ROB 537:
Learning-Based Control

Week 5, Lecture 1

Policy Gradient, Eligibility Traces, Transfer Learning (Matt Taylor)

Announcements:

Project background due Today

HW 3 Due on 10/30

Midterm Exam on 11/6

Reading:

Transfer Learning survey by Taylor & Stone

Oregon State University

Policy Gradient

- Gradient Decent

$$MSE(\vec{\theta}_t) = \sum_{s \in S} p(s) \left(V^{\pi}(s) - V_t(s)\right)^2$$

- Basic parameter update (based on uniform p(s)):

$$\Delta\vec{\theta}_t = -\frac{1}{2} \alpha \nabla_{\vec{\theta}} \left(V^{\pi}(s_t) - V_t(s_t)\right)^2$$

  – Update parameters based on the negative of the gradient

Oregon State University

Gradient Descent Sarsa

- Gradient descent for state-action values:

$$\Delta\vec{\theta}_t = \alpha \left(Q^{\pi}(s_t,a_t) - Q_t(s_t,a_t)\right) \nabla_{\vec{\theta}} Q_t(s_t,a_t)$$

$$\Delta\vec{\theta}_t = \alpha \left(r_t + \gamma\, Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)\right) \nabla_{\vec{\theta}} Q_t(s_t,a_t)$$

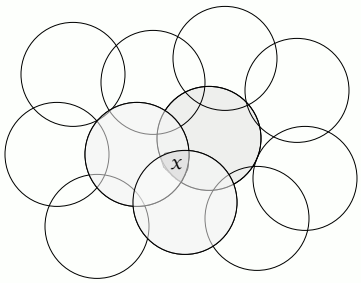
Linear Methods

- Special case of gradient descent methods:
- V_t is linear with respect to the parameters θ_t
- There is a vector of features ϕ for each state
- Function approximation becomes:

$$V_t(s) = \sum_{i=1}^{n} \theta_t(i)\ \phi_s(i)$$

Coarse Coding

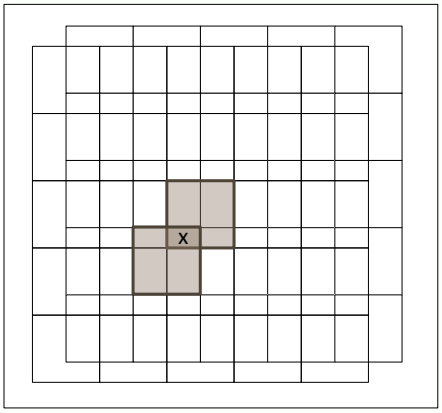
- Cover the space with concentric features (for example circles)
- Weights are 0 or 1
- Value is based on coverage



Oregon State University

Tile Coding

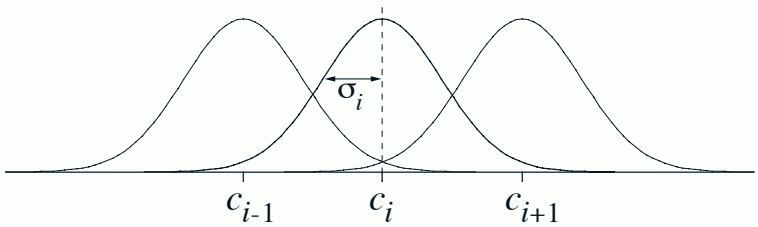
- Exhaustive partitions (tilings)
- Same number of features as number of tilings



Oregon State University

Radial Basis Functions

- Coarse coding generalized to continuous value features
- Each feature can be weighted by a real number (instead of 0 or 1)



Oregon State University

Oregon State University

Eligibility Traces

- Eligibility Traces are the basic mechanism of assigning credit in Reinforcement Learning
  - In TD(λ), λ refers to an eligibility trace
- Basic Idea:
  - Determine how many rewards will be used in evaluating a value function
  - Determine which states are eligible for an update
  - Connect TD learning and Monte Carlo Learning

Oregon State University

Eligibility Traces

- Forward view:
  - Theoretical
  - Connect TD learning and Monte Carlo learning
    - In TD learning only previous state is eligible
    - In Monte Carlo, all visited states are eligible
  - Intuition: How far ahead do you need to look to figure out value of a state
- Backward view:
  - Mechanistic
  - Temporary record of events
    - Visiting states, taking actions, Receiving Rewards
  - Intuition: How many state value pairs do I need to store to figure out value of a state

Oregon State University

Forward View

- Recall Monte Carlo methods for a T step episode:

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$$

  – The update is based on all the returns

- In TD backup:

$$R_t^{(1)} = r_{t+1} + \gamma\, V_t(s_{t+1})$$

  – The update is based on 1-step backup

Why restrict ourselves to these two extremes?

