# Reinforcement Learning

*material from Marie desJardin, Lise Getoor, Jean-Claude Latombe, Daphne Koller, Stuart Russell, Dawn Song, Mark Hasegawa-Johnson, Svetlana Lazebnik, Pieter Abbeel, Dan Klein, Lisa Torrey*


1

## Bookkeeping

- HW4 due 11/20
- Project phase II (code final) now due 12/7
- Final paper now due 12/15
- We will not use 12/17
- No office hours Thursday
- Last time
  - "Probabilistic planning"—learning action policies
  - Value iteration (lots); policy iteration (some)
- Today
  - Reinforcement learning
  - Project work

2

## Review: What is ML?

- ML is a way to get a computer to do things without having to explicitly describe what steps to take.
- By giving it **examples** (training data)
- Or by giving it **feedback**
- It can then look for patterns which explain or predict what happens.
- The learned system of beliefs is called a **model**.

3

## Review: Representation

- A learning system must have a **representation or model** of what is being learned.
- This is what changes based on experience.
- In a machine learning system this may be:
  - A mathematical model or formula
  - A set of rules
  - A decision tree
  - A policy
  - Or some other form of information

6

# Review: Formalizing Agents

- Given:
  - A state space S
  - A set of actions $a_1, \ldots, a_k$ including their results
  - Reward value at the **end of each trial** (series of action) (may be positive or negative)
- Output:
  - A **mapping from states to actions to take**
  - Which is a **policy**, π

7

# Learning Without a Model

- We saw how to learn a value function and/or a policy from a transition model
- What if we don't have a transition model?
- Idea #1: Build one
  - Explore the environment for a long time
  - Record all transitions
  - Learn the transition model
  - Apply value iteration/policy iteration
  - Slow, requires a lot of exploration, no intermediate learning
- Idea #2: Learn a value function (or policy) directly from interactions with the environment, while exploring

8

## Reinforcement Learning

- We often have an agent which has a **task** to perform
  - It takes some actions in the world
  - At some later point, gets feedback on how well it did
  - The agent performs the same task repeatedly
- This problem is called **reinforcement learning**:
  - The agent gets positive reinforcement for tasks done well
  - And gets negative reinforcement for tasks done poorly
  - Must somehow figure out which actions to take next time

9

## Characteristics of Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d data)
  - Agent's actions affect the subsequent data it receives

10

# Reinforcement learning

- It is a family of problems
  - Sequential decision making

Game playing

Self-driving car

Conversational System

Slide: Hongning Wang, CS@UVA

11

# Reinforcement learning

- A typical (narrow) view of the problem formulation

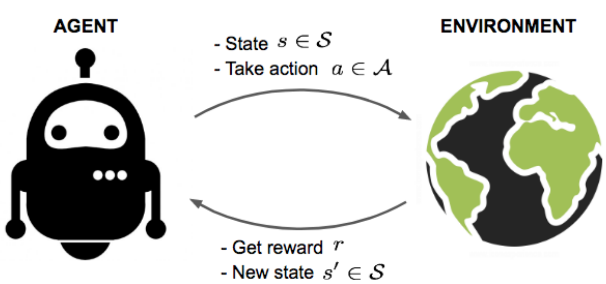



*Image credit: Lil'Log*

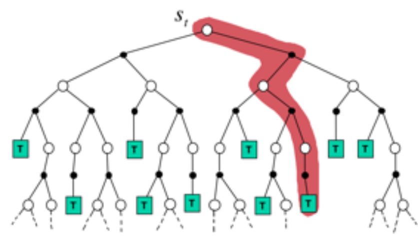
Slide: Hongning Wang, CS@UVA

12

# Reinforcement learning

- It is a family of solutions
  - Taking a series of actions to maximum cumulative return

Planning




~~Planning while~~ learning
Reinforcem
ent

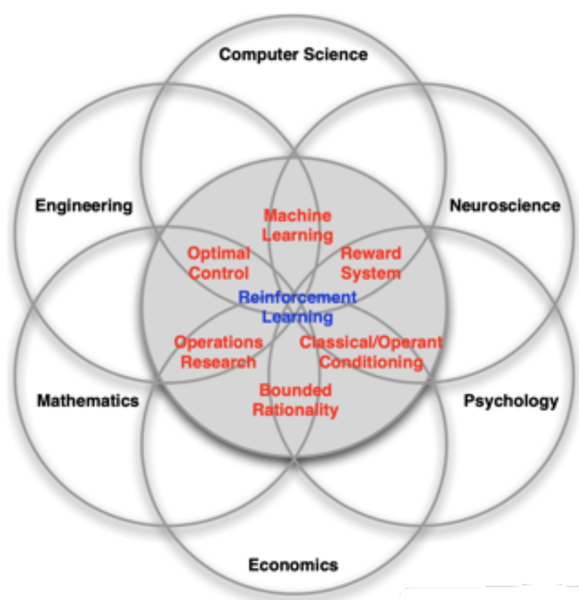
*Image credit: David Silver, "Model-Free Prediction"*

