Artificial Intelligence

Lecture 14：Reinforcement Learning

Xiaojin Gong

2022-05-30

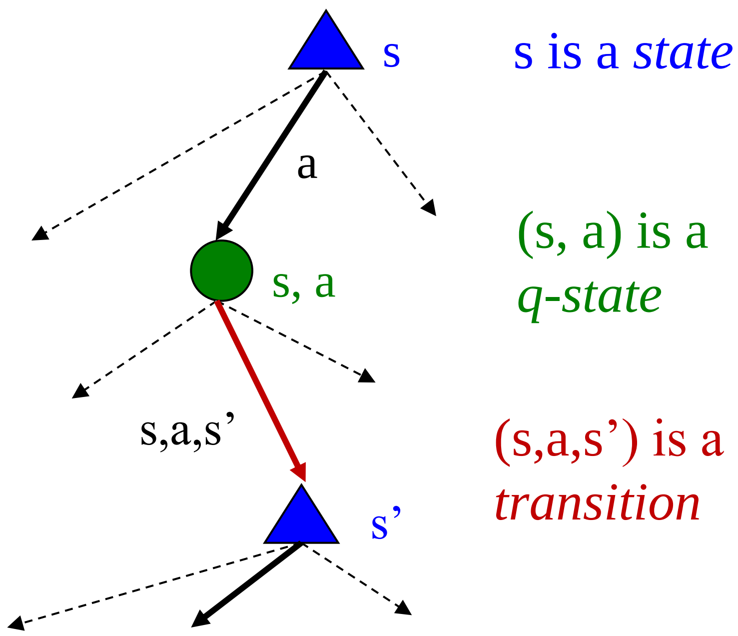
Outline

- Reinforcement Learning
  - Model-based Learning
  - Model-Free Learning

- Deep Reinforcement Learning

Review: Markov Decision Processes

- Markov decision processes:
  - States *S*
  - Actions *A*
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount γ)
  - Start state s_0



- Quantities:
  - Policy = map of states to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state (max node)
  - Q-Values = expected future utility from a q-state (chance node)

- Optimal values define optimal policies

Review: MDP Algorithms

- The Value Iteration Algorithm

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- The Policy Iteration Algorithm

