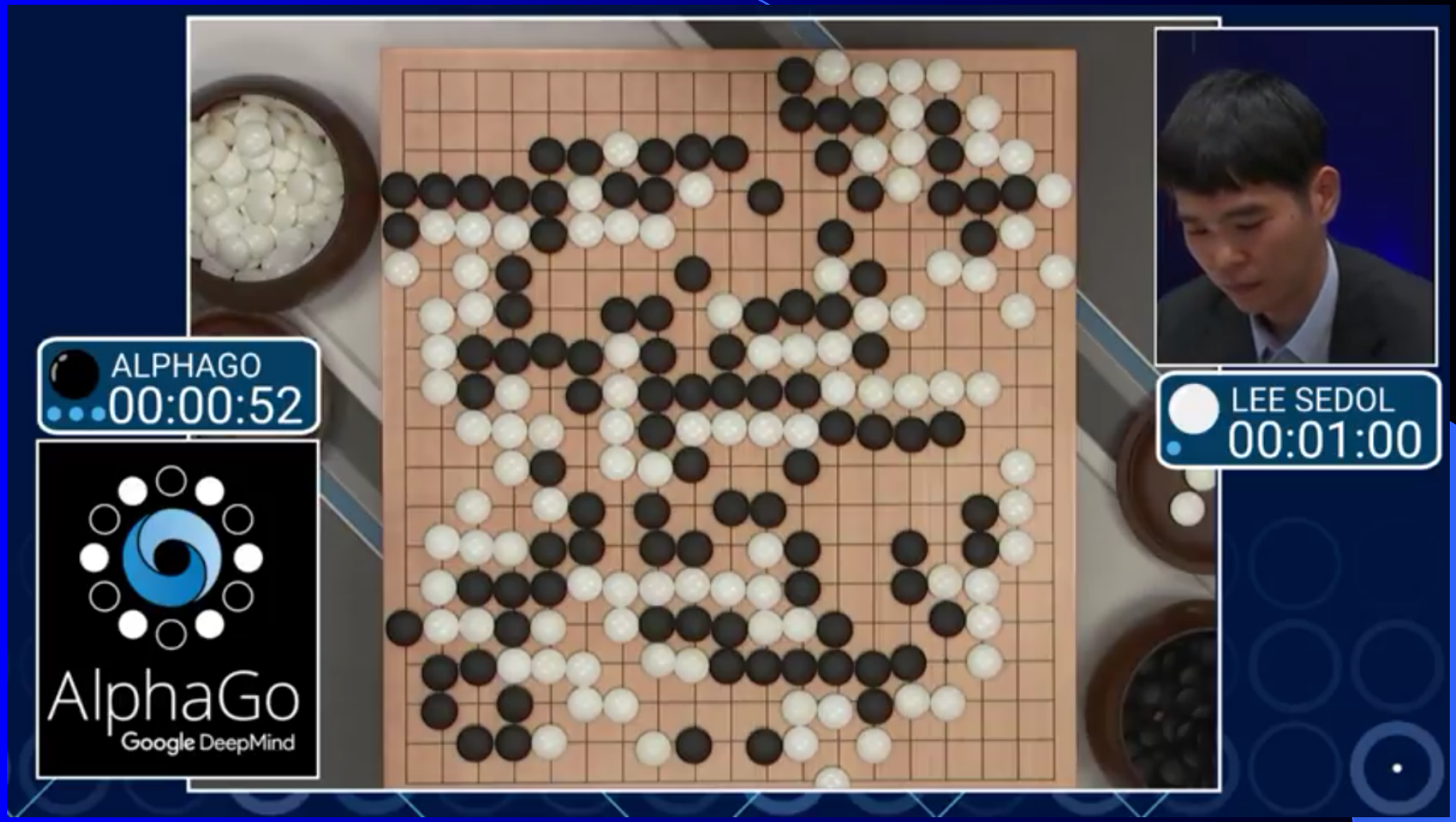
Reinforcement Learning



Reinforcement Learning (RL)

- Variation on Supervised Learning
- Exact target outputs are not given
- RL systems learn a mapping from states to actions by trial-and-error interactions with a dynamic environment
- Some variation of reward is given either immediately or after some steps
  - Chess
  - Path Discovery
- Deep RL (RL with deep neural networks) – Recently demonstrating tremendous potential
  - Especially nice for applications (e.g. games) where it is easy to generate data through simulation or self-play

RL Basics

- Agent (sensors and actions)
- Can sense state of Environment (position, etc.)
- Agent has a set of possible actions
- Actual rewards for actions from a state are usually delayed and do not give direct information about how best to arrive at the reward
- RL seeks to learn the optimal policy: which action the agent should take given a particular state, to achieve the agents eventual goal (e.g. maximize reward)
  - Trial and error approach – explore the action space and update action policy based on rewards