Slide: Hongning Wang, CS@UVA

13

# Summary: reinforcement learning

- It is a family of problems
  - Sequential decision making
- It is a family of solutions
  - Planning and learning
- It is a collection of fields that study the problems and solutions

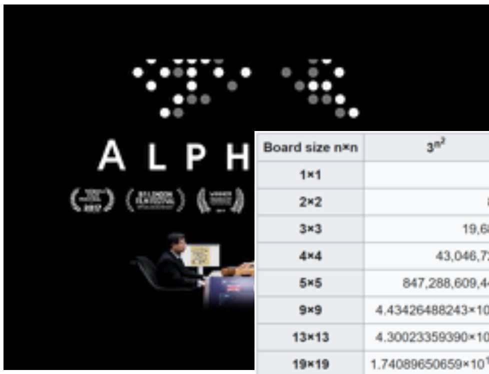



Slide Hongning Wang, Image David Silver

14

# Why reinforcement learning

- Sequential decision making is everywhere




| Board size n×n | $3^{n^2}$ | Percent legal | $L$ (legal positions) (A094777)[11] |
|---|---|---|---|
| 1×1 | 3 | 33.33% | 1 |
| 2×2 | 81 | 70.37% | 57 |
| 3×3 | 19,683 | 64.40% | 12,675 |
| 4×4 | 43,046,721 | 56.49% | 24,318,165 |
| 5×5 | 847,288,609,443 | 48.90% | 414,295,148,741 |
| 9×9 | $4.43426488243\times10^{38}$ | 23.44% | $1.03919148791\times10^{38}$ |
| 13×13 | $4.30023359390\times10^{80}$ | 8.66% | $3.72497923077\times10^{79}$ |
| 19×19 | $1.74089650659\times10^{172}$ | 1.20% | $2.08168199382\times10^{170}$ |

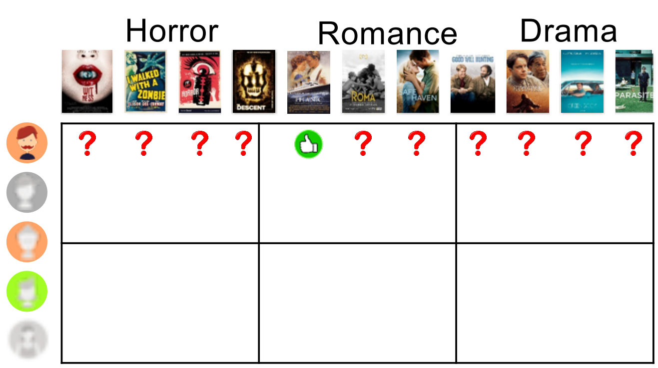

2016 Source: https://en.wikipedia.org/wiki/Go_and_mathematics 1997

*Slide: Hongning Wang, CS@UVA*

15

# Why reinforcement learning

- Sequential decision making is challenging
  - Huge unknown search space

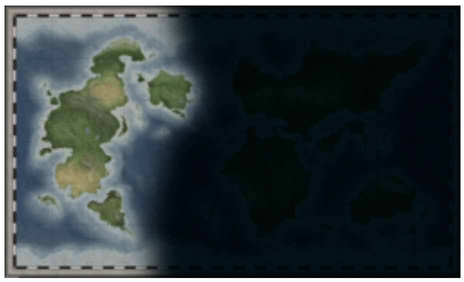



*Slide: Hongning Wang, CS@UVA*

16

## Why reinforcement learning

- Sequential decision making is challenging
  - Huge unknown search space
    - Supervised ML: generalize to unseen
    - RL: what to generalize



Slide: Hongning Wang, CS@UVA

17

## Reinforcement Learning (RL)

- RL algorithms attempt to find a **policy**
  - Maximizing cumulative reward for the agent over the course of the problem
- Typically represented by a **Markov Decision Process**
- RL differs from supervised learning:
  - Correct input/output pairs are never presented
  - Sub-optimal actions never explicitly corrected

18

## Typical Applications

- Robotics
  - Helicopter control
  - Robo-soccer
- Board games
  - Checkers
  - Backgammon
  - Go/Atari
- Scheduling
  - Dynamic channel allocation
  - Inventory problems

