Reinforcement Learning

Nicholas Ruozzi
University of Texas at Dallas

based on the slides of Ronald J. Williams

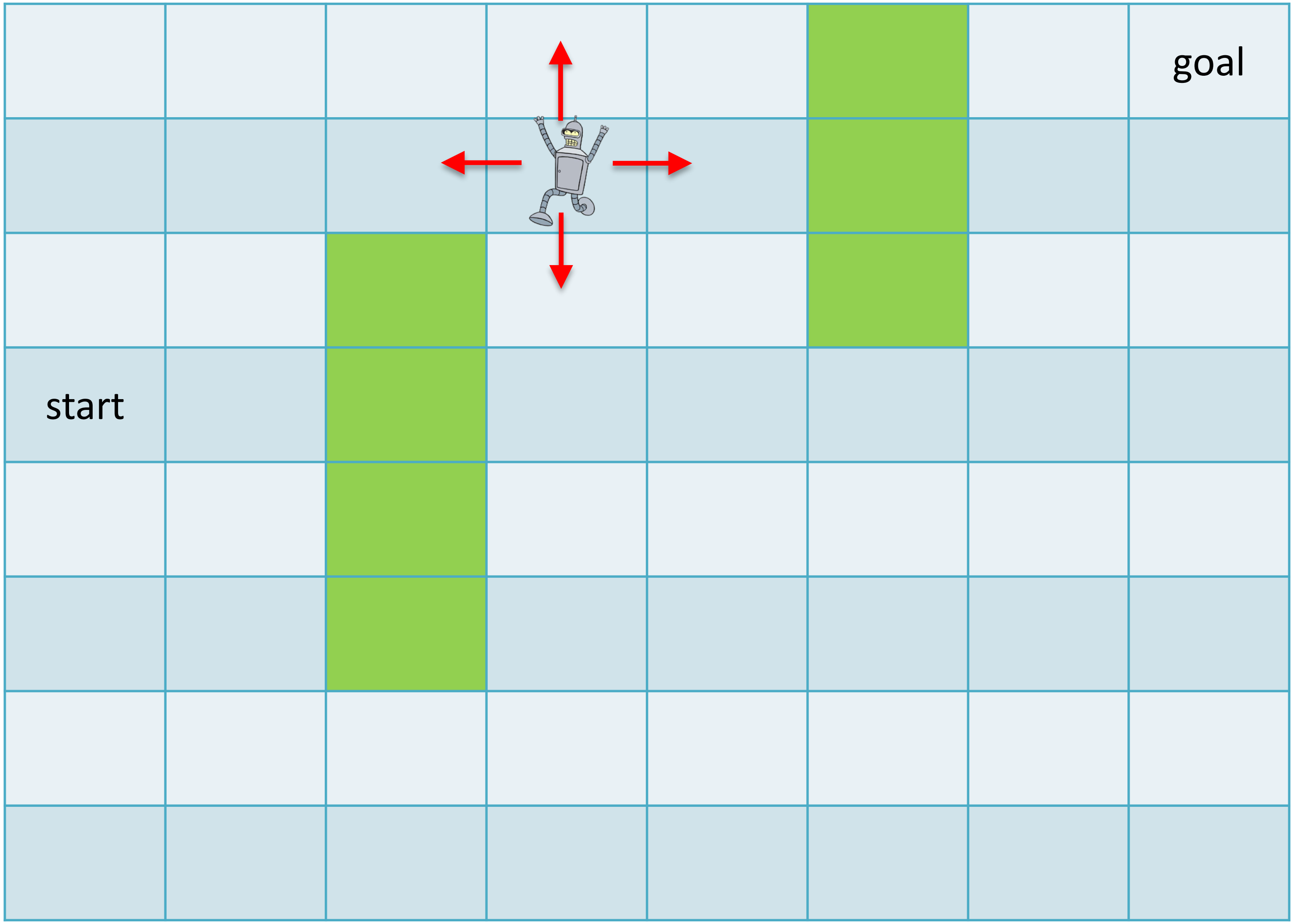
Reinforcement Learning

- Autonomous “agent” that interacts with an environment through a series of actions
  - E.g., a robot trying to find its way through a maze
    - Actions include turning and moving through the maze
  - The agent earns rewards from the environment under certain (perhaps unknown) conditions
- The agent’s goal is to maximize the reward
  - We say that the agent learns if, over time, it improves its performance

