Deep Reinforcement Learning: Q-Learning on Custom Environment

Sukru Yasin Bursali
Computer Engineering,
Osmangazi University
Eskisehir, Turkey
152120161048@ogrenci.ogu.edu.tr

***Abstract*—This paper surveys the eld of reinforcement learning from a computer-science perspective. It is written to be accessible to researchers familiar with machine learning. Reinforcement learning is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment.This paper focuses on Q-Learning. Q-Learning is a model-free form of machine learning, in the sense that the AI "agent" does not need to know or have a model of the environment that it will be in. The same algorithm can be used across a variety of environments. In this study, we created custom environment and trained our agent.**

Keywords— Q-Learning, Reinforcement Learning, AI agent

I .Introduction

Our reality contains environments in which we perform numerous actions. Sometimes we get good or positive rewards for some of these actions in order to achieve goals. We strengthen our actions in order to get as many rewards as possible. In that description of how we pursue our goals, we framed for ourselves a representative analogy of reinforcement learning.

The integration of reinforcement learning and neural networks contains a long history (Sutton and Barto, 2018; Bertsekas and Tsitsiklis, 1996; Schmidhuber, 2015). With recent exciting achievements of deep learning (LeCun et al., 2015; Goodfellow et al., 2016), making the most of massive information, powerful computation, new algorithmic techniques, mature code packages and architectures, and robust resource, we've got been witnessing the renaissance of reinforcement learning (Krakovsky, 2016), especially, the mix of deep neural networks and reinforcement learning, i.e., deep reinforcement learning (deep RL).

Deep learning, or deep neural networks, has been prevailing in reinforcement learning within the last many years, in games, robotics, natural language process, etc. we have been witnessing breakthroughs, like deep Q-network (Mnih et al., 2015) and AlphaGo (Silver et al., 2016a); and novel architectures and applications, like differentiable neural computer (Graves et al., 2016), asynchronous strategies (Mnih et al., 2016), dueling network architectures (Wang et al., 2016b), worth iteration networks (Tamar et al., 2016), unsupervised reinforcement and auxiliary learning (Jaderberg et al., 2017; Mirowski et al., 2017), neural design style (Zoph and le, 2017), dual learning for machine translation (He et al., 2016a), spoken dialogue systems (Su et al., 2016b), info extraction (Narasimhan et al., 2016), target-hunting policy search (Levine et al., 2016a), and generative adversarial imitation learning (Ho and Ermon, 2016), etc. power would push the frontiers of deep RL further with respect to core components, mechanisms, and applications.

In our case, we have a player(blue), which aims to navigate its way as quickly as possible to the food(green), while avoiding the enemy(red). We could make this high definition, but we already knew we're going to be breaking it down into observation spaces. Instead, we just started in a discrete space. We studied between 10x10 and 20x20 pixels. We keep it simple because of technical difficulties. Because larger environment ment us larger Q-Table. And this would be in terms of space it takes up in memory as well as time it takes for the model to actually learn. So, our environment will be a 15 x 15 grid, where we have 1 player, 1 enemy, and 1 food. We just had the player able to move, in attempt to reach the food, which will yield a reward.

As we engage in the environment, we used .predict() method to figure out our next move (or move randomly). When we use .predict() method, we will get the 3 float values, which are our Q values that map to actions. Then we used argmax on these, like we would with our Q Table's values. We will then "update" our network by doing a .fit() based on updated Q values. When we do this, we will actually be fitting for all 3 Q values, even though we intend to just "update" one. There have been DQN models in the past that serve as a model per action, so you will have the same number of neural network models as you have actions, and each one is a regressor that outputs a Q value, but this approach isn't really used.

