Deep Reinforcement Learning with Successive Over-Relaxation and its Application in Autoscaling Cloud Resources

Indu John
Department of Computer Science and Automation
Indian Institute of Science
Bangalore, India
indu@iisc.ac.in

Shalabh Bhatnagar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore, India
shalabh@iisc.ac.in

***Abstract*—We present a new deep reinforcement learning algorithm using the technique of successive over-relaxation (SOR) in Deep Q-networks (DQNs). The new algorithm, named SOR-DQN, uses modified targets in the DQN framework with the aim of accelerating training. This work is motivated by the problem of auto-scaling resources for cloud applications, for which existing algorithms suffer from issues such as slow convergence, poor performance during the training phase and non-scalability. For the above problem, SOR-DQN achieves significant improvements over DQN on both synthetic and real datasets. We also study the generalization ability of the algorithm to multiple tasks by using it to train agents playing Atari video games.**

***Index Terms*—reinforcement learning, deep learning, cloud computing, resource allocation, atari games**

I. Introduction

Reinforcement learning (RL) framework involves an agent that learns to behave adaptively by interacting with its environment. By combining deep neural networks with reinforcement learning, deep reinforcement learning algorithms have been shown to be successful in a wide variety of control tasks. In particular, Deep Q-network proposed in [1] achieves human-level performance in complex domains like Atari video games, using only low level information such as screen images and scores as inputs.

Typically, deep RL algorithms require a large number of iterations (sometimes of the order of millions) in the training phase before they attain good performance in the evaluation phase. For example, DQN was trained on a total of 50 million frames for Atari games [2]. While this is not a concern in simulated environments, it becomes prohibitive in real world applications where data is expensive. Our objective in this paper is to develop an algorithm that learns faster, thus reducing the training costs.

Our algorithm uses the technique of successive over-relaxation in deep Q-networks. The key idea is as follows. The Q-function that a DQN approximates is known to be the fixed point of the Bellman operator. A technique known as successive relaxation can be applied to generalize the Bellman operator with an additional parameter such that the contraction factor of the generalized Bellman operator is less than that of the Bellman operator [3]. When the relaxation parameter is greater than 1, the method is known as successive over-relaxation (SOR). We apply the SOR technique to modify the targets in the DQN algorithm in order to make the learning faster. The new algorithm is named SOR-DQN. It has the same computational complexity as DQN.

For evaluating our algorithm, we consider the problem of autoscaling resources for cloud applications. Cloud computing provides individuals and organisations with a shared pool of configurable computing resources. One of the key attractions of the cloud is its elastic nature i.e., its ability to increase or decrease the amount of resources allocated to an application depending on the changing requirements. To efficiently utilize elasticity of clouds, the decisions on resource allocation need to be made algorithmically, adaptively and in real-time [4]. Moreover, they must respect the performance requirements of the application, as specified in the Service Level Agreement (SLA) between the cloud provider and the client.

While several RL based solutions exist already [5], [6] for this problem, they suffer from issues such as slow convergence, non scalability and poor performance during the learning period. We propose the use of SOR-DQN as a scalable and efficient RL algorithm for adaptive resource provisioning to cloud applications. The deep neural network addresses the scalability issue, while the SOR technique accelerates the training phase. We test the algorithm on simulated and real workloads to demonstrate its superiority over DQN. Further, we study the generalization ability of our algorithm using a set of 48 Atari 2600 games involving diverse tasks that was used in the evaluation of DQN.

The rest of the paper is organized as follows. Section II discusses the mathematical background and related work in this area. We present our algorithm in Section III followed by a description of the problem of auto-scaling cloud applications in Section IV. The experimental results in cloud computing

This work was supported in part by the Robert Bosch Centre for Cyber Physical Systems, Indian Institute of Science and a DST project under the ICPS scheme. The second author also acknowledges support from the J. C. Bose National Fellowship.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

and Atari games are given in Section V. Finally, Section VI concludes the paper and points out directions for further research.

