DNN Tree Search for Bayesian Reinforcement Learning to Machine Intelligence

Anil Kumar Yadav, Ajay Kumar Sachan

Abstract- Bayesian model-based reinforcement learning can be formulated as a partially observable Markova decision process (POMDP) to provide a principled framework for optimally balancing exploitation and exploration. Then, a POMDP solver can be used to solve the problem. If the prior distribution over the environment's dynamics is a product of dirichlet distributions, the POMDP's optimal value function can be represented using a set of multivariate polynomials. Unfortunately, the size of the polynomials grows exponentially with the problem horizon [3]. During machine learning agent required lots of training inputs of execution cycle. Due to this situation look up table contain huge amount of data base. In this paper, we observe the use of dynamic neural network tree search (DNNTS) algorithm for large POMDPs, to solve the Bayesian reinforcement learning problem. The keen idea of DNN tree search is to train agent and act as a NN classifier to help agent for taking self decision without prior knowledge of the system during data learning .We will show that such an algorithm successfully searches for a near-optimal policy and achieve goal. Experiments show that the used DNN methods improve performance of Bayesian reinforcement learning in the context of training episodes, reward and discount rate.

Keywords: Bayesian reinforcement learning, machine learning, DNN tree search, POMDP

I. INTRODUCTION

Reinforcement learning (RL) provides a structure for concurrently acting and learning in unknown environments [1, 2]. To act well in such situations, reinforcement learning algorithm has to handle the exploration-exploitation trade-off- it needs to balance actions that reduce its uncertainty about the environment with actions that exploit what it already knows. RL has had some remarkable practical successes in various areas, including learning to play checkers, backgammon job-scheduling, chess, dynamic channel allocation and others. Traditionally, RL algorithms can be divided into two major approaches: model-free and model-based. Model free approaches attempt to directly learn the optimal policy by approximating the cost-to-go of each state, called a value function. These methods often have large variance and poor trade-off between exploration/exploitation. On the other hand, model-based approaches attempt to learn a model of the environment and then compute the optimal policy based on that learnt model. These approaches normally have better trade-off between exploration/exploitation. However both of them are impractical to learn online due to intensive computation and poor trade-off ability.

Manuscript Received on September 2014.

Dr. Ajay Kumar Sachan, had completed his Ph.D in Department of (Computer Science & Engineering) from Rajeev Gandhi Technical University, Bhopal, India.

Anil Kumar Yadav, received the B.Tech in Department of (CSE), UPTU and M.Tech (I.T), SATI, RGPV, Bhopal, India.

One approach to reduce this problem is to use DNN model based RL. Because it will trade-off searching/ utilization and uses less data required [4]. Bayesian reinforcement learning can be represented as a partially observable Markov decision process (POMDP) problem. Its policy is a mapping from the posterior distribution (or history of observations) to an action. This POMDP problem can be solved by an online planning algorithm. In addition, its policy at each step can be considered as a suggestion of the appropriate action for online Bayesian reinforcement learning. When representing Bayesian reinforcement learning as a POMDP, the posterior distribution of parameters given an observation is often conveniently represented in closed form as a product of Dirichlet distributions. It was shown [6, 3] in that the optimal value functions in Bayesian reinforcement learning. It can be represented using a set of multivariate polynomials. Unfortunately, the size of the polynomial set grows exponentially with the problem horizon, severely limiting the applicability of the method [3]. In this paper, we study a research and application of dynamic neural network based reinforcement learning [35], leaning agents take sequential actions with the goal of maximizing a reward signal, which may be time-delayed. That much knowledge acquired by agent takes large repetitive loops in every episode. It may affect on discount rate and training time. For example, an agent could learn to play a game by being told whether it wins or loses, but is never given the "correct" action at any given point in time. The RL structure has gained fame as learning methods have been developed that are capable of handling gradually more complex problems. RL research focuses on improving the speed of learning by exploiting domain expertise with varying amounts of human-provided knowledge. Common approaches include Q-learning, TDN and swarm learning rather than simple one-step actions; and efficiently abstracting over the state space [10-15], so that the agent may generalize its experience more efficiently. The insight behind dynamic neural network (DNN) is used to learning data memorization for look table to train the agent. Many papers were compared different methods among discount rate and training episodes [7, 8, 9].Dynamic neural network in RL is a vital topic to address at this time for three reasons. First, in recent years RL techniques have achieved notable successes in difficult tasks which other machine learning techniques are either unable or ill-equipped to address (e.g., TD Gammon Treasure 1994, job shop scheduling Zhang and Dietterich 1995, elevator control Crites and Barto 1996, helicopter control Ng et al. 2004 Robot Soccer Keep away Stone et al. 2005).

