Application of Q-Learning Neural Network in Quadrotor Path Planning with Limited Environment Perception

Yupeng Li YUPENGLI@SEAS.UPENN.EDU
Junfan Pan JFPAN@SEAS.UPENN.EDU
Jiatong Sun JTSUN@SEAS.UPENN.EDU

Abstract

For general obstacle avoidance trajectory and path planning for quadrotors, it is common to use A* and Dijkstra's algorithms for a given map. However, when it comes to an unknown environment and given limited perception of the environment, these two algorithms would not hold anymore. Thus we are trying to use reinforcement learning neural network to resolve this problem as proposed in (Bing-Qiang Huang, 2005)

1. Introduction

Obstacle avoidance is a key part for autonomous quadrotors. For environments already known, there have already been optimal and fast path generation algorithms such as A* and Dijkstra. These algorithms can help navigate the quadrotors when the map and obstacles are already revealed before planning the path. However in real world situations like search and rescue post disaster and archaeological excavation, we don't always have access to the whole map. Obviously, it is not cost effective and safe to let human do the job instead. We hope to use quadrotors to save more people post disasters and better explore the sites before excavating for the artifacts. Thus, a method which can navigate the quadrotors in complex and unknown environments is highly demanded.

Among all the machine learning techniques we studied in class, reinforcement learning is the best solution in this case. Since our agent, the quadrotor, would need to make decisions in unknown situations about where to go based on the surrounding environment and learn what to do and what not based on the environment's feedback.

Using reinforcement learning, the quadrotors can then learn the desired actions like going straight or going diagonal as outputs providing current states including position, the desired target location and surrounding obstacles within visible range. For each pair of state x and action a, the quadrotor is trying to learn the state-action Q value $Q(x, a)$ through Q-learning. After training under substantial amount of maps, the quadrotor would be able to predict the correct action by picking the state-action pair with highest Q value given a specific state.

The real-world states and actions can be complex. Since there are large number of state-action pairs in autonomous quadrotor obstacle avoidance problem, the Q values can hardly be stored and updated in a matrix as in the Mountain Car problem. Thus we are using a neural network to store and predict these Q values. The neural network here serves as a table where we can look up the expected Q values given state and action. Also, it allows the quadrotor to reasonably interpolate Q values for state-action pairs that has not been visited before.

Putting these together, we are trying to use a technique which combines the idea of reinforcement learning and neural networks to navigate autonomous quadrotors in environment unknown. We hope this would help in real-world excavation and rescue.

2. Methods

2.1. Generate Random Maps

In order to initialize the neural network, a random map generator is required. This random map can generate a world map with randomly positioned blocks within a user-defined boundary, and generate random start and goal position for the quadrotor. Besides, to ensure a valid training path, we must guarantee the existence of path. Therefore, several space constraints need to be satisfied among those randomly generated blocks and also between the start and the goal point.

To create random obstacles, a class method called **ran-**

University of Pennsylvania, CIS 419/519 Course Project.

dom_block() is created inside the class World. This class method returns a world object describing boundary, random blocks volume and position, start and goal position parameterized by input arguments, which are lower bound and upper bound of the world, block width, block height, number of blocks, robot radius and safety margin. In addition, all space constraints are satisfied during generation process. For instance, the distance between two random blocks should be larger than two times of safety margin plus robot radius, blocks should never go beyond the boundary, start and goal position should be at least one margin plus robot radius from the closest point on the blocks, etc.. After that, all values of boundary, block, start point and goal point are stored in a list of dictionaries called world_data and then can be directly fed into Sandbox to generate flight simulation output, which are also our training data.

2.2. Obtain Warm-up State-Action Pairs

Based on the randomly generated maps, A* algorithm is implemented with L2 norm as heuristic to generate an optimal path. The path returns a list of positions and subsequent actions to the main function.

The main function then expand the state by adding the obstacles distance within range of 20 indices. If there is an obstacle within 20 indices in a certain direction, the distance to that obstacle would be added to the state vector at that location of that certain direction. Otherwise, it would return a zero telling that there is no obstacle in range.

