Landing, Gracefully

Pouya Rezazadeh / Kongphap Wongpattanukul / Dong Hee Song
CS221, **Artificial Intelligence: Principles and Techniques** (Fall 2019) Stanford University

Introduction

- Autonomous drones and rockets have a spectrum of applications from urban transportation and reusable rockets to drug delivery and fugitive methane leak detection
- Effective control systems are needed, especially for challenging tasks of landing and takeoff
- This project applies AI to drone maneuvering using *LunarLander-v2* simulation environment from *OpenAI Gym* [1]
- The objective is to land the vehicle on the target *fast*, *safely*, and *efficiently*. We use reinforcement learning, in particular Q-Learning, to train an agent to achieve this goal.

Fig 1. Urban Air Transport (left), Methane Leak Detection (center), Autonomous Rockets (right)

[2],[3],[4]

Model: LunarLander-v2

- The lander maneuvers by engaging thrusters (with a noisy outcome) and consuming fuel
- State has 8 components:
  - horizontal and vertical position, horizontal and vertical velocity, angle and angular velocity, and left and right leg contact
- Control agent can take four actions
  - (i) do nothing, (ii) fire main engine (push up), (iii) fire left engine (push right), and (iv) fire right engine (push left)
- Vehicle starts from the top of the screen (with random initial velocity) and landing pad is always at coordinates (0,0)
- Each simulation episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Firing side engine is -0.03 points each frame. Solved is 200 points.

References

1. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," arXiv preprint arXiv:1606.01540, 2016
2. www.openaccessgovernment.org/
3. internetofbusiness.com/ge-commercial-drone-detect-gas-leaks/
4. www.theverge.com/2018/12/5/
5. Mao, Hongzi, et al. "Resource management with deep reinforcement learning." *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016.

Methodology

Q-Learning

Q-learning: model-free RL algorithm; estimates optimum policy (π_{opt}). Optimum Q-value ($\hat{Q}_{opt}$) estimated using the state and action:

$$\hat{Q}_{opt}(s,a) = f(s,a)$$

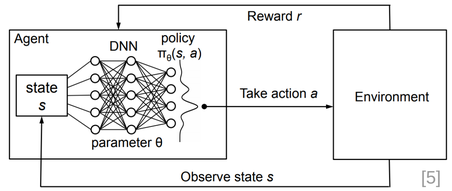
Given a sequence (state (s), action (a), reward (r), new state (s')), want to achieve optimal Q-value: discounted future reward given optimal policy:

$$\hat{Q}_{opt} \approx r + \gamma\left(\max_a \hat{Q}(s',a)\right)$$

Loss function for this regression problem (mean squared loss):

$$MSE = \sqrt{\frac{1}{N}\sum_i (y_i - \hat{y}_i)^2}$$

Exploration of algorithm: determined by epsilon greedy policy



[5]

Fig 2. Deep Q-Learning

1. Linear Function Appx.

Linear models were used to estimate the optimum Q-value ($\hat{Q}_{opt}$) .

$$\hat{Q}_{opt}(s,a) = w \cdot \phi(s,a)$$

where w is the sparse weight vector and $\phi(s,a)$ is the feature vector. Our feature vector (ϕ) is based on applying indicator function to the discretized state values (i.e. positions, angle, and velocities) and actions. Each continuous state variable is rounded to its nearest decimal place e.g. zero decimal place or first decimal place. Our discretization scheme is defined as:

$$\phi_i = \begin{cases} 1(\lfloor s \rfloor = s_i, a = a_i) & \text{, round down} \\ 1(\lceil s \rceil = s_i, a = a_i) & \text{, round up} \end{cases}$$

where s is the current state variable, a is the selected action, s_i is all possible discretize state variable values, and a_i is all possible actions. s could be single state variable or a combination of them e.g. position, a tuple of angle and angular velocity, etc.

