Generic Game Playing Agent Using Deep Reinforcement Learning

Subhojeet
Department of Computer Science
Indian Insitute of Technology
Guwahati, India
subhojeet@iitg.ernet.in

Dhruv Kohli
Depratment of Mathematics
Indian Institute of Technology
Guwahati, India
dhruv.kohli@iitg.ernet.in

Dheeraj Khatri
Department of Computer Science
Indian Insitute of Technology
Guwahati, India
d.khatri@iitg.ernet.in

***Abstract*—One of the machine learning method to build a game playing agent is by using reinforcement learning. But the matrix representation of the state-action value function in the traditional Q-learning algorithm prevents us from applying reinforcement learning in building agents in an environment with infinite number of states. In this project, we tackle this problem by representing the state-action value function in Q-learning with a convolution neural network. And, our aim is to build a generic agent that learns to play any game (tested on limited atari games only) from the sensory input (screen of the game as input) only, by integrating the concepts of convolution neural network in deep learning and markov decision process in reinforcement learning [1].**

***Keywords*—Deep reinforcement learning, markov decision process, state-action value function, Q-learning, CNN**

I. Introduction

Imagine a hot cup of coffee placed before a kid. The kid touches the cup and immediately removes his hand as he felt the burning sensation in his fingers. The kid might repeat same mistake of touching the hot cup but will eventually learn, not to touch the hot cup to prevent burns. This example covers the essence of reinforcement learning where the agent initially explores the environment and then exploits the information acquired while exploring, to make moves that increases his reward. So, we start with an introduction to reinforcement learning, elaborate on the markov decision process with relevant mathematics, state the Q-learning algorithm and then bring in the convolution neural networks as state-action value function to describe deep reinforcement learning. Then, we describe our model that is capable of learning to play any game (tested on atari games only), describe the training details and the results obtained. Finally, we state our conclusion, future work and the progress of future work followed by the appendix.

II. Reinforcement Learning

Here, the learner does not receive a labeled dataset, in contrast with supervised learning. Instead, he collects information through a course of actions by interacting with the environment. In response to an action, the learner or agent receives two types of information: his current state in the environment and a real-valued reward, which is specific to the task and its corresponding goal. Fig. 1 shows the general scenario of reinforcement learning. Unlike supervised learning, there is no fixed distribution according to which the instances are drawn; the choice of a policy defines the distribution.

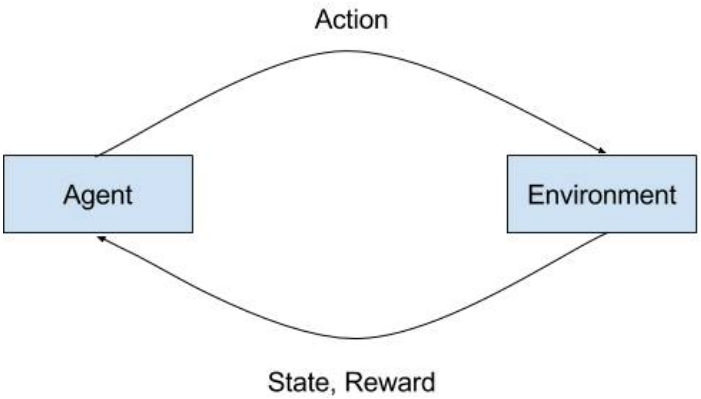

Fig. 1. Representation of general scenario of Reinforcement learning

The objective of the agent is to maximize his reward and thus to determine the best course of actions, or policy, to achieve the objective. **However, the information he receives from the environment is only the immediate reward corresponding to the action just taken. No future or long-term reward feedback is provided by the environment.**

A. Exploration vs Exploitation Trade-off

The agent also faces the dilemma between exploring unknown states and actions to gain more information about the environment and exploiting the information already collected to optimize his reward.

Two main settings are possible:

1) Environment model is known to agent. Then the problem is reduced to **planning**.
2) Environment model is unknown to agent. Then, he faces **learning** problem. This will be our main concern.

B. Markov Decision Process Model

A MDP is defined by:

1) Set of states, S.
2) Set of actions, A.