Second, classical machine learning techniques such as rule induction and classification are sufficiently grown-up that they may now easily be leveraged to assist with DNN. Third, they can be very effective at speeding up learning [35]. In the following, we describe some related works. Then, we start with the Q learning formulation of RL in Sect. 2.1. Next, we describe the DNN algorithm in sect. 2.2 and DNNRL in Sect. 3. Then, experimental results are presented in Sect. 4. Finally, we conclude with some discussions on the future research in Sect. 5.

II. RELETED WORK

Bayesian reinforcement learning has been studied many research works such as [24, 25–28,29, 30,31, 32,33, 34]. POMDP solver to find a policy that can then be used as a reinforcement learning algorithm MCBRL algorithms represented larger transition matrices. While our algorithms used NN classifier, which scales up efficiently with problems having larger transition matrices. Monte-Carlo tree search (MCTS) algorithm [3]. It also preferentially expands the search tree by maintaining hard upper and lower bounds on the values for each state and action so as to direct the rollouts: action is chosen greedily according to the upper bound, there is an alternative approach in RL, called PACMDP, Efficient in dealing with the trade-off between explorationand exploitation [5, 19, 20, 22, 23]. PAC-MDP algorithms use exploration actions gathering necessary information, and then later exploit this information to choose optimal or near-optimal actions, which maximize the cumulative reward. Recently, Bayesian methods combined with PACMDP approach were also developed to build a better exploration model [21]. These methods were proved to give lower sample complexity bounds. The algorithm that we used in this paper is a modification of the Q learning algorithms (QA) method in [35].

A. Existing Approaches for Reinforcement Learning

In this paper [15], we have discussed about a RL problem, whose environment is formulated as a state action Q(s,a) and try to find optimal policy, Q-learning can be formulated as follows: $Q(s_{t,}a_t)= r_{t+1}+\gamma\max Q(s_{t+1}, a)$

$R_1=R\gamma(x_1, i)$

Where α is the learning rate and γ (0<γ<1) is discount rate. r_{t+1} is the reinforcement signal in t+1 moment. Essentially, the estimate for Q (s_t,a_t), the value of the state action pair at time t is updated using the best estimated value of the next state. At any state action pair Q (s_t ,a_t) and single reward R1 and next state possible. Where X_1 is the total steps that the agent move from initial state to the final reward state in episode. And i, is the count steps that the agent move from some initial state to the current state. γis 0.9 discount rate. $R_1$is the reward agent is given when it achieve the goal state and it is expressed as a numeric value (credit) with parameter γ, X1, t. For each state action pair in the episode to learn more effective behavior (state action pair with large value).it is important that the policy is rational. Q learning usually stores Q value relative with every state action in a lookup table [16]. In Q-learning, there are sequences of episodes, learning steps repeat in every episode, in n^th time.