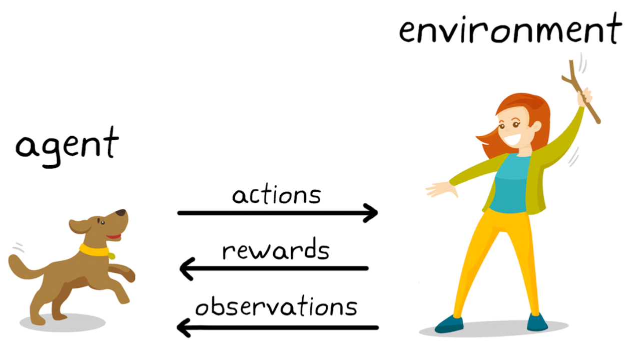
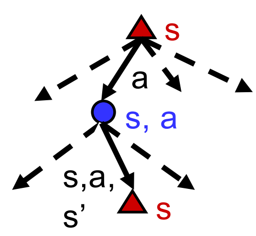



Image: https://www.mathworks.com/discovery/reinforcement-learning.html

19

## Recap: Defining MDPs

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




- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

20

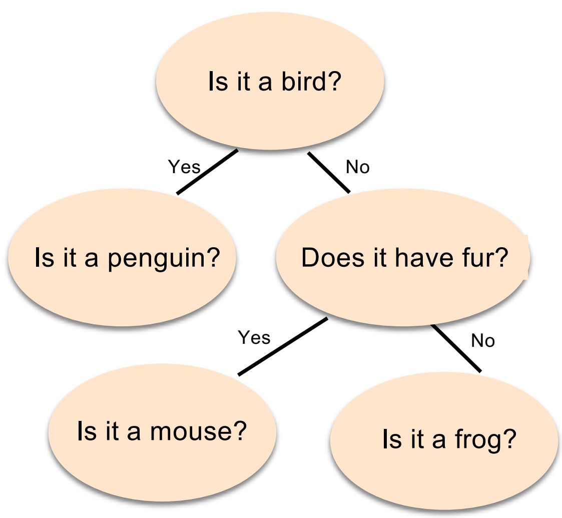
# Animals Game

- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.
- Computer: **Tell me a question that distinguishes a frog from a mouse.**
- Human: Does it have fur?
- Computer: What is the answer for a frog?
- Human: No.

21

# Animals Behind the Scene

Computer: Is it a bird?
Human: No
Computer: Is it a frog?
Human: No
Computer: What is it?
Human: A mouse
Computer: Tell me a question that distinguishes a frog from a mouse.
Human: Does it have fur?
Computer: What is the answer for a frog?
Human: no




After several rounds…

22

## Animals Guessing Game Architecture

- All of the parts of ML Architecture:
  - The Representation is a sequence of questions and pairs of yes/no answers (called a binary decision tree).
  - The Actor "walks" the tree, interacting with a human; at each question it chooses whether to follow the "yes" branch or the "no" branch.
  - The Critic is the human player telling the game whether it has guessed correctly.
  - The Learner elicits new questions and adds questions, guesses and branches to the tree.

23

## Reinforcement Learning

- This is a simple form of **Reinforcement Learning**
- Feedback is at the end, on a **series** of actions.
- Very early concept in Artificial Intelligence!
- Arthur Samuels' checker program was a simple reinforcement based learner, initially developed in 1956.
- In 1962 it beat a human checkers master.

www-03.ibm.com/ibm/history/ibm100/us/en/icons/ibm700series/impacts/

24

## Reward in reinforcement learning

- A scalar feedback signal about the taken action
  - Suggest good/bad **immediate** consequence of the action
    - Score in Atari game
    - User clicks/purchase in a recommender system
    - Change of black-box function value
  - Delayed feedback
    - GO game
    - Generate a sentence in chat-bot
  - Goal of learning – maximize cumulative rewards
    - Reward hypothesis: *"All goals can be described by the maximization of expected cumulative reward."*

*Slide: Hongning Wang, CS@UVA*

25

## More about rewards

- A reward $R_t$ is a scalar feedback signal
- Indicates how well agent is doing at step $t$
- The agent's job is to maximize cumulative reward

**Reinforcement learning is based on the reward hypothesis:**

- "All goals can be described by the maximization of expected cumulative reward"
- (Do we believe this?)