3) Start state, $s_0 \in S$.
4) Reward Probability

$$P(r_{t+1}|s_t, a_t) \;\; where \; r_{t+1} = r(s_t, a_t)$$

5) State transition probability

$$P(s_{t+1}|s_t, a_t) \;\; where \; s_{t+1} = \delta(s_t, a_t)$$

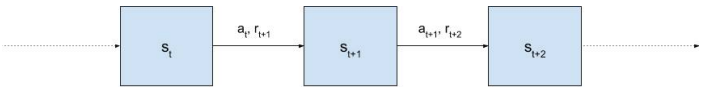

Fig. 2. Illustration of states and transitions of MDP at different times

We also define $\pi : S \rightarrow A$ as the policy function mapping a state S to an action A. In a discrete time model, actions are taken at a set of decision epochs $\{0, ..., T\}$. We deal with infinite horizon i.e. T tends to infinity. Now, the agent's objective is to find a policy that maximizes his expected reward.

C. Policy Value, State-Action Value Function and Q-Learning

We define the policy value $V_\pi(s)$ in the case of infinite horizon as the expected reward of the agent starting at state s and following policy π i.e.

$$V_\pi(s) = E\left[\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+\tau}, \pi(s_{t+\tau}))|s_t = s\right] \tag{1}$$

where T tends to infinity.

The policy value $V_\pi(s)$ obey the following system of linear equations (Bellman's equation):

$$\forall s \in S, \;\; V_\pi(s) = E\big[r(s, \pi(s))\big] + \gamma \sum_{s'} Pr\left[s^{'}|s, \pi(s)\right] V_\pi(s^{'}) \tag{2}$$

Refer appendix for the proof.

The Bellman's equation can also be written as:

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V} \tag{3}$$

where,
$\mathbf{P}$ is the state transition probability matrix, $\mathbf{R} = E[r(s, \pi(s))]$, γ is the discount for future rewards and $\mathbf{V}$ is the unknown policy value matrix

For a finite MDP, Bellman's equation admits a unique solution given by

$$\mathbf{V}_0 = (\mathbf{I} - \gamma \mathbf{P})^{-1}\mathbf{R} \tag{4}$$

Refer appendix for the proof.

The optimal policy value at a given state s is thus given by

$$V_{\pi^*}(s) = max_\pi V_\pi(s) \tag{5}$$

Finally, we define the optimal state-action value function Q^* for all $(s, a) \in S \times A$ as the expected return for taking action $a \in A$ at state $s \in S$ and then following the optimal policy:

$$Q^*(s, a) = E\big[r(s, a)\big] + \gamma \sum_{s' \in S} Pr\big[s^{'}|s, a\big] V^*(s^{'}) \tag{6}$$

One can observe that the optimal policy can be given by

$$\forall s \in S, \;\; \pi^*(s) = argmax_{a \in A} Q^*(s, a) \tag{7}$$

Thus, the knowledge of the state-vale function Q^* is sufficient for the agent to determine the optimal policy, without any direct knowledge of the reward or transition probabilities.

To learn the optimal state-action value function, Q-learning algorithm can be used as shown below:

Algorithm 1 Q-learning Pseudocode [2]

procedure Q-LEARNING(π)
  $Q \leftarrow Q_0$ ▷ initialization e.g, $Q_0 = 0$
  for $t \leftarrow 0$ to T **do**
    $s \leftarrow SELECT\_STATE()$
    for each step of epoch t **do**
      $a \leftarrow SELECTACTION(\pi, s)$ ▷ Policy π from ϵ-greedy
      $r' \leftarrow REWARD(s, a)$
      $s' \leftarrow NEXTSTATE(s, a)$
      $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r' + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
      $s \leftarrow s'$
  return Q

III. Deep Reinforcement Learning

In real world scenarios, the number of states are very large (infinite) which prevents us from representing the state-action value function as a matrix. This is the case with the atari games too. The number of states (configuration of the screen of the game) are infinite and hence, we represent our state-value function using a convolution neural network (instead of a matrix), which takes state-information (screen) as input and outputs scores representing the expected reward corresponding to each action. The optimal action is the one corresponding to maximum score.