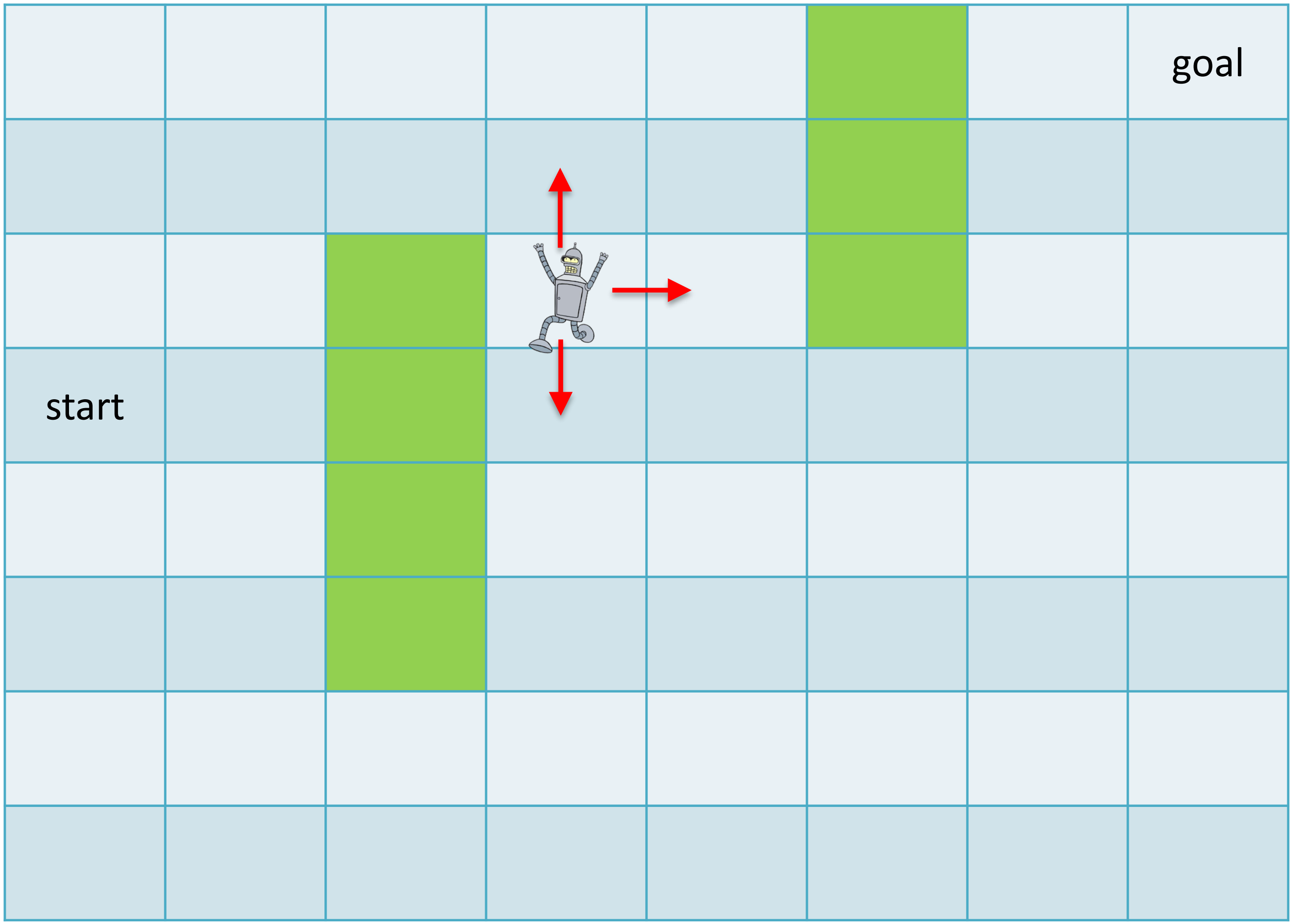
Reinforcement Learning

- Often formalized (mathematically) as Markov Decision Processes (MDPs) or Partially Observable Markov Decision Processes (POMDPs)
- MDPs are described by series of states (state of the environment) and a collection of actions corresponding to each state (allowable actions that change the state of the environment)
  - The next state depends (perhaps probabilistically) on only the current state and the chosen action
  - Each state/action pair has an associated reward (possibly probabilistic)
- Markov chains are a simple form of MDP with only one action and no rewards

