International Journal of Scientific Research & Engineering Trends
Volume 5, Issue 3, May-June-2019, ISSN (Online): 2395-566X

An Overview of Deep Q Learning

Bhavesh Dhera bhavesh.dhera@gmail.com
M. Mani Teja murarishettymani@gmail.com
G. Yashwanth Reddy
Dept. of Computer Science & Engg.
Gokaraju Rangaraju Institute of Engineering &Technology
Hyderabad, India

***Abstract*- This paper gives a bird's eye view of Reinforcement Learning, an area under Artificial Intelligence. It starts with the history of reinforcement learning and its traditional models then goes on to delve deeper into one of the modern algorithm called Deep Q-Learning. It is aimed at discussing the intuition behind Deep Q-Learning and ways to implement it. Also discussed are various aspects to overcome drawbacks of Q Learning.**

***Keywords* – Reinforcement Learning, Q-Learning, Deep Q-learning, Deep Q Network.**

I. INTRODUCTION

1. Reinforcement Learning

Reinforcement learning is learning what to do--how to map situations to tell which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. This is called the delayed reward and is one of two important characteristics of Reinforcement Learning (The other being trial-and-error search).

Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in machine learning, statistical pattern recognition, and artificial neural networks. Supervised learning is learning from examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. One of the challenges that arise in reinforcement learning and not in other kinds of learning is the trade-off between exploration and exploitation.

To obtain a lot of rewards, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing rewards. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain a reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task.

The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. A number of factors are involved in the dataflow and decision process of Reinforcement Learning. These help decide what action to take in a particular state. A few are discussed below.

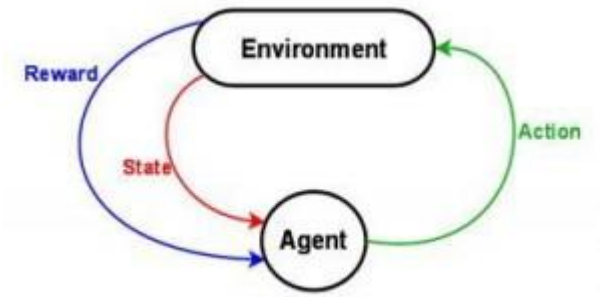

Fig.1 Reinforcement Learning

1.1 Input- The input should be an initial state from which the model will start
1.2 Output- There are many possible output as there are variety of solution to a particular problem
1.3 Training- The training is based upon the input, the model will return a state and the user will decide to reward or punish the model based on its output.
The model keeps continues to learn. The best solution is decided based on the maximum reward. The 'deep' in deep reinforcement learning means it is a consequence of applying deep learning to reinforcement learning. The model uses Q values to determine the best action.

In traditional learning this is based on the Policy and in Q-learning is based on bellman equation whence the data is extracted from the Q-table. In deep reinforcement learning or deep Q-learning we use a neural network to approximate the Q-values and decide future actions so as to maximize reward. Deep Q-Learning is known to fare better than policy gradients or the Q-Table in case of the presence of a large number of states.

2. Motivation

Deep Learning agents are still unresolved because what is happening in the network isn't completely clear to

many scholars. There are researchers working in this field to discover and solve real-life problems using Reinforcement Learning. Solving Game AI's using Deep Learning can be a futuristic work as we expect the agent to learn as humans do in a trial-and-error and reward-penalty basis. Playing with a bot that has been highly trained on a particular game can make it really challenging for the human player and also help in developing new insights on how to play the game better.

3. History on Game Bots

Most of the game bots in the past were hard-coded, i.e., most of the conditions are given and the bots act according to them. Such bots are known as Scripted Bots. Later, IBM's Deep Blue computer defeated Garry Kasparov, a Grandmaster in 1997. From 2007, modern Gaming introduces more technical bots which can react according to realistic markers, such as sound made by a player and the footprints they leave behind.

Nintendo started a competition called "Mario AI Championship" in which the players had to create a game bot that can play the Mario game. The bots were built using Neural Networks and cannot see beyond the computer screen. In 2014, Google acquired 'Deep mind', an AI-based company that uses reinforcement learning to make a bot that is not pre-programmed but can play arcade games like Space Invaders and Breakout.

4. Observations

If we ever want to do better than take random actions at each step, it'd probably be good to actually know what our actions are doing to the environment. The environment's `step` function returns exactly what we need. In fact, `step` returns four values. These are:

4.1 Observation (object**)-** an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.

4.2 Reward (float)- amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.

4.3 Done (**Boolean**)- whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)

4.4 Info (dict) - diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change). However, official evaluations of your agent are not allowed to use this for learning. This is just an implementation of the classic "agent-environment loop". Each time step, the agent chooses an action, and the environment returns an observation.

II. PREVIOUS WORK

Reinforcement Learning is situated in between supervised learning and unsupervised learning. It deals with learning in sequential decision-making problems where the feedback is limited. The previous model used has been a Q-learning algorithm to facilitate reinforcement learning.

1. Q-learning

In Q-learning we define a function Q(s,a) representing the discounted future reward when we perform action a in state s, and continue optimally from that point on.

$$Q(st, at) = max\pi Rt + 1 \qquad (1)$$

The way to think about Q(s,a) is that it is the best possible score at the end of game after performing action a in state s. It is called Q-function, because it represents the quality of certain action in given state. We can't estimate the score at the end of the game. But as a theoretical construct we can assume existence of such a function.

$$\pi(s) = \text{argmax}_a Q(s,a) \qquad (2)$$

Here π represents the policy, the rule how we choose an action in each state. Just like with discounted future rewards we can express Q-value of state s and action a in terms of Q-value of next state s′.

