# CSCI 5622 Machine Learning

# ML Reinforcement Learning

| DATE | TOPIC | DUE |
|---|---|---|
| | | ***Peer Feedback Overdue*** |
| Wed, Dec 9 | Reinforcement Learning | Peer Feedback Grades |
| Fri, Dec 11 | ---------- | ***Final Paper Due*** |
| W 16th 4PM | ***1777 Exposition Dr, Room 102*** | Final Presentations |
| .. (Dec 16) | ***Mandatory attendance & participation*** | ***Ask Questions*** |

www.RodneyNielsen.com/teaching/CSCI5622-F09/


Instructor: Rodney Nielsen

Assistant Professor Adjunct, CU Dept. of Computer Science

Research Assistant Professor, DU, Dept. of Electrical & Computer Engr.

Research Scientist, Boulder Language Technologies

# ML Reinforcement Learning

- Control learning
- Control policies that choose *optimal* actions
- Q-Learning
- Convergence

# Control Learning

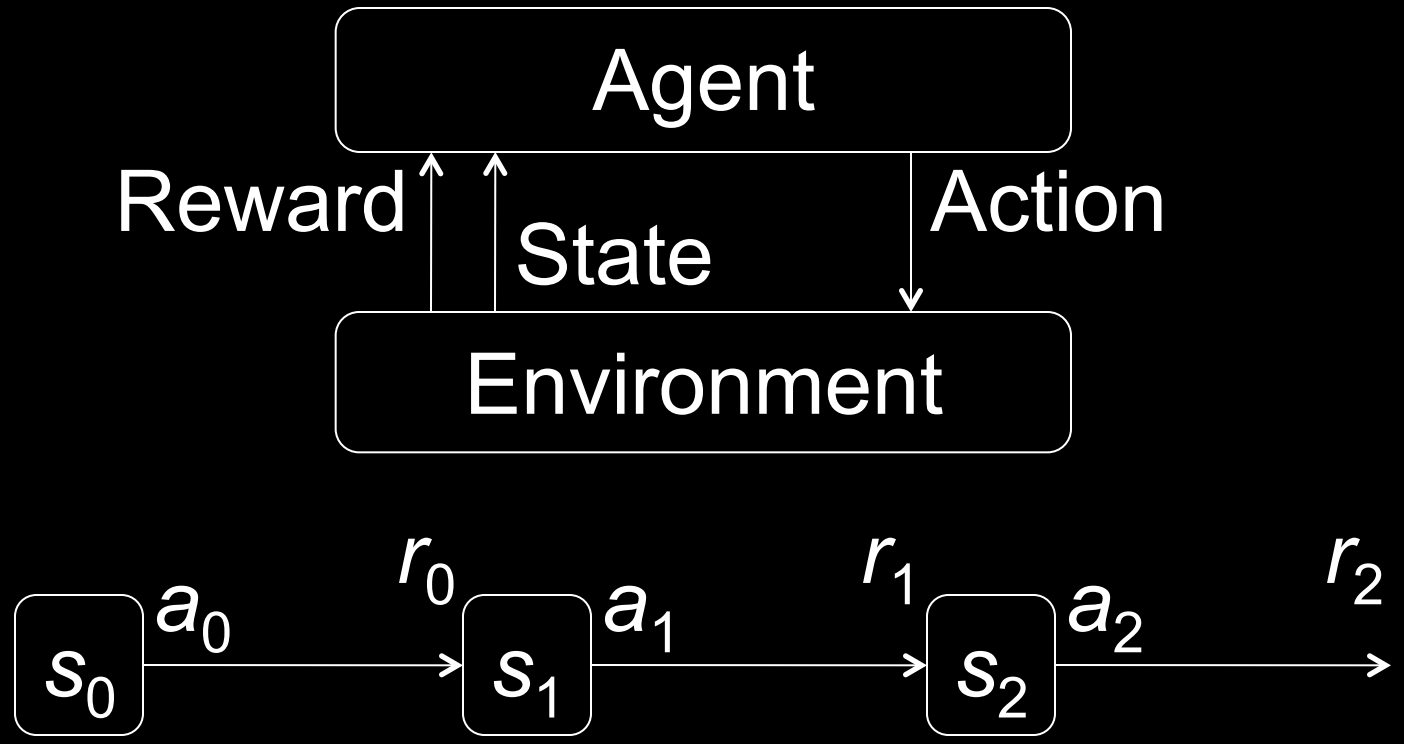
ML

- Learning to choose actions:
  - Robot learning to dock with battery charger, …
  - Learning optimal factory machine settings
  - Learning to play a game like backgammon
- Problem characteristics
  - Delayed reward
  - Opportunity for exploration
  - State may only be partially observable
  - Possibly multi-task learning from same inputs