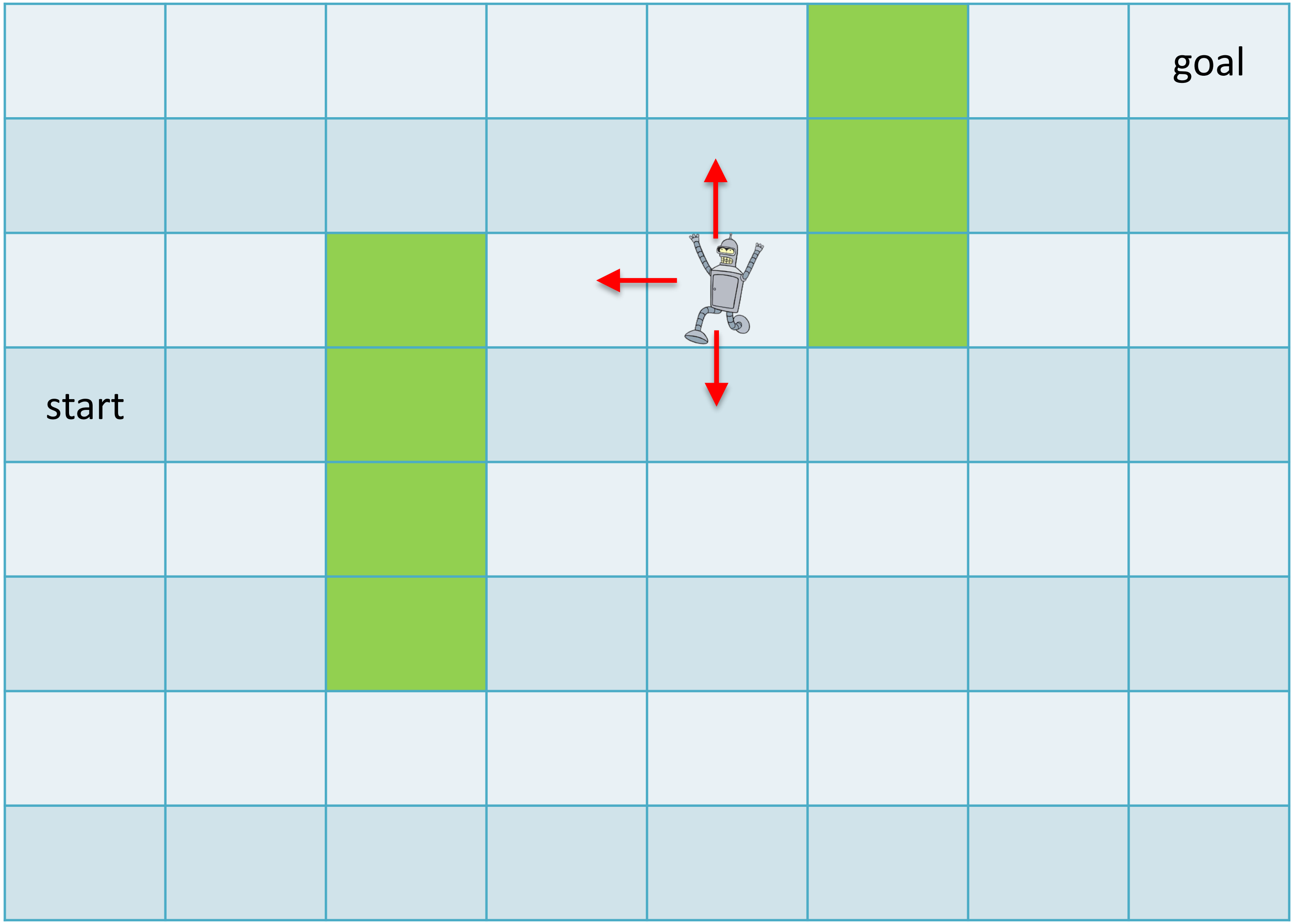
Example



Example



Example



MDPs

- Rewards can be positive or negative
  - E.g., the robot might receive a small penalty each time it takes a step that does not reach the goal
- Objective of the learning process is to develop a **policy** (a way to choose actions given the current state) to maximize the reward
  - Could be difficult to do as rewards may be delayed
    - E.g., the robot receives a reward for reaching the end of the maze, but only penalties in-between

MDPs

- Agent at step t
  - Observes the state of the system
  - Selects an action to perform
  - Receives some reward
- This process is repeated indefinitely

Policies

- A policy is the prescription by which the agent selects an action to perform
  - Deterministic: the agent observes the state of the system and chooses an action
  - Stochastic: the agent observes the state of the system and then selects an action, at random, from some probability distribution over possible actions

Applications of MDPs

- Robot pathfinding
- Planning
- Elevator scheduling
- Manufacturing processes
- Network routing
- Game playing

