Speeding up Reinforcement Learning-based Information Extraction Training using Asynchronous Methods

Aditya Sharma
Indian Institute of Science
Bangalore, India
adisharma075@gmail.com

Zarana Parekh[*]
DA-IICT
Gandhinagar, India
zaranaparekh17@gmail.com

Partha Talukdar
Indian Institute of Science
Bangalore, India
ppt@iisc.ac.in

Abstract

RLIE-DQN is a recently proposed Reinforcement Learning-based Information Extraction (IE) technique which is able to incorporate external evidence during the extraction process. RLIE-DQN trains a single agent *sequentially*, training on one instance at a time. This results in significant training slowdown which is undesirable. We leverage recent advances in parallel RL training using asynchronous methods and propose RLIE-A3C. RLIE-A3C trains multiple agents in *parallel* and is able to achieve upto 6x training speedup over RLIE-DQN, while suffering no loss in average accuracy.

1 Introduction

Extracting information about an event (or entity) involves multiple decisions, as one first needs to identify documents relevant to the event, extract relevant information from those documents, and finally reconcile various values obtained for the same relation of the event from different sources (Ahn, 2006). Search based methods for Information Extraction have been increasingly investigated (West et al., 2014); (Hegde and Talukdar, 2015); (Zhang et al., 2016); (Bing et al., 2017).

(Kanani and McCallum, 2012) combine search and Information Extraction (IE) using Reinforcement Learning (RL), with the goal of selecting good actions while staying within resource constraints, but don't optimze for extraction accuracy. More recently, (Narasimhan et al., 2016) proposed a RL-based approach to model the IE process outlined above. We shall refer to this approach as RLIE-DQN in this paper. RLIE-DQN trains an RL agent using Deep Q-Network (DQN) (Mnih et al., 2015) to select optimal actions to query for documents and also reconcile extracted values.

DQN trains a single agent *sequentially*, updating parameters based on one instance at a time. Each such instance is sampled from the entire training data, also called the *experience replay*. Such sequential experience replay-based training results in slow learning, while requiring high memory and computation resources. In order to overcome this challenge, A3C (Asynchronous Advantage Actor-Critic), an asynchronous deep RL training algorithm, has been proposed recently (Mnih et al., 2016). A3C trains multiple RL agents in *parallel*, each of which estimates gradients locally, and asynchronously updates globally shared parameters. Recent work has explored applications for A3C in varied domains (Fernando et al., 2017), (Mirowski et al., 2016).

In this paper, we propose RLIE-A3C which uses A3C-based parallel asynchronous agents for training. This is in contrast to the sequential DQN training in RLIE-DQN. Differences between the training regimes of the two methods are shown in Figure 1. Through experiments on multiple real-world datasets, we find that RLIE-A3C achieves upto 6x training speedup compared to RLIE-DQN, while suffering no loss in accuracy. To the best of our knowledge, this is the first application of asynchronous deep RL methods in IE (and also in NLP), and we hope this paper will foster further adoption and research into such methods in the NLP community. RLIE-A3C code is available at https://github.com/adi-sharma/RLIE_A3C

[*] Research carried out during an internship at the Indian Institute of Science, Bangalore.

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2641–2646
Copenhagen, Denmark, September 7–11, 2017.

