Deep Q Learning in Stabilization of Inverted Pendulum

S. Suganthi Amudhan, Bhavin Sedani, Ved Vyas Dwivedi

Abstract*: Reinforcement Learning is a concept in which an agent takes actions in an environment to maximize its aggregate reward. Agents in Reinforcement Learning algorithms are penalized for bad actions and rewarded for good ones. Deep Reinforcement learning is used to overcome some shortcomings in Reinforcement learning like lack of variety in the open source collection and change in task's difficulty based on reward or set of actions. In a larger environment, inference of new states from already explored states is a difficult task due to its time & space complexity. Hence the Q-value function which represents the quality value is approximated by Deep Q-learning that uses a neural network for the same. The results of implementing Deep Q Learning (DQN), Double DQN, Dueling DQN, Noisy DQN & DQN Prioritized Experience Replay techniques & their performance in stabilization of inverted pendulum are highlighted in this paper. Deep Reinforcement learning can be applied in various platform like for recognition, perception in computer vision, for simulation to real robot control in robotics, for sequence generation, translation in Natural Language processing, for Poker, Bridge StarCraft in games, for pricing, trading risk management in finance, for e- commerce, customer management in business management, for diagnosis using Electronic Medical Records in healthcare and in adaptive decision control.

Keywords *: Deep Q Learning, Inverted Pendulum, OpenAI gym, Reinforcement learning, Deep Reinforcement Learning*

I. INTRODUCTION

Reinforcement Learning helps the environment in finding where an action is required and selecting action with the highest reward along with the best method for attaining the same. In some conditions, usage of reinforcement learning model can be avoided where enough data is available to solve using supervised learning method and in large action spaces.

The important characteristics of reinforcement learning are that there is no supervisor, only reward signal is used. Sequential decision making is used. Performance time is vital parameter to judge the efficiency. Feedback is always delayed and actions will supply successive data.

Challenges of Reinforcement Learning

- Proper reward design should be involved.
- Speed of learning depends on the number of parameters.
- Observability of Realistic environments is limited.
- Overloading of states may weaken the results.

Reinforcement Learning generally elaborates on actor-critic or Q learning. Actor-critic learning involves estimate of long term usage of each state & choosing optimal action of each state. Both these techniques can be applied to all adaptive optimal control problems but Q learning is simpler & converges faster. Applications include robotics for industrial automation, machine learning, data processing, business strategy planning, creating custom based training material to students etc.

Q learning has two problems: Firstly, as the number of states increases, it would also increase the amount of memory to save and update the table Secondly, the amount of time taken for exploration and achieving Q-table would be very large. Here comes the necessity of approximating these Q with neural networks. This leads to Deep Q Learning techniques and their merits.

Deep Q learning and their advanced techniques like Deep Q Learning DQN, Double DQN, Dueling DQN, Noisy DQN & DQN Prioritized Experience Replay techniques has been applied to popular control problem of an unstable inverted pendulum and compared.

II. REINFORCEMENT LEARNING

Reinforcement learning improves action by learning from its mistakes to attain the desired performance of an agent [1]. An initial environment has a current state (S_t) where best action (A_t) is executed. A reward (R_t+1) is returned for a given action along with a new state (S_t+1) and the process is repeated until termination as in Fig.1.

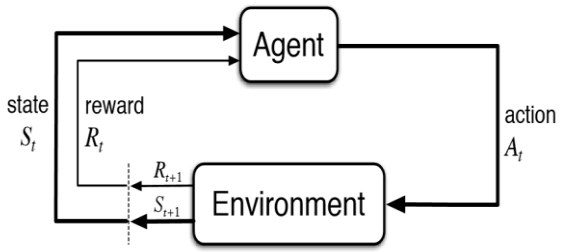

Fig.1.Reinforcement Learning
(source:https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning)

An environment along with an agent comprising observation space and action space are created. After every run, new environment & new state is set. An action is got from an agent on the basis of the current state until termination condition is satisfied. Then a new state is received along with the associated reward. The reward comes from the environment and that defines whether lower or higher reward is better. Then it remembers the state, action, reward, next state, terminal state and perform Experience Replay.

