Knows What It Knows: A Framework for Self-Aware Learning

Lihong Li LIHONG@CS.RUTGERS.EDU
Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Thomas J. Walsh THOMASWA@CS.RUTGERS.EDU
Rutgers University, Department of Computer Science, 110 Frelinghuysen Dr., Piscataway, NJ 08854

Recent reinforcement-learning (RL) algorithms with polynomial sample-complexity guarantees (e.g. (Kearns & Singh, 2002)) rely on distinguishing between instances that have been learned with sufficient accuracy and those whose outputs are still unknown. This partitioning allows algorithms to directly manage the exploration/exploitation tradeoff. However, prior frameworks for measuring *sample complexity* in supervised learning, such as Probably Approximately Correct (PAC) and Mistake Bound (MB), do not necessarily maintain such distinctions, so efficient algorithms for learning a model in these paradigms can be insufficient for efficient RL. In this work, we describe the Knows What It Knows (KWIK) framework (introduced by Li et al. (2008)), which intrinsically relies upon the known/unknown partitioning, embodying the sufficient conditions for sample-efficient exploration in RL. We show that several widely studied RL models are KWIK-learnable and derive polynomial sample-complexity upper bounds within this framework.

A KWIK algorithm begins with an *input set X output set Y*, and observation set Z. The *hypothesis class* H consists of a set of functions from X to Y: $H \subseteq (X \to Y)$. The *target function* $h^* \in H$ is unknown to the learner. H and parameters ϵ and δ are known to both the learner and environment. The environment selects a target function $h^* \in H$ adversarially. The agent then repeats the following:

1. The environment selects an input $x \in X$ adversarially and informs the learner.
2. The learner predicts an output $\hat{y} \in Y \cup \{\perp\}$ where $\perp$ means "I don't know".
3. If $\hat{y} \neq \perp$, it should be accurate: $|\hat{y} - y| \leq \epsilon$, where $y = h^*(x)$. Otherwise, the entire run is considered a failure. The probability of a failed run must be bounded by δ.
4. If $\hat{y} = \perp$, the learner makes an observation $z \in Z$ of the output, where $z = y$ in the deterministic case, $z = 1$ with probability y and 0 with probability $1 - y$ in the Bernoulli case, or $z = y + \eta$ for zero-mean random variable η with additive noise.

Over a run, the total number of steps on which $\hat{y} = \perp$ must be bounded by $B(\epsilon, \delta)$, ideally polynomial in $1/\epsilon$, $1/\delta$, and parameters defining H. We refer to $B(\epsilon, \delta)$ as the KWIK bound. Note that this bound should hold even if $h^* \notin H$, although, obviously, outputs need not be accurate in this case.

Like PAC, a KWIK algorithm is not allowed to make mistakes. Like MB, inputs to a KWIK algorithm can be selected adversarially. Instead of bounding mistakes, a KWIK algorithm must have a bound on the number of label requests ($\perp$) it can make. By requiring performance to be independent of the distribution, a KWIK algorithm can be used in cases in which the input distribution is dependent in complex ways on the KWIK algorithm's behavior, as can happen in on-line or active learning settings. We note that any KWIK algorithm can be turned into an MB algorithm (and therefore PAC) by simply guessing instead of signaling $\perp$, but cases exist where the reverse is not possible.

We now present some fundamental KWIK algorithms to give a flavor of how the framework can be applied in simple situations. The *memorization* algorithm can be used when the input space X is finite and outputs are observed noise free. To achieve this bound, it simply keeps a mapping $\hat{h}$ initialized to $\hat{h}(x) = \perp$ for all $x \in X$. When the environment chooses an input x, the algorithm reports $\hat{h}(x)$. If $\hat{h}(x) = \perp$, the environment will provide a label y and the algorithm will assign $\hat{h}(x) := y$. It will only report $\perp$ once for each input, so the KWIK bound is $|X|$.

The *enumeration* algorithm keeps track of $\hat{H}$, the version space. Each time the environment provides input $x \in X$, the algorithm computes $\hat{L} = \{h(x) | h \in \hat{H}\}$. If $|\hat{L}| = 1$, it means that all hypotheses left in $\hat{H}$ agree on the output for this input, so this element is returned. If $|\hat{L}| > 1$, two hypotheses in the version space disagree and the algorithm returns $\perp$ and receives the true label y, pruning the version space accordingly. It then computes an updated version space $\hat{H}' = \{h | h \in \hat{H} \wedge h(x) = y\}$ that contains at least one fewer hypothesis than the old version space. If $|\hat{H}| = 1$ at any point, $|\hat{L}| = 1$, and the algorithm will no longer return $\perp$. Therefore, $|H| - 1$ is the maximum number of $\perp$ the algorithm can return.