Oregon State University

Forward View: n-step TD

- Why not look at 2-step backup:

$$R_t^{(2)} = r_{t+1} + \gamma\, r_{t+2} + \gamma^2 V_t(s_{t+2})$$

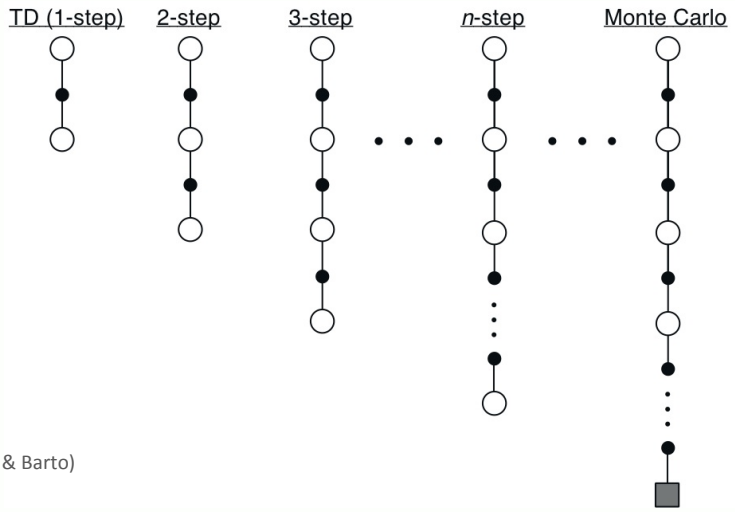
- How about n-step backup?

$$R_t^{(n)} = r_{t+1} + \gamma\, r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$$

- Called "*corrected n-step truncated return*" or *n-step return*

Oregon State University

Forward View



(Sutton & Barto)

Backward View

- Causal, incremental view
- Implementable

- Introduce a new memory variable: eligibility trace $e_t(s)$
  - On each step eligibility trace for each state decays by γλ
  - Eligibility trace of one state increased by 1

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

  - λ is trace decay parameter
  - γ is the discount rate used previously

Intuition

- At any time eligibility traces keep track of which states have been recently visited
  - Recently is quantified by γλ
- The traces capture how "eligible" a state is for an update
- Recall basic reinforcement learning:

$$\boxed{NewEstimate \leftarrow OldEstimate + Stepsize\ (Target - OldEstimate)}$$

Now, update all values based on eligibility trace

Oregon State University

Implementation

$$\boxed{NewEstimate \leftarrow OldEstimate + Stepsize\ (Target - OldEstimate)}$$

- For TD, the error was:

$$\delta_t = r(s_t) + \gamma\ V(s_{t+1}) - V(s_t)$$

- Then update becomes:

$$V_{t+1}(s) = V_t(s) + \alpha\ \delta_t\ e(s_t) \qquad \text{for all } s \in S$$

Oregon State University

Special Cases

- Let's explore two special cases based on this update rule

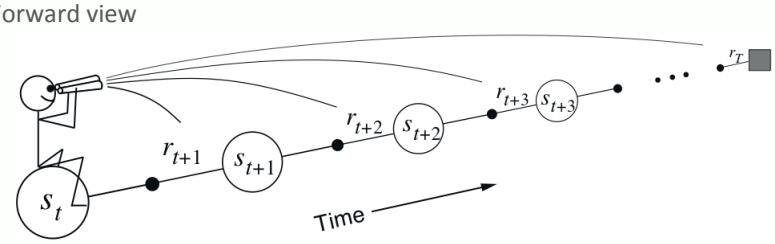
$$V_{t+1}(s) = V_t(s) + \alpha\ \delta_t\ e(s_t) \qquad \text{for all } s \in S$$

- For λ=0:
  - All traces are 0 at t except state visited at t: s_t
  - Reduces to TD update rule
- For λ=1:
  - All states receive credit decaying by γ
  - This is the Monte Carlo updates
    - If γ=1 as well, there is no decay in time and we have undiscounted Monte Carlo updates