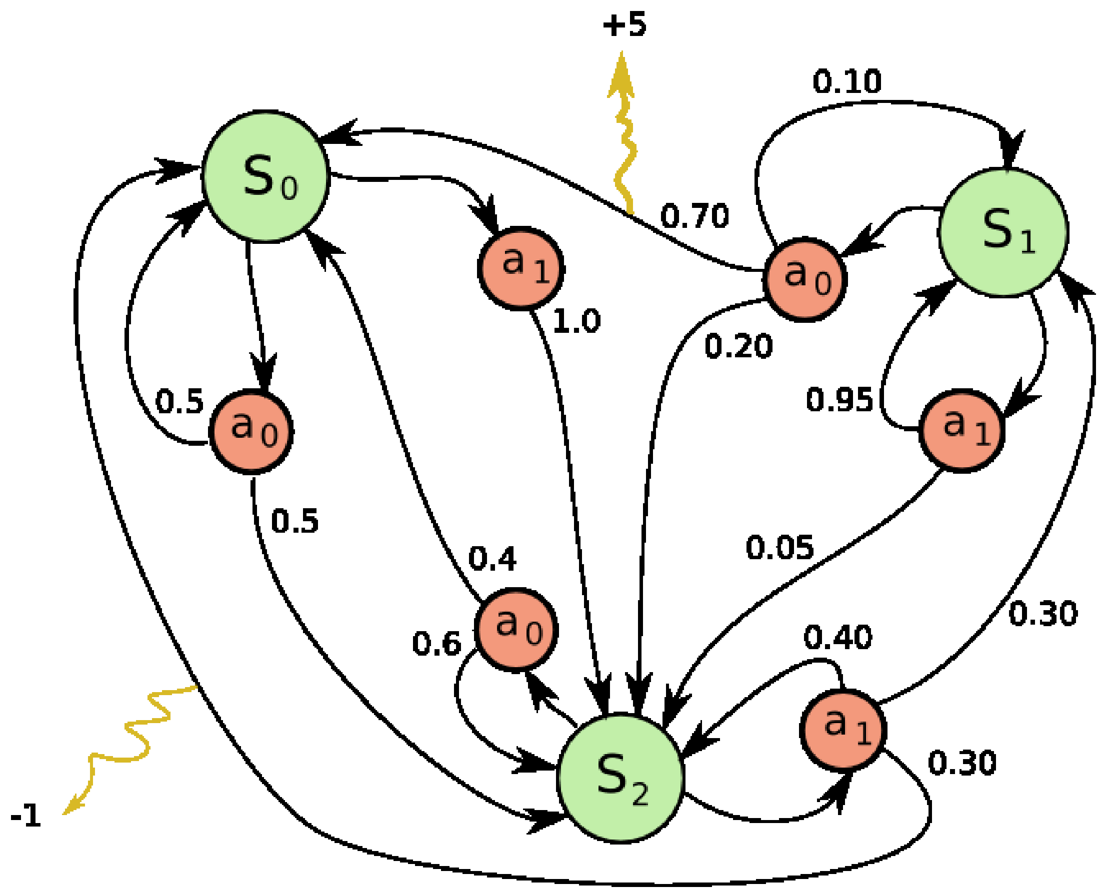
Formal Definition

- A deterministic MDP consists of the following
  - A finite set of states S
  - A set of allowable actions A_s for each $s \in \mathrm{S}$
  - A transition function $T: S \times A \rightarrow S$
  - A reward function $R: S \times A \rightarrow \mathbb{R}$
- In the general case, T and R can be stochastic functions (we'll worry about the deterministic case today)

Formal Definition

- A stochastic MDP consists of the following
  - A finite set of states S
  - A set of allowable actions A_s for each $s \in$ S
  - A transition function $T: S \times A \times S \rightarrow [0,1]$
    - $T(s, a, s')$ is the probability of transitioning from s to s' upon taking action a
  - A reward function $R: S \times A \times S \rightarrow \mathbb{R}$
    - $R(s, a, s')$ is the reward obtained by taking action a in state s and arriving in state s'

MDPs



src: Wikipedia

Cumulative Reward

- A policy is a mapping from states to actions, $\pi: S \to A$
  - Policies can be deterministic or stochastic
- Let $r(t)$ denote the reward at time t
- The objective is to find a policy that maximizes the cumulative (discounted) reward

$$r(0) + \gamma r(1) + \gamma^2 r(2) + \cdots$$

where $\gamma \in (0,1)$ is a discount factor necessary to make the sum converge (also applied in economic contexts to prefer future rewards at a discounted rate)

Value Function

- How can we evaluate the quality of policy π?