Step 1 generate randomly starting state (s_n)
Step 2 search available actions (a_n)
Step 3 selects any one action randomly.
Step 4 if checks previously taken same action then repeat from step one
Step 5 now check for goal
Step 6 if goal achieved than next episode
Step 7 else, store (s_n , a_n) in temp array
Step 8 update Q_{n-1} (s_n ,a_n) according to.
$R1=R\gamma(x_1\text{-}i)$
Step 9 now generate next state (s_{n+1}), using state action
Step 10 if goal achieve then next Step 11 else repeat above step until stop criterion is satisfied. In Mat lab, we have trained the agent in the form of state action pair or Q-table. In our Implementation had size of (100x4) for <10x10> grid world problem. Where α is the learning rate and γ (0<γ<1) is the discount factor that reduces the Influence of future expected rewards. So technically speaking, Q learning is evaluated in terms of q-value and rewards of each agent over every trial.

B. DNN Approaches for Reinforcement Learning

In this section, we had described about dynamic neural network algorithm.DNN is effective decision making unit as a NN classifier to take as an input/action label. Dynamic neural network reduced q-table and agent should learn during real time operation. Those algorithms are as follows.
Step: 1 predict next state by NN
Step: 2 if decision by NN and agent both are same
(I) then predict next state is goal
(ii) Exit (achieved goal)
Step: 3 else (Both are not same)
Step: 4Update NN

C. DNN Development for Bayesian Reinforcement Learning

Dynamic neural network (DNN) was introduced in [17, 18] for classification and learning continuous or large complex data sets. Here, we represent DNN with Q learning because for random policy generation, it has no knowledge beyond the search tree. Our algorithm is in terms of dynamic neural network Bayesian Reinforcement Learning (DNNBRL) as described. Each node of the search tree is labeled state action with a pair (s_n ,a_n) of a current MDP history in the form of $Q(s_{t,}a_t)=r_{t+1}+\gamma\max Q(s_{t+1}, a)$
There are training algorithms for Bayesian RL as follows.
Step: 1 generate random action an
Step: 2 move to next state according to action
Step: 3 if goal is not achieved
Step: 4 go to step (1)
Step: 5 update reward using R1=Rγ(x1-i)
Step: 6 check for iteration limit
Step: 7 if under iteration limit go to step (1)
Step: 8 else train NN by state action table or look-up-table or q table
Step: 9 predict next state by NN
Step: 10 if decision by NN and agent both are same

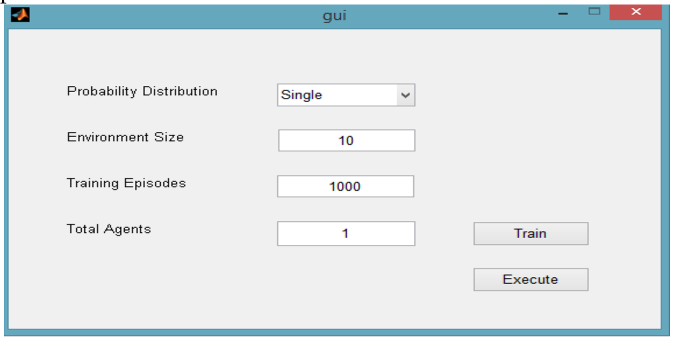
(I) then predict next state is goal
(ii) Exit (achieved goal)
Step: 11 else (Both are not same)
Step: 12 Update NN

III. EXPERIMENTAL RESULT AND DISSCUSSION

In this section, we evaluated the performance of DNNRL on simple problems from the previous problems as 6x6 Maze, and compare with MCBRL algorithms [3]. We have also performed training and testing phase in mat lab simulation as follows.

A. Training Phase

In training phase fig.4.1, showed as the environment size 10,once training episode 1000, single agent and state input, which we call state having the particular value s and action values a , both are presented for agent training. First of all, it will take random action from starting state and search available action take as an input from environment and achieve a new state. If previously taken state-action is equal to the current state then repeat and arrive starting state. Now examine goal state if goal achieved then update next episode. Otherwise, store state action pair (s_t a_t) in look-up-table .Again generated next state (s_{t+1}) using state-action pair. If then goal achieved, then exit. Now let's started train the agent in the form of state-action pair. Then click on train button as given fig. 4.1, after successful completion of training, click execute button and then show result as a fig. 4.2. You can also saw, learned agent travelling in grid world in the form of decision path. Now specify the inputs state S(1,1) and goal G(6,6). State-action pair (10x4) implemented for 10x10 maze problem.