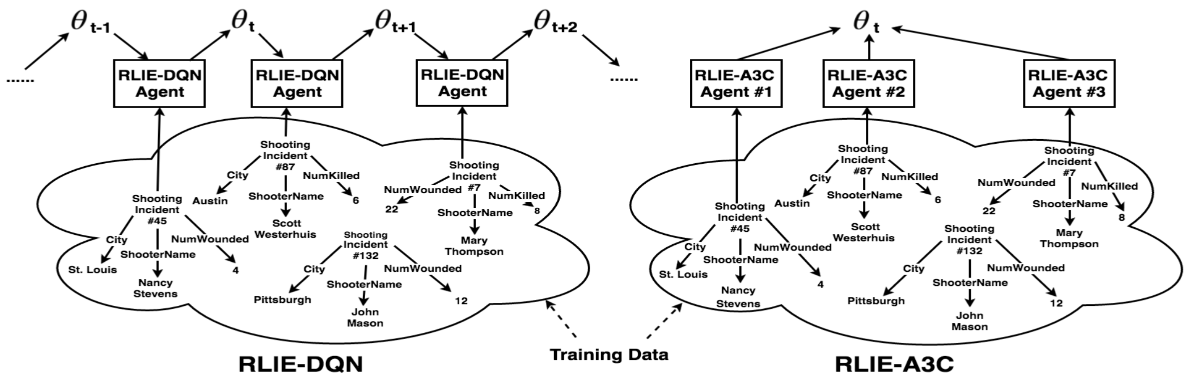

Figure 1: **Left:** DQN-based *sequential* learning framework used in RLIE-DQN (Narasimhan et al., 2016), as discussed in Section 2. At each time step, the agent looks at a specific instance from the training data. **Right:** A3C-based *parallel* learning framework in RLIE-A3C (proposed approach). The parallel agents look at different parts of the training data, estimate parameter update statistics locally, and then perform *asynchronous* updates on the globally shared parameters (θ_t) at time step t. See Section 3 for details. Due to the asynchronous parallel updates, RLIE-A3C achieves significant training speedup without loss in accuracy, as we shall see in Section 4.

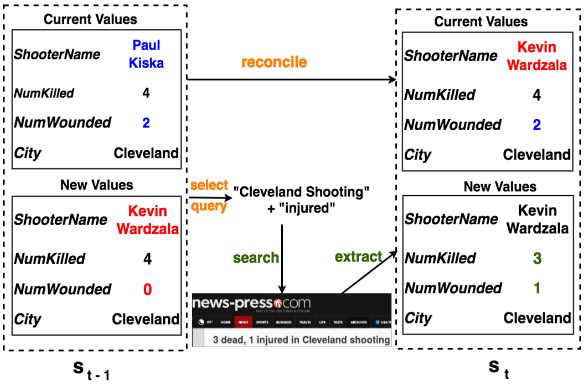

Figure 2: Sample state transition in the MDP of RLIE-DQN (and also RLIE-A3C). In each transition, two actions are carried out: (1) reconcile new values with current values,; and (2) issue query to retrieve other relevant documents and extract values from those documents. Please see Section 2 for details.

2 RLIE-DQN: Information Extraction using Reinforcement Learning

We first present a brief overview of RLIE-DQN (Narasimhan et al., 2016). Given a document to extract information about an event, RLIE-DQN issues a search query to retrieve other documents related to the event, extracts event information from those documents, and finally reconciles values extracted from the documents. If confidence in the extracted values are low, then RLIE-DQN repeats this process with additional queries. This way, RLIE-DQN incorporates evidences from external sources to improve information extraction (IE) from a given source document.

RLIE-DQN models the task as a Markov Decision Process (MDP) in order to reconcile newly extracted information selectively and dynamically. The MDP describes the environment in which the RL agent learns to make decisions. The MDP is represented using a tuple $\langle S, A, R, T \rangle$, where S is the set of states, $A = \{a_d, a_q\}$ is the set of actions, $R(s, a)$ is the reward for taking action $a \in A$ from state $s \in S$, and $T(s^{'}|s, (a_d, a_q))$ is the state transition function. Here, a_d is the reconciliation action, while a_q is the query action. Based on a_d, the agent may accept extracted values for one or all relations, reject all newly extracted values or stop (episode ends).

A sample state transition in RLIE-DQN's MDP is shown in Figure 2. State representation consists of many details such as confidence scores of current and newly extracted relation values, context statistics from which the extractions are performed, etc. But for better readability, only the set of current and new values are shown for the states in this figure. Each transition consists of two actions: reconcile decision and query. The RL agent uses the reconcile action (a_d) to update value of the *ShooterName* relation from *Paul Kiska* to *Kevin Wardzala*. The agent uses the query action (a_q) to issue a new query (*"Cleveland shooting"* + *"injured"*) to retrieve other relevant documents and extract new values 3 and 1 for relations *NumKilled* and *NumWounded*, respectively. The transitions stop whenever a_d is a stop decision.

The reward function at each state, is the cumulative difference of current and previous extracted accuracies, summed over all relations. Also, a negative reward per step is added to the reward in order to penalize the agent for longer episodes, since issuing queries to a search engine is expensive.