# Ex: TD-Gammon

- Learn to play backgammon
- Immediate reward
  - Win: +100
  - Lose: -100
  - Else: 0
- Trained by playing 1.5M games against itself
- Reached world-class player status

# ML Reinforcement Learning




- Goal: Learn an action policy that maximizes reward: $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$; $0 \leq \gamma < 1$

# Reinforcement Learning

- Finite set of states, $S$
- Finite set of actions, $A$
- At time, $t$, the agent observes the state, $s_t$ in $S$
- Chooses an action $a_t$ in $A$
- Receives the immediate reward $r_t$
- With a change of state to $s_{t+1}$

# Markov Decision Processes

- Markov assumption: $s_{t+1}=\delta(s_t,a_t)$ and $r_t=r(s_t,a_t)$
- $s_{t+1}$ and $r_t$ depend only on the current state and action
- The functions $\delta$ and $r$ might be nondeterministic
- $\delta$ and $r$ are not necessarily known to the agent

ML

# Agent's Learning Task

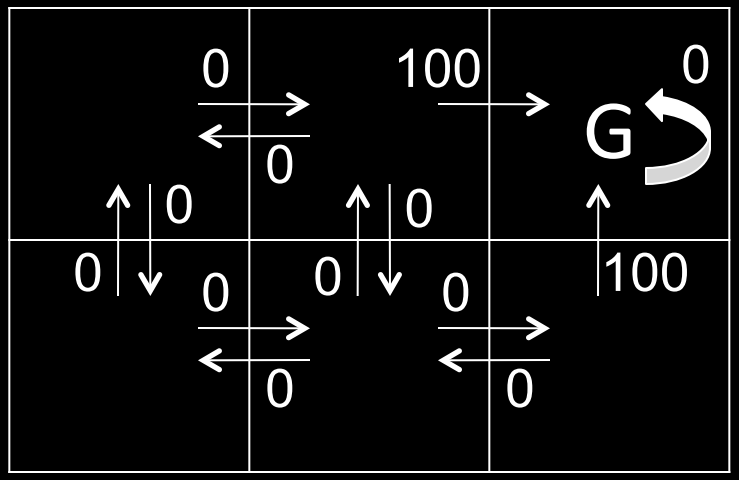
- Execute actions in environment, observe, and
  - Learn action policy $\pi : S \rightarrow A$, maximizing $E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$, starting anywhere in $S$
  - $0 \leq \gamma < 1$ is the discount factor for future rewards
- No training examples of the form <*s*, *a*>
- Instead, training examlples are <<*s*, *a*>, *r*>

# Value Function

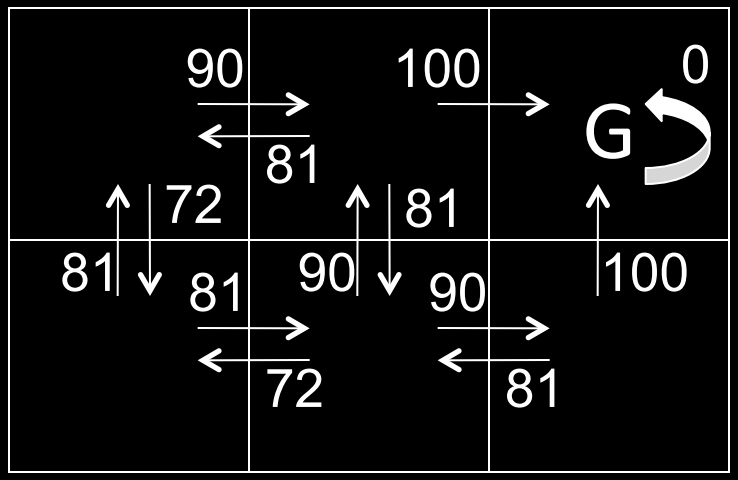
- Consider deterministic worlds
- For each possible policy function π, define the evaluation function $V^{\pi}(s) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \Sigma_{i=0..\infty} \gamma^i r_{t+i}$
- Where the $r_k$ are generated by policy π starting at state *s*
- Goal: Learn the optimal policy π*

