3, 0)$ can be done by directly moving forward by 3 meters), while some other tasks are relatively hard such like task $(3, 3, \frac{2\pi}{3})$.

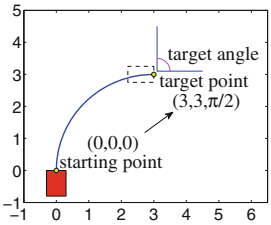


Fig. 2. The experimented task

4.2 The Weak Policy

We use a simple greedy-based method with a customized cost function R_0 as the weak policy π_0. This policy chooses the action that minimizes the expected cost of being executed at time t

$$\pi_0(s_t) = \underset{a_k}{argmin}\, \mathbf{E}[R_0(execute(s_t, a_k))] \tag{9}$$

The cost function R_0 is defined as a weighted summation of the axis distance between the agent and target position, absolute difference between the agent heading and the target heading and the distance to the *optimal line* (the line that passes through the target position (x_g, y_g) with the slope $tan(\theta_g)$), as

$$\begin{aligned} R_0(s_t) =& w_1|x_t - x_g| + w_2|y_t - y_g| + w_3|\theta_t - \theta_g| \\ &+ w_4 \frac{|Ax_t + By_t + C|}{\sqrt{A^2 + B^2 + C^2}} \end{aligned} \tag{10}$$

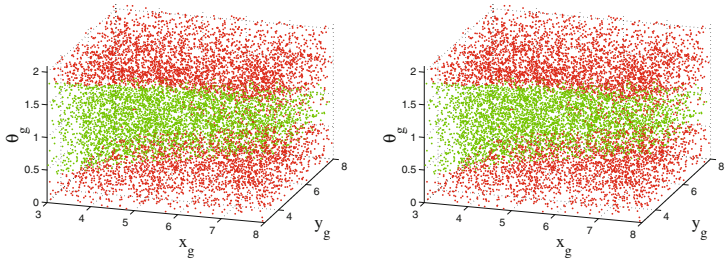
Here, A, B, C are the equations coefficients of *optimal line* and the weights used in the experiments are $w_1 = 0.1$, $w_2 = 0.1$, $w_3 = 0.15$ and $w_4 = 1$.

4.3 Experiment Setup

To measure the performance of our algorithms, policies are evaluated for success rate of task performing. A task is successfully executed if and only if the agent steps into $0.1\,m$ range of the target position and within $0.1\,rad$ error to the target angle, i.e., $\|x_t - x_g, y_t - y_g\| < 0.1\,m$ and $|\theta_t - \theta_g| < 0.1\,rad$, which can be considered as a more strict criterion comparing to the existing work (e.g. [16]).

In our experiments, each policy is tested with a test set containing 200 tasks uniformly sampled from $\mathcal{L}$. We employ 4 start-of-the-art supervise learning algorithms as the direct policy learning methods, i.e., kNN [17], decision tree [18] and random forest [19] implemented in WEKA [20] and SVM [21] in LIBSVM [22]. The parameters of the four classifiers are

- kNN: $k = 7$ (obtained by cross validation)
- Decision tree: Default settings of WEKA
- Random forest: Default settings of WEKA with ensemble size 100
- SVM: Radial Basis Function (RBF) kernel with $\gamma = 0.01$ and C is 200.



(a) real distribution (b) classification result by SVM

Fig. 3. The distribution of successful and failed tasks in tasks space

For task (x_g, y_g, θ_g), we extract 11 arbitrary geometry features from state (x, y, θ), including $x - x_g$, $y - y_g$, $\theta_g - \theta$, $\alpha - \theta$ (α is the angle of $(x_g - x, y_g - y)$), $\|x - x_g, y - y_g\|$ and some other geometry features.

We denote LEWE$_r$, LEWE$_e$ and LEWE$_a$ as LEWE with random sampling, evolutionary sampling and active sampling, respectively. For active sampling strategy, we also use LIBSVM as the SVM solver. Note that for both evolutionary sampling and active sampling strategies, there's no historical data in the very beginning, thus, random sampling strategy is used to sample first 30 successful demonstrations for both cases. The size of candidate set for active sampling is 100. The parameters ρ, c and λ are selected by cross validation. Finally, we run every configuration 200 times and report the mean success rate.