*Slide: David Silver*

26

# RL inputs and outputs

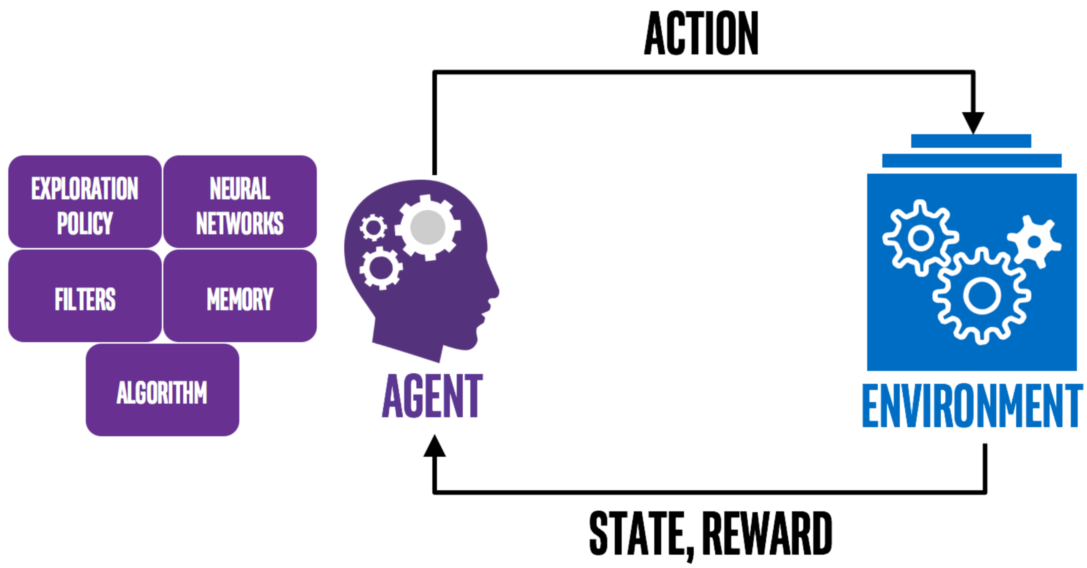
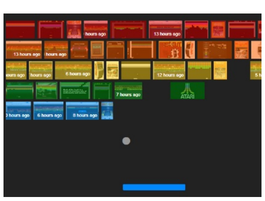



*Image: Hongning Wang, CS@UVA*

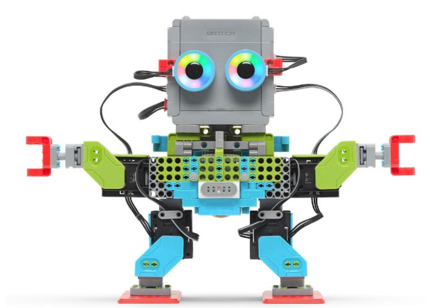
27

# How to take an action

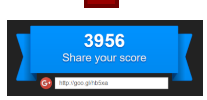
- With respect to the current observation



Observation $o_t$



Action $a_t$



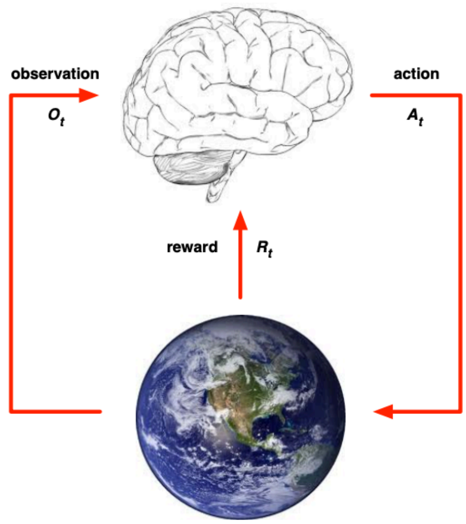

Reward $r_t$

*Slide: Hongning Wang, CS@UVA*

28

## Agent and environment

- At each step t the agent:
  - Executes action $A_t$
  - Receives observation $O_t$
  - Receives reward $R_t$
- The environment:
  - Receives action $A_t$
  - Emits observation $O_{t+1}$
  - Emits scalar reward $R_{t+1}$
- t increments at environment step




*Slide: David Silver*

29

## Reinforcement Learning (cont.)

- Goal: agent acts in the world to maximize its rewards
- Agent has to figure out what it did that made it get that reward/punishment
  - This is known as the **credit assignment problem**
- RL can be used to train computers to do many tasks
  - Backgammon and chess playing
  - Job shop scheduling
  - Controlling robot limbs

30

## Procedural Learning

- Learning how to act to accomplish goals
  - **Given**: Environment that contains rewards
  - **Learn**: A policy for acting
- Important differences from classification
  - You don't get examples of correct answers
  - You have to try things in order to learn

31

## RL as Operant Conditioning

- RL shapes behavior using reinforcement
  - Agent takes **actions** in an environment (in episodes)
  - Those actions **change the state** and trigger **rewards**
- Through experience, an agent learns a policy for acting
  - Given a state, choose an action
  - Maximize cumulative reward during an episode
- Interesting things about this problem
  - Requires solving credit assignment
    - What action(s) are responsible for a reward?
  - Requires both exploring and exploiting
    - Do what looks best, or see if something else is really best?

