Exploring Game Playing AI using Reinforcement Learning Techniques

Preprint · November 2020

DOI: 10.13140/RG.2.2.14522.62400

1 author:

Prabesh Paudel
St. Olaf College
3 PUBLICATIONS **9** CITATIONS

SEE PROFILE

Exploring Game Playing AI using Reinforcement Learning Techniques

Prabesh Paudel
paudel2@stolaf.edu, *MSCS Department, St. Olaf College*

Abstract--Reinforcement learning has proven to be state-of-the-art methods for a lot of machine learning and artificial intelligence tasks from resource management, traffic control to self-driving cars, robotics, and even game playing. This method allows an agent to estimate the expected utility of its state in order to make optimal actions in an unknown environment. This paper explores different reinforcement learning algorithms and their efficiency in playing popular games. We use Q-Learning to train agents to play trivial games like Flappy Bird, and we dive into Deep Reinforcement Learning to train the agents to play more complicated games where the tasks are non-trivial.

Keywords-- Reinforcement Learning, Q-Learning, Deep Q-Learning

1. INTRODUCTION

Reinforcement Learning (RL) is an area of machine learning that operates via an idea of reward or punishment for every action an agent takes and the end goal for the agent is to maximize the cumulative reward. We are going to be discussing Q-Learning and Deep Q Network in this paper. However, these are not the only RL algorithms that exist. Markov Chain Decision Process, SARSA, and DDPG are some of the RL algorithms that are popular today.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

Fig 1: Bellman Equation [Richard E. Bellman, 1951][Source]

The equation above is used to calculate a new Q value each time the agent loops through a state. Q-Learning is an algorithm that improves the agent based on the reward that the agent gets for performing an action. A major part of Q-Learning is the Q table which consists of Q values for each set of actions or movements an agent can take. "Learning" in this case is updating the Q table so that we have the optimal Q value for every action in each state. Once learning is done, the agent can refer back to the Q table to figure out the best possible action for any given state. In this paper, we use Q-Learning to train an agent to play the mountain car and flappy bird. To put it into context, we discussed some scenarios where we created agents that could play games like *Hunt the Wumpus.* However, in those cases, the state space and the action space were finite and we could hardcode some search algorithms like A* or BFS/DFS to find the optimal path to victory. However, we don't have perfect knowledge of the game in a lot of cases and that is where search algorithms fail and reinforcement learning comes in.

One of the major issues with Q-Learning is memory and time. The amount of memory required to save Q values for every possible action in a fairly big game is huge and the time required to explore each state to create the required Q table is simply unrealistic. This is where Deep Q Networks come in. Deep Q-Learning is simply a way of avoiding the Q table as a whole and rather use a neural network to approximate the Q value given the state. In this paper, we use Deep Q-Learning to train our agents to play more complicated games (Minh et. al. 2015).

1.1. Significance

Why should we care about Reinforcement Learning? How is it different from a run off the mill machine learning algorithm? Let's take a step back and think about supervised learning and unsupervised learning. In those kinds of situations you have a pre-existing set of inputs and finite outputs corresponding to the input, or a cluster that you want the input to fall in. So you just train the data you collect in one of the supervised or unsupervised algorithms you see fit.

However, reinforcement learning works in a completely different way. They can deal with large complex problem spaces, meaning it can deal with every possible combination of possible circumstances. Reinforcement learning learns by a process of receiving rewards or punishments for every action it takes, and is able to adapt in unforeseen circumstances accordingly. This can not only be applied to tasks like game playing, where the set of actions and outcomes

are virtually infinite but also tasks like Natural Language Processing and Supply Chain Management can take advantage of RL.

2. RELATED WORK

Q-Learning was first theorized by Chris Watkins in 1989. Q-Learning (Watkins et al, 1992) provides the first convergence proof of the Q-Learning algorithm. Since then a lot of innovation has happened in Q-Learning, one of which includes the use of the Bellman equation which is a mathematical optimization method. Q-Learning uses the Bellman equation to update the Q values in the Q table.

