# REINFORCEMENT LEARNING

ADAM ECK (SUPPLEMENTED BY LEEN-KIAT SOH)

University of Nebraska Lincoln

CSCE 990: Advanced MAS

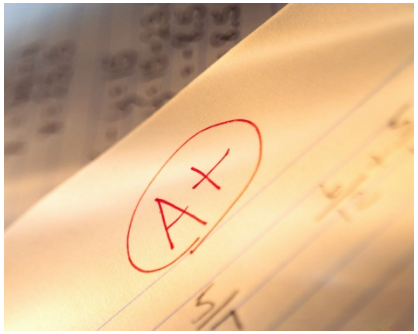
# Machine Learning

- 3 Primary Types of Machine Learning
  - Supervised Learning
    - Learning how to prediction and classify
    - Decision trees, neural networks, SVMs
  - Unsupervised Learning
    - Learning how to grouping and find relationships
    - Clustering: k-Means, spectral
  - Reinforcement Learning (RL)
    - Learning how to act and make decisions
    - Q-learning, Rmax, REINFORCE

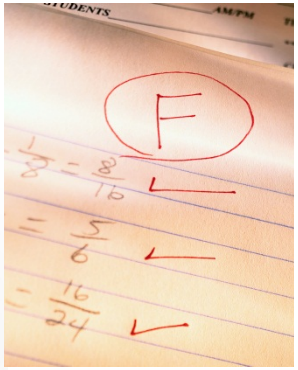
# Reinforcement Learning

- Learn rewards based on environment feedback

**Positive Rewards**




**Negative Rewards**




# Single Agent Reinforcement Learning

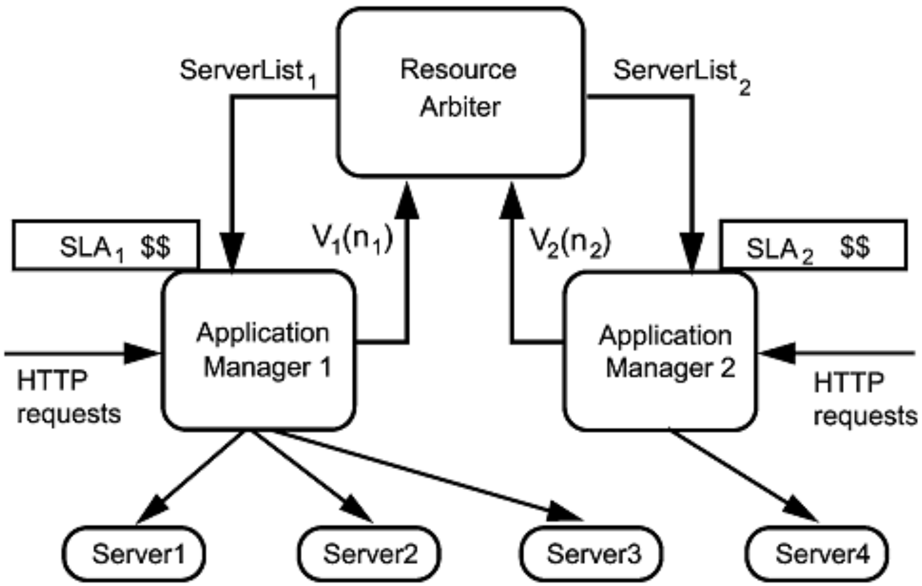
- Framework: Markov Decision Process
  - States S – description of environment
  - Actions A – action taken to change environment
  - Transitions T(s, a, s') – models dynamic changes in environment
  - Reward R(s,a) – numeric result of action

# Single Agent Reinforcement Learning

- Reinforcement Learning Problem
  - Given *S* and *A*
  - Need to learn *R* (and maybe *T*)
    - Mapping of state/action pairs to:
      - Reward values
      - Probabilities of next states
    - From history (state/action/reward sequence)
      - $H = s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots$
  - Use learned rewards to form policy $\pi$
    - Plans of actions maximizing rewards
    - Determines how agent acts, given current state