32

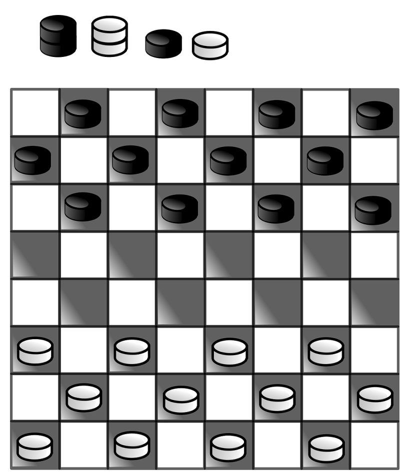
# Types of Reinforcement Learning

- Search-based: evolution directly on a policy
  - E.g. genetic algorithms
- Model-based: build a model of the environment
  - Then you can use dynamic programming
  - Memory-intensive learning method
- Model-free: learn a policy without any model
  - Temporal difference methods (TD)
  - Requires limited episodic memory (though more helps)

33

# Simple Example

- Learn to play checkers
  - Two-person game
  - 8x8 boards, 12 checkers/side
  - relatively simple set of rules: http://www.darkfish.com/checkers/rules.html
  - Goal is to eliminate all your opponent's pieces



https://pixabay.com/en/checker-board-black-game-pattern-29911

34

## Representing Checkers

- First we need to represent the game
- To completely describe one step in the game you need
  - A representation of the game board.
  - A representation of the current pieces
  - A variable which indicates whose turn it is
  - A variable which tells you which side is "black"
- There is no history needed
- A look at the current board setup gives you a complete picture of the state of the game

which makes it a ____ problem?

35

## Representing Rules

- Second, we need to represent the rules
- Represented as a **set of allowable moves** given board state
  - If a checker is at row x, column y, and row x+1 column y±1 is empty, it can move there.
  - If a checker is at (x,y), a checker of the opposite color is at (x+1, y+1), and (x+2,y+2) is empty, the checker must move there, and remove the "jumped" checker from play
- There are additional rules, but all can be expressed in terms of the state of the board and the checkers
- Each rule includes the outcome of the relevant action in terms of the state

36

## What Do We Want to Learn?

- Given
  - A description of some state of the game
  - A list of the moves allowed by the rules
  - **What move should we make?**
- Typically more than one move is possible
  - Need strategies, heuristics, or hints about what move to make
  - **This is what we are learning**
- We learn **from** whether the game was won or lost
  - Information to learn from is sometimes called "training signal"

39

## Simple Checkers Learning

- Can represent some heuristics in the same formalism as the board and rules
  - If there is a legal move that will create a king, take it.
    - If checkers at (7,y) and (8,y-1) or (8,y+1) is free, move there.
  - If there are two legal moves, choose the one that moves a checker farther toward the top row
    - If checker(x,y) and checker(p,q) can both move, and x>p, move checker(x,y).
  - But then each of these heuristics needs some kind of priority or **weight**.

40

# Formalization for RL Agent

- Given:
  - A state space S
  - A set of actions $a_1$, …, $a_k$ including their results
  - **A set of heuristics for resolving conflict among actions**
  - Reward value at the end of each trial (series of action) (may be positive or negative)
- Output:
  - A policy (a mapping from states to preferred actions)

41

# Learning Agent

- The general algorithm for this learning agent is:
  - Observe some state
  - If it is a terminal state
    - Stop
    - If won, **increase** the weight on **all** heuristics used
    - If lost, **decrease** the weight on **all** heuristics used
  - Otherwise choose an action from those possible in that state, using heuristics to select the preferred action
  - Perform the action

42

## Policy

- A complete mapping from states to actions
  - There must be an action for each state
  - There may be more than one action
  - Not necessarily optimal
- The goal of a learning agent is to **tune** the policy so that the preferred action is optimal, or at least good.
  - Analogous to training a classifier
- Checkers
  - Trained policy includes all legal actions, with **weights**
  - "Preferred" actions are **weighted up**

43

## Approaches

- Learn policy directly: Discover function mapping from states to actions
  - Could be directly learned values
    - Ex: Value of state which removes last opponent checker is +1.
  - Or a heuristic function which has itself been trained
- Learn utility values for states (value function)
  - Estimate the value for each state
  - Checkers:
    - How happy am I with this state that turns a piece into a king?