Q-Learning has been used for various tasks after this innovation for various tasks. It wasn't until DeepMind published a paper Playing Atari with Deep Reinforcement Learning in 2015 when Deep Q-Learning became popular. It was one of the first papers that approached game playing using Deep Q-Network. The 2600 Atari gaming system was quite popular in the late 1970s and the early 1980s. The games ran with only 4 kilobytes of RAM on a 210x160 pixel display with 128 colors. However, despite many attempts at creating agents using other machine learning techniques, none of them were quite successful. Mnih et al attempt to provide a proof concept that training agents to play highly complicated games is possible using Deep Q-Learning. The 2019 paper Grandmaster level in StarCraft II using multi-agent reinforcement learning is the state-of-the-art paper on using multi-agent reinforcement learning to not only play StarCraft 2 but reach the Grandmaster level. We don't expect our model to reach that level of complexity.

However cool it may seem, reinforcement learning is not able to perform as well in a physical environment as it can in virtual simulated environments. It is not feasible to take best action in the real world where all the actions come with repercussions and the real world is very dynamic, as opposed to a simulated virtual environment like a videogame. So, even though a lot of papers have discussed the applicability of reinforcement learning in the physical environment, very little have been actually accomplished.

3. METHOD

In this section, we discuss how we create an environment and how different techniques were used to train our agents.

3.1. 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. First, we explore Q-Learning to create an agent that can play Mountain-Car and Flappy Bird. Q-Learning is a foundation for Deep Q-Learning.

3.1.1. State Representation

Mountain Car: Mountain-Car is a fairly simple game powered by OpenAI Gym. A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. [OpenAI]. The state of the car is represented in a tuple that is a coordinate. There are three actions. We know we can take 3 actions at any given time. That's our "action space."

Flappy Bird: Flappy bird is a game where you have a bird and your goal is to keep it alive as it goes through the pipes. The action is moving the bird up and down to get the bird through the pipes.. It works in a way where you tap the screen to move the bird upwards and press nothing to let it fall. Flappy bird seems more complicated but it is very similar in complexity to Mountain Car. We have a discrete state space which consists of three variables, the horizontal distance to the pipes, the vertical distance to the lower pipe and a boolean value that indicates whether the bird is dead or alive.

3.1.2. Q-Learning Algorithm

The main idea of the Q-Learning Algorithm is creating a Q-Table, which is essentially a policy table, that has Q values for all possible combinations of states and actions. The following is the Q-Learning algorithm. (Watkins et al, 1992)

Initialize Q(s, a) arbitrarily
Repeat for each episode:
  Initialize s
  Repeat (for each step of the episode):
    Choose a from s using policy derived from Q
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$;
  until s is terminal

There is a Q-value per action possible per state and we create a table for this. To figure this table, we

either query the environment or we engage in the environment over and over again until we figure it out. For a mountain car, we get a continuous x coordinate, velocity, and reward as a return value. Since making a table for every single continuous value is impossible, we make discrete chunks of them and get a finite Q table. We use the algorithm above to update the Q-table, and we query it whenever we have to run the mountain car independently. The code for the mountain car can be found here.

We take a similar approach to flappy bird. We initialize a Q table and see what action maximizes the reward given the state. We reward the bird with 0 points if it survives and give it -1000 points if it dies. We update our Q table accordingly as described before. The code for the flappy bird game was found on the internet. It is a replica of the flappy bird game made using PyGame. The code for the Q-learning bot can be found here.

3.2. Deep Q-Learning

Building on top of the foundation we created with Q-Learning, we now move on to Deep Q-Learning.

---

*Note: Deep Q-Learning is absolutely not necessary to achieve the goals that we are about to explore. We just use Deep Q-Learning to understand how Deep Reinforcement Learning works!

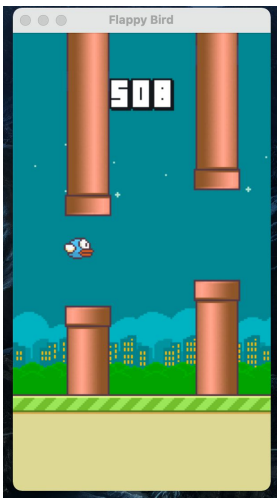
In Q-Learning, we build a Q table for all possible sets of states and actions. But we can sometimes encounter states which might be similar, but not exactly the same as one of the combinations in the Q-Table. This is where Deep Q-Learning comes in. Flappy bird has a relatively small state space so it is easy to discretize the state space and create a Q-table for it. But once we move to more complex games, the state space and action space are way bigger than that of flappy bird and hence the size of the Q-table increases exponentially. Storing such a Q table would be next to impossible. (Chen, 2015)