2.3. Q-values, Rewards and Algorithm

Based on the state and action values from Section 2.2, we warm up the Q-value neural network with $Q(x, a) = 10$ for all the points in the planned trajectory, $Q(x, a) = -100$ for going into obstacles and obstacless and $Q(x, a) = 0$ for other state-action pairs.

Then we follow the algorithm 1 to learn Q and update the neural network in unknown environments until the quadrotor learns how to make an optimal decision. Note that in the algorithm, x stands for current state, a is the action taken, y is the actual next state, r is reward which will be covered below, $0 < \gamma < 1$ is the discounted factor and α is the learning rate.

The reward function is determined using the equation in (Richter C., 2016) as follows:

$$r = -\rho(\frac{D_n - D_c}{D_c}) - \gamma(dt - \frac{d}{V_{max}}) - \beta|\omega|$$

Algorithm 1 Q-learning Algorithm

```
Input: Warmed-up Neural Network $M$
repeat
    Get current state $x$
    Determine optimal action st. $a^*(x)$=argmax$Q(x, a)$
    Choose if to take a random action or the optimal action
    Quadrotor moves and obtains new state $y$
    if Collision Occurs then
        $r = -100$ and return to starting position
    end if
    Update $Q$ using

    $Q(x, a) = Q(x, a) + \alpha[r + \gamma maxQ(y, b) - Q(x, a)]$

    Calculate the difference between updated $Q$ and original $Q$ to train the neural network
until $Q$ converge to $\epsilon$
```

Where ρ, γ and ω are hyper parameters which respectively emphasize on the choice of locally optimal path, choice of the maximum possible velocity and choice of taking a straight path. d, D_n and D_c respectively stands for the distance travelled in the current time step by taking present action a, shortest distance to the destination before and after taking the specified action a.

Since we simplified the model so that we would not be considering the orientation and kinematics of the quadrotors and only focus on the path, the reward function is then trimmed down to:

$$r = -\rho(\frac{D_n - D_c}{D_c})$$

where ρ is set to 100 to prevent the model from casting all the training labels to 0, thus only predicting 0 at the output.

2.4. Neural Network Architecture and Training

The neural networks consists of 2 hidden layers with ReLU as the activation functions. Input for the first hidden layer is a hyper vector with length of 23. The input vector here is our extended state-action pair. First three elements of the vectors are the original state of the quadrotor which in other words, are the current position of the quadrotor. The next 14 elements in the vectors represent the distance to obstacles within 20 indices in up, down, front, back, left, right and main diagonals direction of a cube. The following 3 elements represent the target location and the last three represents the action chosen.

The hidden layers each consists 50 neurons and finally go to the output layer with a single output that is the designated Q value for the given extended state. Since the output is a single value, we are using Mean Squared

Error (MSE) as our loss function and Adaptive Momentum (Adam) as our optimizer.

During Q-learning, when a new state-action pair with the target Q-value is encountered, the train function first determines if the state-action pair is already in the training set. If it is, the training label for this state would be updated to the new Q-value. If not, the new state-action and respective Q-value would be appended to the training set.

After each episode of Q-learning, the new data-set would be fed into the train function and keep training for 2000 epoch. Early stop would be applied if the training loss drops below 10^{-4}. This process would repeat until the Q-value neural network finally converges.

3. Results and Discussions

In the standard approach above, we return to the start point whenever the quadrotor hits an obstacle and update the Q-value with penalty and train the neural network with updated Q's. During the process of training, we discovered a new approach to accelerate the searching process. In this new approach, which is named as "Persistent Searching" approach, the quadrotor does not return to the origin after hitting the obstacles. Instead, it updates the Q-value in the training labels and remove the hitting-obstacles action from the possible action list. Then it continues searching for path to the goal until it finally reaches the goal or reaches maximum steps. Using this new method, the quadrotor would explore the map more efficiently and converge to the desired path faster. The following section compares these in loss, success rate and final path of the quadrotor.