Value Function

- How can we evaluate the quality of policy π?
- A **value** function $V: S \to \mathbb{R}$ assigns a real number to each state
  - A particular value function of interest will be the reward function

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} r(t)$$

where the state at time t is generated from the state at time $t-1$ by applying the action dictated by the policy, $\pi(s_{t-1})$

Technical Notes

- In the case that the rewards, transitions, policy, etc. are stochastic
  - Replace the reward, $r(t)$, with the expected reward under the policy
- An MDP has an absorbing state if there exists a state $s \in S$ such that, with probability one, $T(s, a) = s$ for all $a \in A_s$
  - In this case, if the absorbing state can always be reached, the discount factor is unnecessary

Objective

- Find a policy $\pi^*: S \rightarrow A$ such that

$$V^{\pi^*}(s) \geq V^{\pi}(s)$$

for all $s \in S$ and all policies π

- Any policy that satisfies this condition is called an optimal policy (may not be unique)
- There always exists an optimal policy
    - How do we find it?

Optimal Policies

- Can find an optimal policy via a dynamic programming approach
  - Compute the optimal value, $V^{\pi^*}(s)$, for each state
  - Greedily select the action that maximizes reward
- We can describe the optimal value via a *recurrence relation*

$$V^{\pi^*}(s) = \max_{a \in A_s}\left(R(s,a) + \gamma V^{\pi^*}(T(s,a))\right)$$

  - This is one of the so-called Bellman equations
  - Justifies the greedy strategy (all optimal strategies are "greedy" in this sense)

Bellman Equations

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma V^{\pi}(T(s, \pi(s)))$$

$$V^{\pi^*}(s) = \max_{a \in A_s} \left(R(s, a) + \gamma V^{\pi^*}(T(s, a)) \right)$$

- The first equation holds for any policy while the second must hold for any optimal policy
  - Why?

The Greedy Strategy

- Given a value function $V: S \to \mathbb{R}$, we say that π is greedy for V if

$$\pi(s) \in \arg\max_a \big(R(s,a) + \gamma V(T(s,a))\big)$$

- If π is not an optimal policy, then π' which is greedy for V^π must satisfy $V^\pi(s) \leq V^{\pi'}(s)$ for all $s \in S$
  - This suggests that we can, starting from any policy, obtain a better policy (similar to coordinate ascent)
  - Two questions:
    - Does this process converge?
    - If it converges, is the converged policy optimal?

Value Iteration

- Choose an initial value function V_0 (could be anything)
- Repeat until convergence
  - For each s

$$V_{t+1}(s) = \max_{a \in A_s}\left(R(s,a) + \gamma V_t(T(s,a))\right)$$

- This process always converges to the optimal value, V_*, as long as $\gamma \in (0,1)$,

$$\|V_{t+1} - V_*\|_\infty \leq \gamma \|V_t - V_*\|_\infty \leq \gamma^{t+1} \|V_0 - V_*\|_\infty$$

Example (100 reward at goal, -1 for each step)

100	100	100	100	100		100	100
100	100	100	100	100		100	100
100	100		100	100		100	100
100	100		100	100	100	100	100
100	100		100	100	100	100	100
100	100		100	100	100	100	100
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100

Example (100 reward at goal, -1 for each step)

99	99	99	99	99		99	100
99	99	99	99	99		99	99
99	99		99	99		99	99
99	99		99	99	99	99	99
99	99		99	99	99	99	99
99	99		99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Example (100 reward at goal, -1 for each step)

98	98	98	98	98		99	100
98	98	98	98	98		98	99
98	98		98	98		98	98
98	98		98	98	98	98	98
98	98		98	98	98	98	98
98	98		98	98	98	98	98
98	98	98	98	98	98	98	98
98	98	98	98	98	98	98	98

Example (100 reward at goal, -1 for each step)

87	88	89	90	91		99	100
88	89	90	91	92		98	99
87	88		92	93		97	98
86	87		93	94	95	96	97
85	86		92	93	94	95	96
86	87		91	92	93	94	95
87	88	89	90	91	92	93	94
86	87	88	89	90	91	92	93

Policy Iteration

- Choose an initial policy π_0 (could be anything)
- Repeat until convergence
  - Compute V^{π_t}
  - Choose π_{t+1} to be a greedy policy with respect to V^{π_t}
- This process always converges to an optimal policy