# Examples

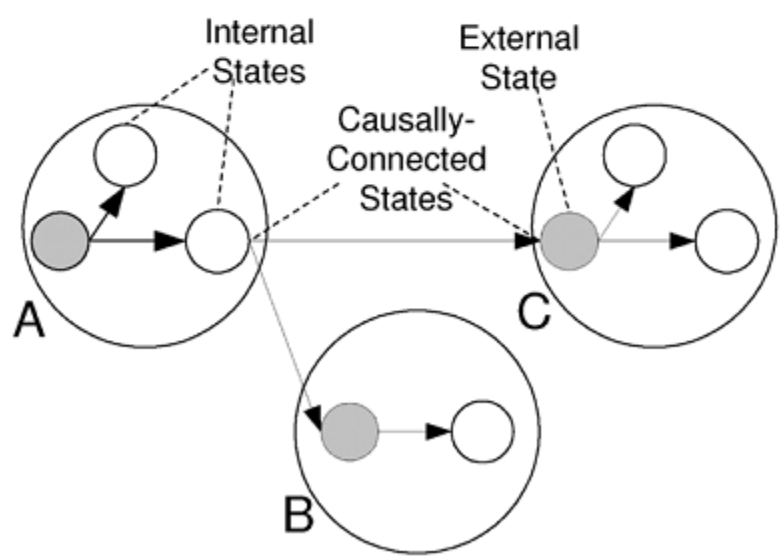
- Web server allocation (Tesauro et. al, 2007)
  - Learn how many servers to assign to applications based on incoming requests
  - Goal: maximize SLA revenue




Source: (Tesauro et al., 2007)

# Examples

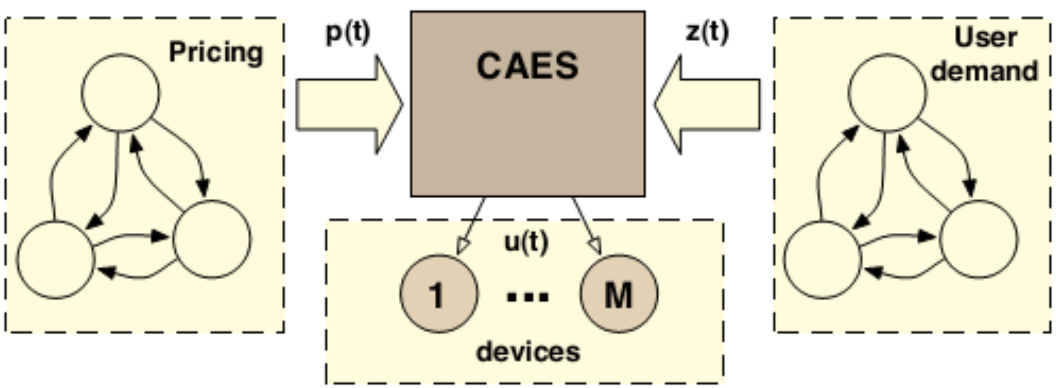
- Ad hoc networks (Dowling et. al, 2005)
  - Learn how to route packets through distributed network
  - Goal: maximize packet delivery and adapt to changing network conditions (e.g., node failure)




Source: (Dowling et al., 2005)

# Examples

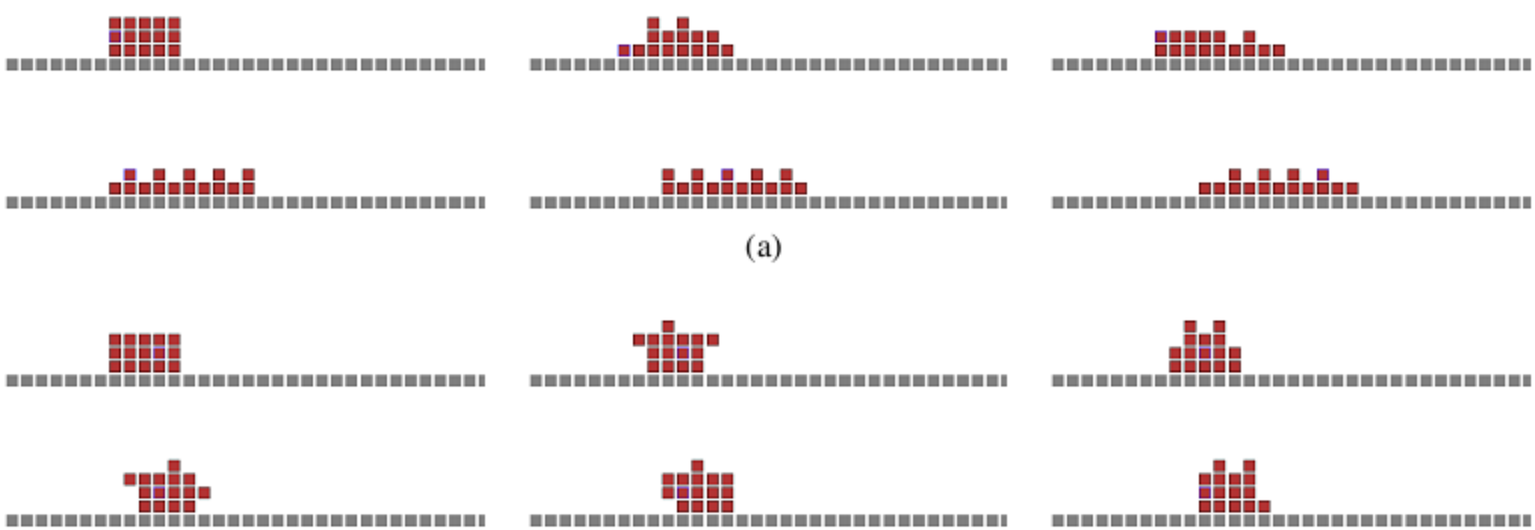
- Smart Grid (O'Neill et. al, 2010)
  - Learn how to allocate energy to residences and optimize schedule of energy usage
  - Goal: Reduce cost of energy usage