44

## Value Function

- The agent knows what state it is in
- It has actions it can perform in each state
- Initially, don't know the value of any of the states
- If the outcome of performing an action at a state is deterministic, then the agent can update the utility value U() of states:
  - U(oldstate) = reward + U(newstate)
- The agent learns the utility values of states as it works its way through the state space

45

## Learning States and Actions

- A typical approach is:
- At state S choose, some action A ← How?
- Taking us to new State $S_1$
  - If $S_1$ has a positive value: increase value of A at S.
  - If $S_1$ has a negative value: decrease value of A at S.
  - If $S_1$ is new, initial value is unknown: value of A unchanged.
- One complete learning pass or **trial** eventually gets to a terminal, deterministic state. (E.g., "win" or "lose")
- Repeat until? Convergence? Some performance level?

46

# Selecting an Action

- Simply choose action with highest (current) expected utility?
- Problem: each action has two effects
  - Yields a **reward** on current sequence
  - Gives **information** for learning future sequences
- Trade-off: immediate good for long-term well-being
  - Like trying a shortcut: might get lost, might find quicker path
- **Exploration** vs. **exploitation**
  - Exploration finds more information about the environment
  - Exploitation exploits known information to maximize reward
  - It is usually important to explore as well as exploit

47

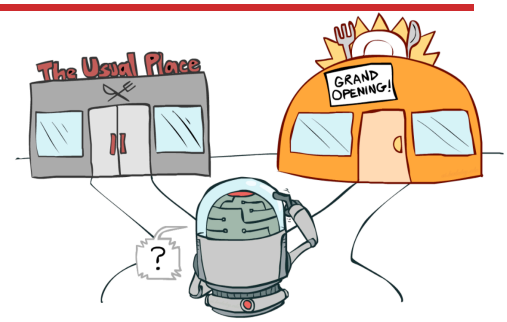
# Exploration vs. Exploitation

- Problem with naïve reinforcement learning:
  - What action to take?
  - **Best apparent action, based on learning to date** } Exploitation
    - Greedy strategy
    - Often prematurely converges to a suboptimal policy!
  - **Random (or unknown) action** } Exploration
    - Will cover entire state space
    - Very expensive and slow to learn!
    - When to stop being random?
- Balance exploration (try random actions) with exploitation (use best action so far)

48

## Exploration vs. Exploitation

- Restaurant Selection
  - Exploitation: Go to your favorite restaurant
  - Exploration: Try a new restaurant
- Online Advertisements
  - Exploitation: Show the most successful advert
  - Exploration: Show a different advert
- Navigation
  - Exploitation: Walk to class
  - Exploration: Try a possible shortcut through a building
- Game Playing
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move




*Slide: David Silver*
*Image: Berkeley AI course*

49

## More on Exploration

- Agent may sometimes choose to explore suboptimal moves in hopes of finding better outcomes
  - Only by visiting all states frequently enough can we guarantee learning the true values of all the states
- When the agent is **learning**, ideal would be to get accurate values for all states
  - Even though that may mean getting a negative outcome
- When agent is **performing**, ideal would be to get optimal outcome
- A learning agent should have an **exploration policy**

50

# Exploration Policy

- Wacky approach (exploration): act randomly in hopes of eventually exploring entire environment
  - Choose any legal checkers move
- Greedy approach (exploitation): act to maximize utility using current estimate
  - Choose moves that have in the past led to wins
- Reasonable balance: act more wacky (exploratory) when agent has little idea of environment; more greedy when the model is close to correct
  - Suppose you know no checkers strategy?
  - What's the best way to get better?

51

# Example: N-Armed Bandits

- A row of slot machines
- Which to play and how often?
- State Space is a set of machines
  - Each has cost, payout, and percentage values
- Action is pull a lever.
- Each action has a positive or negative result
  - …which then adjusts the perceived utility of that action (pulling that lever)

| | | |
|---|---|---|
| ¢25 | ¢95 | $10 |
| $100 | $200 | $900 |
| 0.1% | 0.6% | 10% |

52

## N-Armed Bandits Example

- Each action initialized to a standard payout
- Result is either some cash (a win) or none (a lose)
- **Exploration**: Try things until we have estimates for payouts
- **Exploitation**: When we have some idea of the value of each action, choose the best.