Oregon State University

Forward and Backward Views

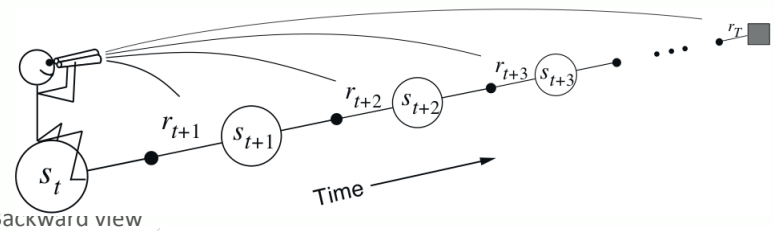
- Forward view



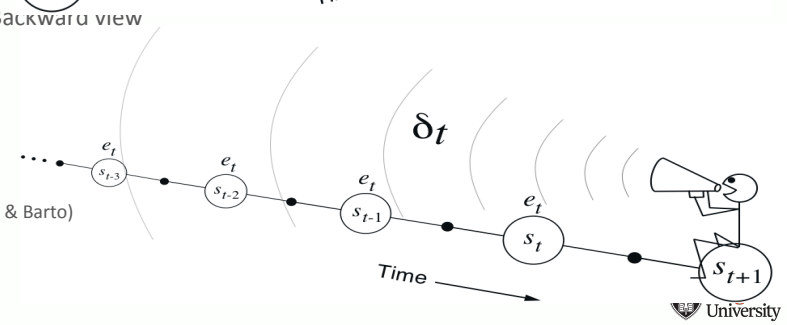
Oregon State University

Forward and Backward Views

- Forward view



- Backward view



(Sutton & Barto)

Sarsa (λ)

- Recall Sarsa learning:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\Big(\underbrace{r(s_t,a_t) + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)}\Big)$$

- Now, we have:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\, \delta_t\, \underbrace{e(s_t,a_t)} \qquad \text{for all } s,a$$

Increase eligibility if s,a picked
Decay eligibility otherwise

Sarsa (λ)

- Recall Sarsa learning:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\Big(r(s_t,a_t) + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\Big)$$

- Now, we have:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\ \delta_t\ e(s_t,a_t) \qquad \text{for all } s,a$$

- where:

$$\delta_t = r(s_t,a_t) + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)$$

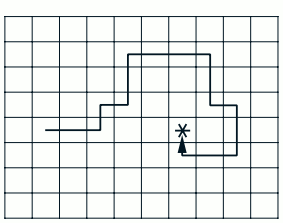
$$e_t(s,a) = \begin{cases} \gamma\,\lambda\, e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\,\lambda\, e_{t-1}(s,a) & \text{otherwise} \end{cases} \quad \text{for all } s,a$$

Oregon State University

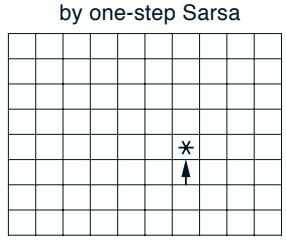
Example: Sarsa (λ)

- Recall Sarsa updates: only last state receives update. All other state action pairs will update when visited/taken next

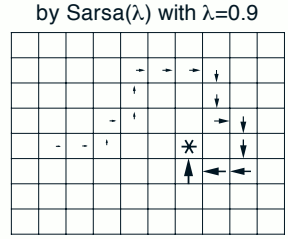
Path taken



Action values increased by one-step Sarsa



Action values increased by Sarsa(λ) with λ=0.9



Oregon State University

Q(λ)

- Recall Q-learning

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left(r(s_t,a_t) + \gamma \max_a Q(s_{t+1},a\ ) - Q(s_t,a_t)\right)$$

- Now, we have:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\ \delta_t\ e(s_t,a_t) \qquad \text{for all } s,a$$

- where

$$\delta_t = r(s_t,a_t) + \gamma \max_a Q(s_{t+1},a\ ) - Q(s_t,a_t)$$

$$e_t(s,a) = I_{ss_t} \cdot I_{aa_t} \begin{cases} \gamma\ \lambda\ e_{t-1}(s,a) + 1 & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a) \\ 0 & \text{otherwise} \end{cases}$$

Oregon State University

Q(λ)

- Q learning with eligibility traces:
  - Like in Sarsa learning, except eligibility trace set to zero after an exploratory move (not greedy)
  - Intuition: two step process
  - Step 1:
    - Either each state-action pair decayed by λ
    - Or all set to 0 after an exploratory step
  - Step 2:
    - Trace for current state action pair incremented by 1

(this is Watkins' Q(λ); There are other versions as well.)