So, our optimal state-action value function will be given by

$$Q^*(s, a; \theta^*) = E\big[r(s, a); \theta^*\big] + \gamma \sum_{s' \in S} Pr\big[s^{'}|s, a; \theta^*\big] V^*(s^{'}) \tag{8}$$

where,
θ^* represents the learnt parameters of our neural network. Also, the optimal policy is given by

$$\forall s \in S, \;\; \pi^*(s) = argmax_{a \in A} Q^*(s, a; \theta^*) \tag{9}$$

IV. Generic Game Playing Agent using Deep-Reinforcement Learning

A. Model

Our model is a convolution neural network which takes last 4 preprocessed frames of the screen as input (84x84x4), applies 32 convolution filters with kernel size (8,8) and stride (4,4) and rectifier non-linearity to get first hidden layers of neurons which is further convolved using 64 filters with kernel size (4,4) and stride (2,2) and rectifier non-linearity to get the second hidden layer of neurons which is further convolved using 64 filters with kernel size (3,3) and stride (1,1) and rectifier non-linearity to get third hidden layer of neurons which is fully connected to a layer of 512 neurons with rectifier non-linearity which is again fully connected to output layer of size equal to number of actions in the minimum legal action

set. Each output unit represents expected reward obtained by the agent if it performs the action corresponding to that output unit. Here, the preprocessing involves conversion to grayscale and taking the maximum of the value of pixel in current frame and the last frame to avoid flickering effect in the games where some objects occur in even or odd frames only.

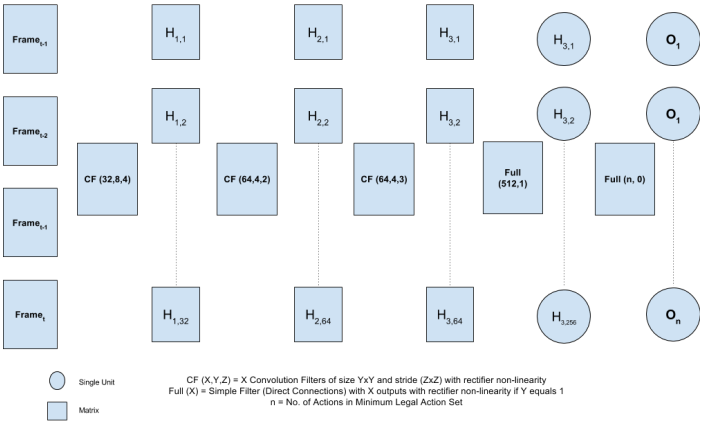

Fig. 3. Convolution Neural Network

B. Training Details

The parameter ϵ in our model simulates the exploration vs exploitation tradeoff. A value of 1 for ϵ implies that the agent will make random actions hence explore the environment and a value of 0 for ϵ implies that the agent will chooses actions based on the model only and hence exploits the acquired information. While training, we decrease ϵ with some appropriate decay rate. So, our agent initially starts by making random actions and collects the information in the form of following tuple: (S_t, a_t, r_t, S_{t+1}, g_t) where S_t denotes the last four states including the current state, a_t denotes the action taken in current state, r_t denotes the reward received, S_{t+1} denotes the last four frames including the next state that the agent got into and g_t is a boolean denoting whether the game is over or not. Also, the parameters of the convolution neural network that acts as the brain of the agent is initialized with random values. After making a move, the agent selects a random batch of information from the collected information and makes a gradient descent step on the euclidean loss between targets and predictions with respect to network parameters θ where the inputs and targets to convolution neural network are described below:

Let $j = 1, , m$ represents the indices of the examples in the sampled batch. Then, input is:

$$I_j = S_t^j \tag{10}$$

and target is:

$$y^j = \begin{cases} r_t^j & if g_t^j \text{is true} \\ r_t^j + \gamma max_{a'} Q(S_{t+1}^j, a_t^j; \theta) & \text{otherwise} \end{cases} \tag{11}$$

The discount value, γ is set to 0.99. We took a batch size of 32, initial ϵ value of 1, and final ϵ value of 0.1. While choosing an action, a uniform random number is generated between 0 and 1. If the generated number is less than ϵ then a random action is taken otherwise an action prdicted by the state-action value function represented by CNN is taken. We used the rmsprop[3] version of gradient descent for updating the parameters. A learning rate of 0.00025, combination coefficient of value 0.95, momentum of value 0.95 and minimum squared gradient of value 0.01 were used in rmsprop.

V. Results

Fig. 4 shows the epoch number versus the total reward and mean Q-value received by the agent in the atari game *Breakout* during testing. Each test epoch comprises of a fixed number of frames to make the rewards in different epoch comparable. The increase in total reward per epoch shows that our agent learnt to play the game. Fig. 5 shows the learning curve for the game *Breakout*. As expected, the learning curve doesn't give much insight on whether the agent has learnt to play in an optimal manner while the mean Q-value per epoch does. Then, Fig. 6 shows a sequence of frames while the agent has learnt to play the game.