After some # of successful trials, or with some statistical **confidence**, or when our value function isn't changing (much), or…

- Clearly this is a heuristic.
- No proof we ever find the best lever to pull!
  - The more exploration we can do the better our model
  - But the higher the cost over multiple trials

53

## Mathematical Model - MDP

- Markov decision processes
- S - set of states
- A - set of actions
- $\delta$ - Transition probability
- R - Reward function

54

# Types of Reinforcement Learning

- Search-based: evolution directly on a policy
  - E.g. genetic algorithms
- Model-based: build a model of the environment
  - Then you can use dynamic programming
  - Memory-intensive learning method
- Model-free: learn a policy without any model
  - Temporal difference methods (TD)
  - Requires limited episodic memory (though more helps)

55

# Types of Model-Free RL

- Actor-critic learning
  - The TD version of Policy Iteration
- Q-learning
  - The TD version of Value Iteration
  - This is the most widely used RL algorithm

56

## Q-Learning: Definitions

- Current state: ***s***
- Current action: ***a***
- Transition function: ***δ(s, a)*** = ***s'***
- Reward function: ***r(s, a)*** ϵ ***R***
- Policy ***π(s)*** = ***a***

Markov property: this is independent of previous states given current state

In classification we'd have examples ***(s, π(s))*** to learn from

- ***Q(s, a)*** ≈ value of taking action ***a*** from state ***s***

57

## The Q-function

- **Q(s, a) estimates the discounted cumulative reward**
  - Starting in state s
  - Taking action a
  - Following the current policy thereafter
- Suppose we have the optimal Q-function
  - What's the optimal policy in state s?
  - The action $argmax_b Q(s, b)$
- But we don't have the optimal Q-function at first
  - Let's act as if we do
  - And update it after each step so it's closer to optimal
  - Eventually it will be optimal!

***Q(s, a)*** ≈ value of taking action ***a*** from state ***s***

58

## Q-Learning: Updates

- The basic update equation

$$Q(s,a) \longleftarrow r(s,a) + \max_b Q(s',b)$$

- With a discount factor to give later rewards less impact

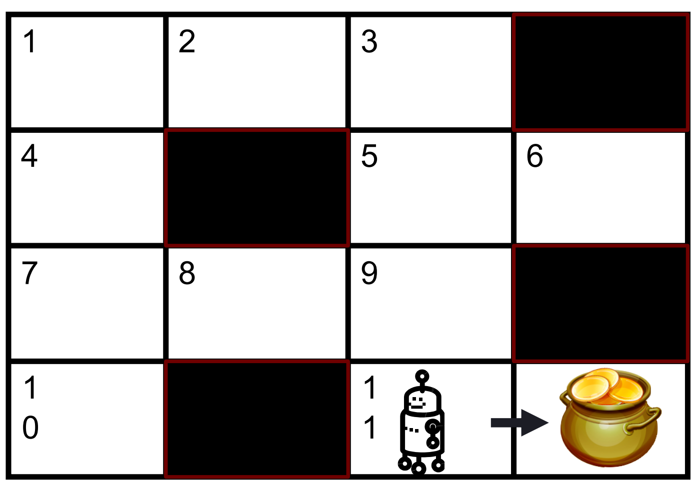
$$Q(s,a) \longleftarrow r(s,a) + \gamma \max_b Q(s',b)$$

- With a learning rate for non-deterministic worlds

$$Q(s,a) \longleftarrow [1-\alpha]Q(s,a) + \alpha[r(s,a) + \gamma \max_b Q(s',b)]$$