Learning a Policy

- Find optimal policy π: S -> A
- $a = \pi(s)$, where a is a possible action and s a state of the environment
- Which actions in a sequence leading to a goal should be rewarded, punished, etc.
- Exploration vs. Exploitation – To what extent should we explore new unknown states (hoping for better opportunities) vs. taking the best possible action based on knowledge already gained
- Markovian? – Do we just base action decision on current state or is there some memory of past states – Basic RL assumes Markovian processes (action outcome is only a function of current state, state fully observable) – Does not directly handle partially observable states (i.e. states which are not unambiguously identified) – can still approximate

Rewards

- Assume a reward function $r_s(a)$ – Common approach is a positive reward for entering a goal state (win the game, get a resource, etc.), negative for entering a bad state (lose the game, lose resource, etc.), 0 for all other transitions.
  - Some reward states are absorbing states (e.g. end of game)
- Discount factor γ: between 0 and 1, future rewards are discounted
- Value Function $V(s)$: The value of a state is the sum of the discounted rewards received when starting in that state and following a fixed policy until reaching a terminal state
- $V(s)$ also called the Discounted Cumulative Reward

Q-Learning

- Just try an action and see what state you end up in and what reward you get. Update the policy based on these results.
- Rather than find the value function of a state, find the value function of an (s,a) pair and call it the Q-value
- $Q(s,a)$ = Sum of discounted reward for doing a from s and following the current policy thereafter
  - $Q^*(s,a)$ represents the *optimal* Q function giving an optimal policy $\pi^*(s)$

Learning Algorithm for Q function

• Create a table with a cell for every (*state*, *action*) pair with zero or random initial values for the hypothesis of the Q values

• Iteratively try different actions from different states and update the table based on the following learning rule (for deterministic environment)

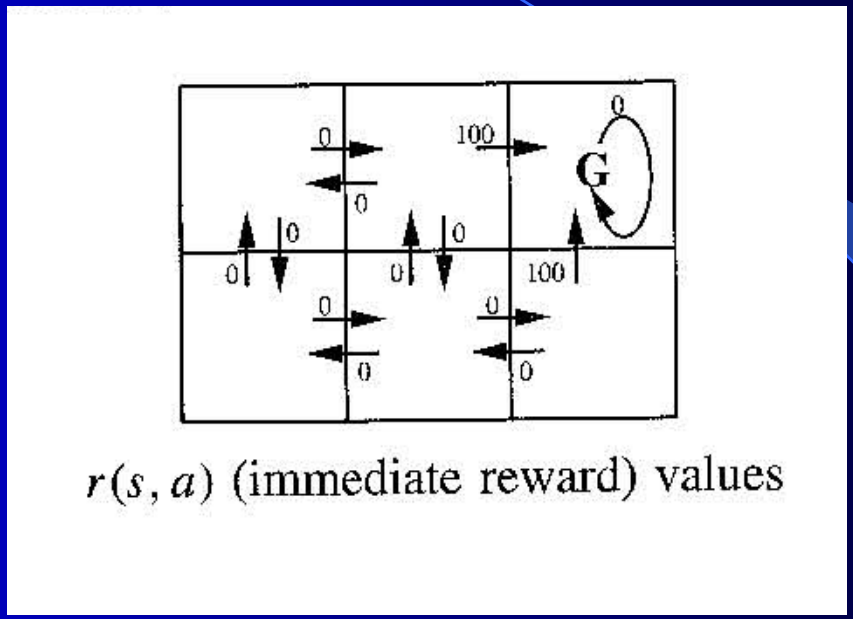
$$Q(s,a) = r_s(a) + \gamma \cdot \max_{a' \in A} \{Q(s',a')\}$$

• Note that this slowly adjusts the estimated Q-function towards the true Q-function. Iteratively applying this equation will converge to the optimal Q-function if

- The system can be modeled by a deterministic Markov Decision Process
  - action outcome depends only on current state (not on how you got there)
- r is bounded ($r_s(a) < c$ for all transitions)
- Each (*state*, *action*) transition is visited infinitely many times

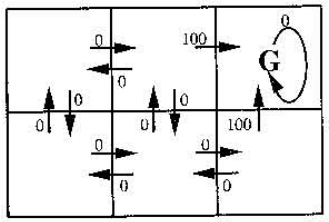
Learning Algorithm for Q function

- Until Convergence (Q-function changing very little) repeat:
  - Choose an arbitrary state *s*
  - Select any action *a* and execute (can use exploitation vs. exploration)
  - Update the Q-function table entry for (*s*, *a*)
- Monotonic Convergence – Once updated once, a Q-value can only increase
- We do not need to know the actual reward and state transition functions. Just sample them (Model-less).