Source: (O'Neill et al., 2010)

# Examples

- Modular Robots (Varshavskaya et. al, 2008)
  - Each robot module learns how to operate with a team
  - Goal: move a robot consisting of multiple modules across an open space



(a)

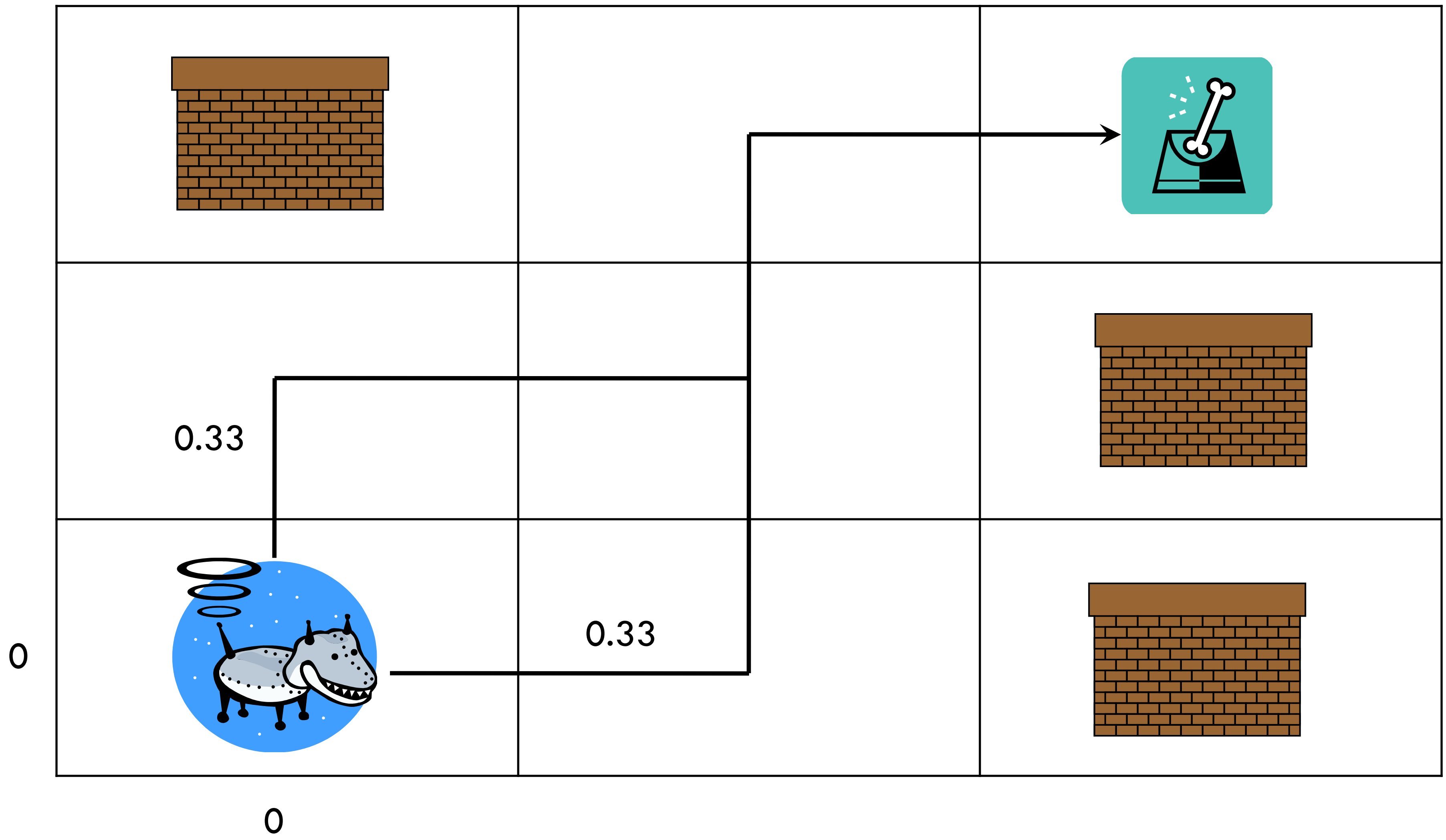
(b)