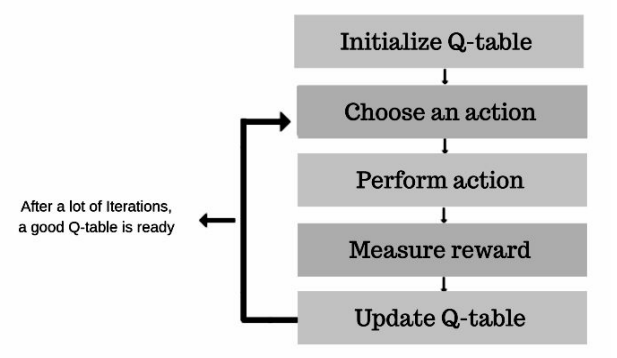

Fig. 1. *Overview Flow of Q-Learning*

II. Background

A. Markov Decision Process

A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

Let us mention formalising this whole process using a concept called a Markov Decision Process or MDP.

Markov Decision Process (MDP) model contains:

- A set of possible world states S
- A set of possible actions A
- A real valued reward function R(s,a)
- A description T of each action's effects in each state

Any random process in which the probability of being in a given state depends only on the previous state, is a markov process. In other words, in the markov decision process setup, the environment's response at time t+1 depends only on the state and action representations at time t, and is independent of whatever happened in the past.

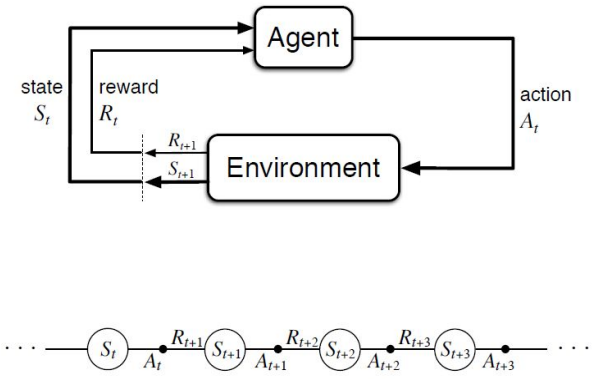

***Fig. 2**. Iteration at each time step*

- St: State of the agent at time t
- At: Action taken by agent at time t
- Rt: Reward obtained at time t

The above diagram clearly illustrates the iteration at each time step wherein the agent receives a reward Rt+1 and ends up in state St+1 based on its action At at a particular state St. The overall goal for the agent is to maximise the cumulative reward it receives in the long run. Total reward at any time instant t is given by:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

where T is the final time step of the episode. In the above equation, we see that all future rewards have equal weight which might not be desirable. That's where an additional concept of discounting comes into the picture. Basically, we define γ as a discounting factor and each reward after the immediate reward is discounted by this factor as follows:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

For discount factor < 1, the rewards further in the future are getting diminished. This can be understood as a tuning parameter which can be changed based on how much one wants to consider the long term (γ close to 1) or short term (γ close to 0).

Can we use the reward function defined at each time step to define how good it is, to be in a given state for a given policy? The value function denoted as v(s) under a policy π represents how good a state is for an agent to be in. In other words, what is the average reward that the agent will get starting from the current state under policy π?

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}$$

E in the above equation represents the expected reward at each state if the agent follows policy π and S represents the set of all possible states. Policy, as discussed earlier, is the mapping of probabilities of taking each possible action at each state (π(a/s)). The policy might also be deterministic when it tells you exactly what to do at each state and does not give probabilities.

Now, it's only intuitive that 'the optimum policy' can be reached if the value function is maximised for each state. This optimal policy is then given by:

$$\pi^* = \arg\max_\pi V^\pi(s) \quad \forall s \in \mathbb{S}$$

B. Deep Q - Learning

The base algorithm for DQN, Q-learning, is value based RL, which is a method that approximates an action value (i.e., a Q-value) in each state. Further, Q-learning is a model-free method such that even if the agent does not have knowledge of the environment, the agent can develop a policy using repeated experience by exploring. In addition, Q-learning is an off-policy algorithm, that is, the action policy for selecting the agent's action is not the same as the update policy for selecting an action on the target value. An algorithm based on Q-learning that approximates the Q-function using DNN is the basis of DQN (Mnih et al., 2013). To prevent DNN from learning only through the experience of a specific situation, experience replay was introduced to sample a general experience batch from memory. Additionally, the DQN algorithm used two separate networks: a Q-network that approximates the Q-function and a target network that approximates the target value needed for the Q-network updated to follow a fixed target (Mnih et al., 2015).

