Reducing T-step Reinforcement Learning to Classification

John Langford and Bianca Zadrozny

[1] TTI-Chicago
[2] IBM T.J. Watson

jcl@cs.cmu.edu,zadrozny@cs.ucsd.edu

Abstract. We discuss two reductions of reinforcement learning to classification: one in the presence of a generative model and the other in the presence of a (weaker) *trace* model. These reductions can be thought of as meta-algorithms for (approximately) solving reinforcement learning given a classification algorithm which succesfully solves certain natural subproblems.
Via the reductions, we provide new reinforcement learning algorithms, which apply to very large (and possibly infinite) state spaces. The structure of the reductions provides insight into the relationship between learning problems, and the trace model reduction applies to an intuitive reinforcement learning domain: *offline* reinforcement learning.

1 Introduction

Why should we be interested in reducing reinforcement learning to classification?

1. A reduction from reinforcement learning to classification implies new algorithms. The new algorithms are given by the composition of a classification algorithm with the reduction techniques outlined here. This automatic extension of the set of available techniques includes "support vector reinforcement learning", "RL with boosted decision trees", etc.
2. A reduction implies new theory. Suppose, for example, that we have some theorem bounding the true error rate of a learned classifier, c. Via the reduction, this also bounds the optimality of a learned policy.
3. Reductions are a method for understanding. Much of computational complexity theory and cryptology is a study of reductions. Knowing which problems reduce to other problems (and how efficiently they do so) provides understanding about the nature and difficulty of these problems.

The goal of reinforcement learning is finding a policy $\pi(a|o^*)$ which maps observation sequences to actions. Functionally, a policy is a (possibly probabilistic) mapping from observations to actions. In the case of only two actions (which we limit consideration to here), the functional form of a policy $\pi : O^* \to A$ is exactly equivalent to a binary classifier $c : X \to \{0, 1\}$ where the "features" are the observations and the "class" (0 or 1) is the choice of action. Given this, it seems plausible that we can construct a policy π given a good learning algorithm for classifiers.

In this paper we turn that plausible conjecture into reality with a precise definition of a set of classification problems which can be used to construct a nonstationary policy π to optimize the T-step reward of reinforcement learning.

As stated in the abstract, we present two reductions. The first reduction is from reinforcement learning with a generative model to classification. Good classification allows us to provide a *global*[3] guarantee on the optimality of the resulting policy. The drawback of this method is that it takes time exponential in the horizon, T. In fact, there exists a family of RL problems such that this exponential is unavoidable, so we precisely characterize the power of classification to solve RL in a generative model.

The second method works on a much weaker *trace model* (defined later) and has a running time polynomial in T. The trace model is weak enough that the reduction can easily apply in *offline* settings, where we cannot "reset" to an arbitrary state and see what alternative outcomes might have arisen. The drawback of this method is that we can only state a local improvement guarantee.

1.1 Past Work

The study of the sample complexity of policy evaluation [7] is the inspiration of this paper. Implicit in naming "the VC dimension of a policy" is the notion of a reduction, which we make explicit here. The value of making the reduction explicit is a more exact understanding of the nature of the reinforcement learning problem and a new class of algorithms for solving these problems. A reduction from reinforcement learning to regression [6] in the presence of a "μ-reset" is similar.

There has been quite a bit of other work recently on reductions of reinforcement learning to classification [1, 10, 3]. The algorithms presented here differ by being provably more robust, not requiring additional assumptions, and working in new settings.

In particular, the algorithms of [10, 3] relies upon approximate policy iteration in order to construct a policy formed from a classifer. Approximate policy iteration only converges to a near-optimal policy when the approximate policy is "almost perfect everywhere". Unfortunately, this l_∞ notion does not translate well into classifier performance where we typically hope to see "almost perfect almost everywhere" (an l_1 notion). Consequently, only 0-error classifiers can be constructed into optimal policies. In contrast, the results here work with an ϵ-error classifier for a small but nontrivial ϵ. In addition, the work here explicitly takes into account varying rewards (unlike [10]).