Source: (Varshavskaya et al., 2008)

# Examples

- Poker Agents
  - Learn how to play based on opponents' behavior and available cards
  - Goal: maximize winnings

# Running Example



# Example Comparison

| | Web Server Allocation | Ad Hoc Networks | Smart Grid | Modular Robots | Poker Agents | Maze |
|---|---|---|---|---|---|---|
| **States S** | # incoming requests | Have packet? Packet transmitted? | Price of energy, user demand | Positions of all robots | Cards, opponent model | Grid location |
| **Actions A** | # servers to assign | Transmit, find neighbors | Allocation of energy | Move module | Raise, check, fold | Movement: N, S, E, W |
| **Transitions T** | Change in requests over time | Transmission success probability | Change in price and demand | Change in team configuration | Changes in cards and model | Change in location |
| **Rewards R** | Revenue $$$ | Cost of sending, decay in learning | User's utility of allocation | +/- if move in correct/incorrect direction | Chips won | Inverse of distance to goal |

# Types of RL

- Model-free RL
  - Learn reward for controller
  - Ignore model parameters
  - Example: Riding a bicycle

- Model-based RL
  - Learn underlying model of environment, then solve
    - Often learn MDP
  - Example: Playing poker

# Types of RL

- Use model-free RL when…
  - Only care about rewards (and not dynamics)
  - Very simple environment with fixed transitions…

    …or very complex environment
  - More concerned with fast learning than optimal performance

- Use model-based RL when…
  - Want to consider dynamics
  - Moderately complex environment with stochastic transitions
  - More concerned with optimal performance and can afford longer learning phase

# Types of RL

- Web server allocation (Tesauro et. al, 2007)
  - Model-free (function approximation with SARSA rule)
- Ad hoc networks (Dowling et. al, 2005)
  - Model-based (CRL)
- Smart Grid (O'Neill et. al, 2010)
  - Model-free (Q-Learning)
- Modular Robots (Varshavskaya et. al, 2008)
  - Model-free (but assume know dynamics *a priori*)

# Types of RL

- Poker Agents
  - Model-based (if opponent modeling)
    - Want to determine how opponent will respond
  - Model-free (if focused only on cards)

- Robotic Maze
  - Model-free if perfect actuators
  - Model-based if actuators can fail

# Q-Learning

- Q-Learning: classic model-free RL algorithm (Watkins, 1989)
  - Simple but powerful and effective
  - Learns reward function as a table, based on current state and chosen action
  - Guaranteed convergence to true reward function with enough exploration
  - Assumes discrete state/action spaces

# Q-Learning

- Learned rewards stored as a Q-table

| | Actions |
|---|---|
| States | Reward Values<br>Q(s,a) |

- Initialize table
  - Equal values
  - Random values
  - A priori information

# Q-Learning

- Update Q-table after every action
  - $Q'(s,a) = (1 - \alpha)Q(s,a) + \alpha\,[R(s,a) + \gamma \max_{a' \in A} Q(s',a')]$