  $\pi^* = \text{argmax}_{\pi} V^{\pi}(s)$

# Rewards

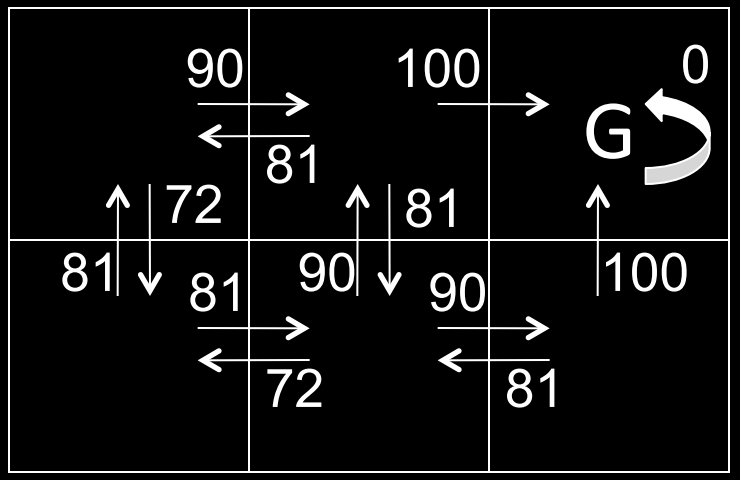



*r*(*s*, *a*) values: immediate reward

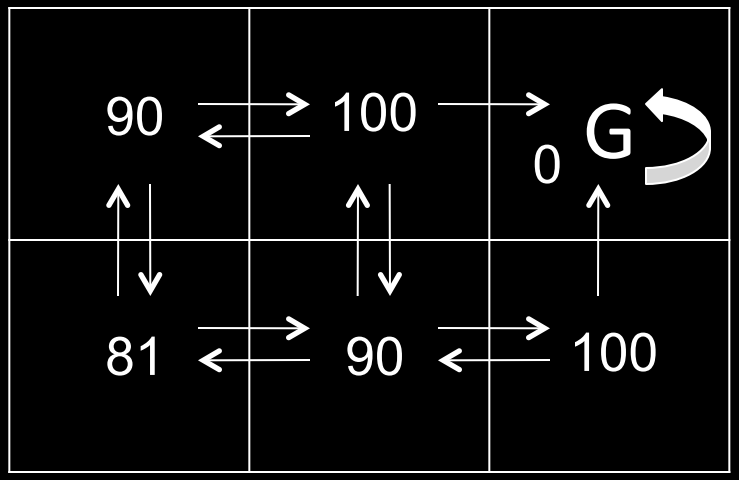



*Q*(*s*, *a*) values

# Rewards

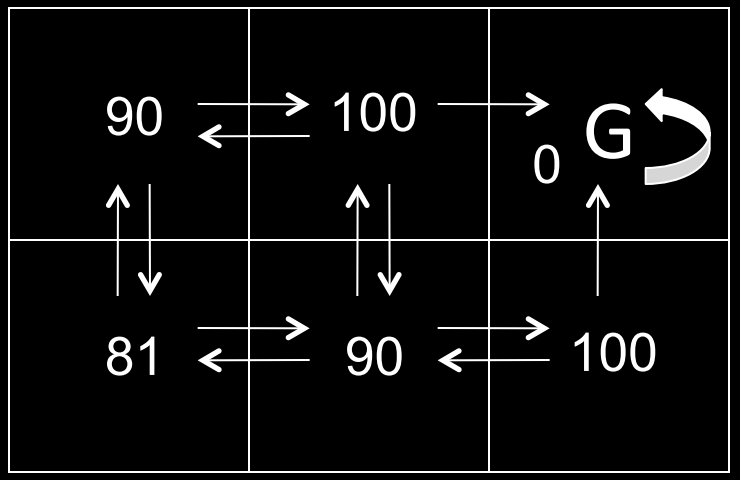



$Q(s, a)$ values

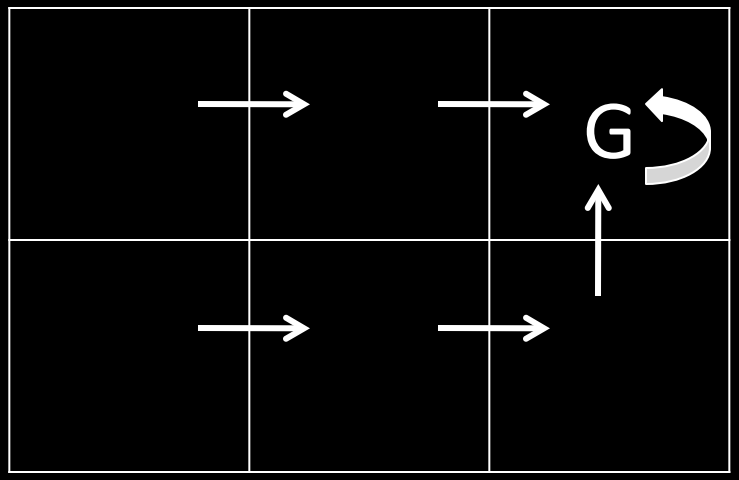



$V^*(s)$ values

# Rewards




$V^*(s)$ values




An optimal policy

# What to Learn

- Option 1: $V^{\pi^*} = V^*$
- It could then do a lookahead search to choose best action from any state $s$ because $\pi^*(s) = \text{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$
- Problem:
  - Works well if agent knows $\delta : S \times A \rightarrow S$ and $r : S \times A \rightarrow \mathbb{R}$
  - But when it doesn't it can't choose actions this way

# $Q$ Function

- Define a new function similar to $V^*$

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

- If agent learns $Q$, it can choose optimal action even without knowing $\delta$

$$\pi^*(s) = \text{argmax}_a \, [r(s,a) + \gamma V^*(\delta(s,a))]$$