The algorithm PSDP [1] is a near-optimal reduction in the sense we use here. However, it requires an *additional* piece of information: the distribution over states of a (near) optimal policy. Here we do not make this strong additional assumption, achieving weaker results with weaker assumptions.

[3] As opposed to local guarantees as provided by policy gradient techniques.

2 Common Elements of Both Reductions

2.1 Definitions

We explicitly consider only the simpler case of a Markov Decision Process (MDP) where the observation sequence is summarized by a state s. A simple argument (see section 5.1) shows this is sufficient for the full reinforcement learning problem, given a generative or trace model.

An MDP is defined on a state set S and action set A. For every state s and action a, there is a probability $P(s'|a,s)$ over next states, s'. For every state s we enter, a reward $R(s) \in [0,1]$ is received. The distribution over initial states is $P(s_0)$. Our goal in reinforcement learning (RL) on an MDP is choosing a (possibly nonstationary) mapping from states to actions which maximizes the expected T-step reward. The expected T-step reward is defined as:

$$\eta(\pi) = E_{s_0,s_1,\ldots,s_T \sim P(s_0),(P(s'|a,s),\pi(a|s))^T} \frac{1}{T} \sum_{i=1}^{T} R(s_i). \tag{1}$$

In other words, $\eta(\pi)$ is the average expected T-step reward given the MDP and a policy π.

A classification problem is defined by an input domain X and discrete output domain Y. Examples are drawn from a distribution D over $X \times Y$. The goal in classification is finding a function $c : X \to Y$ which minimizes errors under the distribution D. The error rate is defined as:

$$E_{x,y \sim D}[I(c(x) \neq y)],$$

where $I(\cdot)$ is the indicator function which is 1 when its argument is true and 0 otherwise. For simplicity, we assume throughout the paper that $|A| = |Y| = 2$.

The reductions use classification for action prediction, where the input for classification is a state ($S = X$) and the output space is the set of actions ($A = Y$). Since classification is optimized with respect to a distribution D over the space $X \times Y$ (or equivalently $S \times A$), an essential element in both reductions is the specification of D.

2.2 From importance weighted classification to classification

One very important tool in both the reductions is a subreduction: from importance-weighted classification to pure classification. In the importance-weighted setting, it is more important to classify some examples correctly than others. In particular, every example has associated with it a (positive, real-valued) importance weight, w, and the goal is to minimize the expected importance of example mistakes:

$$E_{x,y,w \sim D}[w \; I(c(x) \neq y)] \tag{2}$$

Importance-weighted classification is used to model the fact that correctly classifying some examples is more important than others. For example, a bank may be much more interested in the correct classification of large loans as "good" or "bad" rather than small loans.

To model this setting, we assume that there is a distribution D over (input, output, importance) triples (x, y, w). A basic folk theorem[4] from decision theory notes that if we have examples drawn from the following altered distribution over pairs (x, y):

$$\hat{D}(x, y) \equiv \frac{w}{E_{x,y,w \sim D}[w]} D(x, y, w)$$

then optimal error rate classifiers for $\hat{D}$ are optimal importance-weighted minimizers for data drawn from D.

Minimizing the error rate on examples drawn according to $\hat{D}$ is slightly tricky [18], but feasible in a black-box manner for any classifier learning algorithm. With this tool in hand, we have half finished both of the reductions presented here.

3 RL with a Generative Model and a Classifier (RLGen)

We first state a simple reduction from reinforcement learning to classification which provides a guarantee of a near optimal policy when the learned classifiers have a small error rate with respect to a particular distribution. More precisely, we prove that if every classifier learned has error rate at most ϵ, the policy π learned by the algorithm has expected T-step reward at most $\frac{T+1}{2}\epsilon$ less than the expected T-step reward achieved by optimal policy for the MDP.

All of the results in this section rely on a generative model which has two operations:

1. start_state() returns a state drawn from $P(s_0)$.
2. next_state(s, a) returns a state s and a reward $R(s)$ drawn from $P(s'|a, s)$ for an action a and state s.

