Solving Lunar Lander with Deep Reinforcement Learning

Shivani Bhakta

Why Lunar Lander?

- Need for space exploration
- Need for safe landing in different uncertain conditions (gravity, terrain, external force, engine issues, fuel etc)
- Can also be used for landing on earth in different conditions
- Has many applications in robotics control.

Classic Reinforcement Learning

- Why current classic Q-Learning method are not good enough?
  - Curse of Dimensionality
  - Computationally heavy for large state and action space.
  - Will not give optimal policy
  - Will take a long time to run even if computational power is available.
  - Gets hard to manage the Q-value matrix over time.

Deep Reinforcement Learning

Deep Q-Learning with experience replay

- It uses a neural network as a nonlinear function approximator to get the action values.
- They can be unstable due to the correlation between the consecutive states.
    - To fix this we use something called experience replay.
- There is also a correlation between the Q-values and the target Q-values.
    - This can be solved using a separate target network and updating it.

Deep Q-Learning with experience Replay

Algorithm 1 Deep Q-Learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ.
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode* = *1,M* **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for *t=1,T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end
end

1. Experience Replay

A memory buffer is used to record every experiences and use randomly sampled minibatch to train the Q-Network.

2. Q-Target Network

Maintain a fixed target network for the target values and update these values after a every few (pre decided) number of steps.

3. ϵ-greedy policy

We choose the best action (max q value) with probability (1-ϵ), otherwise, we choose a random action.

References

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529–533.
- Lecture Slides.