II. Background and Related work

The interaction between the agent and the environment in a reinforcement learning problem is modelled mathematically using a Markov Decision Process (MDP) which has the following components : The set of states S, the set of actions A, the transition probability function $P : S \times A \times S \to [0, 1]$, the reward function $r : S \times A \to \mathbb{R}$ and the discount factor γ. The agent tries to learn the best action sequence or *policy* that maximizes the expected *sum of discounted rewards* (return) over a period of time. Formally, a policy π is a mapping from states to actions. The goal is to find an optimal policy i.e., one that maximizes over all policies π the expected return or value function given by

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s\right],$$

where s_0 is the initial state and R_t is the possibly random reward obtained at time t with expected value $r(s, a)$ if the state at time t is s and the action chosen is a.

In most of the real world applications the transition probabilities and reward functions are not accurately known a priori. In such cases a model free approach like Q-learning [7] can be used. Here the agent interacts with the environment iteratively to learn Q values for each (state, action) pair. The optimal Q values correspond to the maximum expected return for using action a in a certain state s.

Deep RL algorithms approximate the Q-function using deep neural networks. The optimal Q-function Q^* satisfies the Bellman equation [8] given by

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{b \in A} Q^*(s', b) \qquad (1)$$

This expresses the fact that the optimal expected return by taking action a in the current state s is the sum of the reward obtained for taking action a in state s and the optimal expected return from the next state s'. The optimal policy may be computed from the optimal Q-function as

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a) \qquad \forall s \in S.$$

A. DQN

A Deep Q-network (DQN) with weights θ updates its weights in each iteration i so as to minimize the mean squared error between the right hand side of equation (1) and the current Q estimate given by the network. In order to mitigate the issue of high correlation among consecutive training samples, the technique of experience replay is used. For this, the agent's experiences are stored in a replay buffer D and random samples from this buffer are used to update θ_i. Gradient descent is performed on the following mean squared error loss function

$$L(\theta_i) = \mathbb{E}_{(s,a,s') \sim P}\left[\left(y_j - Q(s, a; \theta_i)\right)^2\right],$$

where $y_j = r(s_j, a_j) + \gamma \max_{b \in A} Q(s_{j+1}, b; \theta_i^t)$ for $(s_j, a_j, s_{j+1}) \in D$. Note that θ_i^t, the network parameters used to compute the target at iteration i are different from θ_i. The Q-network parameters are copied to the target network parameters periodically to make the training stable. Further ϵ-greedy strategy is used for exploration. This means that at each time step, the agent chooses a random action with probability $\epsilon \in (0, 1)$ or the best action based on its current estimate of Q-values (i.e., $\arg\max_{a \in A} Q(s, a; \theta_i)$) with probability $(1 - \epsilon)$.

B. Relaxation methods

Successive relaxation methods are iterative methods for solving systems of linear equations. These are variants of the Gauss-Siedel procedure involving an additional parameter. It is known that for certain positive choices of the relaxation parameter, faster convergence is obtained [9].

C. Related work

In tabular reinforcement learning where the Q-function is maintained as a table of Q-values for each (s, a), methods such as Speedy Q-learning [10] and SOR Q-learning [11] have been proposed to speed up the convergence of the algorithm. In order to reduce the training time in deep RL, methods such as having a supervised pre-training stage for feature learning [12] and fast reward propogation via optimality tightening [13] have been proposed. Several variants of DQN have been proposed such as Double DQN [14], Prioritized Experience Replay [15], Dueling network architecture [16] to improve stability and performance during the test phase. While these methods aim to achieve better performance once training is completed, our aim is to accumulate rewards faster during the training phase itself.

III. Algorithm

We propose a simple modification in the target Q-values to speed up the DQN algorithm. From Equation (1), we see that Q^* is a fixed point of the Bellman operator H defined as $HQ(s, a) := r(i, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{b \in A} Q(s', b)$. It is known that H is a contraction mapping with contraction factor equal to the discount factor of the MDP γ. The method that we propose is based on the Generalized Bellman operator [3] given below, which uses the concept of successive relaxation.

$$H^w Q(s, a) := w\left(r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{b \in A} Q(s', b)\right) + (1 - w) \max_{c \in A} Q(s, c),$$

where $w \in (0, w^*]$ and $w^* := \min_{s,a} \dfrac{1}{1 - \gamma P(s|s, a)}$ depends on the underlying MDP (note that $w^* \geq 1$).