There are fundamental difficulties introduced in attempting to reduce T-step reinforcement learning to a single instance of classification, simply because the optimal T-step policy might be nonstationary. These difficulties are avoided when we reduce to T instances of classification, one for each timestep. These T learned classifiers are denoted c_t for $t \in \{1, ..., T\}$ and the *nonstationary* policy they create is $\pi(a|s, t) = c_t(s)$.

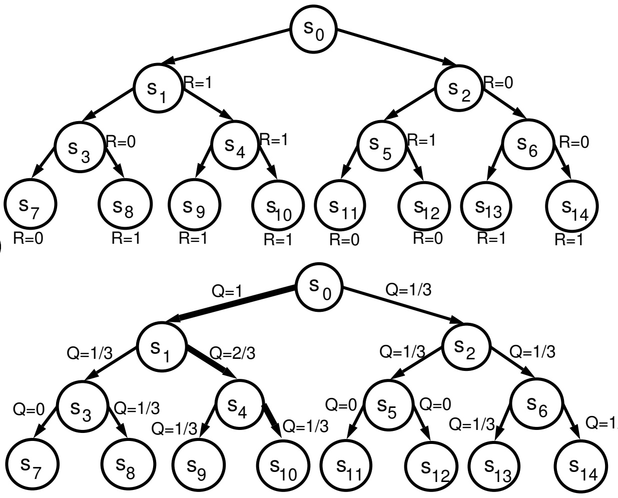
What are the distributions $D_1, ..., D_T$ on which these T classifiers must work well? The distribution on states at timestep i is dependent on the classifiers for previous timesteps. The rewards achieved based upon an action choice are dependent upon the classifiers for later timesteps. Thus, the correct distribution is inherently circular — each learning problem influences the other learning problems. Somewhat surprisingly, it is possible to break this circularity in a manner that allows provable guarantees on global optimality.

[4] We say "folk theorem" here because the result appears to be well-known by some. It is a simplification of many results from decision theory.

We start (as in [7]) with the notion of a trajectory tree. At depth T, this requires $|A|^T = 2^T$ calls to the generative model per tree. A tree is constructed by first calling start_state(), and then using next_state(s, a) to recursively construct a binary tree of depth T. Every tree produces one sample for each of the T classifiers. This one sample is found by computing the empirical value function $\hat{Q}_t(s, a)$ on that tree. The empirical value function is defined recursively as: $\hat{Q}_t(s, a) = \frac{R(s')}{T} + \max_a \hat{Q}_{t+1}(s', a)$ where s' is the state reached in the tree after taking action a in state s. The recursion terminates at $\hat{Q}_T(s, a) = 0$. An example of this calculation is shown in figure 1.

a)

b)



c) $(s_0, 0, 1), (s_1, 1, 2/3), (s_4, 1, 1/3)$

Fig. 1. a) A possible trajectory tree annotated with the values of $R(s)$ for each state, b) the empirical values $\hat{Q}(s, a)$ for each state and action, with the optimal path highlighted (when two actions are maximal, we choose one lexicographically) c) the examples extracted from the tree.

To extract the examples, consider a path through the tree which always takes a maximal value action. If s is the t^{th} state on the path and a is a maximal action, then the (x, y, w) triple is $(s, a, \hat{Q}(s, a))$ for the t^{th} classification problem. The importance $\hat{Q}(s, a)$ is the maximum amount that could be lost if we choose another action rather than a, i.e., misclassify the example. This implicitly assumes that the reward achieved after misclassification is 0, as it might be in the worst case. This pessimal assumption appears to be the only consistent choice, and violating it simply improves the theoretical guarantees we make later.

Note that the above sampling technique implicitly defines a distribution D over the (x, y, w) triples for an importance-weighted classification problem.

Assumption	Upper Bound	Lower Bound
Generative Model	$\left(\frac{\lvert A\rvert T}{\epsilon}\right)^{O(T)}$	$\Omega(\lvert A\rvert^T)$
Generative Model + $\frac{2\epsilon}{T+1}$-Classification	$O(\lvert A\rvert^T)$	$\Omega(\lvert A\rvert^T)$

Table 1. Lower and upper bounds on the sample complexity of T-step reinforcement learning.