There are three approaches to implement a Reinforcement Learning algorithm.

Revised Manuscript Received on December 05, 2019.

***** Correspondence Author

S.Suganthi Amudhan*, Electronics and Communication Engineering, C.U.Shah University, Surendranagar, Gujarat, India.

Dr.Bhavin Sedani, Electronics and Communication Engineering, L.D.College of Engineering, Ahmedabad, Gujarat, India.

Dr.Ved Vyas Dwivedi, Pro Vice Chancellor, C U Shah University Wadhwan City.Surendranagar Gujarat, India.

1) Value-Based:

Maximising a value function **V(s) in** expectation of a long-term return of the current states under policy π.

2) Policy-based:

In a policy-based RL method, action of every state depends on maximum reward in future. Two types of policy-based methods are: Deterministic where same action is got and stochastic in which policy is based on certain probability.

3) Model-Based:

In this Reinforcement Learning method, virtual model is created for that particular environment.

There are two types of Reinforcement:

1. The strength & the frequency of behaviour is increased then Positive Reinforcement occurs. This improves performance but at the same time too much of positive reinforcement may also cause overload of states which may affect the results.
2. The strength of the behaviour occurs due to stopping a condition in Negative reinforcement learning. This increases behaviour but provides only to meet the minimum behaviour

RL is very suitable to system where the model of an environment is known but analytic solution is not present, only simulation model known and in places where only by interaction information can be collected.

Reinforcement learning (RL) is mainly involved in decision making & adaptive control in various machine learning techniques [2]. The agents help in learning & perform even in a complex and indeterminate environment and applied widely for two reasons:

(I) RL is applied in all problems for making a sequence of decisions like example of controlling a robot's motors, making business decisions relating to pricing and inventory management, or playing video games . RL can even be applied to supervised learning problems.

(II) RL algorithms achieve notable results in many complex environments. It relied on problem specific engineering until recent advances in deep learning.

RL implementation is also decelerated due to:

(I) Lack of variety in open source collections of RL environments.

(II) The reward or the set of actions, can change a task's difficulty.

III. DEEP Q LEARNING

Deep Q learning is a Reinforcement Learning technique used in selecting the best action for a given environment using DQNSolver. Each action has its associated Q value, where 'Q' stands for a quality of a given move. To get accurate Q values deep neural networks and linear algebra are to be used.

For each state experienced by the agent, it has to remember it and perform an experience replay. The memory contains tuples of state, next state, action, reward and a Boolean value for indicating the termination of the agent [3]. This keeps on going and the agent memorizes the information until the termination happens as in Fig.2.

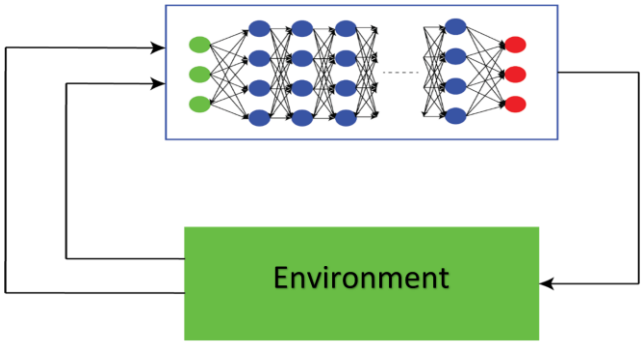

Fig.2.Deep Reinforcement Learning

(source:https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning)

The steps involved in reinforcement learning using deep Q-learning networks (DQNs) are that initially all the past experience is stored by the user in memory, then next action is found using the maximum Q. The loss function is mean square error of predicted & target Q value. This leads to predicting its own value, but since R is the unbiased true reward, it updates its gradient using backpropagation until convergence.