It is proven, see [11], that H^w is a max-norm contraction with contraction factor $(1 - w + \gamma w)$. Moreover, for $w \in [1, w^*]$ and $\gamma \in (0, 1)$,

$$1 - w + \gamma w \leq \gamma. \qquad (2)$$

Let Q' be the unique fixed point of H^w. Then,

$$Q'(s,a) = w\left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{b \in A} Q'(s',b)\right) + (1-w) \max_{c \in A} Q'(s,c). \quad (3)$$

Further, it has been shown [3] that both Q' and Q^* yield the same *optimal value function*. i.e.,

$$\max_{a \in A} Q'(s,a) = \max_{a \in A} Q^*(s,a) \qquad \forall s \in S.$$

Hence, either Q' or Q^* can be used to compute the optimal policy. Based on this idea, an algorithm known as SOR Q-learning has been proposed in tabular reinforcement learning. This algorithm has been shown to converge faster than Watkins' Q-learning for the case $w > 1$, which is known as over-relaxation [11].

We borrow these ideas to the field of deep RL, by modifying the target Q-values in the DQN algorithm. Instead of using the future reward based on the next state as in the Bellman equation (1), the proposed algorithm uses the Generalized Bellman equation (3) which has a weighted combination of the target given by the Bellman equation and the value function of the current state. This combination is expected to make the the Q-values converge faster when $w > 1$ because the contraction factor of the Generalized Bellman operator is less than that of the Bellman operator in this case (see (2)). The target Q-values in our algorithm, which we call SOR-DQN will be as follows.

$$y_j = w(r(s_j,a_j) + \gamma \max_{b \in A} Q(s_{j+1},b;\theta_i^t)) + (1-w) \max_{c \in A} Q(s_j,c;\theta_i^t),$$

where $w > 1$ is an input to the algorithm. The pseudo code of the algorithm is given in Algorithm 1.

For episodic MDPs which have special states designated as the starting state and some terminal states, the targets y_j are set as follows in SOR-DQN.

$$y_j = \begin{cases} r(s_j,a_j), & \text{for terminal } s_{j+1} \\ w(r(s_j,a_j) + \gamma \max_{b \in A} Q(s_{j+1},b;\theta_i^t)) & \\ +(1-w) \max_{c \in A} Q(s_j,c;\theta_i^t) \text{otherwise.} & \end{cases}$$

Gradient descent is used to minimize the mean squared error between the target and the current Q-estimate, as in DQN. By modifying the targets in this way, we expect the algorithm to learn faster i.e., collect more rewards in less time. (note that when $w = 1$, the update rule is same as that of DQN.) This is important in applications for which performance during the initial phase of learning is also crucial. In the next section, we discuss one such application - automatic allocation of resources to cloud applications.

IV. Autoscaling of Cloud Applications

The resource requirements of an application hosted on the cloud keep on changing due to variations in its real-time workload. Autoscaling refers to the automatic allocation or deallocation of resources to a cloud application based on its requirements. The decisions regarding allocation should not only consider the cost of the resources involved, but also the performance requirements of the application such as throughput, response time etc. as specified in the Service Level Agreement (SLA). This problem can be modelled using a Markov Decision Process as follows.

- The state at time t is (W,v), where W is the workload of the application and v is the number of units of resource currently allocated.
- The action or decision at time t is the amount of resources to be allocated or deallocated for the next time period.
- After action a is taken, the new state becomes (W',v') where W' depends on the workload characteristics and $v' = v + a$.
- The reward is the negative of a cost function which is defined as follows.

  $$C((W,v),a) := c \times v + SLA_{penalty}(r),$$

  where c is the cost per unit of resource, r is the response time of the application and $SLA_{penalty}$ is based on the performance requirements of the application as specified in the SLA between the client and the cloud service provider.

While reinforcement learning is a natural solution to this adaptive decision making problem, tabular methods such as Q-learning fail to be computationally feasible for large state and action spaces. Recently, deep reinforcement learning was proposed [17] to address the scalability issue in autoscaling. However, methods have not yet been developed to make the algorithm perform well during the training phase. We propose the use of SOR-DQN to handle this issue.