This set of T classification problems is a reduction of reinforcement learning to importance weighted classification. Coupled with the reduction of importance-weighted classification to misclassification, we have a complete reduction from T-step reinforcement learning to error-minimizing classification. The algorithm for solving T-step reinforcement learning is then very simple: For every classification problem t, learn a classifier c_t and output the policy, $\pi(a|s,t) = c_t(a)$.

3.1 Analysis

We analyze the performance of this reduction, under the condition that the classifier learning algorithm succeeds in producing a small error rate classifier. This condition is clearly sometimes satisfiable as (for example) the parity of the index of the state could perfectly predict the action. It also hides a great deal of complexity, of course, but it allows us to focus on the efficiency of the reduction rather than the classification problem. It is worth noting that various boosting algorithms can reduce the ability to construct an ϵ error-rate classifier to constructing a set of $\frac{1}{2} - \epsilon$ error-rate classifiers.

Theorem 1. *(Learning Translation) If every classifier learned has error rate at most ϵ, the policy π satisfies:*

$$\eta(\pi^*) - \eta(\pi) \leq \frac{T+1}{2}\epsilon,$$

where π^* is an optimal policy for the MDP, i.e., a policy that maximizes $\eta(\pi)$.

This guarantee is as strong as the guarantee made by the sparse sampling algorithm [8] for reinforcement learning, which does not rely upon classification. In adding the assumption that we learn a classifier with small error reduces the sample complexity from $(|A|T/\epsilon)^{O(T)}$ to $O(|A|^T)$. A further reduction in the sample complexity is not possible because a lower bound[5] of $O(|A|^T)$ which appeared in [8] applies here as well. These results are summarized in table 1.

Proof. We analyze the construction outlined above. For learning problem t, let

$$N_t = E_{x,y,w \sim D_t}[w]$$

[5] The lower bound is essentially related to the observation that an arbitrary sequence of actions might lead to a reward, and so every sequence of actions must be explored.

be the expected value of the importance weights, in the t^{th} classification problem. The reduction from cost-sensitive to cost-insensitive classification implies that the optimal classifier has reward at least $N_t(1-\epsilon)$ if all other classifiers are perfect. Because rewards are between 0 and 1, we have that

$$N_t \leq \frac{T-t+1}{T}.$$

A policy is non-optimal if it takes some non-optimal action at some timestep. The expected loss due to errors in the first decision is bounded by $N_1\epsilon$. The expected loss due to errors in the second decision is $N_2\epsilon$, and so forth. The total expected loss in reward is at most:

$$\sum_{t=1}^{T} N_t \epsilon \leq \sum_{t=1}^{T} \frac{T-t+1}{T}\epsilon = \frac{T+1}{2}\epsilon$$

implying that the expected reward is at least $N_1 - \frac{T+1}{2}\epsilon$.

If π^* is the optimal policy and $\eta(\pi^*)$ is the reward of the optimal policy, then $N_1 \geq \eta(\pi^*)$ since the expected value of largest path is greater than or equal to the value of the optimal policy. ▮

The assumption that our classifiers have error rate ϵ hides issues of sample complexity. Roughly speaking, the necessary sample complexity is $2^T m$ where m is the number of samples required to realize error rate ϵ for all of the classifiers.

A careful examination of the proof reveals that we are using the generalization in the classification learning algorithm to combat the optimism inherent in considering the path with maximal value.

4 RL with Trace Model and a Classifier (RLTrace)

This second reduction works in the presence of a trace model, which has the following operation:

- trace(π): returns a trace from the execution of a (probabilistic) policy π on the MDP, that is, a sequence of state-action-reward triples: $(\langle s_0, a_0, r_1\rangle, \langle s_1, a_1, r_2\rangle, \ldots, \langle s_{T-1}, a_{T-1}, r_T\rangle)$, where s_0 is drawn from $P(s_0)$, each a_i is drawn from $\pi(a|s_i)$, each s_i (for $0 < t < T$) is drawn from $P(s'|s_{i-1}, a_{i-1})$ and $r_i = R(s_i)$.