Probability Distribution	Single
Environment Size	10
Training Episodes	1000
Total Agents	1

Fig. 4.1 10×10 Maze Problems with Agent Training Model

B. Testing Phase

In testing phase fig.4.2 showed, if action is possible then trained agent moved in the direction of, upwards, downwards, right side or left side. If the movement is not possible due to the border of the grid world, do nothing and decide the next action again at random. This would be repeated until the agent reaches the goal. The maximum number of steps should be determined depending on the size of the grid world or cell. Now we looked at the result of a random movementof agent in fig 4.2. It has actions to move into (x,y+1),(x,y-1),(x-1,y) and (x+1,y).some cell have walls at the boundaries with their adjacent cells, and the movement of the agent is blocked by the walls and the edge of the grid world. In addition, hereno roll back condition occurred in .Fig.4.2.In this section, we experimented with the 10×10 maze problem as in [3]. In this problem, we assume environment size as matrix 10x10 and there are 4 possible actions {L, R, U, D} where L is move left side, R is move right side, U is move upward and D is move downward. Agent moves one step upwards, downwards, to the right, or to the left, if the action is possible. If the movement is not possible due to the border of the grid world, do nothing and decide the next action again at random. This would be repeated until the agent reaches the goal. The maximum number of steps should be determined depending on the size of the maze problem. We now look at the result of a random move by agent in a mentioned above. See the Figure. 4.2. Where an agent reaches the goal cell, it gains the reward 100.the value of discount rate parameter is set to be 0.999. Comprehensive way to remove loops and find shortcuts from episode for speeding up convergence, While the start cell is (1, 1) and fixed, the goal cell (6, 6) and is determined at random. The agent perceives its own coordinates (x, y), and has four possible actions to take: moving up, moving down, moving left and moving right, that is to say, it has actions to move into (x,y+1),(x,y-1),(x-1,y) and(x+1,y).some cell have walls at the boundaries with their adjacent cells, and the movement of the agent is blocked by the walls and the edge of the grid world. In addition, there are no roll back condition occurred here.

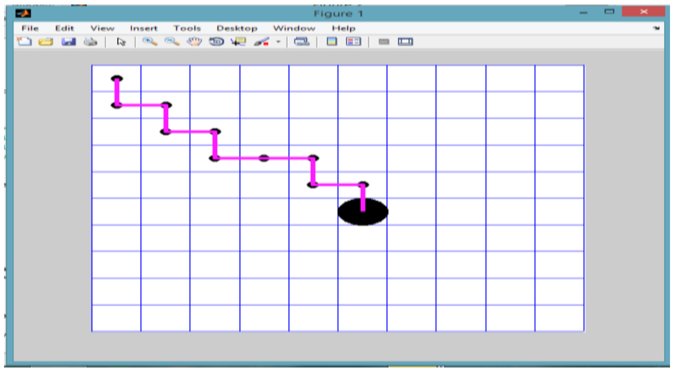


Fig. 4.2 Show as 10x10 Maze Problem, how Agent Travel Shortest Route During 50 Trials Over 500 Episodes. Agent Starting Moves from (1, 1) and Reached at the Goal Sate at (6, 6), without a-Priori Information

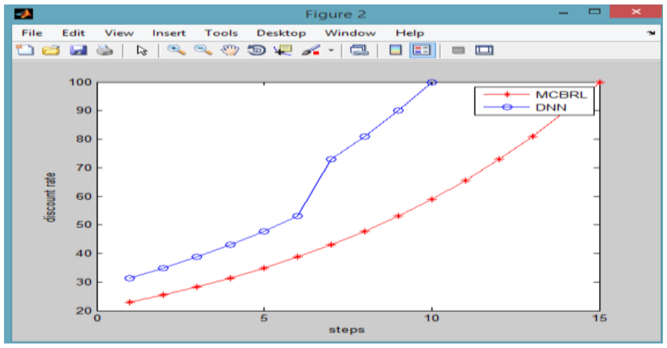


Fig. 4.3 ´Show, the Corresponding Performance Graph of Fig. 4.2, from the Shortest Route to the Goal, During 50 Trials Over 500 Episodes

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The results shown in fig 4.3, that DNN is more effective than MCBR, which can quickly learn and find the aiming goal as described in Fig. 4.2. The performance of DNN with MCBRL is effective in terms of discount rate vs. episodes.