Algorithm 1 Deep Q-network with Successive Over-relaxation (SOR-DQN)

```
Input: Initial weight vector $\theta$, discount factor $\gamma$, relaxation parameter $w > 1$, number of iterations $N$, size of replay memory $M$, minibatch size $b$, exploration rate $\epsilon$, initial state $s_0$, frequency of target updation $C$
Initialize replay memory $D$ to capacity $M$
Initialize action value function $Q$ with weights $\theta$ and target action value function $\hat{Q}$ with weights $\theta^t = \theta$
for $i = 0, 1, 2, ..., N-1$ do
    With probability $\epsilon$, select a random action $a_i$
    Otherwise select $a_i = \arg\max_{a \in A} Q(s_i, a; \theta)$
    Execute action $a_i$ and observe reward $r_i$ and next state $s_{i+1}$
    Store transition $(s_i, a_i, r_i, s_{i+1})$ in $D$
    Sample random $b$ size minibatch $B$ from $D$
    for $(s_j, a_j, r_j, s_{j+1}) \in B$ do
        Set $y_j = w(r(s_j,a_j) + \gamma \max_{b \in A} Q(s_{j+1}, b; \theta_i^t)) + (1-w) \max_{c \in A} Q(s_j, c; \theta_i^t)$
        Perform gradient descent on $(y_j - Q(s_j, a_j; \theta))^2$
    Every $C$ steps reset $\hat{Q} = Q$
```

V. Experiments

In this section, the proposed algorithm is experimentally evaluated. We first consider the interesting application of autoscaling cloud resources that was described in Section IV. The resource under consideration is the number of virtual machines (VMs) allocated for the application. The cost incurred by the proposed algorithm is compared to that of DQN on both simulated and real workloads. Next, we study the performance of both the algorithms on a set of 48 Atari games. The relaxation parameter w was chosen as 1.3 in all the experiments.

A. Cloud computing

1) Parameters: In the MDP model, the action space was set as

$$A = \{-10, -8, ..., 0, ..., 8, 10\}.$$

The constant c was set as 0.1 and the penalty function was chosen as $SLA_{penalty}(r) = r$. The same neural network was used for both DQN and SOR-DQN. Since the state is low-dimensional, we used a simple fully connected network consisting of 3 hidden layers. The number of neurons in these layers was chosen to be $4, 8, 16$ respectively. 'ReLU' activation function was used in all layers except the output layer where 'linear' activation was used. The output layer has 11 neurons corresponding to the number of actions in the MDP model. A schematic diagram of the network is shown in Figure 1.

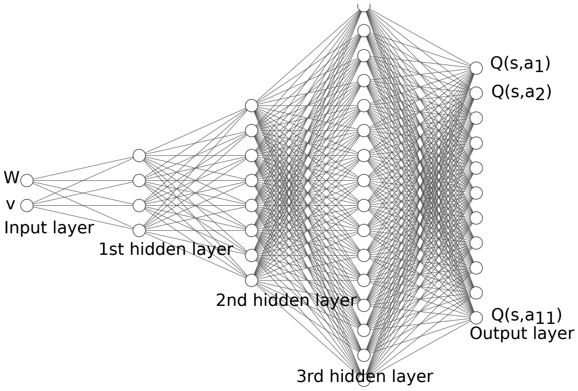

Fig. 1. Neural network used in Cloud computing experiments

2) Simulated data: We use the CloudSim [18] platform to simulate cloud environment. CloudSim is a Java based framework for modeling and simulation of cloud computing infrastructure and services. The workload W of the application for each hour was generated such that $W = min(x, 50)$ where $x \sim Poisson(25)$. The probability mass function of $Poisson(\lambda)$ distribution with mean $\lambda = 25$ is shown in Figure 2. The average total cost incurred by DQN and SOR-DQN (averaged over 5 simulation runs) against time is plotted in Figure 3 for a duration of 1000 hours. It is seen that SOR-DQN achieves a significantly lower cost as compared to DQN right from an early stage of training.

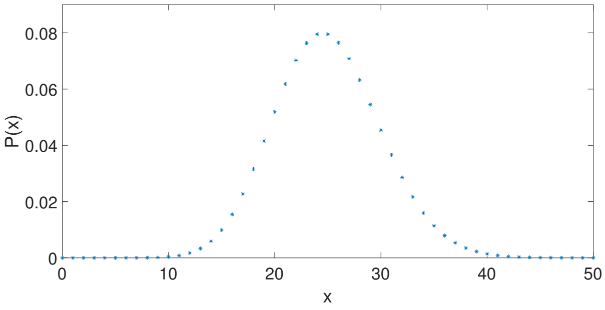

Fig. 2. Probability mass function of Poisson distribution with mean $\lambda = 25$