Experience replay is performed to update Q value which predicts a best next state and reduces correlation between successive actions taking past experiences from the memory. The new Q is calculated by multiplying the best Q for a given action with the discount factor (Gamma) and adding it to the current state reward. Gamma is between zero and one, normally 0.9 is the discount factor which tells an agent to not rely too much on the next state. Here Q value is updated with the cumulative discounted future rewards.

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha).Q(s_t,a_t) + \alpha.(r_{t+\gamma}.\max Q_{(s_{t+1},a)}) \quad (1)$$

This helps in possible convergence and tries to predict its own output in simple cases like an inverted pendulum stabilisation problem. Double DQN's or Dueling DQN's techniques are used in complex problems where more advanced techniques are needed to converge [4].

In DQN the agent's brain is a deep neural network. The input of the neural network will be the state and the number of the actions that an agent can take will decide the output number of neurons. For training a neural network the targets would be the Q values of each of the actions and the input would be the state that the agent is in.

A. Double DQN

Double DQN uses two identical neural network models. One model learns during the experience replay and the other one is a replica of the last episode of the first model. In DQN, Q is calculated with the reward added to the next state maximum Q value. If every time the Q value is a high number for a certain state, which will become higher every time. It may happen that for state s action a is always higher than action b, then action a will get chosen every time for state s. If for some memory experience action b becomes the better action for state s, then as the neural network is trained to give a much higher value for action a, it is difficult to train the network to understand that action b is the

better action in certain conditions. So use of a secondary model which is the replica of the main model from the last episode and since the difference between the values of the second model are lower than the main model, we use this second model to attain the Q value:

$R_t + \gamma\, Q^c\,(s_{t+1}.\text{argmax}\ Q\ (s_{t+1}.a'))$ (2)

The index of the highest Q value from the main model is used to obtain the action from the second model [5].

B. Dueling DQN

The difference in Dueling DQN is in the structure of the model. The model is created in a way to output the formula

$Q(s.a)=V(s)+A(s.a)$ (3)

V(s) stands for the Value of state s and A is the Advantage of action a. The Value of a state is independent of action s. If an agent might have the same Q value in each of the actions, there would be no good action derived. If every action has the same result, then the Advantage of each action will have the same value. Now, if we subtract the mean of all the Advantages from each advantage, we get zero (or close to zero) and Q value would actually be the Value that the state has.

$Q(s.a)=V(s)+A(s.a)-1/n\sum A(s.a')$ (4)

So Q value would not overshoot [6]. The states that are independent of action would not have a high Q value. The output of the model would be the state Value plus the Advantage of actions. But for training the model same Q value for targets are used as before:

$Q(s_t.a_t)_= R_t + \gamma \max Q(s_{t+1}.a')$ (5)

C. Noisy Net (Noisy DQN)

The agent sometimes explores the environment regardless of the Q value or neural network. The exploration was a probability that reduced over time which would have a probability of 1 reducing to, for instance, 0.01.Noisy Net gives noise to the output of the neural network and the agent explores the environment whenever there is a noise in the neural network output and different action gets a higher value when the real action to be taken is another one.

The way to do this is by defining different weights for the neural network. The output of a neuron is the sum of the inputs multiplied by the weight of their connection to the corresponding neuron. But in Noisy Net W = Mu + Sigma * epsilon where Mu is a Variable off random initialization, Sigma is a Variable of constant initialization and epsilon is actually noise with a random value between zero and one. So the fully connected layers would have weights like this to learn and over time if exploration is not needed anymore the value of sigma would be close to zero to neutralize the epsilon.

D.DQN with Prioritized Experience Replay

The agent will start taking actions in an environment and memorize the experience as a tuple of state, next state, action, reward and a Boolean value for indicating the termination of the agent. In Experience Replay step, a particular batch would get chosen from the memory and training the neural network on that particular batch. In that case we have to consider the important experience memories for the batch.

If the Q-value from the next state is far different than the Q-value from the current state that means that the importance is high whether the Q-value in the next state increases or decreases. This deference is called Temporal Difference error (TD error) [7].