3.1. Warm-up Neural Network

In the original Q-learning algorithm, we introduced a warm-up neural network training procedure to initialize weights and bias parameters of the network. By feeding the extended state-action pairs obtained by the the A* algorithm from known maps into the neural network, we can expect to obtain a quite powerful pre-trained model to accelerate successive Q-learning training procedure. However, considering limited access to powerful computational resources, it is only possible to warm-up the neural network with very limited number of maps. In this case, we found that instead of facilitating the quadrotor to explore the path more effectively, this warm-up neural network actually impeded the quadrotor from learning the unknown maps. Therefore, we decided to directly train the initial neural network in unknown environment without warm-up procedure to focus on the implementation of reinforcement learning considering the feasibility of our project.

3.2. Training Loss

We monitored the training loss of each epoch throughout all episodes for both approaches. Figure 1 and Figure 2 show the two training loss curves, where the horizontal coordinate corresponds to the cumulative epochs and the vertical coordinate corresponds to the mean squared error (MSE loss).

In addition to recording loss, we simultaneously implemented the early stopping method. For the whole training process, we consistently keep a record of the lowest loss. If the lowest loss has not been updated for more than 30 epochs, the training process will be terminated. As Q-learning procedure continues and searching likelihood decays, the quadrotor is more likely to follow an existent successful path, therefore accelerates training process. This effect can be visualized from the following figures that the total number of cumulative epochs is significantly fewer than 200000 (100 Q-learning episodes times 2000 maximum epochs per training process).

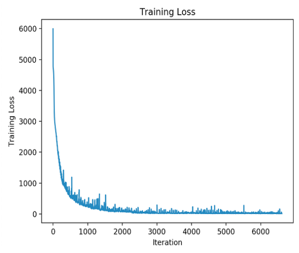

Figure 1: Standard Approach Loss

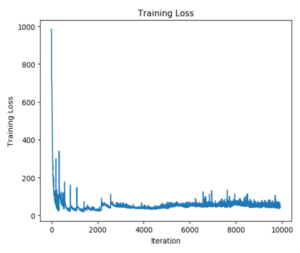

Figure 2: Persistent Searching Loss

Figure 1 manifests the training loss generated by standard approach. Despite of those spikes which take place when a Q-leaning episode completes, the training loss steadily falls over the whole training process and finally converges to approximately zero. The final steady and trivial loss indicates that a stable path has been discovered.

Figure 2 manifests the training loss generated by persistent searching approach. Compared with standard approach, the strengths of the persistent searching method is quite obvious. In the early epochs, the training loss of persistent searching approach drops much faster than the standard one. This is understandable because the agent manages to gather more information, especially around positions where collisions take place.

3.3. Success Rate

We saved the success rate over every five episodes for both two approaches. Figure 3 and 4 show these two success rate curves.

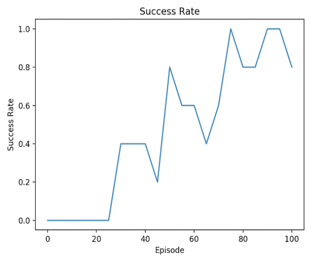

Figure 3: Standard
Approach Success Rate

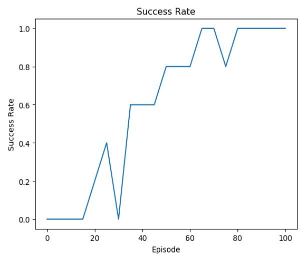

Figure 4: Persistent
Searching Success Rate

From the figures above, we can see a clear rising trend of the success rate. Both rates increase to approximately 100% eventually, indicating that the quadrotor has learned a stable path towards the goal.

The difference between these two figures is that the persistent searching agent uses relatively fewer Q-learning episodes to find the first successful path. This is because that this improved method is devised to gain more information during each episode. In addition, we can observe that persistent searching stabilized at a successful rate of 80% or higher much earlier than the standard searching method.