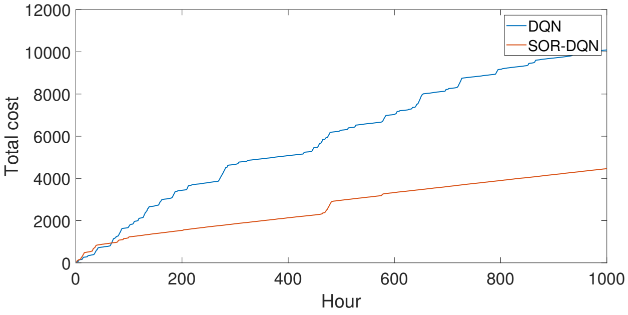

Fig. 3. Total cost incurred by DQN and SOR-DQN algorithms on simulated data

3) Real data: Next, the experiment is repeated with real workloads of web servers that are publicly available [19]. We consider two datasets : 1. Two weeks of HTTP logs of Clarknet server from August 28, 1995 to September 10, 1995 (ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area). 2. One month of HTTP logs of NASA server from July 1, 1995 to July 28, 1995. These logs are preprocessed to get the timeseries data of the number of hourly requests, which is shown in Figures 5 and 6 respectively. We ran the two algorithms and compared the total cost incurred. The results are in Figures 7 and 8. It is seen that SOR-DQN outperforms DQN on real data as well. These experimental results confirm our assertion that SOR technique in deep reinforcement learning is useful for improving performance during the training phase.

B. Atari games

Atari 2600 video games is an RL testbed implemented in the Arcade Learning Environment [20], which consists of diverse and interesting set of tasks (which are episodic in nature). It was shown in [1] that a single DQN agent was able to successfully learn to play several of these games, using only the video input and the scores as humans do. We compare the performance of SOR-DQN against that of DQN *during the training phase* on 48 Atari games as in [2], using the same neural network and other hyperparameters in [2]. After performing $N = 50,000$ training iterations for both the agents we note the scores denoted as $Score_{DQN}$ and $Score_{SOR-DQN}$ respectively. The percentage improvement