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a') \qquad (3)$$

This is called the Bellman equation**.** If you think about it, it is quite logical – maximum future reward for this state and action is the immediate reward plus maximum future reward for the next state. The main idea in Q-learning is that we can iteratively approximate the Q-function using the Bellman equation**.** In the simplest case the Q-function is implemented as a table, with states as rows and actions as columns.

III. DEEP Q LEARNING

As discussed in the previous section, we use Bellman equation to approximate Q value of an action. Since neural networks are known to perform universal approximations, we used a neural network to approximate the Q value of a future action. But there are other things involved in using a neural network.

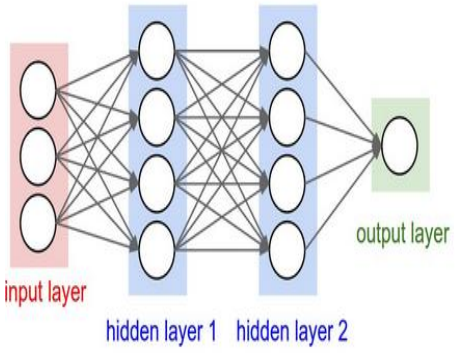

Fig.2 Neural Network to approximate Q value

1. Experience Replay

Deep Q-Networks tend to forget what they've learned. Experience Replay is a technique to overcome this issue. We sample a random batch of experiences from the large table and train the model on it again. To perform experience replay we store the agent's experiences. This means instead of running Q-learning on state/action pairs as they occur during simulation or actual experience, the system stores the data discovered for [state, action, reward, next, state] - typically in a large table. Note this does not store associated values - this is the raw data to feed into action-value calculations later. The learning phase is then logically separate from gaining experience, and based on taking random samples from this table. You still want to interleave the two processes - acting and learning - because improving the policy will lead to different behavior that should explore actions closer to optimal ones, and you want to learn from those. However, you can split this how you like - e.g. take one step, learn from three random prior steps etc.

The Q-Learning targets when using experience replay use the same targets as the online version, so there is no new formula for that. The loss formula given is also the one you would use for DQN without experience replay. The difference is only *which* s, a, r, s', a' you feed into it. In DQN, the Deep Mind team also maintained two networks and switched which one was learning and which one feeding in current action-value estimates as bootstraps. This helped with stability of the algorithm when using a non-linear function approximate.

2. Discounted Future Reward

To perform well in long-term, we need to take into account not only the immediate rewards, but also the future awards we are going to get. Given one run of Markov decision process, we can easily calculate the total reward **for** one episode:

$$R = r1 + r2 + r3 + \cdots + rn \quad (4)$$

Given that, th**e** total future reward **from** time point *t* onward can be expressed as:

$$Rt = rt + rt + 1 + rt + 2 + \cdots + rn \quad (5)$$

But because our environment is stochastic, we can never be sure, if we will get the same rewards the next time we perform the same actions. The more into the future we go, the more it may diverge. For that reason it is common to use discounted future reward instead:

$$Rt = rt + \gamma rt + 1 + \gamma 2rt + 2 \ldots + \gamma n - trn \quad (6)$$

Here γ is the discount factor between 0 and 1 – the more into the future the reward is, the less we take it into consideration. It is easy to see, that discounted future reward at time step *t* can be expressed in terms of the same thing at time step *t*+1:

$$Rt = rt + \gamma(rt + 1 + \gamma(rt + 2 + \cdots)) \quad (7)$$

If we set the discount factor γ=0, then our strategy will be short-sighted and we rely only on the immediate rewards. If we want to balance between immediate and future rewards, we should set discount factor to something like γ=0.9. If our environment is deterministic and the same actions always result in same rewards, then we can set discount factor γ=1. A good strategy for an agent would be to always choose an action, that maximizes the discounted future reward.

IV. CONCLUSION

We have described the Game bots project as an infrastructure for multi-agent systems research that supports different platforms and is widely available. Reinforcement Learning is a widely used real-world problem-solving technique and a Deep Learning based game bot can solve any Atari and Flash game without having any prior information about the game environment. It only needs to be fed the raw pixels of the game and the game controls.A well-trained and well-designed game bot can outperform human gamers.

REFERENCES

[1] Gal A. Kaminka, Manuela M. Veloso, Steve Schaffer, Chris Sollitto, Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, and Sheila Tejada. "GameBots: A Flexible Test Bed for Multiagent Team Research"Communications of the ACM, 45(1):43–45, January 2002.

[2] "Demystifying Deep Reinforcement Learning"(https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/)

[3] Andrej Karpathy's blog (http://karpathy.github.io/2016/05/31/rl/)

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

[5] Richard Sutton and Andrew Barto."Reinforcement Learning: An Introduction." MIT Press,1998.

[6] Andrew Moore and Chris Atkeson. "Prioritized sweeping: Reinforcement learning with lessdata and less real time." Machine Learning, 13:103–130, 1993..

[7] Marc Belle mare, Joel Veness, and Michael Bowling. "Sketch-based linear value function approximation." In Advances in Neural Information Processing Systems 25, pages 2222–2230,2012.