59

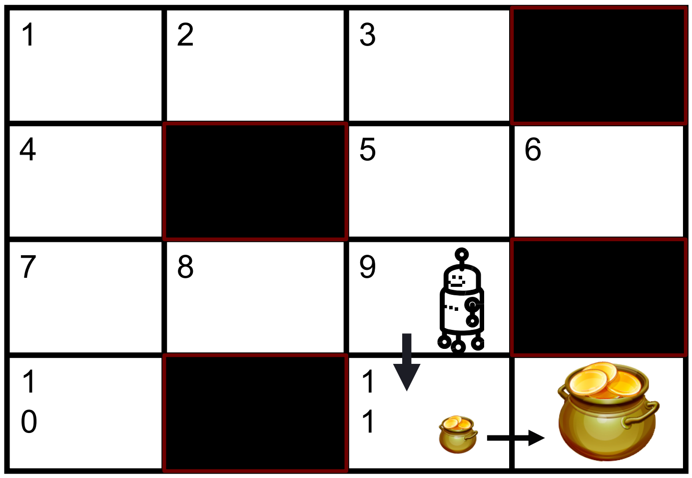
## Q-Learning: Update Example




$Q(s_{11}, a_{\rightarrow}) =$ 

60

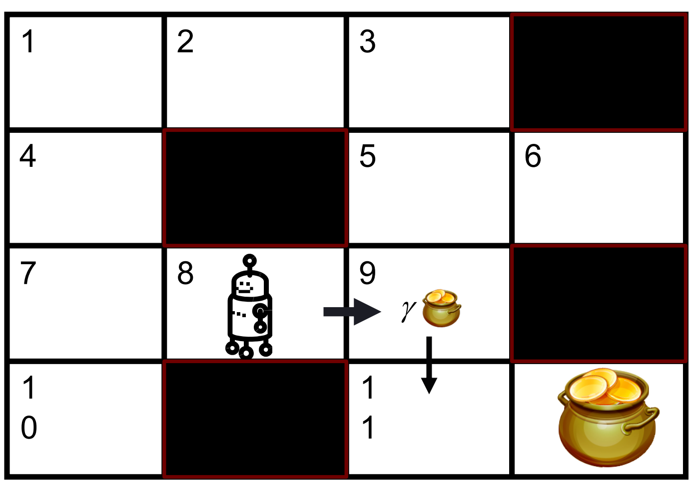
## Q-Learning: Update Example




$Q(s_9, a_\downarrow) = 0 + \gamma$

61

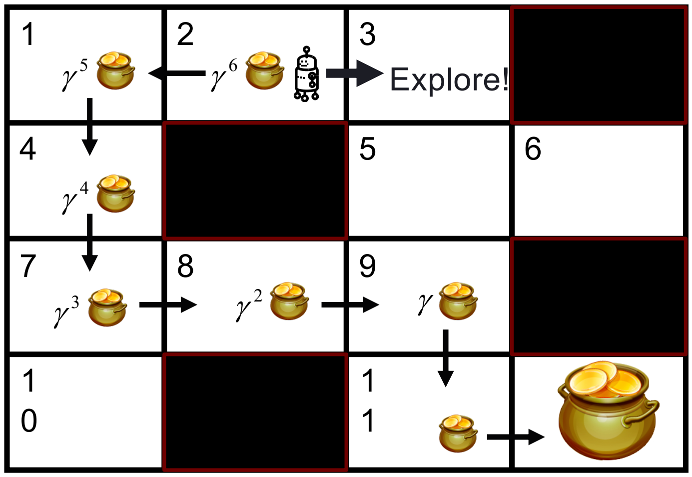
## Q-Learning: Update Example




$Q(s_8, a_\rightarrow) = 0 + \gamma^2$

62

# The Need for Exploration




$$\arg\max Q(s_2, a) = \leftarrow$$
$$best = \rightarrow$$

63

# RL Summary 1:

- **Reinforcement learning systems**
  - Learn **series** of actions or decisions, rather than a single decision
  - Based on feedback given at the end of the series
- A reinforcement learner has
  - A goal
  - Carries out trial-and-error search
  - Finds the best paths toward that goal

64

## Exploration/Exploitation

- Can't always choose the action with highest Q-value
  - The Q-function is initially unreliable
  - Need to explore until it is optimal
- Most common method: ε-greedy
  - Take a random action in a small fraction of steps (ε)
  - Decay ε over time
- There is some work on optimizing exploration
  - Kearns & Singh, ML 1998
  - But people usually use this simple method

65

## Q-Learning: Convergence

- Under certain conditions, Q-learning will converge to the correct Q-function
  - The environment model doesn't change
  - States and actions are finite
  - Rewards are bounded
  - Learning rate decays with visits to state-action pairs
  - Exploration method would guarantee infinite visits to every state-action pair over an infinite training period

66

## Challenges in Reinforcement Learning

- Feature/reward design can be very involved
  - Online learning (no time for tuning)
  - Continuous features (handled by tiling)
  - Delayed rewards (handled by shaping)
- Parameters can have large effects on learning speed
- Realistic environments can have partial observability
- Realistic environments can be non-stationary
- There may be multiple agents

67

## RL Summary 2:

- A typical reinforcement learning system is an active agent, interacting with its environment.
- It must balance:
  - Exploration: trying different actions and sequences of actions to discover which ones work best
  - Exploitation (achievement): using sequences which have worked well so far
- Must learn **successful sequences of actions** in an uncertain environment

68

## RL Summary 3

- There are **many** sophisticated RL algorithms
  - Most notably: probabilistic approaches
- Applicable to game-playing, search, finance, robot control, driving, scheduling, diagnosis, …

69