Oregon State University

Back to Gradient Descent Sarsa

- Gradient descent for state-action values:

$$\Delta\vec{\theta}_t = \alpha \left(Q^{\pi}(s_t,a_t) - Q_t(s_t,a_t)\right) \nabla_{\vec{\theta}} Q_t(s_t,a_t)$$

$$\Delta\vec{\theta}_t = \alpha \left(r_{t+1} + \gamma\, Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)\right) \nabla_{\vec{\theta}} Q_t(s_t,a_t)$$

- Gradient descent Sarsa:
  - 1-step backup
  - Extend to eligibility traces

Oregon State University

Gradient Descent Sarsa(λ)

- Gradient descent Sarsa(λ):

$$\Delta\vec{\theta}_t = \alpha\, \delta_t \vec{e}_t$$

  - where

$$\delta_t = r_{t+1} + \gamma\, Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$

$$\vec{e}_t = \gamma\, \lambda\, \vec{e}_{t-1} + \nabla_{\vec{\theta}} Q_t(s_t,a_t)$$

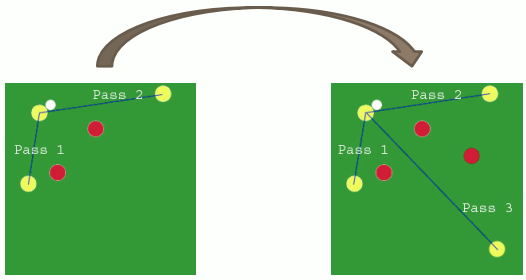
Oregon State University

Transfer Learning

So far, we learned all policies from scratch
- Can be unnecessarily slow

- Humans can use past information
  - Soccer with different numbers of players

- How do we leverage learned knowledge in novel tasks
  - Bias learning : speedup method



Transfer Learning

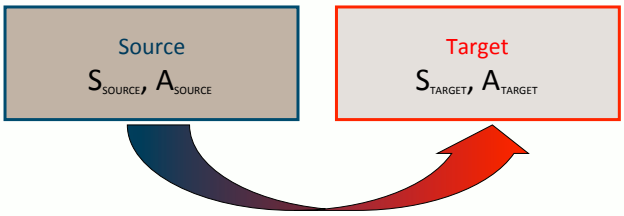
Matthew E. Taylor and Peter Stone.
"Transfer Learning for Reinforcement Learning Domains: A Survey."
Journal of Machine Learning Research, 10(1):1633–1685, 2009.

(Slides courtesy of Matt Taylor)

Oregon State University

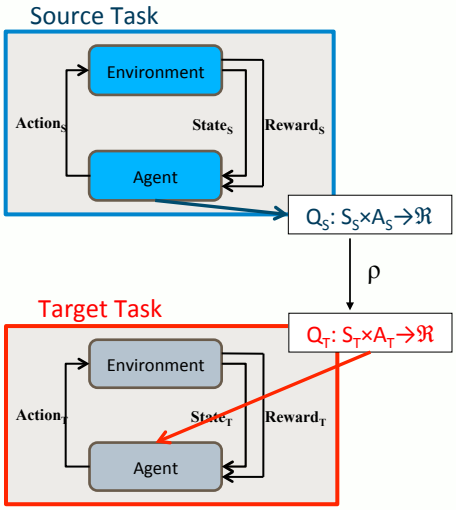
Primary Questions

- Is it possible to transfer learned knowledge?
- Is it possible to transfer *without* providing a task mapping?
- Focus on reinforcement learning tasks
  - Plenty of work in supervised settings