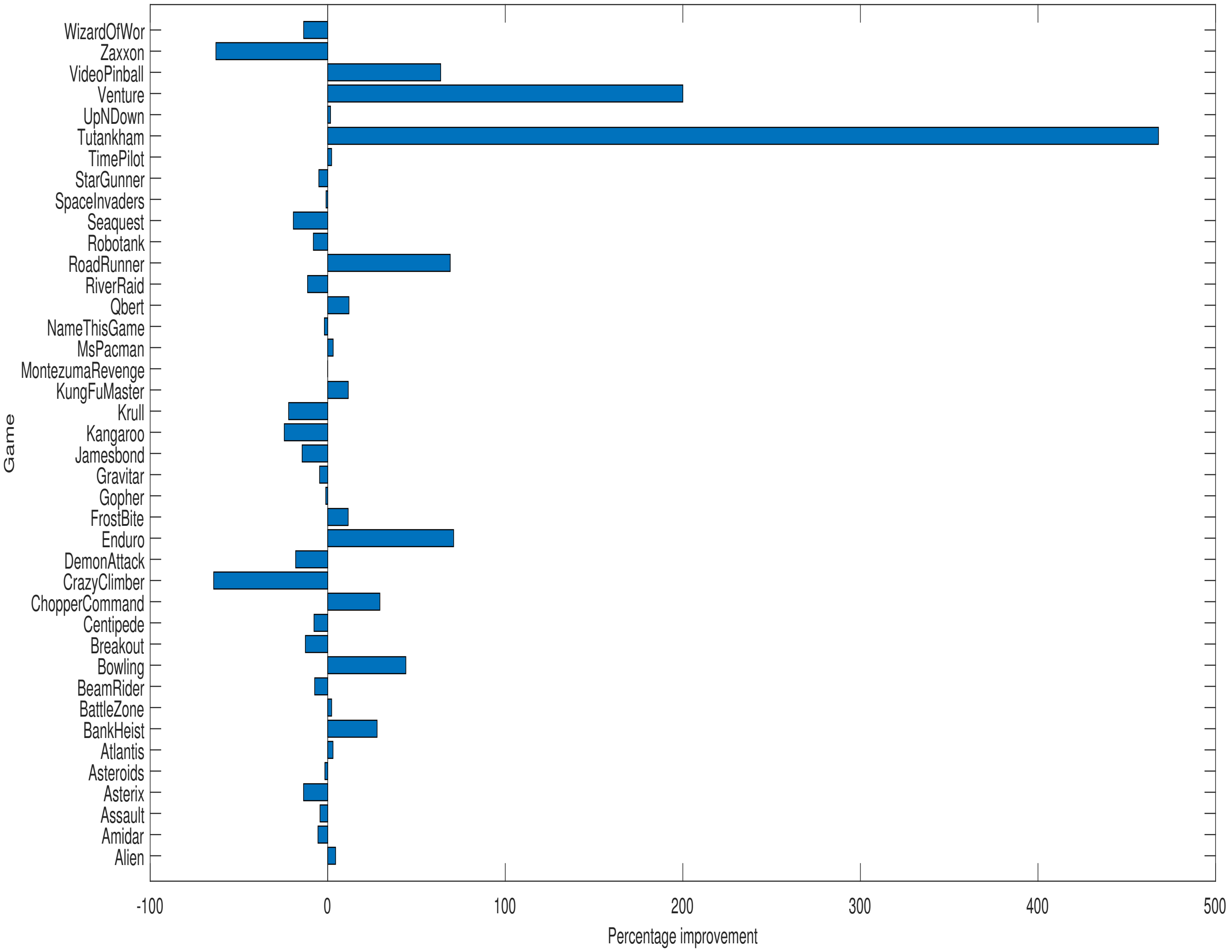

Fig. 4. Percentage improvement of SOR-DQN over DQN on Atari games

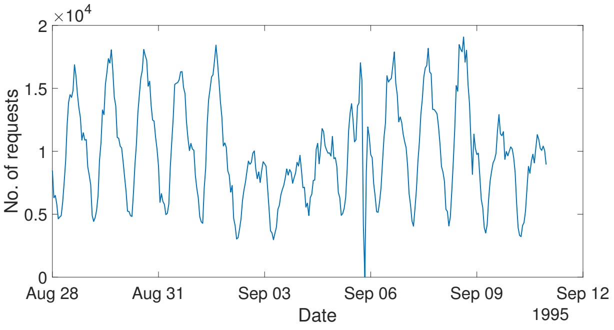

Fig. 5. Hourly workload of ClarkNet server for two weeks

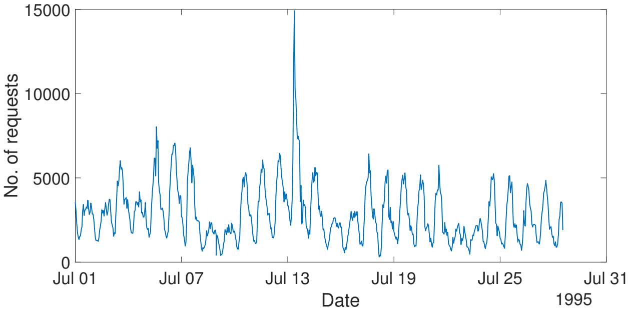

Fig. 6. Hourly workload of NASA server for two months

(P.I.) of SOR-DQN with respect to DQN on a game is calculated as

$$P.I. = \frac{Score_{SOR-DQN} - Score_{DQN}}{Score_{DQN}} \times 100$$

The results for the individual games is shown in Figure 4. (We have omitted 7 games on which both the algorithms gave negative scores - Boxing, DoubleDunk, FishingDerby, IceHockey, Pong, PrivateEye, Tennis as well as the game Freeway on which DQN attained a score of 0 whereas SOR-