$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

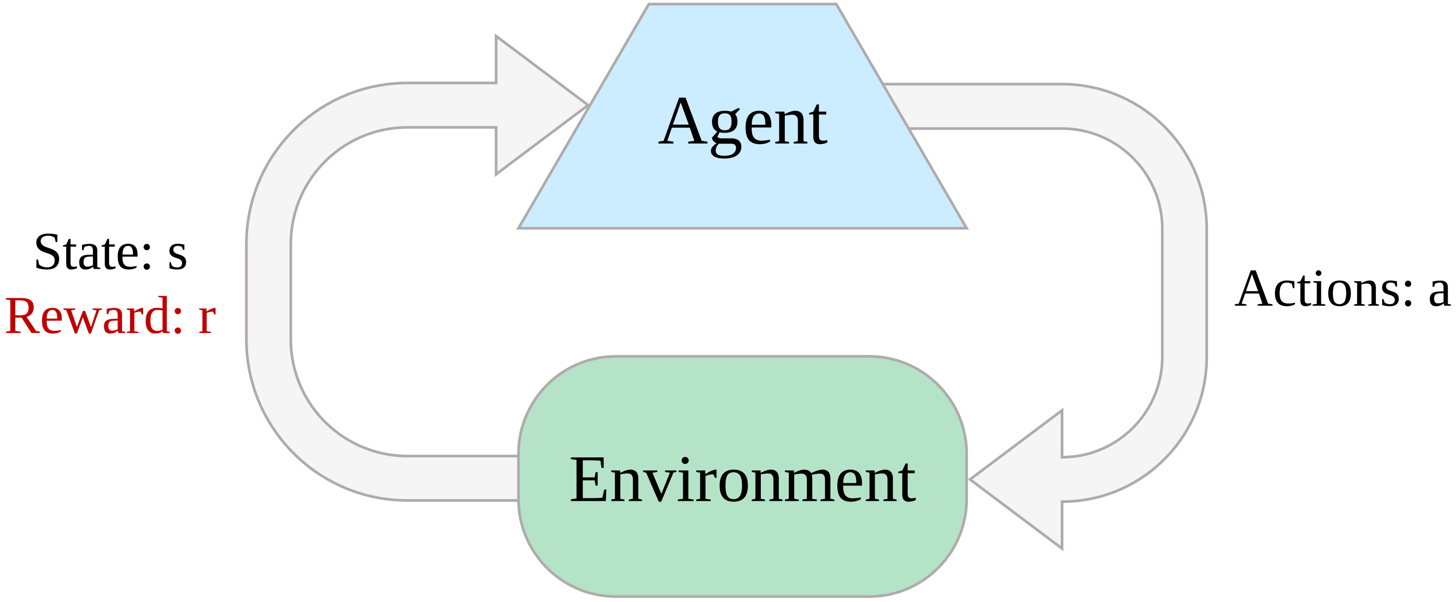
Reinforcement Learning

- Assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Looking for a policy $\pi(s)$

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Reinforcement Learning

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!



Reinforcement Learning

- Model-Based Learning
  - Learn an approximate model based on experiences
    - Transition model + Rewards
  - Solve for values as if the learned model were correct
- Model-Free Learning
  - Passive Reinforcement Learning
    - Directly evaluate values for each state under π
    - Policy evaluation
  - Active Reinforcement Learning

Model based RL

Value based RL

Policy based RL

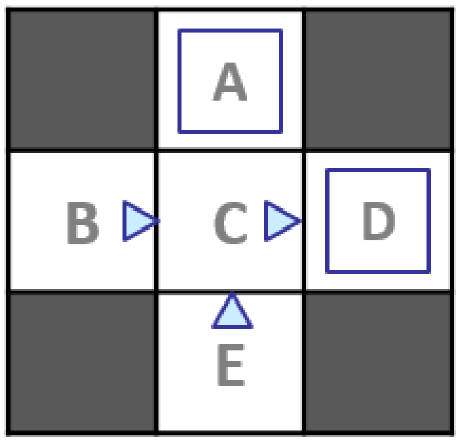
Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\widehat{T}(s, a, s')$
  - Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s')
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Model-Based Learning

- Learn empirical MDP model

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\widehat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

$\widehat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

Model-Free Learning

- Passive Reinforcement Learning
  - Directly evaluate values for each state under π
  - Policy evaluation
- Active Reinforcement Learning

Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

Direct Evaluation

- Goal: Compute values for each state under π

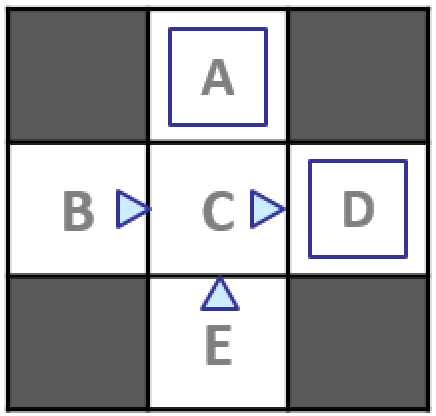
- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

- This is called direct evaluation

Direct Evaluation

▪

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

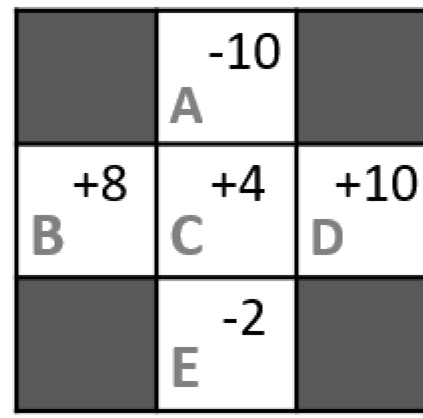
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Policy Evaluation

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$

  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!
- Key question: how can we do this update to V without knowing T and R
- In other words, how to take a weighted average without knowing the weights?