This model is weaker than the generative model since we can simulate the trace operation using the initial_state and next_state operations but the opposite is not true. The model is motivated by (for example) problems where a large company interacts with many individual (hopefully independent) people according to some policy. The company can change the policy, but it can never "reset" people and see what alternative outcomes might have arisen from an interaction.

Again we are going to reduce the problem of finding a good policy to T instances of classification, one for each timestep. The T learned classifiers are denoted c_t for $t \in \{1, ..., T\}$ and the nonstationary policy they create is $\pi(a|s,t) = c_t(s)$.

The optimization starts with some arbitrary initial policy $c_1, ..., c_T$ and new classifiers c_t are learned iteratively. For each optimization problem, we learn a new classifier c'_t, then compare the two policies, $c_1, ..., c'_t, ...c_T$ and $c_1, ..., c_t, ..., c_T$ using Monte Carlo integration, as provided by the trace model. If the new policy has a provably larger expected value, we replace $c_t \leftarrow c'_t$, then move on to the next optimization problem. We iterate over $t = 1, 2, ...T, 1, 2, ..., T, 1, 2, ...$ with the process halting after we observe T nonupdates in a row.

The reduction for timestep t works by first calling the operation trace(π') m times, where the policy $\pi'(a|s,i)$ is given by $c_i(s)$ for $i < t$ and $i > t$. At timestep t, $\pi'(a|s,i)$ is a uniform distribution. Each returned trace is made into an example of the form (x, y, w), where $x = s_t$, $y = a_t$ and $w = \frac{1}{T}\frac{\sum_{i=t+1}^{T} r_i}{\pi'(a_t|s_t)}$.

Let

$$W(s_t, a_t) = \frac{1}{T}\sum_{i=t+1}^{T} R(s_i)$$

where $s_{t+1} \sim P(s'|s_t, a_t)$ and $s_{i+1} \sim P(s'|s_i, \pi'(s_i))$ for $t < i \leq T$.

We show next that $E[W(x, c_t(x))]$ is the expected value of the reward obtained by following $c_t(x)$ in step t while following π' in all the subsequent steps.

The following theorem shows that this expected value can be rewritten in a way that allows us to use a classifier learner that maximizes (2) to learn the t^{th} step classifier c_t that maximizes $E[W(x, c_t(x))]$ using the examples (x, y, w).

Theorem 2. *Let X be an arbitrary state space and $Y = \{0, 1\}$ be a discrete action space. Assume we have examples (x, y) drawn from a distribution D over $X \times Y$. Let $W : X \times Y \to [0, 1]$ be a stochastic function and let $c_t : X \to Y$ be a deterministic function. Assume that $\forall x, y : \; P(y|x) > 0$. Then:*

$$E_D[W(x, c_t(x))] = E_D\left[\frac{W(x,y)}{P(y|x)} I(c_t(x) = y)\right]$$

Proof.

$$\begin{aligned}
E_D\left[\tfrac{W(x,y)}{P(y|x)} I(c_t(x) = y)\right] &= E_D\left[\tfrac{W(x,y)}{P(y|x)} \middle| c_t(x) = y\right] P(c_t(x) = y)\\
&= E_D\left[\tfrac{W(x,c_t(x))}{P(c_t(x)=y|x)} \middle| c_t(x) = y\right] P(c_t(x) = y)\\
&= \int_X \tfrac{W(x,c_t(x))}{P(c_t(x)=y|x)} P(x|c_t(x) = y) P(c_t(x) = y) dx\\
&= \int_X \tfrac{W(x,c_t(x))}{P(c_t(x)=y|x)} P(x, c_t(x) = y) dx\\
&= \int_X W(x, c_t(x)) P(x) dx\\
&= E_D[W(x, c_t(x)]
\end{aligned}$$

In the first step of the proof, we use the fact $E[a\ I(b = c)] = E[a|b = c]P(b = c)$, which is easy to show. In the second step, we know that $c_t(x) = y$ because of the conditioning event, so we make substitutions accordingly. In the third step, we use the definition of conditional expectation. In the fourth step, we use Bayes' rule. The following steps are trivial. ▌