Q-Values

- For learning, it will be useful to express value functions in terms of Q-value functions

- For a policy π, $Q^\pi: S \times A \to \mathbb{R}$ is defined to be the value of the policy π starting from state s where the first action is taken to be a

$$Q^\pi(s,a) = R(s,a) + \gamma V^\pi(T(s,a))$$

- For any optimal policy π^*, $V^{\pi^*}(s) = \max_a Q^{\pi^*}(s,a)$

- A policy π is said to be greedy with respect to Q if

$$\pi(s) \in \arg\max_a Q(s,a)$$

Reinforcement Learning

- The above is simply the theory of MDPs
  - We haven't seen any "learning" yet
    - All transition and reward functions were assumed to be known in advance
- The setting for reinforcement learning:
  - The agent is the learner whose task is to maximize its respective rewards
  - All rewards and transitions are unknown and must be learned through trial and error (key complication in the learning setting)

Approaches to RL

- Learn the MDP first, then use value/policy iteration
- Learn only the values (don’t learn the MDP or explicitly model it)
  - Can be advantageous in practice as MDPs can require a significant amount of storage to specify completely
- Hybrid approaches of learning and planning...

Q-Learning

- Choose an initial state-value function $Q(s, a)$
- Let s be the initial state of the environment
- Repeat until convergence
  - Choose an action a for the current state s based on Q
  - Take action a and observe the reward r and the new state s'
  - Set $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a')\right)$
  - Set $s = s'$

Q-Learning

- Choose an initial state-value function $Q(s, a)$
- Let s be the initial state of the environment
- Repeat until convergence
  - Choose an action a for the current state s based on Q
  - Take action a and observe the reward r and the new state s'
  - Set $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a')\right)$
  - Set $s = s'$

α is called the learning rate

Q-Learning

- How should we pick an action to take based on Q?

Q-Learning

- How should we pick an action to take based on Q?
  - Shouldn't always be greedy (we won't explore much of the state space this way)
  - Shouldn't always be random (will take a long time to generate a good Q)
- ϵ-greedy strategy: with some small probability choose a random action, otherwise select the greedy action

Reinforcement Learning

- If the state space is large, these techniques are intractable (what if it is continuous?)
  - Need different algorithms for this setting, but we already know a few!
  - If the goal is to learn $Q(s,a)$, we could use techniques from supervised learning
    - Generate a collection of noisy observations using Q-learning
    - Use a supervised learning algorithm (e.g., a neural network, $k-NN$, etc.) to approximate the Q function