Policy Evaluation

- Sample-based policy evaluation
  - We want to improve our estimate of V by computing these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$$

  - Idea: Take samples of outcomes s' (by doing the action!) and averag

$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$

$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$

$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n}\sum_i sample_i$$

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values

- In this case:
  - Learner makes choices!
  - You actually take actions in the world and find out what happens…

Q-Value Iteration

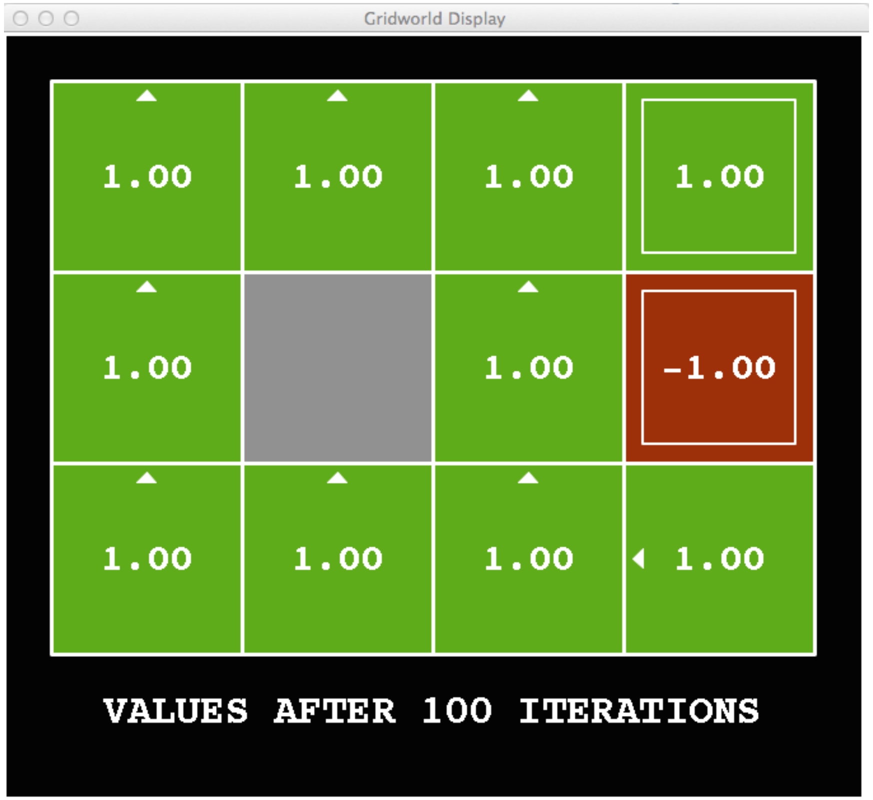
- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth $k+1$ values for all states:

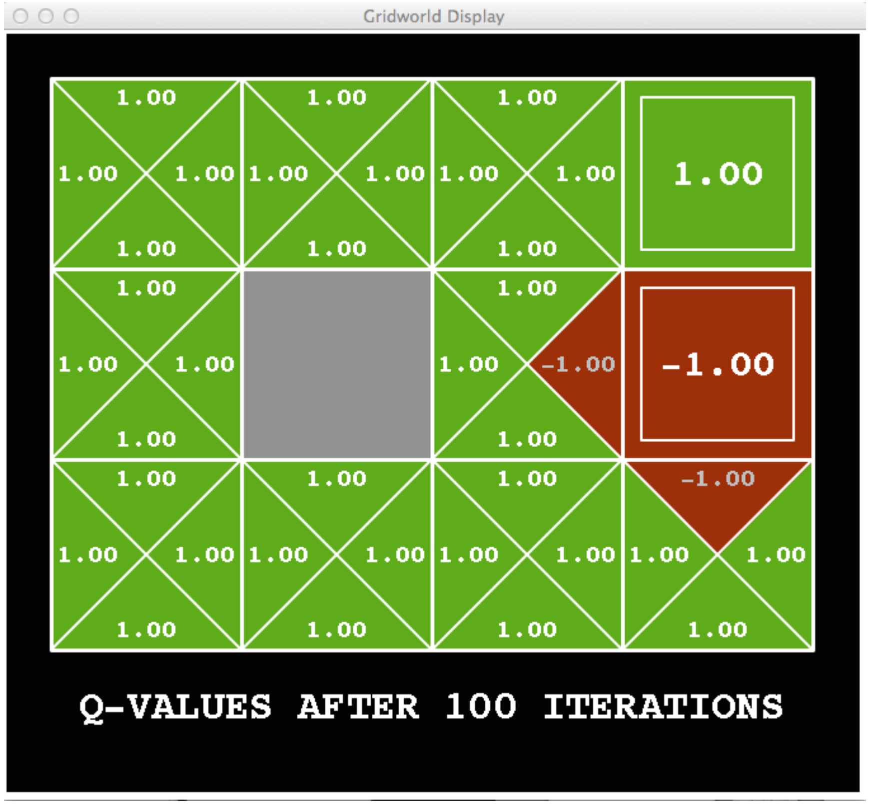
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

- But *Q-values* are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth $k+1$ q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Q-Value Iteration





Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

- Learn *Q(s,a)* values as you go
  - Receive a sample *(s, a, s', r)*
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

Q-Learning Properties

- Q-learning converges to optimal policy
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

Reinforcement Learning

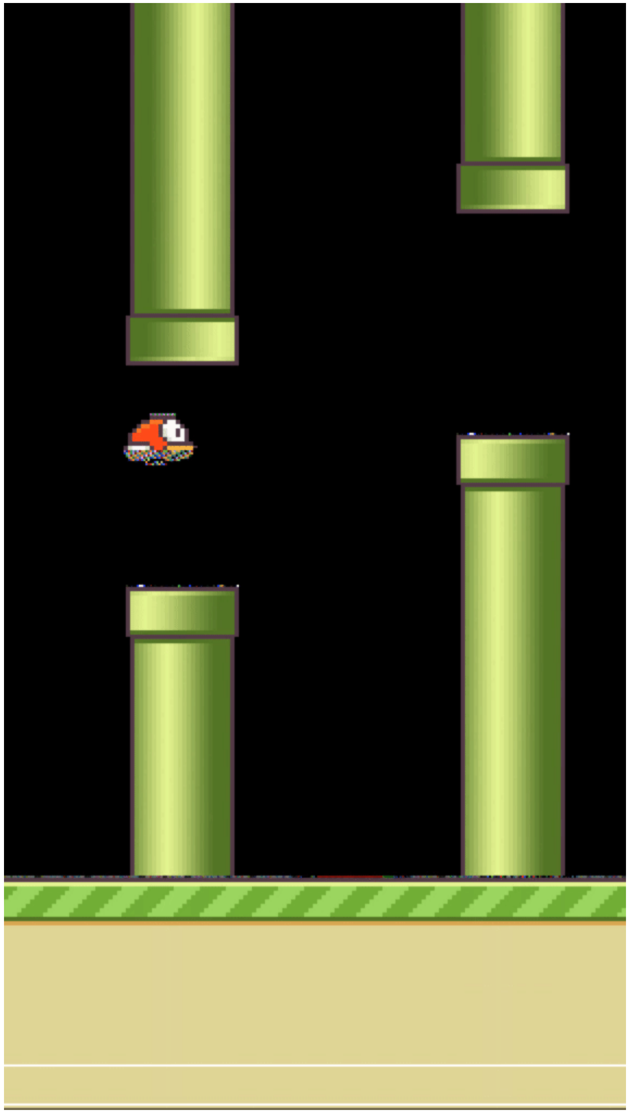
- Model-based RL
  - Build a transition model of the environment
  - Plan (e.g. by lookahead) using model
- Policy-based RL
  - Search directly for the optimal policy π^*
  - This is the policy achieving maximum future reward
- Value-based RL
  - Estimate the optimal value function $Q^*(s, a)$
  - This is the maximum value achievable under any policy