Deep Q-Learning (DQN): Let $Q(s, a)$ be the measure of long-term cumulative reward obtained by taking action a from state s. The RL agent makes use of $Q(s, a)$ to select the next action from a from state s. Q-learning (Watkins and Dayan, 1992) is a popular technique to estimate this function, in which the function for optimal Q-value is estimated using the Bellman equation (Sutton and Barto, 1998) $Q_{i+1}(s, a) = \mathbb{E}_{(s,a)}[R(s, a) + \gamma \max_{a'} Q_i(s', a')|s, a]$. Here, $R(s, a)$ is the immediate reward and γ is the discount factor for the value of future rewards.

For high dimensional state spaces, Deep Q-Network (DQN) (Mnih et al., 2013) approximates $Q(s, a)$ as $Q(s, a; \theta)$ using parameters θ of a deep network. RLIE-DQN used such a DQN-based agent to learn optimal policy, as shown in Figure 1.

3 Proposed Approach: RLIE-A3C

The DQN-based agent used in RLIE-DQN is sequential, as it moves from one instance to another to update parameters. This can result in significant training time slowdown, especially in large data settings. Instead of using experience replay of the DQN algorithm for stabilizing updates, we consider a framework with multiple asynchronous agents, each of which explore different areas of the environment in parallel.

Asynchronous Advantage Actor-critic (A3C) (Mnih et al., 2016) is a recently proposed deep RL algorithm which makes use of parallel agents for parameter estimation. We replace DQN with A3C in RLIE-DQN and call the resulting method RLIE-A3C – our proposed approach (See Figure 1).

At time instant t, RLIE-A3C poses decision policy $\pi_d(a_d|s_t)$, and query policy $\pi_q(a_q|s_t)$ as probability distributions over candidate actions a_d and a_q, respectively. RLIE-A3C also calculates the state value function $V(s_t)$, as an estimate of the cumulative long-term reward obtained starting from state s_t. Owing to the large continuous state space of the problem, RLIE-A3C approximates each of these three functions using three separate deep neural networks, which are parametrized by θ_d, θ_q, and θ_v, as $\pi_d(a_d|s_t) \approx \pi_d(a_d|s_t; \theta_d)$, $\pi_q(a_q|s_t) \approx \pi_q(a_q|s_t; \theta_d)$ and $V(s_t) \approx V(s_t; \theta_v)$. These parameters are updated by agents working in parallel. Based on the policies, each agent selects the query and decision actions to be performed and updates it's state accordingly. This is repeated up to t_{max} steps or until a terminal state is reached.

Local Gradient Calculation: The agents estimate parameter gradients using a local copy of the network parameters, and then perform *asynchronous* updates on the globally shared parameters θ_d, θ_q, and θ_v. Hence, the policy and value functions are jointly estimated. The policy gradient update equations calculated over the local copy of network parameters of each parallel RLIE-A3C agent p are as follows:

$$\begin{aligned} d\theta_x^p \leftarrow d\theta_x^p &+ \nabla_{\theta_x^p} \log \pi_d(a_{x_i}, s_{;}\theta_x^p) A(s_i, a_{x_t}; \theta_v^p) \\ &+ \beta_x \nabla_{\theta_x^p} H(\pi_x(s_t; \theta_x^p)) \end{aligned} \quad (1)$$

where, x $\in$ {d, q} for decision and query. The advantage function $A(s_t, a_t; \theta_v) = \sum_{i=0}^{k-1} \gamma^i R_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$ above significantly reduces the variance of the policy gradient, where γ is the discount factor and k can vary from state to state and is upper-bounded by t_{max}. Since the gradient updates are accumulated, training stability increases. Further, exploration is encouraged by introducing the entropy regularization term $\beta_x \nabla_{\theta_x^p} H(\pi_x(s_t; \theta_x^p))$ in the equation above. Here, H is the entropy function and β_x controls dominance of the entropy term.

The gradient update for the parameters of value function V is calculated locally by every parallel agent p as follows:

$$d\theta_v^p \leftarrow d\theta_v^p + \partial(G_t - V(s_t; \theta_v^p))^2 / \partial\theta_v^p$$

where $G_t = \mathbb{E}_{(s)}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}]$ is the return.