Oregon State University

Value Function Transfer



Q_S not defined on S_T and A_T

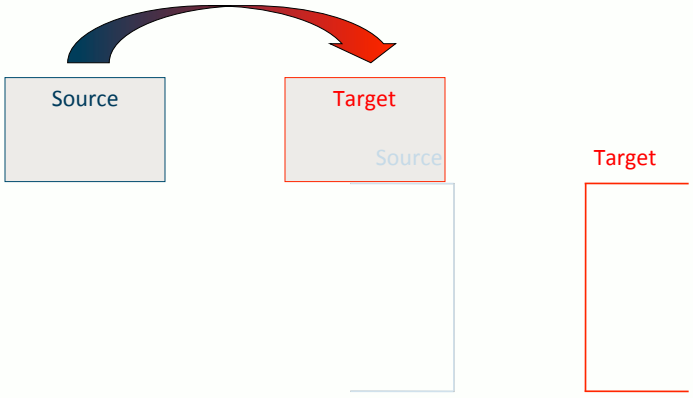
$$\rho(Q_S(S_S, A_S)) = Q_T(S_T, A_T)$$

Action-Value function transferred

ρ is task-dependant:
relies on inter-task mappings

Oregon State University

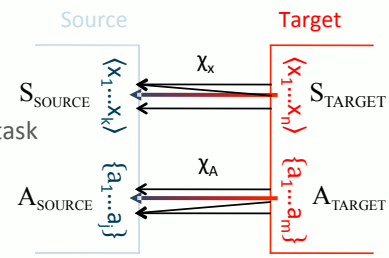
Inter Task Mappings



Oregon State University

Inter Task Mappings

- χ_x: $s_{target} \rightarrow s_{source}$
  - Given state variable in target task (some x from $s=x_1, x_2, \ldots x_n$)
  - Return corresponding state variable in source task
- χ_A: $a_{target} \rightarrow a_{source}$
  - Given action in target task
  - Return corresponding action in source task



- Intuitive mappings exist in some domains (Oracle)
- Used to construct transfer function

Oregon State University

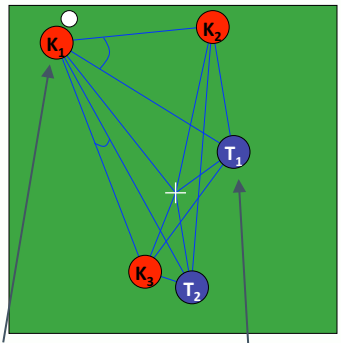
Q Value reuse

$$Q(s,a) = Q_{sourceFA}(\chi_X(s), \chi_A(a)) + Q_{targetFA}(s,a)$$

- Can be considered a type of reward shaping
- Directly use Q values from source task to bias learner in target task
- Not function-approximator specific
- No initialization step needed between learning the two tasks
- Drawbacks: an increased lookup time and larger memory requirements

Oregon State University

Keepaway [Stone, Sutton, and Kuhlmann 2005]



Keeper with ball may *hold ball* or *pass* to either teammate

Both **takers** move towards player with ball

Goal: Maintain possession of ball

3 vs. 2:

5 agents
3 (stochastic) actions
13 (noisy & continuous) state variables

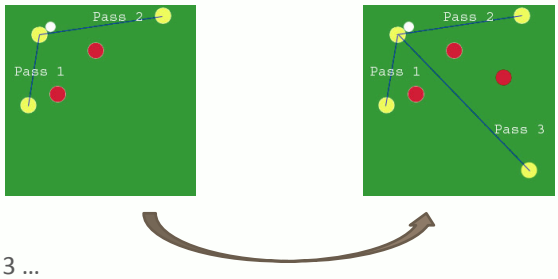
4 vs. 3:

7 agents
4 actions
19 state variables

Oregon State University

Transfer Learning in Keepaway

- 3 vs. 2 Keepaway

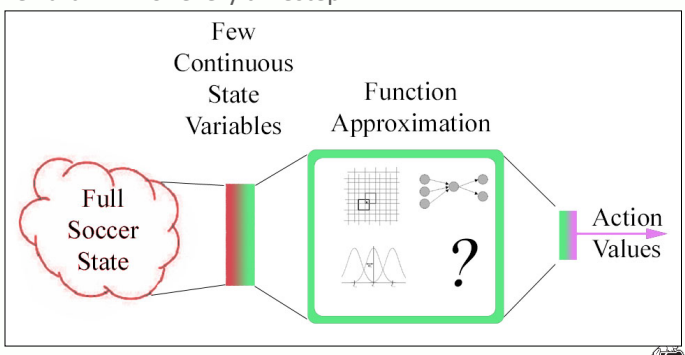


- Or 4 vs. 3 …
- Can learning transfer from 3 vs. 2 to 4 vs. 3?