4.4 Experiment Results

We firstly investigate how the weak policy acts for this class of tasks. We randomly sample 10000 tasks from $\mathcal{L}$ and apply the weak policy to perform on these tasks. Figure 3a shows the task space, where the green points are the tasks accomplished (about 41 %) and the red ones are those not accomplished by the weak policy. It is easy to observe that the successful tasks are separated well from failed ones. Moreover, we apply a standard SVM classifier (with RBF kernel, C is 200 and γ is 0.1) to classify these tasks with a training set consists of only 75 randomly chosen tasks (30 successful v.s. 45 failed). The classification result is shown in Fig. 3b with an accuracy rate of about 88 %, which implies the learnability of easy and hard tasks.

We then investigate the first question, i.e., can LEWE improve the weak policy. Figure 4 plots the success rates against the number of executed demonstrations ranged from 30 to 300, for the weak policy as well as LEWE with the four learning algorithms and the three sampling strategies. We first observe that LEWE has a higher success rate than the weak policy, only except when using SVM with less than 60 demonstrations. It can also be observed that the success rate of LEWE increases as the number of the executed demonstrations increases. When 300 demonstrations are executed, LEWE achieves at least 0.67 success rate

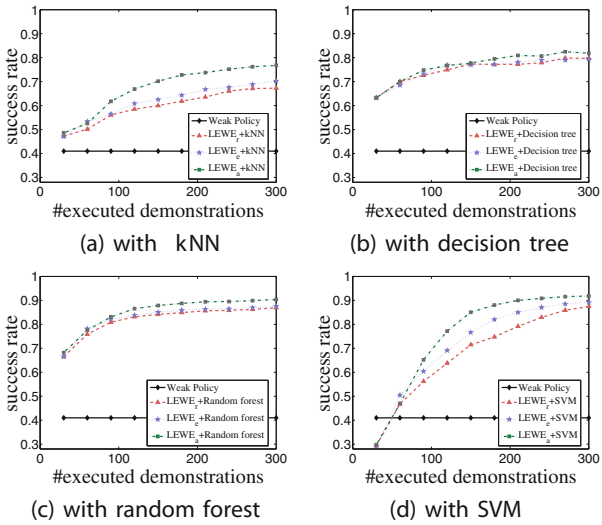


Fig. 4. The success rates of LEWE against the weak policy

Table 1. Success rates of LEWE after 300 executed demonstrations.

Algorithm	LEWE$_r$	LEWE$_e$	LEWE$_a$
kNN	0.67 ± 0.13	0.70 ± 0.11	$\mathbf{0.77 \pm 0.06}$
Decision tree	0.80 ± 0.11	0.79 ± 0.11	$\mathbf{0.82 \pm 0.11}$
Random forest	0.87 ± 0.05	0.88 ± 0.05	$\mathbf{0.90 \pm 0.04}$
SVM	0.87 ± 0.09	0.89 ± 0.07	$\mathbf{0.92 \pm 0.03}$

and as high as 0.92 success rate while that of weak policy is 0.41. Therefore, it is clear that LEWE effectively improves the weak policy.

For the second question, i.e., is the proposed active sampling better than the other, we look into Fig. 4, where each plot also compares the three sampling strategy with a policy learning algorithm. The parameter selected for ρ, c and λ are 0.6, 2.5 and 1.0. For active sampling, the GMM with only one component in the task space shows to be the best in this case, which in fact degrades into a Gaussian distribution. It can be observed that the curves of active sampling are almost always above the comparing curves, particularly when the number of executed demonstrations is large. Table 1 compares the success rates after 300 executed demonstrations, where it can be found that active sampling is significantly better than the compared approaches by the t-tests with 95 % confidence. To further investigate how the sampling strategies work, we plot the number of executed demonstrations against the sampled ones in Fig. 5a. It can be observed that random sampling executes more demonstrations than evolutionary and active sampling for sampling the same number of successful demonstrations, which confirms our design that evolutionary and active sampling treat the samples much smarter so that the waste of failed executions is fewer. Meanwhile from

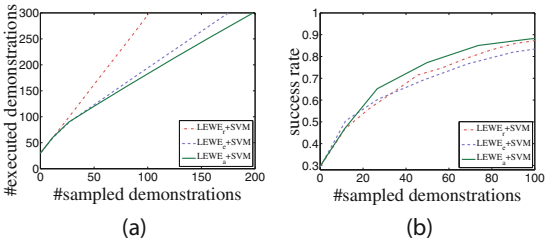


Fig. 5. Effectiveness of sampling strategies

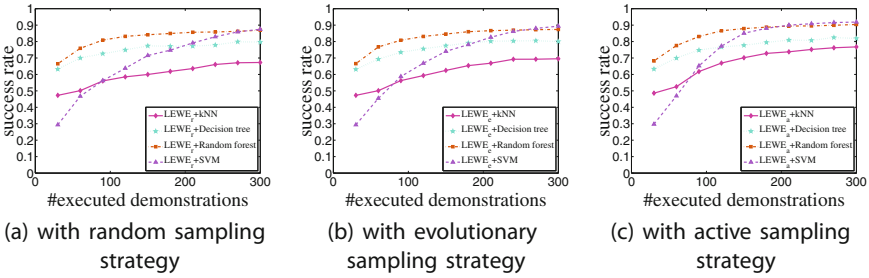


Fig. 6. The success rates of LEWE with different direct policy learning algorithms