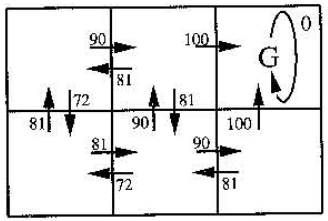


$r(s, a)$ (immediate reward) values

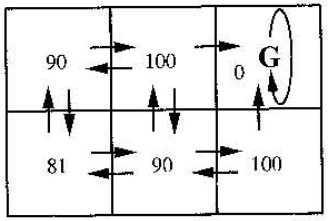
Assume all initial Q-values are 0 and discount factor is .9



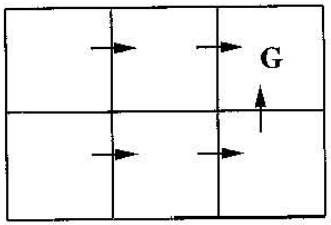
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values

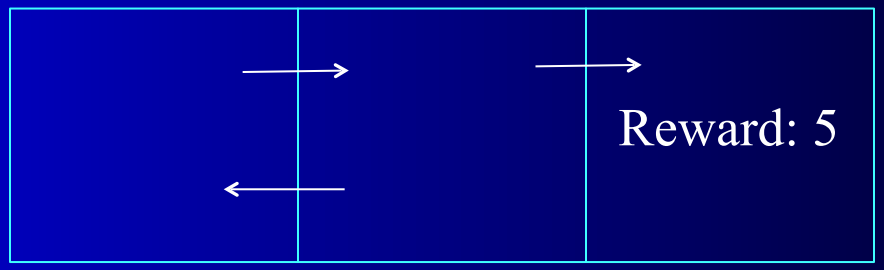


One optimal policy

FIGURE 13.2
A simple deterministic world to illustrate the basic concepts of Q-learning. Each grid square represents a distinct state, each arrow a distinct action. The immediate reward function, $r(s, a)$ gives reward 100 for actions entering the goal state **G**, and zero otherwise. Values of $V^*(s)$ and $Q(s, a)$ follow from $r(s, a)$, and the discount factor $\gamma = 0.9$. An optimal policy, corresponding to actions with maximal Q values, is also shown.

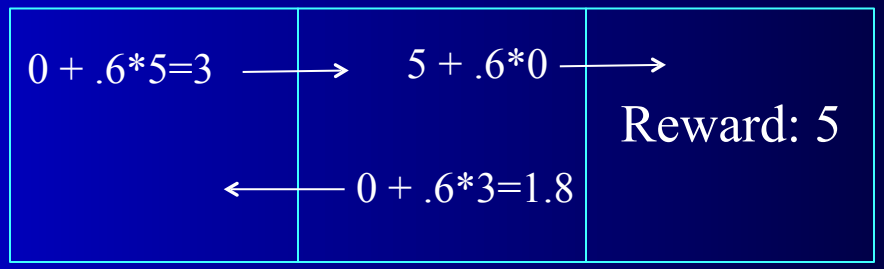
Q-Learning Challenge Question

- Assume the deterministic 3 state world below (each cell is a state) where the immediate reward is 0 for entering all states, except the rightmost state, for which the reward is 5, and which is an absorbing state. The only actions are move right and move left (only one of which is available from the border cells). Assume a discount factor of .6, and all initial Q-values of 0. Give the final optimal Q values for each action in each state and describe the optimal policy.



Q-Learning Challenge Question

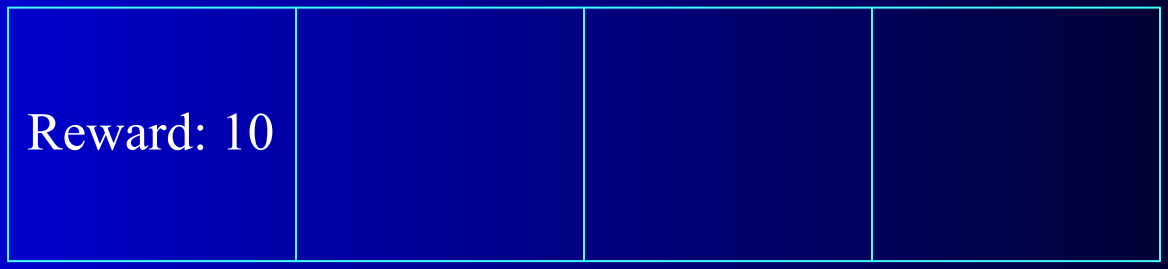
- Assume the deterministic 3 state world below (each cell is a state) where the immediate reward is 0 for entering all states, except the rightmost state, for which the reward is 5, and which is an absorbing state. The only actions are move right and move left (only one of which is available from the border cells). Assume a discount factor of .6, and all initial Q-values of 0. Give the final optimal Q values for each action in each state and describe the optimal policy.



Optimal Policy: "Choose the Right!!"

