Learn how to play Flappy Bird with Deep Reinforcement Learning

Presented by: Tao Ni
Flappy Bird

The player controls a bird, attempting to fly between columns of green pipes without hitting them.
5 elements

- **Agent:** bird
- **Environment:** the green pipes world
- **State:** location of the bird
- **Action:** flap or not
- **Reward:** a scalar
An episode

$s_0, a_0, r_1, s_1, a_1, r_2, s_2, ..., s_{n-1}, a_{n-1}, r_n, s_n$

The bird died.
Game over.
Return

- $G_t$ is the discounted total future reward from current state all the way to the terminal state.
- $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-t} r_n = r_{t+1} + \gamma G_{t+1}$
- $\gamma$ is the discount factor between 0 and 1
- $\gamma$ accounts for the uncertainty in the future.
\( Q(s, a) \)

- \( Q(s, a) \) is the **expected return** after performing action \( a \) in state \( s \).
- \( Policy \ \pi(s) = \arg\max_a Q(s, a') \)

- **New task:** approximate the \( Q! \)
Bellman Expectation Equation

1. \( G_t = r_{t+1} + \gamma G_{t+1} \)
2. \( Q(s, a) = r + \gamma \max_{a'} Q(s', a') \), where \( S_t = s, S_{t+1} = s' \)
Pseudo Code

initialize $Q[\# \text{ states}, \# \text{ actions}]$ arbitrarily
Observe initial state $s$
repeat
    select and carry out an action $a$
    observe reward $r$ and new state $s'$
    $Q[s,a] = Q[s,a] + \alpha (r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
    $s = s'$
until convergence
Deep Reinforcement Learning
A huge $Q(s, a)$ table

- Using screenshots is an universal way to represent states of any game.
- In the DeepMind paper, $256^{84 \times 84 \times 4} \approx 10^{67970}$ possible game states
Parametrization using CNN

- $Q(s, a) = f(s, a|\theta)$
- $\theta_{k+1} \leftarrow \theta_k - \eta \nabla_{\theta} \text{Div}(f(s, a|\theta_k), Q_{s,a}^{\text{target}})$
- $Q_{s,a}^{\text{target}} = r + \gamma \max_{a'} f(s', a'|\theta_k)$
Limited Capacity of Q function

- A table of Q values will never forget any values that you write into it.
- But, modifying the parameters of a Q function will affect its overall behavior.
- When we are fitting the parameters to match one $(s,a)$ pair, it can change the function’s output at any other $(s',a')$ pair since $\theta$ is updated.
- If some pair is not visited for a long time, the function’s output can diverge considerably from the values previously stored there.
Action Replay

- Current data can look very different from data from several episodes ago if the policy changed significantly.
- The idea of action replay is to create a replay buffer holding past experiences, so we can train the Q function using this data.
Deep Q network with Action Replay

Initialize $\theta_0$;
Initialize buffer with some random episodes;

for each episode $e$ do
    Start at initial state $S_1$
    for $t = 1 \ldots$ Terminate do
        Draw $A_t$ from $S_t$ according to $\epsilon$-greedy policy derived from $\theta_t$
        Observe $R_{t+1}, S_{t+1}$
        Add $S_t, A_t, R_{t+1}, S_{t+1}$ to the buffer
        Sample from the buffer a batch of tuples $S, A, R, S_{new}$
        Choose $A_{target} = \arg\max_a f(S_{new}, a|\theta_t)$
        $Q_{target} = R + \gamma f(S_{new}, A_{target})$
        $\theta_{t+1} = \theta_t - \nabla_\theta \|Q_{target} - f(S, A|\theta_t)\|^2$
    endFor
defend