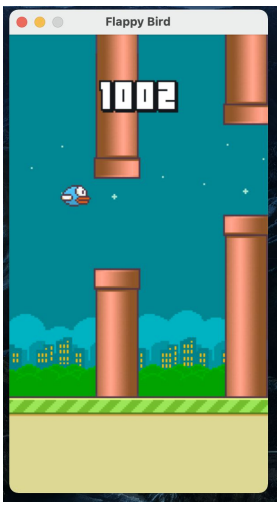
Initialize replay memory
Initialize DQN to random weights
Repeat
  New episode (new game)
  Initialize state s_0
  Repeat
    Extract x_t from raw pixel data update state s_t with x_t
    Add experience
    $e_t = (\varphi(s_t-1), a_t-1, r_t-1, \varphi(s_t))$ to replay memory
    Take best action
    $a_t = arg\ min_{a \in actions}\ Q(st\ ,\ a)$ with exploration if training
    Uniformly sample a batch of experiences from the replay memory
    Backpropagate and update DQN with the minibatch
    Update exploration probability ε
    if C updates to DQN since last update to target network **then**
      Update the target Q-network
      $Q(s, a) \leftarrow Q(s, a)$
    end
    Update state s_t with a_t
    Update current reward r_t and total reward totalReward
    Update game parameters (bird position, etc.)
    Refresh screen
  Until flappy bird crashes;
  Restart Flappy Bird
Until convergence or number of iterations reached;

Deep Q-learning algorithm for Flappy Bird,
Deep Reinforcement Learning for Flappy Bird[2015]
Kevin Chen [Source]

So, we use Neural Networks to predict a Q-value for possible actions given a state. The Deep Q-Network 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. It is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards.

With the introduction of neural networks, rather than a Q table, the complexity of our environment can go up significantly, without necessarily requiring more memory. The aforementioned is the Deep Q-Learning algorithm that is used in the flappy bird program that can be found here.





Our Deep Q-Network is trained on raw pixel values observed from the game screen at each time step. We feed the raw images to a Convolutional Neural Network with Max Pooling layers and the output layer has the same dimension as the action space, which in this case is two. One of them corresponds to moving the bird upwards and the other one corresponds to doing nothing. At each time step, the network performs whichever action corresponds to the highest Q-value using a greedy policy. The network is trained using the algorithm shown above.

4. RESULTS

The results of the experiments were interesting. Starting off with Mountain Car, we assumed it would not take many iterations for the Q-Learning algorithm to figure out how to reach the top of the mountain but to our surprise it took the algorithm over 2000 episodes to reach an optimal Q-table. However, it was not the case for flappy bird. The algorithm was able to figure out the game in under 250 episodes and the game reached a superhuman level. The game basically never died and was able to reach the max score every single time. At this point we figured that Deep Q-Learning for this task would be an absolute overkill but we wanted to explore it anyways.

As expected it was able to reach a maximum score on every single trial as well. However, it took a while to train the network because the image had to be preprocessed and it took awhile for the neural network to converge to an optimal model.

5. CONCLUSION

Reinforcement Learning has a lot of potential. Game playing is one small thing that RL is good at and it has been explored extensively in recent years. Recently, a new toolkit called NLPGym has been released which gives the user an environment where they can test Reinforcement Learning on Natural Language Processing tasks. I would love to explore this myself and I am actually doing an *Independent Research* with Matthew Richey on this next semester, which I really look forward to.

Future work building on this project might include accomplishing tasks that have a bigger action and state spaces and possibly replicating some of more complicated papers like *Grandmaster level in StarCraft II using multi-agent reinforcement learning* by Vinyals et. al. Future of reinforcement learning is very bright and I can see these techniques being applied in a lot of fields.

6. REFERENCES

[1] Kaundinya, V., Jain, S., Saligram, S., Vanamala, C. K., & B, A. (2018). Game Playing Agent for 2048 using Deep Reinforcement Learning. *Nciccnda*. doi:10.21467/proceedings.1.57

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2015). Playing Atari with Deep Reinforcement Learning. *DeepMind Research*. doi:https://arxiv.org/pdf/1312.5602.pdf

[3] Vinyals, O., & Babuschkin, I. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*. doi:https://doi.org/10.1038/s41586-019-1724-z

[4] Watkins, C. J. (1998). Q-Learning. *Kluwer Academic Publishers*.