Q-Learning Homework

- Assume the deterministic 4 state world below (each cell is a state) where the immediate reward is 0 for entering all states, except the leftmost state, for which the reward is 10, and which is an absorbing state. The only actions are move right and move left (only one of which is available from the border cells). Assume a discount factor of .8, and all initial Q-values of 0. Give the final optimal Q values for each action in each state and describe an optimal policy.



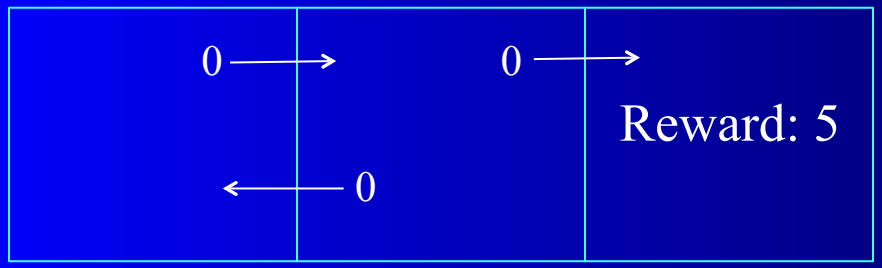
Exploration vs Exploitation

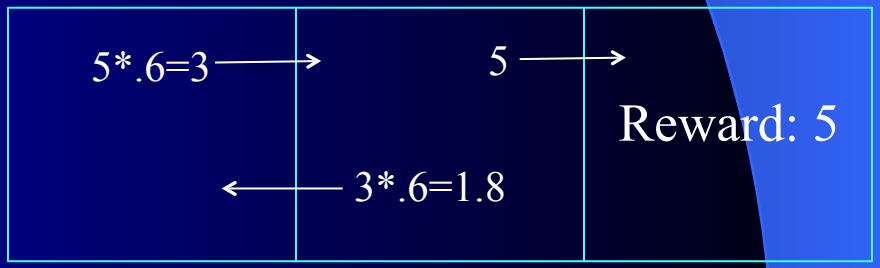
- Choosing action during learning (Exploitation vs. Exploration) – 2 Common approaches
- Softmax:
  - Can increase *k* (constant > 1) over time to move from exploration to exploitation
- ε-greedy: With probability ε randomly choose any action, else greedily take the action with the best current Q value.
  - Start ε at 1 and then decrease with time

Episodic Updates

- Note that much efficiency could be gained if you worked back from the goal state (like you did on the challenge question). However, with model free learning, we do not know where the goal states are, or what the transition function is, or what the reward function is. We just sample things and observe. If you do know these functions then you can simulate the environment and come up with more efficient ways to find the optimal policy with standard DP algorithms (e.g. policy iteration).

- One thing we can do for Q-learning is rather than start with a new state each step, continue going from state to state until reaching a reward state (episode). Then, propagate the discounted Q-function update all the way back to the initial starting state, allowing multiple updates. This can speed up learning at a cost of memory. This approach is often used but it is not the "true" learning algorithm





Deep Q-Learning Example

- Deep convolutional network trained to learn Q function
- To overcome Markov limitation (partially observed states) the function approximator can be given an input made up of *m* consecutive proceeding states (Atari and Alpha zero approach) or have memory (e.g. recurrent NN), etc.
  - Early Q learning used linear models or shallow neural networks
- Using deep networks as the approximator has been shown to lead to accurate stable learning
  - Learns all 49 classic Atari games with the only inputs being pixels from the screen and the score, at above standard human playing level with no tuning of hyperparameters.
  - Alpha-Zero

Replace Q-table with a Function Approximator

- Train a function approximator (e.g. ML model) to output approximate Q-values
  - Use an MLP/DNN in place of the lookup table, where it is trained with the inputs *s* and *a* with the current Q-value as output
  - Avoids huge or infinite lookup tables (real values, etc.)
  - Allows generalization from all states, not just those seen during training
  - We are not training with the optimal Q-values (we don't know them)
  - Initial Q-values are random based on initial random weights
  - For each update the training error is the difference between the network's current Q-value output (generalization) and the updated Q-value expectation from our standard update equation above
    - If current Q-value is .4 and the updated Q-value is .7, then output error propagated back is .3
  - Converged when Q-values are no longer changing much

Reinforcement Learning Summary

- Learning can be slow even for small environments
  - Can be great for tasks where trial and error is reasonable or can be done through simulation
- Large and continuous spaces can be handled using a function approximator (e.g. MLP)
- Deep Q learning: States and policy represented by a deep neural network – more in CS 472
- Suitable for tasks which require state/action sequences
  - RL not used for choosing best pizza, but could be used to *discover* the steps to create the best or a better pizza
  - But need mechanism to efficiently explore (e.g. simulator, self-play, etc.)
- With RL we don't need labeled data. Just experiment and learn!