2. Deep Q-Learning with Neural Networks

Neural Networks were used to better estimate the optimum Q-value ($\hat{Q}_{opt}$)

$$h_i = f(w_i \cdot h_{i-1})$$
$$\hat{Q}_{opt}(s,a) = w_n \cdot h_n$$

where w_i is the weight of hidden layer i, h_i is the output value of hidden layer i, and $f(z)$ is an activation function. Note that the input layer is corresponded to the number of states and the output layer is corresponded to the number of actions.

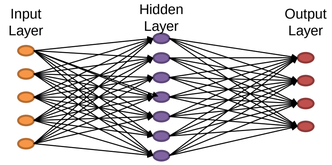


Fig 3. Sample Deep Neural Network

- Network: consisted of 8 inputs, 2 hidden layers and 4 outputs (8, 150/ReLU,120/ReLU, 4/Linear)
- Optimizer: ADAM
- Loss function: mean squared error
- Experience replay*: used to stabilize algorithm and enhance efficiency

Hyper-Parameters

- # training examples
- # epochs
- Batch size (experience replay)
- Dropout rate
- Learning rate
- Weight decay
- lambda scaling
- Maximum moves
- Initial exploration probability
- Minimum exploration probability
- Exploration probability decay

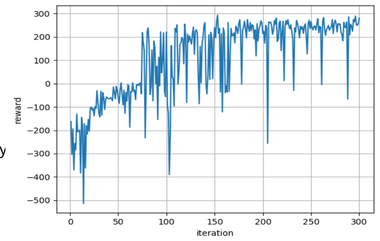


Fig 4. Score vs. number of iterations for best performing deep Q-learning algorithm

* Technique storing observed (s,a,r,s') in memory, and sampling them to create a batch for the neural network.

Results and Discussion

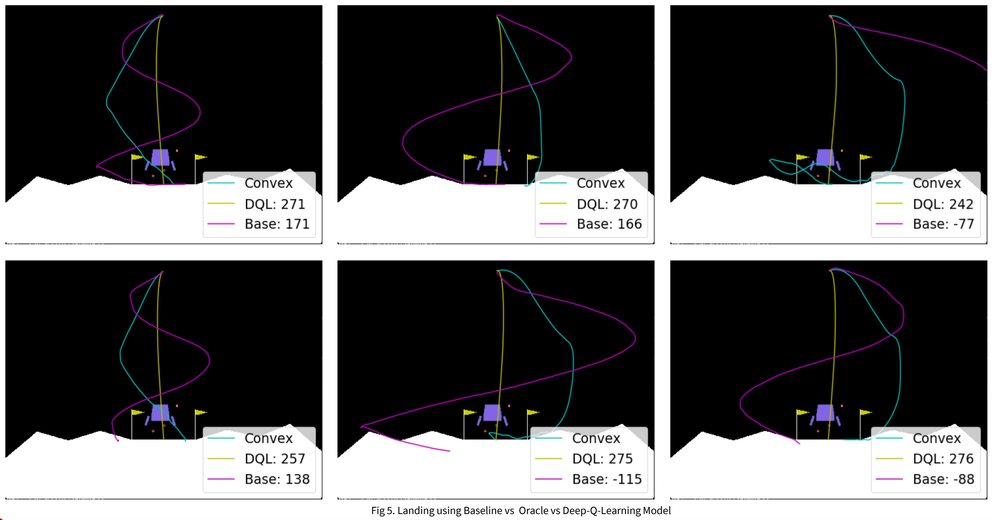


Fig 5. Landing using Baseline vs Oracle vs Deep-Q-Learning Model

Case Name	Algorithm	Score (1000 run avg)
Baseline	Heuristic Control	-87.21
Features rounded to 0-decimal places	Linear Function Appx	-29.46
Features rounded to 1-decimal place	Linear Function Appx	-7.14
DNN without Memory Replay	Deep Q-Learning	35.8
DNN with Memory Replay	Deep Q-Learning	212.82

- Agent trained with Deep Q-Learning (DQL) with experience replay can consistently land the lunar lander
- DQL agent consistently outperforms the baseline
- DQL with experience replay shows strong potential for training optimal control agents for planetary vehicles