Fig. 5b, it is clear that active sampling makes a better utilization of the samples than evolutionary sampling, which confirms our design that active learning utilizes full information. Finally, we investigate the third question, i.e., is generalization ability of the policy learning algorithm important. Figure 6 compares LEWE with the four learning algorithms with each sampling strategy. It can be observed that random forest always takes the most advantage except that SVM is comparable with random forest when there are 300 demonstrations. It is consistent with the experience that random forest and SVM are the two state-of-the-art supervised learning approaches [23]. We recommend using random forest as it has a more stable performance and fewer parameters to be tuned.

5 Conclusion

In this paper, we propose the LEWE framework for an agent to improve its performance from a weak policy using imitation learning. We incorporate active sampling for LEWE to smartly choose demonstrations to practice. Experiments verified the effectiveness of the LEWE framework as well as the active sampling strategy. The work verifies the possibility that an agent can acquire demonstrations to improve its performance by itself. The future work will combine the learning policy with other reinforcement learning approaches towards a better performance.

Acknowledgements. This research was supported by the Jiangsu Science Foundation (BK2012303), the 2013 State Grid Research Project, and the Baidu Fund (181315P00651).

References

1. Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. Cambridge University Press, Cambridge (1998)
2. Schaal, S.: Is imitation learning the route to humanoid robots. Trends Cogn. Sci. **3**(6), 233–242 (1999)
3. Argall, B., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Rob. Auton. Syst. **57**(5), 469–483 (2009)
4. Atkeson, C., Schaal, S.: Robot learning from demonstration. In: Proceedings of the ICML'97, San Francisco, USA, pp. 12–20, July 1997
5. Choi, J., Kim, K.: Inverse reinforcement learning in partially observable environments. In: Proceedings of IJCAI'09, Barcelona, Spain, pp. 1028–1033, July 2009
6. Jetchev, N., Toussaint, M.: Task space retrieval using inverse feedback control. In: Proceedings of ICML'11, Bellevue, WA, USA, pp. 449–456, June 2011
7. Zhang, D., Cai, Z., Nebel, B.: Playing tetris using learning by imitation. In: Proceedings of GAMEON'10, Leicester, UK, pp. 23–27, November 2010
8. Ng, A., Russell, S.: Algorithms for inverse reinforcement learning. In: Proceedings of ICML'00, Stanford, USA, pp. 663–670, June 2000
9. Ziebart, B., Maas, A., Bagnell, J., Dey, A.: Maximum entropy inverse reinforcement learning. In: Proceedings of AAAI'08, Chicago, USA, pp. 1433–1438, July 2008
10. Bentivegna, D.: Learning from Observation Using Primitives. Ph.D. thesis, College of Computing, Georgia Institute of Technology (2011)
11. Bentivegna, D., Atkeson, C.: Learning from observation using primitives. In: Proceedings of ICRA'11, Seoul, Korea, pp. 1988–1993, May 2001
12. Silver, D., Bagnell, J., Stentz, A.: Perceptual interpretation for autonomous navigation through dynamic imitation learning. Robot. Res. **70**, 433–449 (2011)
13. Settles, B.: Active learning literature survey. Computer Sciences Technical Report, University of Wisconsin-Madison (2009)
14. Huang, S., Jin, R., Zhou, Z.: Active learning by querying informative and representative examples. In: NIPS'11, pp. 892–900 (2011)
15. Beyer, H., Schwefel, H.: Evolution strategies-a comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002)
16. Argall, B., Browning, B., Veloso, M.: Learning robot motion control with demonstration and advice-operators. In: Proceedings of IROS'08, Nice, France, pp. 399–404, September 2008
17. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley, New York (2001)
18. Quinlan, J.: C4.5: Programs for machine learning. Morgan kaufmann, San Franscisco (1993)
19. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
20. Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Franscisco (2005)
21. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (2000)
22. Chang, C., Lin, C.: Libsvm: a library for support vector machines. ACM Trans. Intell. Syss. Technol. **2**(3), 27 (2011)
23. Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. In: Proceedings of ICML'06, Pittsburgh, PE, pp. 161–168 (2006)