$$\pi^*(s) = \text{argmax}_a \, Q(s,a)$$

- $Q$ is the evaluation function the agent will learn

# Training Rule to Learn $Q$

- Note $Q$ and $V^*$ closely related

$$V^*(s) = \max_{a'} Q(s,a')$$

- Which allows us to write $Q$ recursively as

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma V^*(\delta(s_t,a_t))$$
$$= r(s_t,a_t) + \gamma \max_{a'} Q(s_{t+1},a')$$

- Let $\hat{Q}$ denote learner's current approximation to $Q$. Consider training rule:

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

where $s'$ is the state resulting from applying action $a$ in state $s$

# ML *Q* Learning in Deterministic Cases

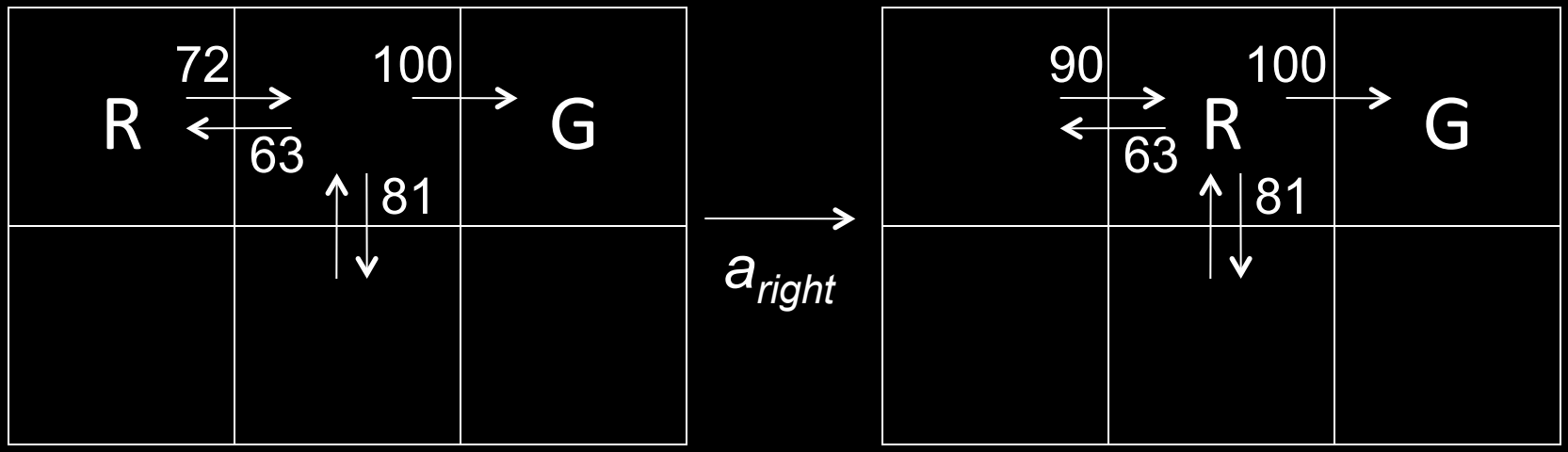
- For each *s*, *a* initialize table entry $\hat{Q}(s,a) \leftarrow 0$
- Observe current state *s*
- Do forever:
  - Select an action *a* and execute it
  - Receive immediate reward *r*
  - Observe the new state *s'*
  - Update the table entry for $\hat{Q}(s,a)$ as follows:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