Global Parameter Update: The parameters θ_d, θ_q and θ_v are learnt using stochastic gradient descent with RMSprop (Tieleman and Hinton, 2012). The standard non-centered RMSProp update is used by the parallel agents p to update the shared parameters asynchronously using the gradients obtained from Equation (1), as follows:

$$g = \alpha g + (1 - \alpha)(d\theta_x^p)^2 \;\; ; \;\; \theta_x \leftarrow \theta_x - \eta \frac{d\theta_x^p}{\sqrt{g + \epsilon}}$$

where x $\in$ {d, q, v}, α is the decay factor, η is the learning rate, ϵ is the smoothing constant and

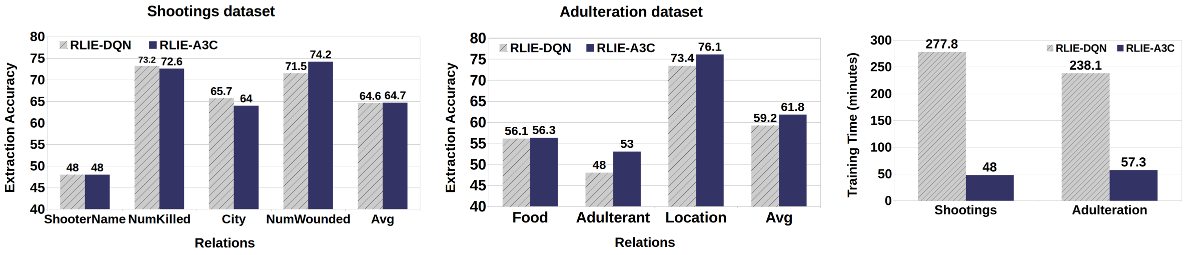

Figure 3: **Left, Middle Panels**: Extraction accuracy of the baseline (RLIE-DQN) and our system (RLIE-A3C) on the Shooting Incidents and Food Adulteration datasets. **Right Panel**: Training time comparison between RLIE-DQN and RLIE-A3C. Overall, we observe that RLIE-A3C results in upto 6x speedup over RLIE-DQN, without any loss in average extraction accuracy. This is our main result. Please see Section 4.1 for details.

g is the moving average of element-wise squared gradients. The pseudo-code for RLIE-A3C can be found in the Appendix.

4 Experiments and Results

Setup: We compare RLIE-A3C against RLIE-DQN using the same protocol and hyperparameters as reported in (Narasimhan et al., 2016). Also, we experiment with the same two datasets used in that paper: the Gun Violence Archive[1] and the Foodshield EMA database[2]. The train, dev and test datasets contain 372, 146 and 146 source articles respectively for the Shooting incident cases and 292, 42 and 148 source articles respectively for the food adulteration cases. We used the implementation of RLIE-DQN provided by the authors of that system. For more details on the dataset and other parameters, we refer the reader to (Narasimhan et al., 2016).

RLIE-A3C: This is our proposed method which is described in Section 3. For the sake of fair comparison, the network, base classifier, and evaluation metrics are same as that of RLIE-DQN. Each of the three deep networks in RLIE-A3C, one each for $\pi(a_d|s)$, $\pi(a_q|s)$ and $V(s)$, is built using two linear layers with 20 hidden units, followed by ReLUs. MaxEnt classifier is used as the base extractor, and the model is evaluated on the entire *test* set for 1.6 million steps. The *dev* set is used to tune all hyperparameters, which can be found in the Appendix. For RLIE-A3C, the evaluation is carried out 50 times after training and the average accuracy values are taken over the top 5 evaluations, as done in (Mnih et al., 2016).

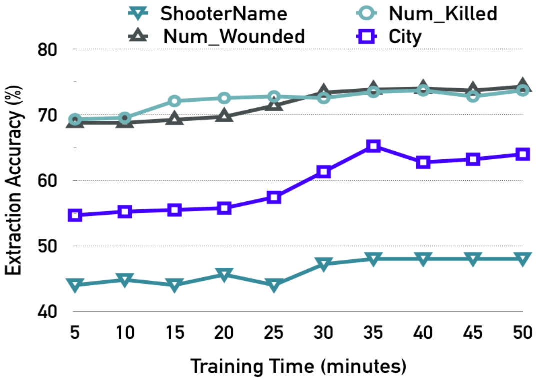

Figure 4: Evolution of accuracy of RLIE-A3C for the four relations on *test* set of the *Shooting Incidents* dataset. Please see Section 4.1 for details.