The KWIK framework is not restricted to deterministic problems. One fundamental KWIK algorithm for learning stochastic concepts is the *coin learning algorithm*, where we have a biased coin whose unknown probability of heads is p. We want to learn an estimate $\hat{p}$ that is accurate ($|\hat{p} - p| \leq \epsilon$) with high probability ($1 - \delta$). Each time the algorithm says $\perp$, it gets an independent trial that it can use to compute $\hat{p} = \frac{1}{T}\sum_{t=1}^{T} z_t$, where $z_t \in \{0, 1\}$ is the tth observation in T trials with 1 for HEAD and 0 for TAIL. The number of trials needed before we are $1 - \delta$ certain our estimate is ϵ accurate can be computed using a Hoeffding's bound: $T \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} = O\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right)$.

These and other simple solutions form the backbone of more intricate KWIK algorithms for learning complex models in dynamic environments. For instance, one can consider a higher level version of the memorization algorithm, the *input-partition* algorithm, which learns several disjoint KWIK-learnable classes in parallel. Such a meta-algorithm can be used to learn a Markov Decision Process (MDP) consisting of n states and m actions. An agent observes state–action–next-state transitions and must predict the probabilities for transitions it has not yet observed. In the model-based setting, the algorithm learns a mapping from the size $n^2 m$ input space of state–action–next-state combinations to probabilities via Bernoulli observations. Thus, the problem can be solved via the input-partition algorithm over a set of individual probabilities learned via coin learning. The resulting KWIK bound is $O\left(\frac{n^2 m}{\epsilon^2} \ln \frac{nm}{\delta}\right)$. Online exploration in this and other MDP forms can be driven using an "optimism in the face of uncertainty" heuristic, resulting in a PAC-MDP agent (see the description of KWIK-Rmax by Li (2009) for details). Given this result, it is important to determine what types of RL models are KWIK learnable. We list some of the more prominent ones here.

1. **Linear Dynamics** – Linear functions are a common model for the dynamics of continuous environments. Strehl & Littman (2008) showed a *noisy* d-dimensional linear function can be KWIK-learned with a KWIK bound of $\tilde{O}(d^3)$, and polynomial dependencies on $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.
2. **Action Outcomes with Gaussian Distributions** – Learning the parameters of a Gaussian can be done using a variant of the coin-learning algorithm, and multiple actions can be dealt with using input-partition, just like in the flat MDP case (Brunskill et al., 2008).
3. **Dynamic Bayes Nets with Known Structures** – For any given factor in the DBN, given its parent values, the probability table for its next value can be computed using the coin-learning algorithm as in the flat MDP case. Multiple factors can be learned in parallel and predictions made using the "cross product" of these individual predictors (Li et al., 2008).
4. **Dynamic Bayes Nets with Unknown Structures** – In the case where the DBN structure is not known, multiple hypotheses about the structure can be considered at any given time, using a high level version of the enumeration algorithm described earlier. Learning the DBN parameters for each of these hypotheses proceeds as in the known structure case (for details see Diuk et al. (2009)). This bound significantly improves on a previously derived bound for this problem.

In addition to unifying the analysis of model-based RL, the KWIK framework can also be used to construct sample-efficient model-free algorithms by considering a hypothesis space of value functions or Bellman errors (see (Li, 2009) for details). Thus, the KWIK framework outlined in this paper serves as a powerful unifying tool for the analysis and development of RL algorithms across a rich set of function classes.

References

Brunskill, E., Leffler, B. R., Li, L., Littman, M. L., & Roy, N. (2008). CORL: A continuous-state offset-dynamics reinforcement learner. *UAI* (pp. 53–61).

Diuk, C., Li, L., & Leffler, B. R. (2009). The adaptive k-meteorologists problem and its application to structure discovery and feature selection in reinforcement learning. *ICML*.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, *49*, 209–232.

Li, L. (2009). *A unifying framework for computational reinforcement learning theory*. Doctoral dissertation, Rutgers University, New Brunswick, NJ.

Li, L., Littman, M. L., & Walsh, T. J. (2008). Knows what it knows: A framework for self-aware learning. *ICML*.

Strehl, A. L., & Littman, M. L. (2008). Online linear regression and its application o model-based reinforcement learning. *NIPS*.