$$s \leftarrow s'$$

# Updating $\hat{Q}$

ML




$$\hat{Q}(s_1,a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2,a')$$

$$= 0 + 0.9 \max \{63, 81, 100) = 90$$

- If reward is non-negative, then for all $\langle s,a,n \rangle$

$$\hat{Q}_{n+1}(s,a) \geq \hat{Q}_n(s,a), \text{ and}$$

$$0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$$

# Convergence

- $Q\hat{}$ converges to $Q$.
- The book shows a proof for the case of a deterministic world where we see each $<s,a>$ visited infinitely often

# Nondeterministic Case

- What if reward and next state are non-deterministic?
- We redefine $V$, $Q$ by taking expected values

$$V^{\pi}(s) \leftarrow E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

$$= E[\Sigma_{i=0..\infty} \gamma^i r_{t+i}]$$

$$Q(s,a) \leftarrow E[r(s,a) + \gamma V^*(\delta(s,a))]$$

# Nondeterministic Case

- *Q* learning generalizes to nondeterministic worlds
- Alter training rule to:

  $\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \max_{a'} \hat{Q}_{n-1}(s',a')]$
- where

$$\alpha_n = 1 / (1 + visits_n(s,a))$$

- Can still prove convergence of $\hat{Q}$ to Q

# ML Temporal Difference (TD) Learning

- *Q* learning: reduce discrepancy between successive *Q* estimates
- One step time difference:

$$Q^{(1)}(s_t,a_t) \leftarrow r_t + \gamma \max_a Q\hat{}(s_{t+1},a)$$

- Why not two steps?

$$Q^{(2)}(s_t,a_t) \leftarrow r_t + \gamma r_{t+1} + \gamma^2 \max_a Q\hat{}(s_{t+2},a)$$

- Or *n*?

$$Q^{(n)}(s_t,a_t) \leftarrow r_t + \gamma r_{t+1} + \ldots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a Q\hat{}(s_{t+n},a)$$

- Blend all of these

# TD Learning

- Blend all of time steps:

$$Q^{\lambda}(s_t,a_t) \leftarrow (1-\lambda)[Q^{(1)}(s_t,a_t) + \lambda Q^{(2)}(s_t,a_t) + \lambda^2 Q^{(3)}(s_t,a_t) + \ldots]$$
$$= r_t + \gamma[(1-\lambda)\max_a \hat{Q}(s_t,a_t) + \lambda Q^{\lambda}(s_{t+1},a_{t+1})]$$

- TD($\lambda$) algorithm uses above training rule
- Sometimes converges faster than $Q$ learning
- Converges for learning $V^*$ for any $0 \leq \lambda \leq 1$
- TD-Gammon used this algorithm

# Questions

ML

Questions???


© Rodney D. Nielsen

# ML Presentations

- Wed, Dec 9: Tony & Xinyu


© Rodney D. Nielsen

# Projects

ML

- Will stay until all questions are addressed, or
- Please make an appointment if you have any questions at all regarding your project


© Rodney D. Nielsen