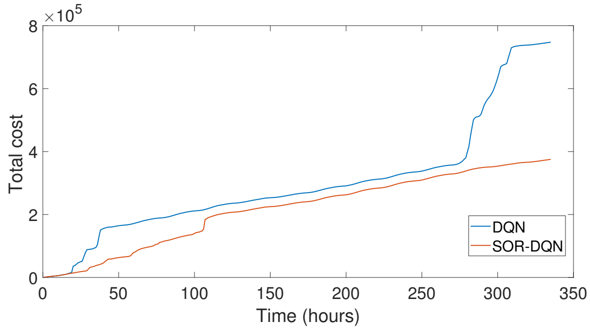

Fig. 7. Total cost incurred by DQN and SOR-DQN on ClarkNet dataset

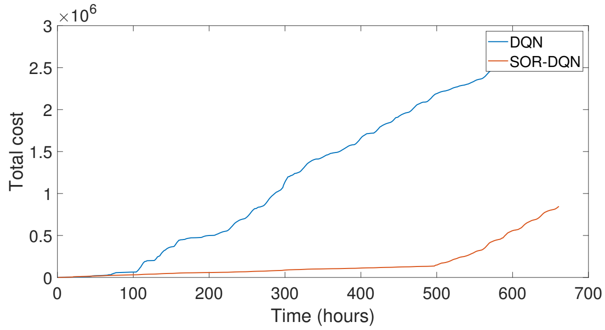

Fig. 8. Total cost incurred by DQN and SOR-DQN on NASA dataset

DQN attained a score of 556.) The overall improvement (O.I.) is calculated as the mean of the percentage improvement on individual games.

1) Discussion: It was found that the overall improvement achieved by SOR-DQN over DQN across all games put together is as good as 17.5%, although the number of games on which SOR-DQN achieves higher score is slightly less than half. It may be noted that we used the same constant value of the parameter w for all the games in the implementation of SOR-DQN and did not adapt the same for different games. If the value of w is adapted for each game individually, much better results will be obtained for SOR-DQN.

VI. Conclusion and Future work

We proposed a new deep reinforcement learning algorithm, SOR-DQN, to improve the performance of the RL agent during the training phase. The update rule of the algorithm is a simple modification of the update rule for DQN, motivated by the successful application of successive over-relaxation techniques in tabular reinforcement learning. Through experimental study on simulated and real datasets, it was seen that the proposed algorithm incurs lower training cost as compared to DQN in autoscaling cloud applications. Further, it was found to have a generalization ability better than that of DQN on Atari games.

In the future, we would like to explore the possibility of applying the successive over-relaxation technique in other deep RL methods, such as those for continuous control [21]. Further, the choice of the parameter w plays an important role in the algorithm. It would be useful to develop a heuristic to determine the value of w for each problem instance. We believe that with a good heuristic for choosing w, the generalization ability of the algorithm would improve significantly.

References

[1] 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.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] D. Reetz, “Solution of a markovian decision problem by successive overrelaxation,” *Zeitschrift für Operations Research*, vol. 17, no. 1, pp. 29–32, 1973.

[4] I. John, A. Sreekantan, and S. Bhatnagar, “Efficient adaptive resource provisioning for cloud applications using reinforcement learning,” in *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2019, pp. 271–272.

[5] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, “Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow,” in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, 2011, pp. 67–74.

[6] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, “A review of auto-scaling techniques for elastic applications in cloud environments,” *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014.

[7] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[8] D. P. Bertsekas and J. N. Tsitsiklis, “Neuro-dynamic programming: an overview,” in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, vol. 1. IEEE, 1995, pp. 560–564.

[9] D. Young, “Iterative methods for solving partial difference equations of elliptic type,” *Transactions of the American Mathematical Society*, vol. 76, no. 1, pp. 92–111, 1954.

[10] M. G. Azar, R. Munos, M. Ghavamzadaeh, and H. J. Kappen, “Speedy q-learning,” 2011.

[11] C. Kamanchi, R. B. Diddigi, and S. Bhatnagar, “Successive over-relaxation Q-learning,” *IEEE Control Systems Letters*, vol. 4, no. 1, pp. 55–60, 2019.

[12] G. V. Cruz Jr, Y. Du, and M. E. Taylor, “Pre-training neural networks with human demonstrations for deep reinforcement learning,” *arXiv preprint arXiv:1709.04083*, 2017.

[13] F. S. He, Y. Liu, A. G. Schwing, and J. Peng, “Learning to play in a day: Faster deep reinforcement learning by optimality tightening,” *arXiv preprint arXiv:1611.01606*, 2016.

[14] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[16] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, “Dueling network architectures for deep reinforcement learning,” *arXiv preprint arXiv:1511.06581*, 2015.

[17] C. Bitsakos, I. Konstantinou, and N. Koziris, “Derp: A deep reinforcement learning cloud system for elastic resource provisioning,” in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 21–29.

[18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[19] M. F. Arlitt and C. L. Williamson, “Web server workload characterization: The search for invariants,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 126–137, 1996.

[20] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.