So for each memory we have TD error as below:

$TD=|Q(s,a)-Q(s+1,a)|$ (6)

Then the probability of an experience memory being chosen would be:

$p_i=(TD_i+\varepsilon)^a/\sum_k^{\text{memory size}} (TD_k+\varepsilon)^a$ (7)

Now, here epsilon is a small value to prevent division by zero and alpha is a random value between zero and one. So p will be the probability that an experience is important and the batch considers experience probabilities. Here training the network happens stochastically. So, if an experience has a high probability, the neural network would over fit on this particular experience

This problem can be solved by multiplying the value below to the training loss:

$\text{Importance}= (1/p_i*1/\text{memory size})^b$ (8)

Where b is a value starting from zero and gradually reaching to 1. So in this formula the importance is calculated from the distribution that the experience came from. So if a probability is high it won't get chosen all the time. Therefore, the training loss will be calculated as:

$J=1/m\sum(y-y_{hat})^2 * \text{Importance}$ (9)

This following algorithm is the pseudo code for Deep-Q Learning with Experience Replay.

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise state s_t
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Set $s_{t+1} = s_t$
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$
  end for
end for

(source:https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning)

The pseudo algorithm can be summarised as which incorporates T states with some probability epsilon to select random action or otherwise select the maximum value generated in the state s_t to execute action and observe reward r_t. Then storing this change in the memory and replacing s_{t+1} with s_t and based on the transitions, setting of output y_j is done. In brief it can be seen that temporal differences of action-value Q(s,a) are calculated and used to update Q(s,a) until Q(s,a) has converged using gradient descent.

IV. APPLICATION OF INVERTED PENDULUM CONTROL USING DQN

A pole is attached to a cart, which is controlled by applying a force to the cart [8]. The pendulum starts upright, and the goal is to stabilize the pole in moving cart as in Fig.3.

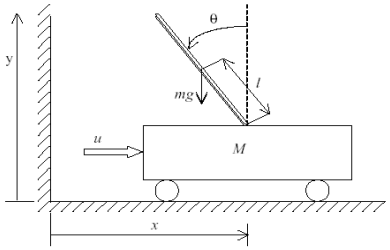

Fig.3.Inverted Pendulum

The equations of motion for the system are derived using the following parameters

M is the mass of the cart and *m* the mass of the pole,
l the distance from the pivot to the center of mass of the pole
I the moment of inertia of the pole about the pivot,
x the cart's position
θ the angle the pole makes with the vertical and
u the horizontal control force imparted to the cart.

A reward is provided for every run when the pole remains upright. An assumption that the action stops when the pole is oscillating more than 24 degrees from vertical on either direction, or the cart moves more than 2.4 units from the centre.

The assumptions are applied in OpenAI gym environment and were able to achieve stabilization of pole in a moving cart efficiently. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the agent, and is compatible with TensorFlow or Theano [9]. The gym library is mainly used in testing reinforcement learning algorithms in different environment. It also allows writing of algorithms. This solved the problem with an average reward of 195 in 100 trails as shown in Fig.4. In similar fashion the advanced techniques of DQN are also explored with different number of trials & plotted against their respective scores as in Fig.5.

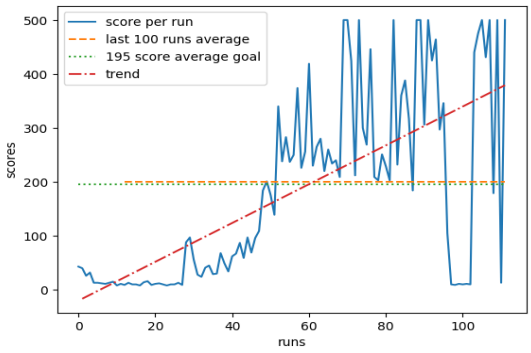

Fig.4.Learning performance of DQN for inverted pendulum

(source:https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning)