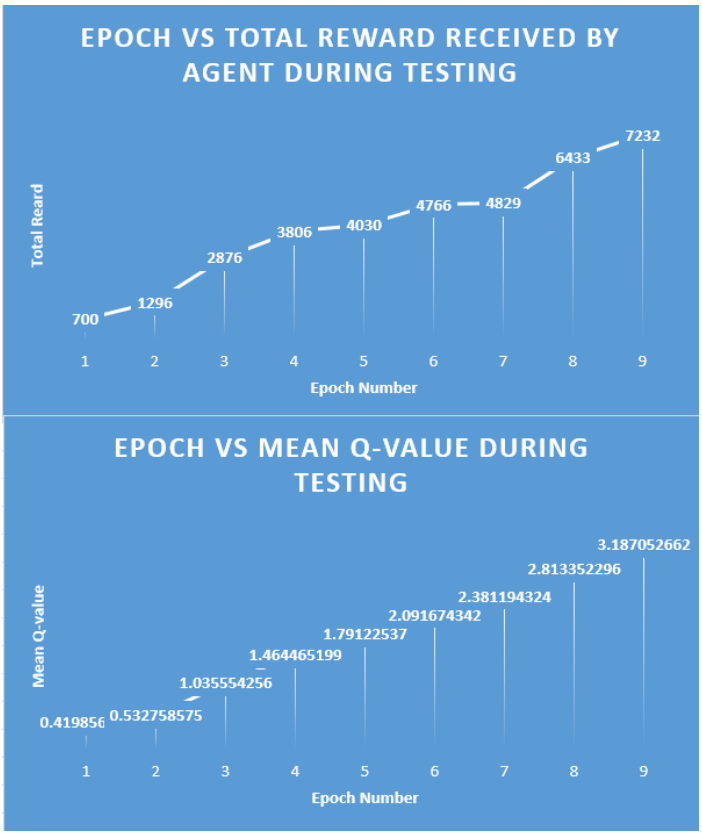

Fig. 4. Epoch number versus the total reward and mean Q-value received by the agent in the game *Breakout* during testing

VI. Conclusion

We conclude that our model resembles the way human beings learn to play or act in an environment in the sense that when a positive reward is provided corresponding to an action of an agent in a particular state, then the agent is biased towards making that action in that particular state so as to increase his score by receiving a positive reward. Similarly, the agent avoids to make those actions which results in a

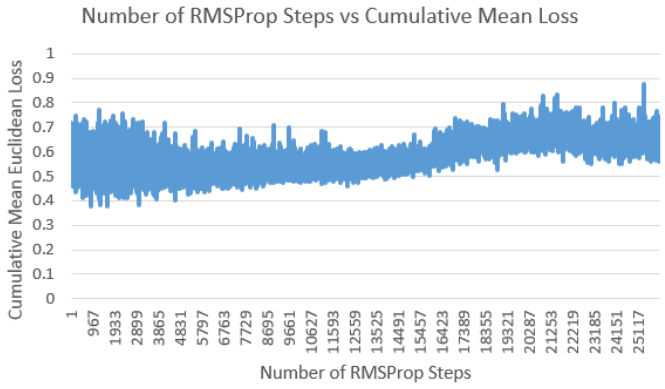

Fig. 5. Learning curve for the game *Breakout*

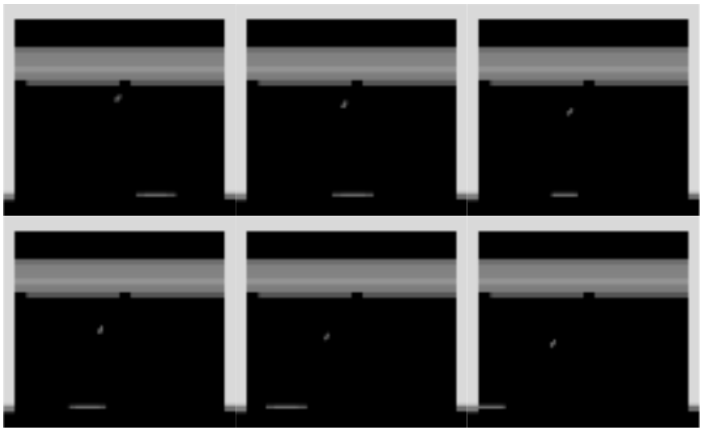

Fig. 6. A sequence of frames while the agent has learnt to play the game

negative reward. When the environment is unknown, then a human will try to explore the environment by making random actions from a set of legal actions and this same behaviour is exhibited by our model by making use of the parameter ϵ as explained in the training details. One can say that the learnt set of parameters represents information relevant for playing the game with respect to which the model is trained. But, there is no way of expressing this information which is in contrast to the way humans learn. We elaborate on this problem in the next section.