C. Cumulative Training Reward vs. Trials for 10×10 Maze

In order to access, fig.4.4, we found that our algorithms are better than MCBRL in the context of commutative reward vs. trials because DNN learn within 10 steps.

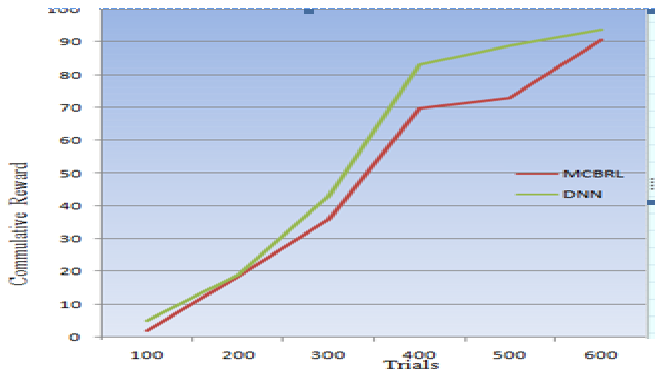


Fig. 4.4 Cumulative Training Reward vs. Trials for 10×10 Maze

IV. CONCLUSION

We studied the use of dynamic neural network for Bayesian reinforcement learning problems to machine intelligence. The use of dynamic neural network in Bayesian learning I found that agent did not required a lot of training input cycle during execution .Because it used NN classifier to classify and avoid repetitive or once learned data. The results that we have presented show that the algorithm is able to achieve superior performance in Bayesian reinforcement learning in fig.4.3 and fig 4.4, in the context of discount rate vs. steps and commutative reward vs. trials. Further research challenges and constraints of DNN are elaborated. Finally, in this section of paper we discussed some future research direction with conclusion of this work to intend that it helps to many researchers who are working for improvement in Bayesian reinforcement learning with SVM and others.

REFERENCE

[1] Alpayadin, Introduction to machine learning, MIT press, Cambridge (2005).

[2] S.N & S.N, principal of soft computing, , (2008).

[3] Ngo AnhVien , Dang, Monte-Carlo tree search for Bayesian reinforcement learning, Springer, New York,(2013) pp.345-353.

[4] Matthew E. Taylor, Transfer Learning for Reinforcement Learning Domains: A Survey, Journal of Machine Learning Research, (2009),pp.1633-1685.

[5] Brafman RI, Tennenholtz, R-max—a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res ,(2002),pp.213–231.

[6] ManjeevanSeera, CheePeng Lim, A hybrid intelligent system for medical data classification", Elsevier, (2013), pp.2239-2249.

[7] Andrew James, Julian, Cartesian Genetic Programming encoded Artificial Neural Networks: A Comparison using Three Benchmarks,(2013), pp.1005-1012.

[8] Pankaj Deep,Inderveer ,Cloud based intelligent system for delivering health care as a service, (2013), pp.346-359.

[9] Ima, H., Karo, Y,Swarm Reinforcement Learning Algorithms Based on Sara Method, (2008),pp. 2045–2049.

[10] Karbasian, H., Maida, N,Improving Reinforcement Learning Using Temporal Deference Network EUROCON,(2009), pp. 1716–1722.

[11] Quinn, L., Ming, C.Z., The Research on the Spider of the Domain-Specific Search Engines Based on the Reinforcement Learning,(2009),pp. 588–592.