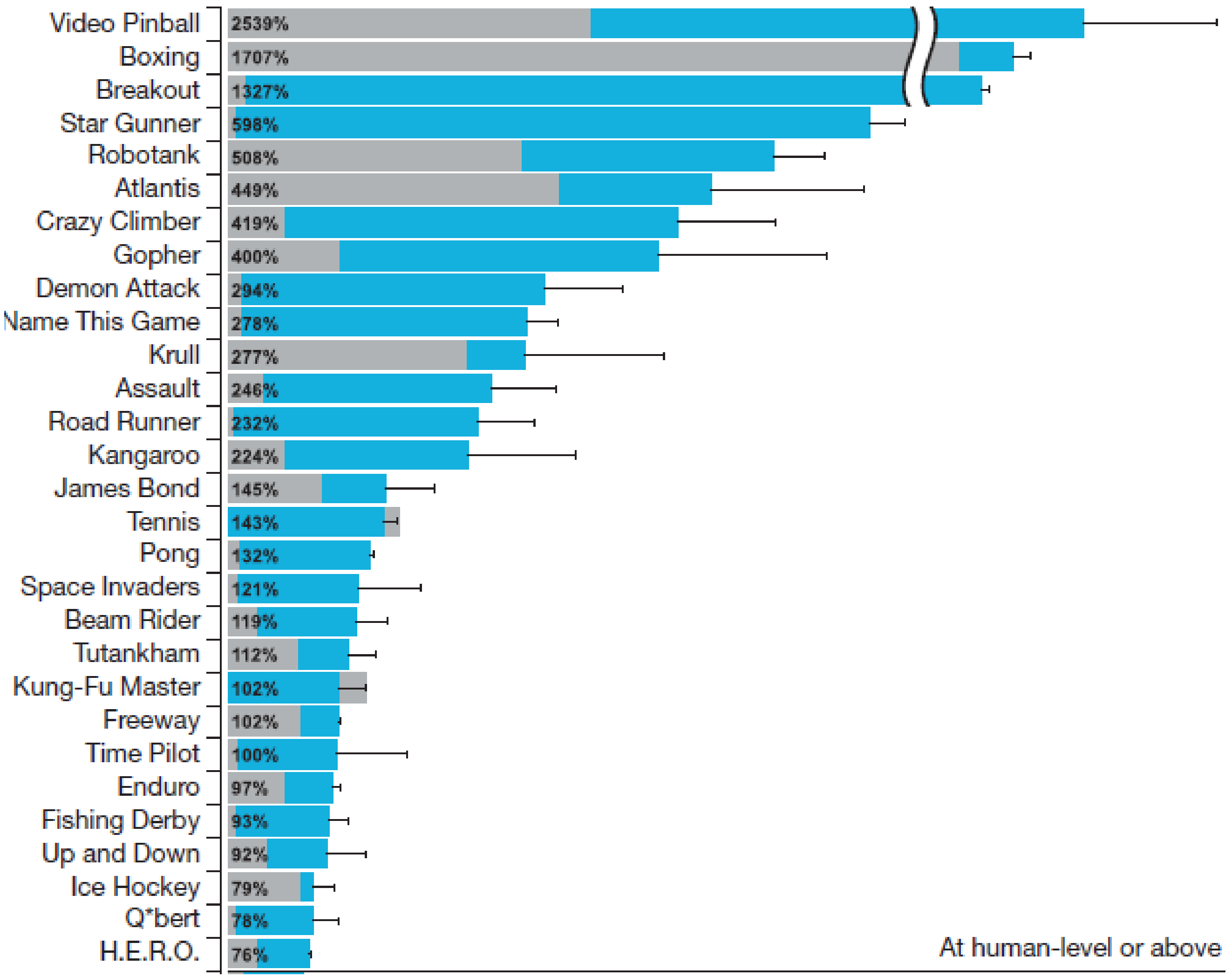
“Deep” Q-Learning

- If the Q function is approximated by a neural network, the correctness guarantees for Q-learning no longer apply
  - Learning might converge poorly or not at all
- In practice, experience replay has been shown to result in better learning performance
  - The idea is that every time a state action pair is explored by the Q-learner, that pair is added to a replay set with its corresponding reward and transition
  - At each iteration, the replay set is sampled and the samples are used to update the weights of the neural network

