RLEX: A DBMS For Reinforcement Learning

Sanjay Krishnan
UC Berkeley
sanjaykrishnan@berkeley.edu

1. ABSTRACT

The marriage of data management and machine learning is one of the most significant trends in recent database research. Industry and academia have built systems to improve the efficiency of model training [3,5,7,20], studied problems in learning on normalized data [16,19], designed libraries for in-database learning [2,8, 10], explored how to apply data cleaning before learning [15]. As database researchers, it is natural to think machine learning as simply a form of high-dimensional aggregate query processing. However, unlike traditional SQL analytics, learning systems potentially create feedback loops where the learned models affect the future training data. For example, if a model recommends Pop music to European users–future data would have fewer examples of European preferences on other genres.

To address such problems, we argue that learning systems need to move beyond the supervised learning paradigm and add support for Reinforcement Learning (RL) – drawing inspiration from recent successes of Google DeepMind on AlphaGo project [1]. Rather than predicting a single label for a single feature vector, RL algorithms learn stateful *policies* that output a sequence of predictions (called actions). As the name implies, these policies are learned through iterative trial-and-error (called exploration); optimizing an aggregate reward (utility of the predictions). In the music recommendation example, one could imagine a system that learns to recommend songs based on those songs previously heard by a user to maximize the total aggregate time spent on the platform.

Such is the promise of RL, and in principle, RL subsumes all supervised learning models. However, this generality comes at a steep price. RL algorithms can spend significant amounts of time exploring before learning a policy of value. It is further the case that many state-of-the-art RL algorithms have to store the entire execution trace of the system for learning [4]. While these algorithms do exploit parallelism [17] and GPUs [18], much of these results are restricted to simulated problems, where exploration is relatively inexpensive, and the data is by definition clean. Transferring these results to real systems requires addressing challenges in geo-distribution, data quality, physical design in these execution traces. We propose RLEX a DBMS system for managing RL execution traces which builds on our work on the SampleClean project [14] and reliable RL in surgical robotics [11].

Anatomy of a Modern RL Algorithm: Especially when Neural Networks are involved, the current state-of-the-art in RL is to use a technique called *experience replay*, where the entire execution trace of an RL algorithm is stored including all historical data encountered and actions taken during exploration [4]. An RL algorithm is first initialized with a random policy. Using this random policy it takes actions and appends the results of these actions to the execution trace. To update the policy, the algorithm samples a batch of tuples (actions, results) from this execution trace. It compares these tuples to its current policy and updates the policy in a way to favor actions that result in a higher reward. Thus, the algorithm sequentially learns more and more about the environment over time.

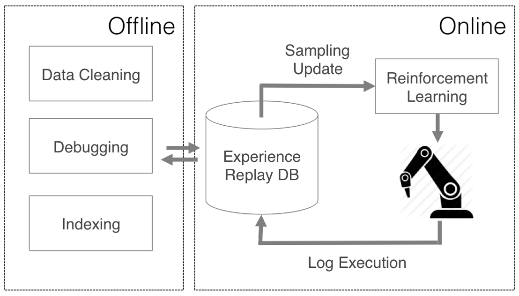

Figure 1: Reinforcement Learning algorithm learn control policies through iterative trial-and-error. State-of-the-art algorithms accumulate experiences in an "experience replay" database, and use samples from this database to update their policies. RLEX is a DBMS system to manage such a database inspired from our experiences in surgical robotics.

RLEX Architecture: We describe some of the major architectural components of RLEX.

1. *The Sampler:* There are several reasons why the execution trace may have to be stored in a normalized or otherwise partitioned schema. Sampling normalized schemas is subtle since coin-flip sampling does not always commute with join operations. The RLEX Sampler users a hash-based algorithm as in [13] or [9] to approximate Bernoulli sampling across equality joins. To apply such a sampling algorithm, we will have to efficiently maintain indices on the execution trace.
2. *The Cleaner:* Many RL applications will interact with the physical world such as self-driving cars, HVAC systems, and personal robots. As the community learned 15 years ago [6], sensor systems are notorious for dirty, missing, and duplicated data. Execution traces can potentially accumulate large dirty datasets, and even more frightening, dirty data from one device can potentially cause problems for other devices. The RLEX Cleaner uses an ensemble of constraint-based error detection and quantitative outlier detection to remove potentially erroneous data.
3. *The Debugger:* To debug and diagnose failures of RL applications, we will have to use time-series transition analysis techniques [12]. These techniques identify common failure modes and states that trigger changes in behavior.

2. REFERENCES

[1] Google deepmind. https://deepmind.com.

[2] Keystone ml. http://keystone-ml.org/.

[3] Tensor flow. https://www.tensorflow.org/.

[4] S. Adam, L. Busoniu, and R. Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2012.

[5] A. Crotty, A. Galakatos, and T. Kraska. Tupleware: Distributed machine learning on small clusters. In *IEEE Data Eng. Bull.*, 2014.

[6] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[7] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-rdbms analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 325–336. ACM, 2012.

[8] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The madlib analytics library or MAD skills, the SQL. In *VLDB*, 2012.

[9] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. 2016.

[10] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[11] S. Krishnan, A. Garg, R. Liaw, L. Miller, F. T. Pokorny, and K. Goldberg. HIRL: hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards. *CoRR*, abs/1604.06508, 2016.

[12] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. In *International Symposium of Robotics Research. Springer STAR*, 2015.

[13] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska. Stale view cleaning: Getting fresh answers from stale materialized views. In *VLDB*, 2015.

[14] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, T. Kraska, T. Milo, and E. Wu. Sampleclean: Fast and reliable analytics on dirty data. *IEEE Data Eng. Bull.*, 38(3):59–75, 2015.

[15] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[16] A. Kumar, J. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1969–1984. ACM, 2015.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[19] D. Olteanu and M. Schleich. F: Regression models over factorized views. *Proceedings of the VLDB Endowment*, 9(13), 2016.

[20] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.