- $\alpha$ = learning rate
  - Balances old knowledge with new information
- $\gamma$ = discount rate
  - Determines how "forward thinking" the agent is
    - Myopic vs. non-myopic
  - Accounts for uncertainty in future rewards

# Q-Learning

- Policy for choosing actions
  - Pick action with highest reward in current state

$$\pi(s) = \underset{a \in A}{\operatorname{argmax}}\ Q(s,a)$$

  - Looks myopic, but is actually non-myopic
    - Future rewards already considered in Q-table
    - Assuming $\gamma > 0$

# RMax

- RMax: popular model-based RL algorithm (Brafman and Tennenholtz, 2002)
  - Simple but powerful and effective
  - Represents learned functions as tables
  - Assumes discrete state/action spaces

  - Also ***learns state transitions***
  - Probably Approximately Correct (PAC) learning algorithm
  - Converges in polynomial time

# RMax

- Maintain tables for *both* rewards and transitions
  - Still based on states/action pairs, like in Q-Learning

- Initialization
  - Assume all rewards equal to same value
    - Value = maximum possible reward value (RMax)
  - Assume fixed transitions to special state
    - Don't know in advance what states lead to other states

# RMax

- Update tables after $k$ fixed number of interactions with the environment for a state/action pair
  - Often $k = 5, 10, 20$, etc.

- Reward updates
  - Store **first** reward experienced for a state/action
  - Store **expected** reward over $k$ iterations for a state/action
  - Calculate **probabilities** of different rewards based on $k$ rewards

- Transition updates
  - Count number of state transitions after state/action
  - Calculate probabilities based on first $k$ transitions

# RMax

- Policy for choosing actions
  - Build a MDP model based on learning and solve
  - Maximize current and future rewards from the current state, considering state transitions
    - Discount future rewards since uncertain transitions

$$V(s) = \max_{a \in A} R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s')$$

$$\pi(s) = \operatorname*{argmax}_{a \in A} R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s')$$

  - Can limit forward search to *n* future actions

# Exploration vs. Exploitation

- Difficult problem: should I keep learning, or use what I’ve learned?
  - Use what I’ve learned
    - More current rewards, less future rewards
  - Additional learning
    - More future rewards, less current rewards

- **Exploration**: try to learn about uncertain state/action pairs
- **Exploitation**: maximize rewards based on learned information

# Exploration vs. Exploitation

- Different methods to balance exploration and exploitation (Vermorel and Mohri, 2005):
  - ε-greedy
    - Explore random action with probability ε (e.g., 10%)
    - Exploit best action with probability 1-ε
  - Softmax: **similar to humans (Daw et. al, 2006)**
    - Choose actions with probabilities based on value of rewards

$$P(a|s) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{a' \in A} e^{\frac{Q(s,a')}{T}}}$$