Oregon State University

Learning Keepaway

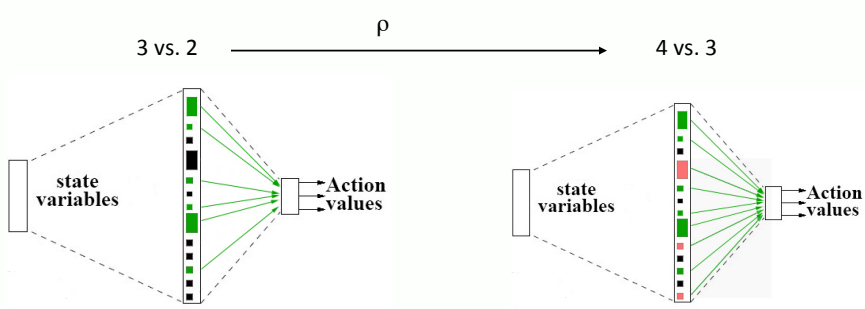
- Sarsa update
  - RBF and neural network approximation successful
- $Q^{\pi}(s,a)$: Predicted number of steps episode will last
  - Reward = +1 for every timestep



Oregon State University

ρ's Effect on function approximator

- For each weight in 4 vs. 3 function approximator:
  - Use inter-task mapping to find corresponding 3 vs. 2 weight

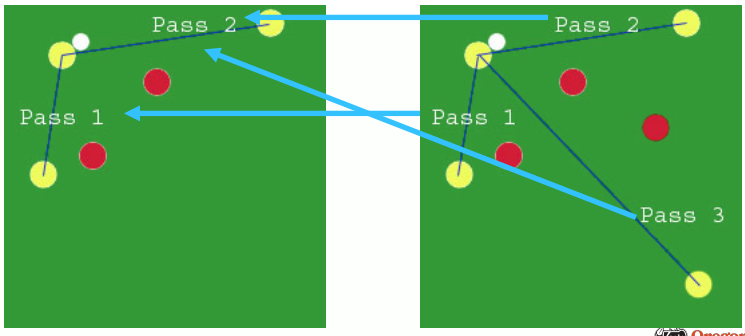


Oregon State University

Keepaway Hand-coded x_A

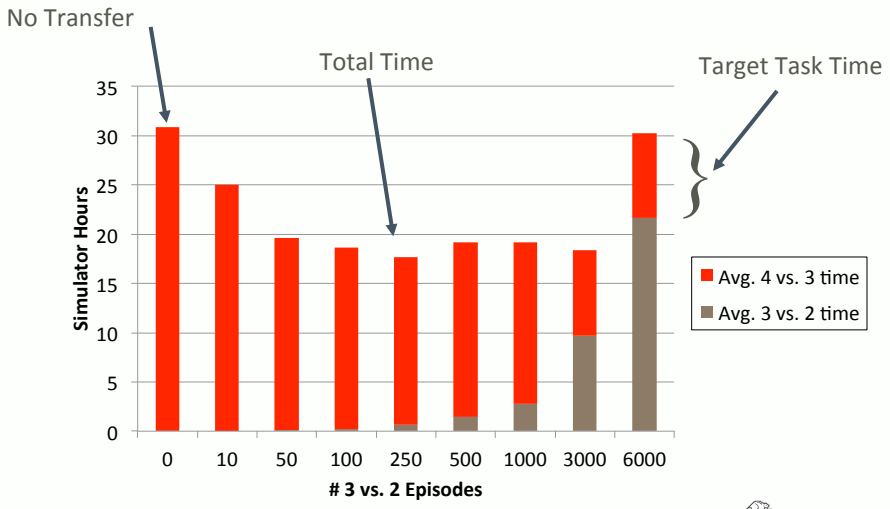
Actions in 4 vs. 3 have "similar" actions in 3 vs. 2

- $Hold_{4v3} \rightarrow Hold_{3v2}$
- $Pass1_{4v3} \rightarrow Pass1_{3v2}$
- $Pass2_{4v3} \rightarrow Pass2_{3v2}$
- $Pass3_{4v3} \rightarrow Pass2_{3v2}$



Oregon State University

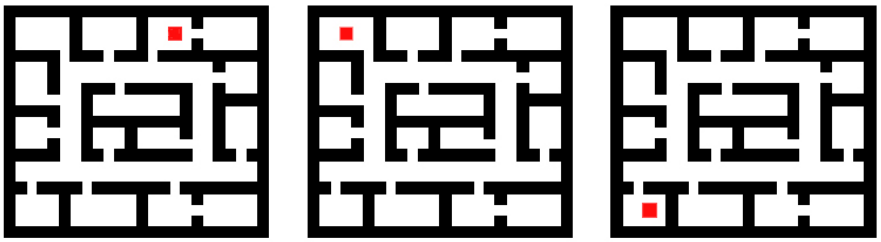
Value Function Transfer: Time Threshold in 4 vs. 3