Deep Reinforcement Learning

- Use deep network to represent
  - Value function
  - Policy
  - Model
- Optimize value function / policy /model end-to-end
- Using stochastic gradient descent

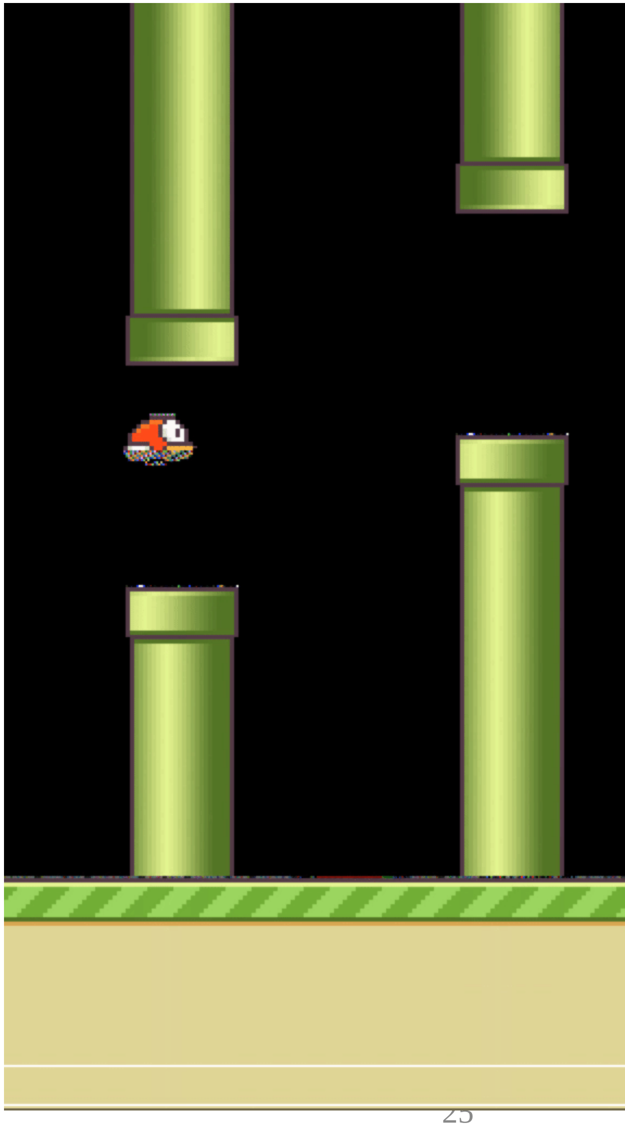
Example: DRL for Flappy Bird

-



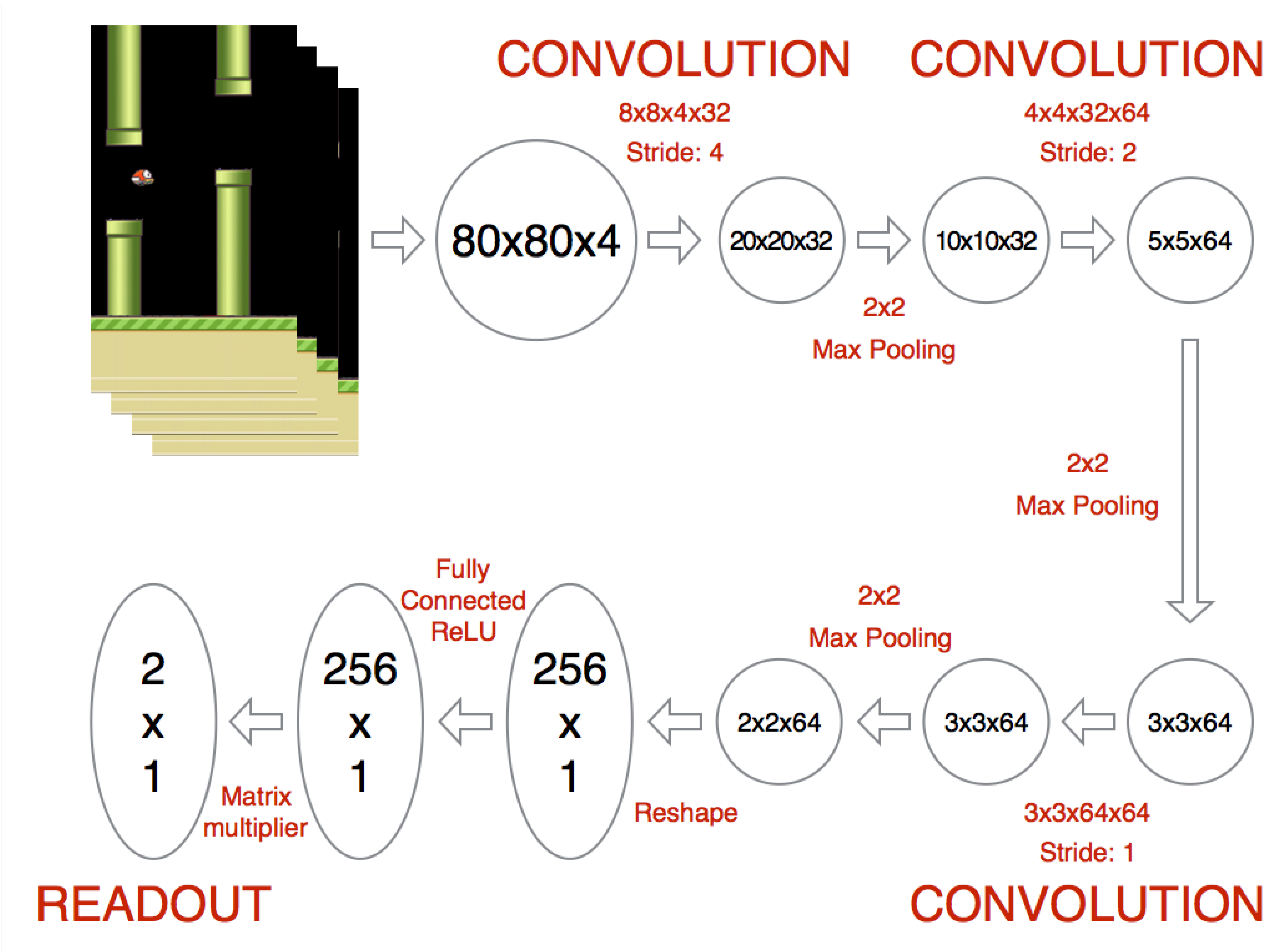
Example: DRL for Flappy Bird

- State space
  - Discretized vertical distance from lower pipe
  - Discretized horizontal distance from next pair of pipes
  - Life: Dead or Living
- Actions
  - Click
  - Do nothing
- Rewards
  - +1 if Flappy Bird still alive
  - -1000 if Flappy Bird is dead
- 6-7 hours of Q-learning



Example: DRL for Flappy Bird

▪



Readings

- Artificial Intelligence
  - Chapter 21.1-3

- Final Project
  - Due by July 6, 2022
- Final Exam
  - 2022年06月21日(14:00-16:00), 玉泉教4-304.