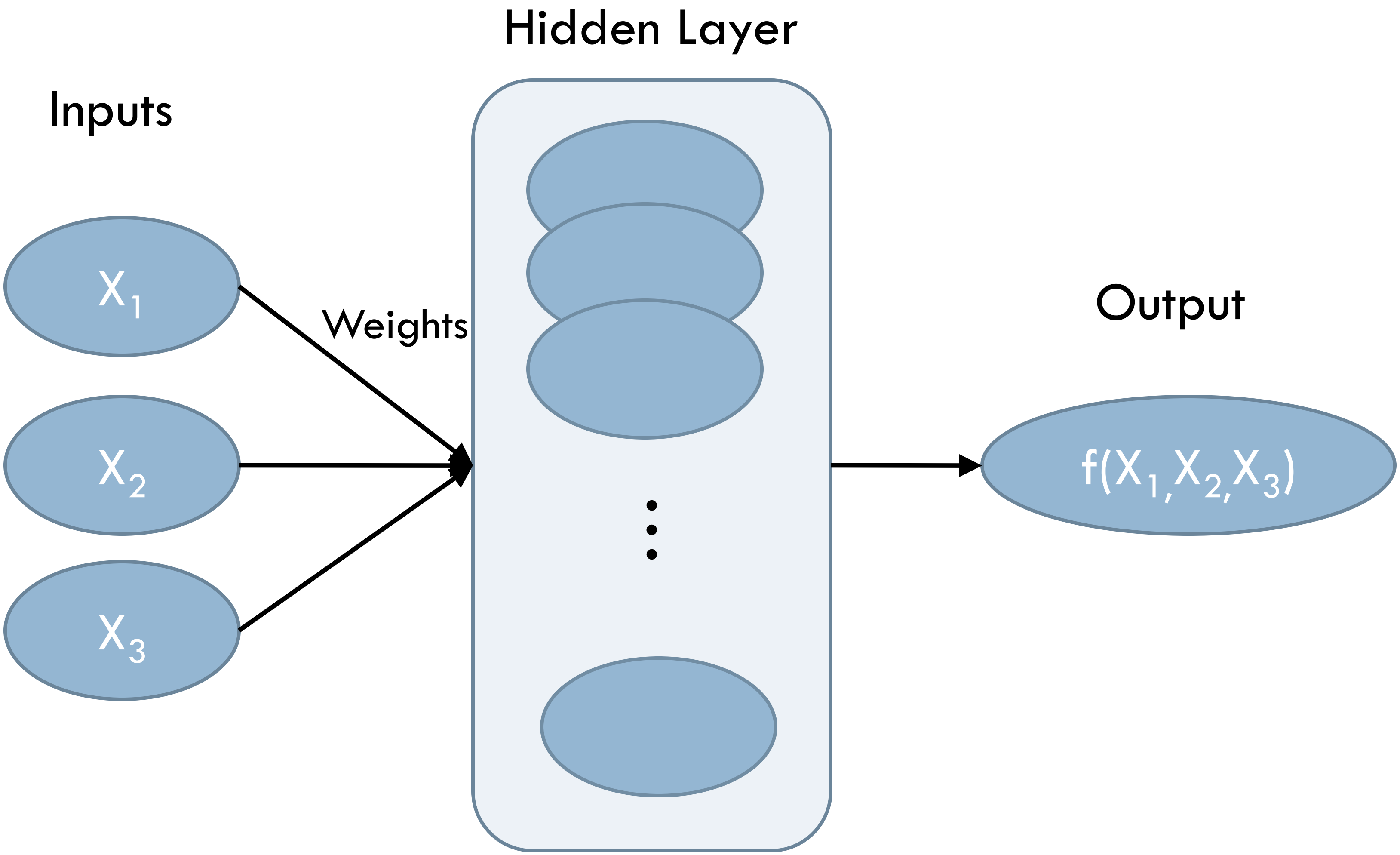
    - Higher rewards = more likely to be chosen

# Continuous RL

- Both Q-Learning and RMax assume discrete state/action spaces
  - Valid assumption in many MAS
    - Can convert continuous spaces into discrete
      - By assigning *bins* to ranges of continuous values

- What if continuous?
  - Need to use function approximation
    - Learn a generic model of reward (and maybe transition) function output based on inputs
    - No tables
  - Common approach: neural networks

# Neural Networks



# Continuous RL

- REINFORCE (Williams, 1992)
  - Train neural network to learn both reward function R and policy $\pi$
    - Reward function predicts rewards based on current state and action inputs
    - Policy probabilistically chooses actions given current state input based on learned rewards
      - Similar to Softmax, but done implicitly within the neural network
  - Use *eligibility* backpropagation to train the policy
    - Different from neural network use in supervised learning

# Summary

- Use RL to learn how to act and make decisions
  - Maximize rewards learned from interactions with the environment
- Different types of algorithms
  - Model-free: focus just on rewards
    - e.g., Q-Learning
  - Model-based: learn full model of environment, then solve the model
    - e.g., RMax
- Exploration vs. Exploitation
  - Control learning vs. using learning

# More on RL: Model-free vs. Model-based

- the main difference between model-free and model-based RL is that
  - ***model-based also learns the underlying dynamics of the environment*** (the stochastic $T$ function in fully observable environments), whereas …
  - that ***knowledge is ignored in model-free***
    - $T$ is very rarely deterministic in the real-world, but learning updates do not happen until $s'$ is known in Q-learning, so there is no need to consider $T$
- The other advantage of learning $T$ explicitly is that the agent can actually do planning in model-based RL
  - with T, it can project possible future states during planning
  - That isn't explicitly possible with model-free algorithms such as Q-learning