VII. Future Work

Almost all existing machine learning models ignore the availability of (potentially very large) memory. Recently, Reasoning, Attention and Memory (RAM) based models have come into picture where the traditional machine learning models like neural networks are coupled with an external memory component. In RAM-based models, the neural network interacts with this external memory component using an attention mechanism to make inference with reasoning where the reasoning component follows from the way of interaction. We aim to propose an extension of the above model using the ideas of RAM-based models like Neural Turing Machine [4] [5], Memory Networks [6] and End to End Memory Networks [7], so that our complete model/agent gives a reasoning on the action that it makes. For instance, consider the game *Breakout* and the sequence of frames observed by our model in Fig. 6.

Now, our current model/agent moves to the left. But why? A human agent might answer that since the ball is moving towards left the action taken is to move left. This is the part that our current model doesn't answer. It has just learnt some set of parameters that once fitted in the neural network predicts optimal action to take. But hopefully the RAM-based agent will be able to reason that "since the ball is moving towards left, I should also move towards left to prevent death".

VIII. Progress of Future Work

We have implemented the Neural Turing Machine and the End to End Memory Networks in other projects (CS565 and Thesis Project). We are at the stage of designing the extended version of our agent by combining the ideas from Deep Reinforcement Learning, Neural Turing Machine and End to End Memory Networks.

Appendix A
Proofs

A. Proof for Bellman equations

The policy value $V_\pi(s)$ obey the following system of linear equations (Bellman equations):

$$\forall s \in S,\ \ V_\pi(s) = E\big[r(s,\pi(s))\big] + \gamma \sum_{s'} Pr\big[s'|s,\pi(s)\big] V_\pi(s') \tag{12}$$

Proof

$$V_\pi(s) = E\Big[\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+\tau}, \pi(s_{t+\tau}))|s_t = s\Big] \tag{13}$$

$$V_\pi(s) = E\big[r(s,\pi(s,\pi(s))\big] + \gamma E\Big[\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+1+\tau}, \pi(s_{t+1+\tau}))|s_t = s\Big] \tag{14}$$

$$V_\pi(s) = E\big[r(s,\pi(s,\pi(s))\big] + \gamma E\big[V_\pi(\delta(s,\pi(s)))\big] \tag{15}$$

The second term in Eqn. 15 is the second term of Eqn. 12

B. Proof of a unique solution of Bellman equation for a finite MDP

For a finite MDP, Bellman equation admits a unique solution given by

$$\mathbf{V}_0 = (\mathbf{I} - \gamma \mathbf{P})^{-1} \mathbf{R} \tag{16}$$

Proof One just needs to show that $(\mathbf{I} - \gamma \mathbf{P})$ is invertible and the remaining part is trivial. Note that the infinite norm of P is:

$$\|\mathbf{P}\|_\infty = \max_s \sum_{s'} |P_{ss'}| = \max_s \sum_{s'} Pr\Big[s^{'}|s,\pi(s)\Big] = 1 \tag{17}$$

This implies that $\|\gamma \mathbf{P}\|_\infty = \lambda < 1$. The Eigenvalues of $\mathbf{P}$ are all less than 1 and $(\mathbf{I} - \gamma \mathbf{P})$ is invertible.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[2] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012.

[3] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[4] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014. [Online]. Available: http://arxiv.org/abs/1410.5401

[5] W. Zaremba and I. Sutskever, "Reinforcement learning neural turing machines," *CoRR*, vol. abs/1505.00521, 2015. [Online]. Available: http://arxiv.org/abs/1505.00521

[6] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *CoRR*, vol. abs/1410.3916, 2014. [Online]. Available: http://arxiv.org/abs/1410.3916

[7] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "Weakly supervised memory networks," *CoRR*, vol. abs/1503.08895, 2015. [Online]. Available: http://arxiv.org/abs/1503.08895