4.1 Results and Discussion

Extraction Accuracy: Experimental results comparing extraction accuracies of RLIE-DQN and RLIE-A3C are presented in Figure 3[3] (left and middle panels). From this figure, we observe that there is no loss in average accuracy in transitioning from RLIE-DQN to RLIE-A3C (in fact there is a slight gain in case of the Adulteration dataset). Please note that, improvement in accuracy is not our primary goal in the paper – it is the training time speedup, as discussed next.

Speedup: Training times of RLIE-DQN and RLIE-A3C over both datasets are compared in the right panel of Figure 3[4]. From Figure 3, we find that RLIE-A3C is able to achieve upto 6x training

[1] http://www.shootingtracker.com/Main_Page

[2] http://www.foodshield.org/member/login/

[3] The tolerance for all the relation accuracy values for RLIE-A3C is within ± 1%

[4] During test, policies learned by RLIE-DQN and RLIE-A3C are executed. Since this is an identical process for both, we don't compare test runtimes in this paper.

speedup over RLIE-DQN. In other words, RLIE-A3C is able to achieve significant speedup in training time over RLIE-DQN, without any loss in average accuracy. This is our main result.

While RLIE-DQN was implemented in Torch, RLIE-A3C was implemented in Python using TensorFlow framework. We note that Torch is known to be faster than TensorFlow (Bahrampour et al., 2015). This makes the speedup gains above even more impressive, and outlines the possibility that further gains may be possible with a Torch-based implementation of RLIE-A3C.

Figure 4 shows evolution of test accuracy of RLIE-A3C for the four relations of the *Shooting Incidents* dataset. For this dataset, state value function of RLIE-A3C converged at 48 minutes. However, from Figure 4, we observe that accuracies converge to the final values much before that.

Why do Asynchronous Methods work? An asynchronous approach fits more naturally with the training data, since different parallel agents look at different events at the same time (see Figure 1), and the model is able to exploit the regularities between events in the dataset. The gradient updates to the global network are less biased and the model is not easily distracted by noise in the data. The asynchronous parallel model is able to converge much faster as compared to replay memory based methods like DQN, as also seen in (Mnih et al., 2016).

5 Conclusion

In this paper, we proposed RLIE-A3C, an asynchronous deep Reinforcement Learning (RL) algorithm for Information Extraction (IE). In contrast to *sequential* training in previously proposed RLIE-DQN (Narasimhan et al., 2016), RLIE-A3C employs *asynchronous parallel* training. This results in upto 6x training speedup, without suffering any loss in average accuracy. We hope that this first application of asynchronous deep RL algorithms will open up more adoption of such techniques in the NLP community.

6 Acknowledgements

We thank members of the MALL Lab, IISc who read drafts of the paper and gave valuable feedback. We also thank the anonymous reviewers for their insightful comments. Special thanks to Karthik Narasimhan for all the helpful discussions. We also gratefully acknowledge support from a gift from Microsoft Research India.

References

David Ahn. 2006. The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*, pages 1–8. Association for Computational Linguistics.

Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2015. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*.

Lidong Bing, Zhiming Zhang, Wai Lam, and William W Cohen. 2017. Towards a language-independent solution: Knowledge base completion by searching the web and deriving language pattern. *Knowledge-Based Systems*, 115:80–86.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. 2017. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

Manjunath Hegde and Partha P Talukdar. 2015. An entity-centric approach for overcoming knowledge graph sparsity. In *EMNLP*, pages 530–535.

Pallika H Kanani and Andrew K McCallum. 2012. Selecting actions for resource-bounded information extraction using reinforcement learning. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 253–262. ACM.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. 2016. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. 2016. Improving information extraction by acquiring external evidence with reinforcement learning. *arXiv preprint arXiv:1603.07954*.

Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2).

Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning*, 8(3-4):279–292.

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. 2014. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526. ACM.

Zhenzhong Zhang, Le Sun, and Xianpei Han. 2016. A joint model for entity set expansion and attribute extraction from web search queries. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3101–3107. AAAI Press.