Oregon State University

Example Transfer Domains

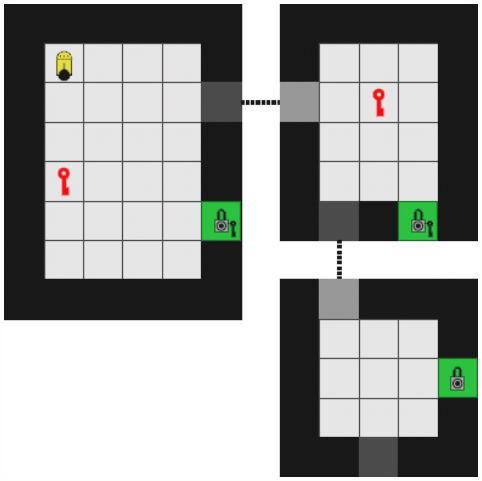
- Series of mazes with different goals [Fernandez and Veloso, 2006]



Oregon State University

Example Transfer Domains

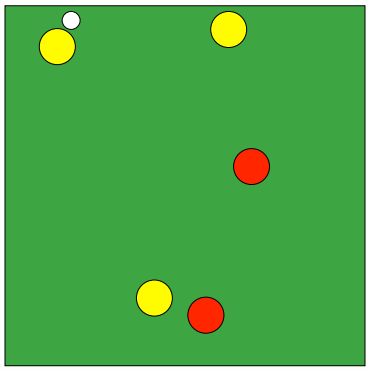
- Series of mazes with different goals [Fernandez and Veloso, 2006]
- Mazes with different structures [Konidaris and Barto, 2007]

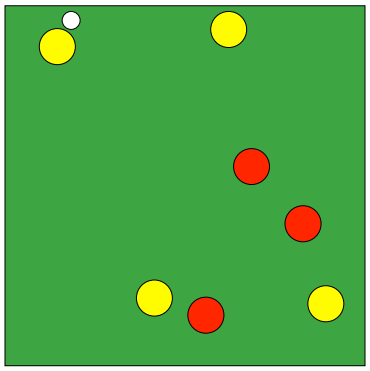


Oregon State University

Example Transfer Domains

- Series of mazes with different goals [Fernandez and Veloso, 2006]
- Mazes with different structures [Konidaris and Barto, 2007]
- Keepaway with different numbers of players [Taylor and Stone, 2005]

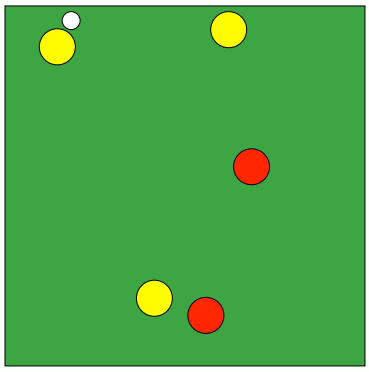


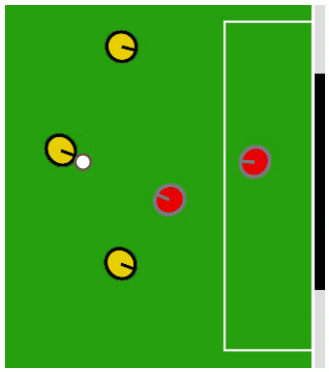


Oregon State University

Example Transfer Domains

- Series of mazes with different goals [Fernandez and Veloso, 2006]
- Mazes with different structures [Konidaris and Barto, 2007]
- Keepaway with different numbers of players [Taylor and Stone, 2005]
- Keepaway to Breakaway [Torrey et al, 2005]





Oregon State University

Transfer Learning

- So far, we talked about transfer learning in a specific domain
  - Task: MDP
  - Domain: Setting for semantically similar task
- What about ***Cross-Domain*** Transfer?
  - Source task could be much simpler
  - Source and Target can be less similar
- What about transferring policies (or just rules) ?