Deep Q-Learning

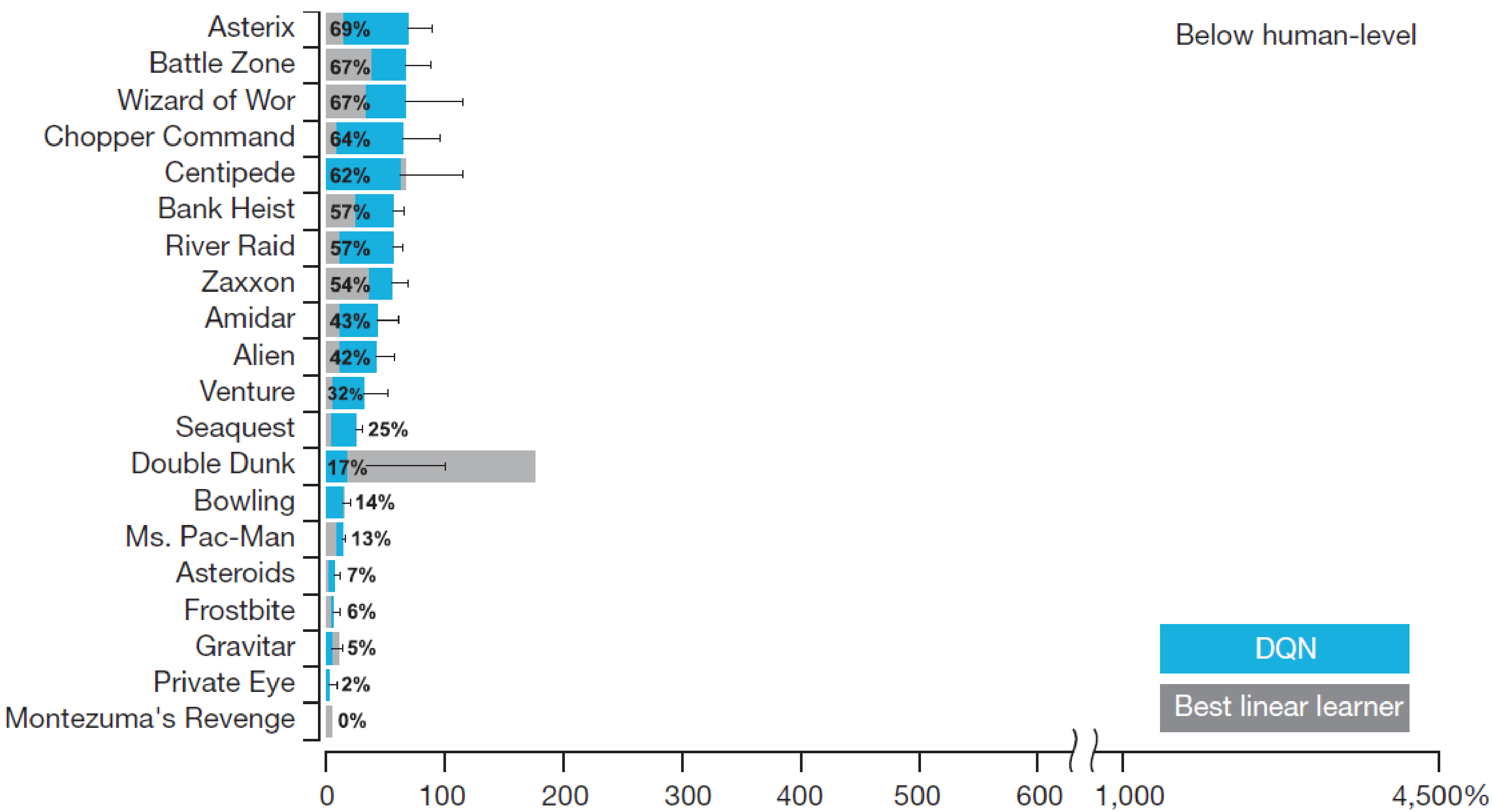
- Choose an initial θ for $Q(\cdot,\cdot\,|\theta)$, an initial state s, and an empty replay set R
- Repeat until convergence
  - Choose an action a for the current state s based on$Q(s,\cdot\,|\theta)$
  - Take action a and observe the reward r and the new state s', add (s, a, s', r) to the replay set
  - Sample $S \subset R$
    - For each element in S, set $y_{(s,a,s',r)} = \left(r + \gamma \max_{a'} Q(s', a'|\theta)\right)$
    - Perform one step of gradient descent starting at θ on $\sum_{(s,a,s',r)\in S}\left(Q(s,a|\theta) - y_{(s,a,s',r)}\right)^2$ to yield θ'
    - Set $\theta = (1-\alpha)\theta + \alpha\theta'$
  - Set $s = s'$

Deep Q-Learning Performance



Playing Atari with Deep Reinforcement Learning [Minh et al.]

Deep Q-Learning Performance



Playing Atari with Deep Reinforcement Learning
[Minh et al.]

Supervised Learning

- Regression & classification
- Discriminative methods
  - k-NN
  - Decision trees
  - Perceptron
  - SVMs & kernel methods
  - Logistic regression
  - Neural networks
- Parameter learning
  - Maximum likelihood estimation
  - Expectation maximization

Bayesian Approaches

- MAP estimation
- Prior/posterior probabilities
- Bayesian networks
  - Naive Bayes

Unsupervised Learning

- Clustering
  - k-means
  - Spectral clustering
  - Hierarchical clustering
- Expectation maximization
  - Soft clustering
  - Mixtures of Gaussians

Learning Theory

- PAC learning
- VC dimension
- Bias/variance tradeoff
- Chernoff bounds
- Sample complexity

Optimization Methods

- Gradient descent
  - Stochastic gradient descent
  - Subgradient methods
- Coordinate descent
- Lagrange multipliers and duality

Matrix Based Methods

- Dimensionality Reduction
  - PCA

Ensemble Methods

- Bootstrap sampling
- Bagging
- Boosting

Other Learning Topics

- Reinforcement learning

Questions about the course content?

For the final...

- You should understand the basic concepts and theory of all of the algorithms and techniques that we have discussed in the course
- There is no need to memorize complicated formulas, etc.
  - For example, if I ask for the sample complexity of a scheme, I will give you the generic formula
- However, you should be able to derive the algorithms and updates
  - E.g., Lagrange multipliers and SVMs, the EM algorithm, etc.

For the final...

- No calculators, books, notes, etc. will be permitted
  - As before, if you need a calculator, you have done something terribly wrong
- The exam will be in roughly the same format
- Exam will emphasize the new material, but ALL material will be tested
- Take a look at the practice exams!

Please evaluate the course!

eval.utdallas.edu