For each example (x, y, w), we have that $w = W(x, y)$. Thus, it follows from this theorem that

$$\frac{1}{m} \sum_{(x,y,w)} \frac{w}{P(y|x)} I(c_t(x) = y),$$

is an unbiased empirical estimate of $E[W(x, c_t(x))]$. Since we know $P(y|x)$, the probability that action y is executed in state x by the policy at step t, we can reweight each example (x, y, w) by $\frac{w}{P(y|x)}$ so that the learned classifier implicity minimizes equation 2 or, equivalently, maximizes

$$E_{x,y,w \sim D}[w \; I(c(x) = y)]$$

Here, we take into account the probability that the arbitrary policy executes an action in a state to estimate the expected value of a different policy. This is inspired by approaches for solving sample selection bias and missing data problems in econometrics and statistics, which take into account the probability that an example is missing [4, 11] to estimate unbiased parameters for a model. Similar importance sampling approaches are also present in the reinforcement learning literature in off-policy learning settings [16, 15], but they have not been used in reductions to classification.

Note that the theorem holds only if $\forall x, y : P(y|x) > 0$, that is, we require that the arbitrary policy chosen for timestep t have non-zero probability of executing each action in each state. However, the reduction degrades gracefully even when this is not the case if we define that

$$\frac{I(c_t(x) = y)}{P(y|x)} = 0$$

when $I(c_t(x) = y) = 0$ and $P(y|x) = 0$. In this case, the reduction makes the (approximately) optimal local improvement for an MDP that is identical to the original one, except that action y is not allowed in state x.

The policy obtained from this reduction is the (approximately) best possible local improvement over the arbitrary policy used for training. "Local" here refers to the fact that we optimize the classifier c_t for each step t in a greedy manner, assuming that the policy remains the same for the other steps. However, we continually change the policy to improve its overall value. The only guarantee is that each classifier c_t added increases the overall policy value.

Even though this is a local reduction it is particularly interesting because it is the first reduction from reinforcement learning to classification that can be used in an offline setting, where we only have traces from a known stochastic policy. This set of traces can be used to simulate the trace model by rejection sampling to produce a subset of traces consistent with the policy π given to the trace model.

A high-level sketch of the T-step reinforcement learning algorithm based on this reduction is given below:

Algorithm RLTrace(Trace model T, Classifier learner C, Initial policy π)

- While π improves
  - For t=1,2,...,T
    1. Generate m traces from π using T.
    2. Construct a sample S_t by generating an example of the form (x, y, w) for each trace, where $x = s_t$, $y = a_t$ and $w = \frac{1}{T} \frac{\sum_{i=t+1}^{T} r_i}{P(a_t|s_t)}$.
    3. Let $c_t = C(S_t)$.
    4. Improve π using c_t and the trace model T.

Output: nonstationary policy π.

We explain the policy improvement step in more detail in section 5.2.

5 Discussion

5.1 Generalizations and Improved Analysis

There are several ways this work can be generalized. In general, we chose the simplest possible mathematical form rather than the most general, so here we consider generalizations of the above analysis.

Application to POMDPs. The results are fully extensible to the POMDP setting where observations are made from a distribution $p(o|s)$ and the states are hidden. The technique used here is the well-known observation that the distribution over hidden states[6] in a POMDP acts according to MDP dynamics. Since a sequence of past observations o^* are a sufficient to describe the distribution over hidden states, we can regard them as labeling the distribution over states. Consequently, instead of predicting from the state s at timestep t, predictions are made based on observations, $o_1, ..., o_t$.

Note that since *any* sequence of observations can be explained by a POMDP, this extension is in some sense model free.

Weak learning assumption. Various boosting algorithms ([17, 12]) can use the weak learning assumption (ability to predict correctly with probability $\frac{1}{2} - \epsilon$) to create a strong learning algorithm.

Discount factor setting. Much of reinforcement learning is analyzed with a discount factor γ rather than a hard horizon T. The analysis here can be applied to a discount factor setting by simply noting that a discount factor of γ implies an "effective" horizon of $T = O(1/\gamma)$.

[6] This is sometimes called a "belief state".

In practical application on real-world reinforcement learning problems, individual classifiers probably do not differ much between timestep 100 and timestep 101. Taking advantage of this fact via multitask learning [2] or learning to learn [14] techniques may result in much reduced sample complexities.