Oregon State University

Rule Transfer Overview

Learn π → Learn D_{source} → Translate (D_{source}) → D_{target} → Use D_{target}

1. Learn a policy ($\pi : S \rightarrow A$) in the source task
   - TD, Policy Search, Model-Based, etc.
2. Learn a decision list, D_{source}, summarizing π
3. Translate (D_{source}) → D_{target} (applies to target task)
   - State variables and actions can differ in two tasks
4. Use D_{target} to learn a policy in target task

Allows for different learning methods and function approximators in source and target tasks

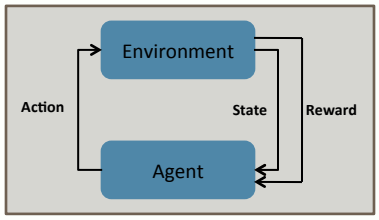
Oregon State University

Rule Transfer Details

Learn π → Learn D_{source} → Translate (D_{source}) → D_{target} → Use D_{target}

- For example, use Sarsa
  - $Q : S \times A \rightarrow$ Return
- Other learning methods possible

Source Task

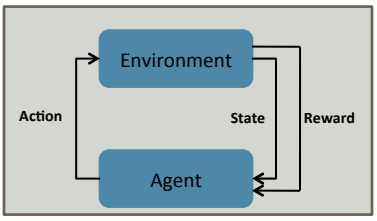


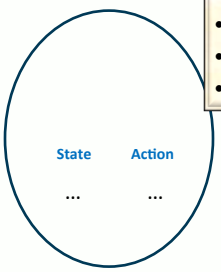
Oregon State University

Rule Transfer Details

Learn π → Learn D_{source} → Translate (D_{source}) → D_{target} → Use D_{target}

- Use learned policy to record S, A pairs
- Extract a rules list (for example: use JRip (RIPPER in Weka))





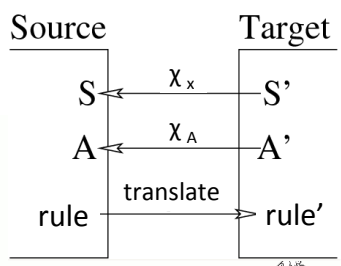
- IF $s_1 < 4$ and $s_2 > 5 \rightarrow a_1$
- ELSEIF $s_1 < 3 \rightarrow a_2$
- ELSEIF $s_3 > 7 \rightarrow a_1$
- ...

Oregon State University

Rule Transfer Details

Learn π → Learn D_{source} → Translate (D_{source}) → D_{target} → Use D_{target}

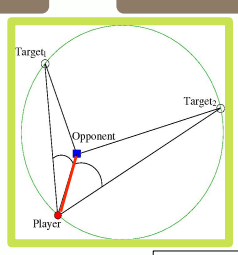
- Inter-task Mappings
- χ_x: $s_{target} \rightarrow s_{source}$
  - Given state variable in target task (some x from $s = x_1, x_2, \ldots x_n$)
  - Return corresponding state variable in source task
- χ_A: $a_{target} \rightarrow a_{source}$
  - Similar, but for actions

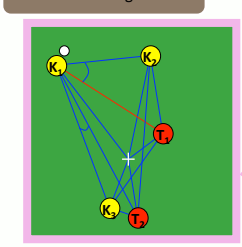


Oregon State University

Rule Transfer Details

Learn π → Learn D_{source} → Translate (D_{source}) → D_{target} → Use D_{target}





χ_A

Stay	Hold Ball
Run_{Near}	Pass to K_2
Run_{Far}	Pass to K_3

χ_x

dist(Player, Opponent)	dist(K_1,T_1)
…	…

IF dist(Player, Opponent) > 4 → Stay → IF dist(K_1,T_1) > 4 → Hold Ball

Oregon State University

Rule Transfer Details

Learn π → Learn D_{source} → Translate (D_{source}) → D_{target} → Use D_{target}

- Many possible ways to use D_{target}
  - Value Bonus (shaping)
  - Extra Action (initially force agent to select)
  - Extra Variable (initially force agent to select)
- Assuming TD learner in target task
  - Should generalize to other learning methods

Evaluate agent's 3 actions in state $s = s_1, s_2$

$Q(s_1, s_2, a_2, a_1) = 5$

$Q(s_1, s_2, a_2, a_2) = 9$

$Q(s_1, s_2, a_2, a_3) = 4$

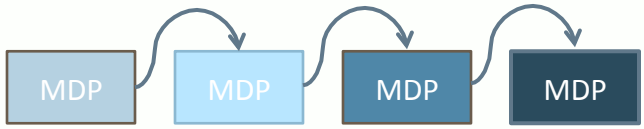
$Q(s_1, s_2, a_4) = 7$ (take action a_2)

$D_{target}(s) = a_2$

Oregon State University

Problem Statement

- Meta-planning problem for agent learning
  - Learn from scratch?
  - Or learn through a series of similar tasks?
  - Start with similar, simple task and move closer to target task



Oregon State University