# More on RL: Model-free vs. Model-based

- In Shoham's book, ***belief-based learning*** is when the agent considers the probabilities of each possible action of the other agents
  - This is an improvement because often the total reward (and thus the Q function) depends not just on the agent's own action, but on the actions of the other agents.
- Belief-based learning could be considered *model-based learning* if the agent learns the *Pr_i* function while it operates in the environment
  - If *Pr_i* is fixed from the start (e.g., to a uniform distribution, or some informed prior), then it wouldn't be model-based learning
  - Although, some might argue that any RL is model-based if the agent has a model of the environment, not necessarily only if it learns that model …

# More on RL: Model-free vs. Model-based

- Even more philosophical …
- In a stochastic game setting (Shoham's book), the transition function represents which normal-form game (i.e., which payout table) ***appears next*** after the agents choose and execute their actions
- In single agent learning, the agent is really ***playing a game against nature*** (so there is only one column in the payout table for the agent itself), and ***nature determines the stochastic next game*** (i.e., state of the environment).
- So in that case, learning the *T* function in a single agent learning problem is equivalent to learning the *Pr_i* function—might be "altogether"—describing what nature will play
- ***Model-based?***

# More on RL

- Videos of AlphaGo: explanatory clips before it beat the Go world champion—Lee Sedol
  - https://deepmind.com/alpha-go
- Videos of Deep Mind playing Atari games earlier, before it moved on to Go
  - https://www.youtube.com/watch?v=V1eYniJ0Rnk
  - https://www.youtube.com/watch?v=r3pb-ZDEKVg
  - http://www.theverge.com/2016/6/9/11893002/google-ai-deepmind-atari-montezumas-revenge

# More on RL: Learning vs. Planning?

- Difference between RL and planning (specifically Q-Learning vs. MDP or POMDP planning)?
- The internal math looks very similar:
  - for both, we create a Q-table (also the Value network learned by AlphaGo) …
  - from which we determine a policy of actions to take (also the Policy network learned by AlphaGo) …
  - As they work longer and longer, both improve over time
- The difference between the two is *what powers the improvement*, and *which direction through time they gain that improvement*

# More on RL: Learning vs. Planning?

- Mitchell's definition of learning: A computer program is said to learn from experience E with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*
- In **RL**, the tasks *T* are whatever the agent is trying to do, the performance measure *P* is usually *discounted cumulative rewards*, and the experience *E* are the ($s'$, $r$) pairs of state transitions and rewards the agent observes after it takes action *a* in state *s*. The more experience *E*, the better the agent performs by learning how the environment changes and how it is rewarded for those changes
- In **planning**, *T* and *P* are the same, but the experience *E* isn't necessary -- the agent ***already knows*** what ($s'$, $r$) it can get after taking action *a* in state *s*. Instead, the agent improves from having more *time* to consider future ($s'$, $r$) pairs -- that is, more contingencies of what it what it might encounter
- So the difference is planning for more possible experiences *in the future*, rather than gaining information from the experiences *it recently saw in the past*

# More on RL: Learning vs. Planning?

- So the difference is planning for more possible experiences ***in the future***, rather than gaining information from the experiences ***it recently saw in the past***

# More Information

Great general reference:

Sutton, R.S. and Barto, A.G. 1998. Reinforcement learning: an introduction. MIT Press:Cambridge, MA.

Available online free at:

http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html

# References


- Brafman, R.I. and Tennenholtz, M. 2002. R-max – A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*. 3. 213-231.
- Daw, N.D. et. al, 2006. Cortical substrates for exploratory decisions in humans, *Nature*. 441. 876-879.
- Dowling, J., et al. 2005. Using Feedback in Collaborative Reinforcement Learning to Adaptively Optimize MANET Routing", *IEEE Transactions on SMC, Part A*, 35(3). 360-372.
- O'Neill, D. et. al. 2010. Residental demand response using reinforcement learning. *Proc. of SmartGridComm'10*. 409-414.
- Tesauro et. al. 2007. On the user of hybrid reinforcement learning for autonomic resource allocation, *Cluster Computing*, 10. 287-299.
- Vermorel, J. and Mohri, M. 2005. Multi-armed bandit algorithms and empirical evaluation, *Proc. of ECML'05*, 437-448.
- Varshavskaya, P. et. al. 2008. Automated Design of Adaptive Controllers for Modular Robots Using Reinforcement Learning. *IJRR*. 27. 505-526.
- Watkins, C.J. 1989. Learning from delayed rewards. Ph.D thesis, Cambridge University.
- Williams, R.J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 229-256.