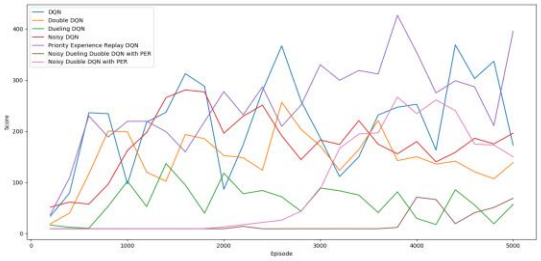

Fig.5.Learning Performance of advanced DQN Methods for inverted pendulum

(source: https://medium.com/@parsa_h_m)

V. CONCLUSION

It is seen that the problems associated with Reinforcement learning are that Q learning increases the memory storage & time for exploration which led to the path of Deep Q learning. Clearly it is understood that DQN is a Reinforcement Learning technique that selects the best action in the given environment. Q value is associated with each and every action, where 'Q' stands for the quality of it.

This clearly gives an insight that for complex problems, some advanced methods like Deep RL techniques, such as Double DQN Networks, Dueling DQN. Noisy DQN and DQN Prioritized Experience replay are used which further mends the learning process. These also provide better performance using even lesser number of runs when compared with traditional algorithms.

It can also be inferred that in Deep learning, targets does not change, so the training process is stable whereas the targets are continuously changing in Deep Reinforcement learning and hence have complex algorithms as there is need for continuous exploration.

These conclusion were drawn easily with all the Deep Q learning techniques being tested for the practical application of an inverted pendulum pole stabilization successfully.

REFERENCES

1. Sutton, Richard S., and Andrew G. Barto. Introduction to reinforcement learning. Vol. 2, no. 4. Cambridge: MIT press, 1998.
2. Sutton, Richard S., Andrew G. Barto, and Ronald J. Williams. "Reinforcement learning is direct adaptive optimal control." *IEEE Control Systems Magazine* 12, no. 2 (1992): 19-22.
3. William Grant Hatcher, Wei Yu, "A Survey of Deep Learning: Platforms Applications and Emerging Research Trends", Access IEEE, vol. 6, pp. 24411-24432, 2018.
4. Sewak, Mohit. "Deep Q Network (DQN), Double DQN, and Dueling DQN." In Deep Reinforcement Learning, pp. 95-108. Springer, Singapore, 2019.
5. Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In Thirtieth AAAI conference on artificial intelligence. 2016.
6. Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015).
7. Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay." arXiv preprint arXiv:1511.05952 (2015).
8. Chao, Yue, Liu Yongxin, and Wang Linglin. "Design of Reinforcement Learning Algorithm for Single Inverted Pendulum Swing Control." In 2018 Chinese Automation

Congress (CAC), pp. 1558-1562. IEEE, 2018
9. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.;Tang, J.; and Zaremba, W. 2016. OpenAI gym. arXiv preprint arXiv:1606.01540

AUTHORS PROFILE

Prof. S. Suganthi Amudhan is working as a Assistant Professor in Electronics and Communication Department at Babaria Institute of Technology, Vadodara. She has a teaching experience of 15 years. She is currently a research scholar of C.U Shah University, Surendranagar. She has received best paper presentation award once.

Dr. Bhavin Sedani is working as a Professor in Electronics and Communication Department at L.D. College of Engineering (Government Engineering College),Ahmedabad. He has teaching experience of 17 years. He has presented more than 46 research papers in various international and national conferences. His 28 research papers are published in various international journals and Scopus indexed IEEE Xplore digital library. He has achieved best paper presentation awards 7 times for his research articles.

Dr.Ved Vyas Dwivedi is working as Pro Vice Chancellor at C U Shah University, Wadhwan City, Surendranagar Gujarat, Indi. He is a Professor in Electronics and Communication Engineering at C.U Shah University and has total 23 years of experience including 5 years of industrial and 18 years of teaching experience. He has published/authored/co-authored several books, research/review articles/papers in refereed international/national journals/conference proceedings, successfully supervised 6 Ph.D. theses (awarded), examined and reviewed many Indian Ph.D. theses. He also has Indian patents; completed research and industry consultancy based projects, delivered many expert/keynotes talks.