In DQN, a multiple output neural network is commonly adopted as the Q-network structure. In this network structure, the input of the neural network is the state, and the output is the Q-value of each action. Using the above technique, we can approximate the Q-value of all feasible actions by updating this multiple output Q-network in parallel with the experience list. To maintain Q-values of infeasible actions, the current Q-value of an infeasible state-action pair is assigned to the target value of the

Q-network output of the corresponding infeasible action to set a temporal difference error of zero. Furthermore, as in DQN, several experience lists are sampled from replay memory, and the Q-network is updated using the experience list batch. A detailed description of the process for updating the Q-network is shown in Fig. 3

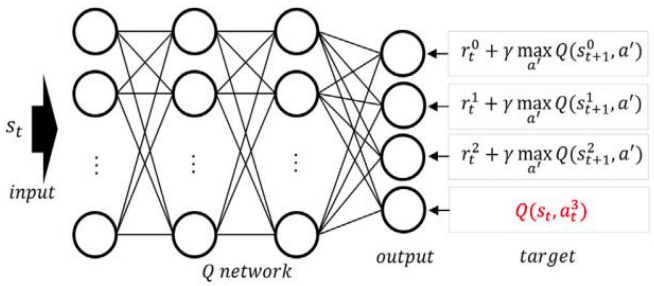

Fig. 3. *Updating a multiple output Q-network*

III. Our Study

A. Environment & Agent Settings

Let us mention our project. The most convenient for our goal was creating our environment. We examined various gym environments and made simple environment 10x10 pixels. Our agent(blue one) are moving with 9 options. These options are 8 ways to move.(right,left,up,down etc.) Every move is effecting score by -5 and if agent goes to the enemy(red dot) score affected by -300. Our action space size is 9. Note that we trained our model with various environment parameters, these values are not exact on every model.

Our coordinate system is based on relative position of green and blue dot. The model is just trained to move based on relative position deltas. One thing this model isn't prepared for, however, if the enemy is close, that in theory, the enemy and player could move on to the same tile. In all of training, the player could move all around the enemy safely. With movement on, this is not the case, so train with movement on is more logical thing to do, and seeing some new capabilities of the player emerge is possible.

The discount variable is a measure of how much we want to care about future reward rather than immediate reward. Typically, this value will be fairly high, and is between 0 and 1. We wanted it high because the purpose of Q Learning is indeed to learn a chain of events that ends with a positive outcome, so it's only natural that we put greater importance on long terms gains rather than short term ones.

The max_future_q is grabbed after we've performed our action already, and then we update our previous values based partially on the next-step's best Q value. Over time, once we've reached the objective once, this "reward" value gets slowly back-propagated, one step at a time, per episode.

B. CNN model & Hyperparameters

CNN model takes observation space values as input. CNN knows where food, agent and enemy is. The DQN neural network model is a regression model, which typically will output values for each of our possible actions. These values will be continuous float values, and they are directly our Q values. We created our model with 1 input, 1 hidden and 1 output layers. For first two layers, we added relu as activation and applied 0.2 dropout. In last layer, our activation function is linear. Lastly we used mean squared error as loss function and adam for optimiser. Note that we used different hyperparameters as well.

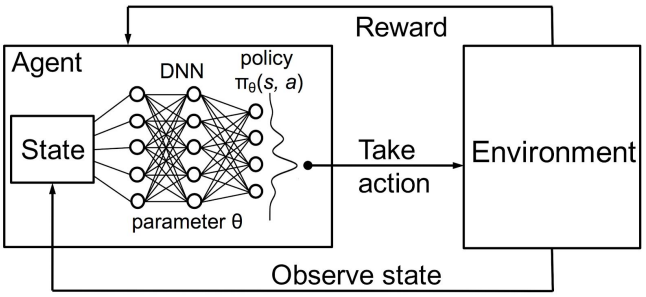

Fig. 4. Deep Reinforcement Learning Model

IV. Conclusion

This paper introduced a deep learning model for reinforcement learning, and demonstrated its ability to master difficult control policies for custom environment, using only raw pixels as input.