[12] Trooper, J.W.C, Optimizing Time Warp Simulation with Reinforcement Learning Techniques, (2007),pp. 577–584.

[13] Dam Silva, R.R., Claudio, An Enhancement of Relational Reinforcement Learning, pp. 2055–2060. IEEE (2008).

[14] Halmahera, K., Tadahiro, Effective integration of imitation learning and reinforcement learning by generating internal reward,(2008) pp. 121–126.

[15] Fang, Z., Tan, Reinforcement Learning Based Dynamic Network Self-Optimization for Heterogeneous Networks, (2009) pp. 319–324.

[16] Anil kumarYadav,Evaluation of Reinforcement Learning Techniques, pp. 1–4. ACM (2010).

[17] Tsung-Hsien, Lee, C.-K., Desin of dynamic neural network to forecast short-term railway passenger demand. (2005), 1651–1666.

[18] Li, H., Kozma, R., A Dynamic neural network metheod for time series prediction using the KIII model,(2003), pp. 347–352.

[19] Kakade S, Kearns MJ, Exploration in metric state spaces. In: International conference on machine learning (ICML),(2003), pp 306–312.

[20] Kearns MJ, Singh SP ,Near-optimal reinforcement learning in polynomial time. Mach Learn 49(2–3):209–232.

[21] Kolter JZ, Ng AY (2009) Near-Bayesian exploration in polynomial time. In: International conference on machine learning (ICML).

[22] Strehl AL, Littman ML (2008) An analysis of model-based interval estimation for Markov decision processes. J ComputSystSci 74(8):1309–1331.

[23] Szita I, Szepesvári C (2010) Model-based reinforcement learning with nearly tight exploration complexity bounds. In: International conference on machine learning (ICML), pp 1031–1038.

[24] Asmuth J, Littman ML (2011) Learning is planning: near Bayesoptimal reinforcement learning via Monte-Carlo tree search. In: Proceedings of the twenty-seventh conference on uncertainty in artificial intelligence, pp19-26.

[25] Dearden R, Friedman N, Russell SJ (1998) Bayesian Q-learning. 1998, pp 761–768.

[26] Duff M (2002) ,Optimal learning: computational procedures for Bayes-adaptive Markov decision processes.

[27] Engel Y, Mannor S, Meir R (2003) Bayes meets bellman,the Gaussian process approach to temporal difference learning. (ICML), pp 154–161.

[28] Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes, pp. 201–208.

[29] Ghavamzadeh M, Engel Y (2006) Bayesian policy gradient algorithms, pp 457–464.

[30] Ghavamzadeh M, Engel Y (2007) Bayesian actor-critic algorithms,pp 297–304.

[31] Poupart P, Vlassis NA, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement learning, pp 697–704.

[32] Strens MJA (2000) A Bayesian framework for reinforcement learning. 2000, pp 943–950.

[33] Vien NA, Yu H, Chung T (2011) Hessian matrix distribution for Bayesian policy gradient reinforcement learning, pp.1671–1685

[34] Wang T, Lizotte DJ, Bowling MH, Schuurmans D (2005) Bayesian sparse sampling for on-line reward optimization.pp 956–963.

[35] Anil Kumar Yadav, Ajay Kumar Sachan, Research and Application of Dynamic Neural Network Based on Reinforcement Learning,springer, AISC ,vol.132, pp. 931–942. 2012.

AUTHOR PROFILE

Dr. Ajay Kumar Sachan, had completed his Ph.D (Computer Science & Engineering) from Rajeev Gandhi Technical University, Bhopal in 2007. He has published more than 25 research papers in National & International Journals. He has more than 21 years teaching & research experience. Presently working as Director in R.I.T.S Bhopal.

Anil Kumar Yadav, recived the B.Tech (CSE), UPTU and M.Tech(I.T), SATI,RGPV Bhopal in 2005 and 2009,respectively and pursuing Ph.D. in the Department of Computer Science and Engineering from IFTM,U.P 2011 onwards. His main research interests include Artificial Intelligence and Machine Learning.