3.4. Final Path

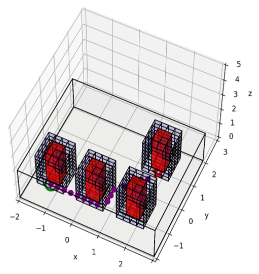

Figure 5: Standard
Approach Path

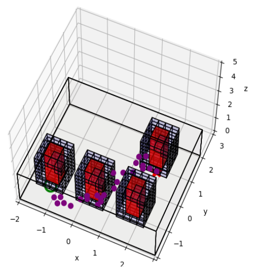

Figure 6: Persistent
Searching Path

The map shown in Fig.5 and Fig.6 above respectively displays the best path from the start to the destination in an unknown map discovered by these two approaches after running 100 episodes of Q-learning procedures. Compared with the path generated by persistent searching, path obtained by standard approach has less waypoints thus is more optimal in terms of path length. The underlying reason is that persistent searching approach is designed to make extensive exploration around where the collision takes place and keeps on searching until the goal or the maximum exploration step limit is reached. Therefore, a complete path without collision discovered by this approach tends to be closer to blocks and thus have more seemingly redundant points in the final path. In contrast, standard approach stops searching once the quadrotor encounters a collision and always restarts from the start position. Therefore, it is more likely to generate a relatively cleaner path comparing to persistent searching approach.

4. Conclusions and Future Prospects

By the end of this project, the quadrotor has been able to train itself and learn how to navigate in a random map by using the trained neural network to predict respective Q values within a couple of Q-learning episodes. It turns out to be helpful in autonomous quadrotor navigation by addressing the case when there is no available map of the job site. Flight environments that are unknown and complex such as excavation sites and rescue scene can take significant advantage of this Q-learning neural network approach.

As illustrated in results and discussions section above, the standard approach and persistent searching both show promising results in terms of increasing the success rate and reducing the training loss. The standard approach provides less scattered path while the persistent searching method explores the map more efficiently. We hope to keep improving on how our agent interacts with the environment during the Q-learning phase so that it could obtain as much information as possible within each training episode.

Due to the impact of Cvoid-19 and the scope of this project, our team has not been able to push the idea of autonomous navigation for quadrotors further. This Q-learning neural-network model can actually be improved to some extent that it would be able to work for most of the given maps of same size but different start, goal and obstacle locations without further Q-learning. That would require training on larger data sets and a more carefully designed neural-network structure.

In future work, we expect to keep improving on the neural-network structure so that it would be capable of handling more state-action pair possibilities. Meanwhile, our group would try to train the model on more extensive data-sets so that it would make more optimal decisions under more complicated circumstances .

References

Bing-Qiang Huang, Guang-Yi Cao, Min Guo. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. volume 1, pp. 85–89. IEEE, 2005. ISBN 0-7803-9091-1. doi: 10.1109/ICMLC.2005.1526924.

Richter C., Roy N., et al. Learning to plan for visibility in navigation of unknown environments. *2016 International Symposium on Experimental Robotics*, pp. 387–398, 2016. doi: https://doi.org/10.1007/978-3-319-50115-4_34.

5. Appendix

5.1. Group Members and Responsibilities

Yupeng Li Constructed the neural network reinforcement learning structure, algorithm and set up the reward function. Wrote a couple of helper functions and the main code for Q-learning. Generated animation for flight and saved as mp4 files for observation.

Junfan Pan Created and implemented the persistent searching approach as an improved Q-learning method. Helped to improve Q-learning codes. Wrote codes for warming up Q-learning neural network, generating random maps with obstacles, plotting success curve and several helper functions.

Jiatong Sun Constructed the neural network structure. Implemented training process with early stopping for execution efficiency. Wrote codes on generating dataloader, plotting learning curve and several other helper functions.

Link to Vimeo https://vimeo.com/415675101

Link to Github https://github.com/iamliguangming/CIS519Project