The $|A|^T$ scaling of the global-optimal approach in section 3 is not viable beyond $T = 20$ or so. The quadratic (in T) sample complexity makes the local approach in section 4 more attractive and might optimistically scale to $T = 10^4$. Like many situations with a local tractable solution and an intractable global solution, a hybrid technique which varies between these approaches is likely useful. We discuss one such hybrid next.

5.2 The RLGen Algorithm

Here, we incorporate the reduction given in Section 3 into a more practical anytime [7] algorithm for solving T-step reinforcement learning in the presence of a generative model and a classifier learner. As the algorithm runs, it obeys succesively stronger properties:

1. (always) Monotonic improvement.
2. (with polynomial sample complexity) ϵ-error classification implies an ϵ-local optima(similar to section 4).
3. (with exponential complexity) ϵ-error classification implies $\frac{T+1}{2}\epsilon$-global optimality (as per section 3).
4. (with very exponential complexity) ϵ-error classification is always possible (as per [8]).

Each of the above guarantees holds in a representation-independent fashion. This implies that it is essentially *always* possible to succeed at reinforcement learning with the above algorithm with more or less work required, depending on the classification learning algorithm. We first present the high-level algorithm next and then discuss each one of its elements.

Algorithm RLGen(Generative model G, Classifier learner C, Initial policy π)

1. Pick complexity parameters for 1 or more classification problems.
2. For each classification problem i, construct a sample S_i. from the classification distribution using G, π and the complexity parameters.
3. For each classification problem i, let $c_i = C(S_i)$.
4. Improve π using the c_i.

Anytime Output: nonstationary policy π

The first step in the algorithm is picking a set of complexity parameters which affect exactly how long the next steps of the iteration take. There are three parameters: P, m and k. The parameter P is the set of classification problems. Each classification problem i is specified by a set of consecutive The other two

[7] By "anytime" we mean that the algorithm runs continuously and can be queried for the policy at any moment.

parameters $m > 0$ and $k > 0$ are positive integers. The parameter m is the number of examples given to each learning algorithm and k controls the noisiness of these examples (large k means low noise). The time complexity of these choices turn out to be $O(m|A|^{|P|k})$ as shown by the next step. We choose (P, m, k) by iterating over (P, m, k) of low complexity until step (4) results in no significant improvement amongst these choices. Then, the complexity is increased.

The second step constructs a dataset S_i for every learning problem i in P. This is done as in the global reduction in section 3, except that the tree has $|A|^{k|P|}$ leaves. In particular, for for each classification problem i, we pick a timestep t_i uniformly from the set of timesteps defining the learning problem. We draw an initial state, then for each timestep t, we branch on all actions k times if there is a learning problem i such that $t_i = t$ and otherwise we act according to the current policy. Once the tree is constructed, we calculate approximate value functions, pick a principle branch, and extract samples from the principle branch as in section 3 with one modification: we average over the k samples from each action.

The third step simply calls the classification algorithm on the constructed datasets S_i to learn new classifiers c_i.

The last step is a policy update step. There are two policies under consideration, the current policy π and a new policy π' formed from π except that at each timestep t in some classificiation problem $i \in P$ we substitute the classifier:

$$\pi'(s,t) = c_i(s) \text{ if } t \in i \in P \text{ or } \pi(s,t) \text{ otherwise}$$

To update the policy, we conduct a linesearch to find the optimal mixture policy:

$$\max_{\alpha \in [0,1]} \eta(\alpha\pi' + (1-\alpha)\pi)$$

Here the policy $\alpha\pi' + (1-\alpha)\pi$ is the policy which for each timestep t acts according to π' with probability α and π otherwise. Note that the conservative policy iteration analysis of [6] implies that such an α always exists whenever the classifiers have a sufficiently small error rate. In practice, $\alpha = 1$ may be common (and always holds when there is only one well-solved classification problem at one timestep.).