Our best accuracy is %83 and that result obtained when red and green dots were also moving. Note that these dots are not agents, they are just randomly moving. All configurations are reported in related document.

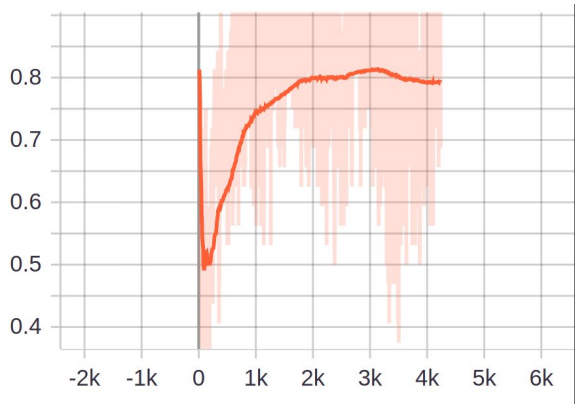

Fig. 6. Accuracy Result

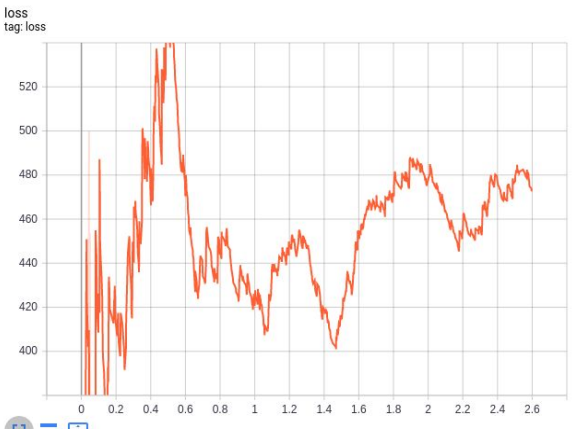

Fig. 7. Loss Result

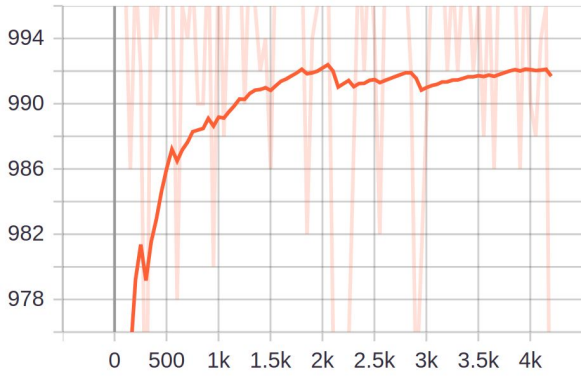

Fig. 8. Max Reward Result

V. REFERENCES

[1] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In the International Conference on Machine Learning (ICML).

[2] Agrawal, P., Nair, A., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In the Annual Conference on Neural Information Processing Systems (NIPS)..

[3] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In ACM Conference on Computer and Communications Security (ACM CCS).

[4] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[5] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.

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

[8] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 2013.

[9] Marc G Bellemare, Joel Veness, and Michael Bowling. Investigatingcontingency awareness using atari 2600 games. In AAAI, 2012.

[10] Marc G. Bellemare, Joel Veness, and Michael Bowling. Bayesian learning of recursively fac-tored environments. In Proceedings of the Thirtieth International Conference on Machine Learning (ICML 2013), pages 1211–1219, 2013.

[11] Playing Atari with Deep Reinforcement Learning,Volodymyr Mnih,Koray Kavukcuoglu,David Silver,Daan Wierstra,Alex Graves,Ioannis Antonoglou,Martin Riedmiller