Now, we can show that the claimed properties hold:

1. Monotonic improvement: Since we use conservative policy iteration, the policy always improves.
2. ϵ-error classification implies an ϵ-local optima. In particular after choosing P = one learning problem over all timesteps a few times, we are guaranteed to achieve a policy π with the property that an ϵ loss classifier can not improve the policy more than ϵ. Again, this follows from the analysis in [6].
3. ϵ-error classification implies $\frac{T+1}{2}\epsilon$-global optimality: At some point the complexity is sufficient so that RLGen computes the global reduction in 3.
4. ϵ-error classification is always possible: At an even later point, the sparse sampling results [8] imply that classification is always possible with a classifier that has access to the generative model itself.

5.3 Implications

The structure of the reductions suggests new directions of research. What is a good multitask learning algorithm when the target concept drifts? Can lifelong learning techniques be applied?

A reduction implies that theory about classification also applies to reinforcement learning. This was partially covered in [7], but there are new implications.

1. The PAC-Bayes bound [13] implies that stochastic policies have a *smaller* sample complexity than might otherwise be expected. In fact, application of PAC-Bayes bounds to policies is somewhat more natural than for classification because the use of stochastic policies has (historically) been more common than the use of stochastic classifiers. Noting that stochastic policies have a lower-than-expected sample complexity for learning is a direct improvement on the analysis of [7].
2. Shell bounds [9] could be applied to policy evaluation and might result in a much smaller sample complexity when many of the possible classifier chains/policies are "obviously bad".

There are other implications of course — the magic of a reduction is that it applies to *future* work on classification.

Many learning problems have now been reduced to binary classification. These include ranking, regression, reinforcement learning, multiclass classification, and cost-sensitive classification. Many learning problems such as clustering and unsupervised learning have also *not* been reduced to classification. Can we classify all learning problems into equivalence classes based upon the families of reductions they fall into?

References

1. Drew Bagnell, Sham Kakade, Andrew Ng, and Geoff Schneider, "Policy Search by Dynamic Programming", NIPS 2003.
2. Rich Caruana, "Multitask Learning", Thesis, Carnegie Mellon University, 1997.
3. Alan Fern, SungWook Yoon, Robert Givan, "Approximate Policy Iteration with a Policy Language Bias", NIPS 2003.
4. James Heckman, "Sample selection bias as a specification error", Econometrica, 1977.
5. Sham Kakade, "On the Sample Complexity of Reinforcement Learning", Thesis, University College London, 2003.
6. Sham Kakade and John Langford, "Approximately Optimal Approximate Reinforcement Learning", ICML 2002.
7. Michael Kearns, Andrew Ng, Yishay Mansour, "Approximate Planning in Large POMDPs via Reusable Trajectories" Advances in Neural Information Processing Systems 12, MIT Press, 2000.
8. Michael Kearns, Andrew Ng, Yishay Mansour, "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes" Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence Morgan Kaufmann, 1999, pages 1324–1331.

9. John Langford and David McAllester, “Computable Shell Decomposition Bounds” COLT 2000.
10. Michail Lagoudakis and Ron Parr, ”Reinforcement Learning as Classification: Leveraging Modern Classifiers”, ICML 2003.
11. Roderick Little and Donald Rubin, “Statistical Analysis with Missing Data”, Chapter 3, Wiley, 2002.
12. Yishay Mansour and David McAllester, “Boosting using branching programs”, COLT2000.
13. David McAllester, “PAC-Bayesian Model Averaging”, COLT 1999.
14. Joseph O’Sullivan, “Transferring Learned Knowledge in a Lifelong Learning Mobile Robot Agent” In European Workshop on Learning Robots, 1998.
15. Andrew Ng, Ronald Parr and Daphne Koller. “Policy Search via Density Estimation”, NIPS 1999.
16. Doina Precup, Rich Sutton and Sanjoy Dasgupta. “Off-policy temporal-difference learning with function approximation”, ICML 2001.
17. Robert Schapire, “The Boosting Approach to Machine Learning”, MSRI Workshop on Nonlinear Estimation and Classification, 2002.
18. Bianca Zadrozny, John Langford, and Naoki Abe, “Cost-Sensitive Learning by Cost-Proportionate Example Weighting”, ICDM 2003.