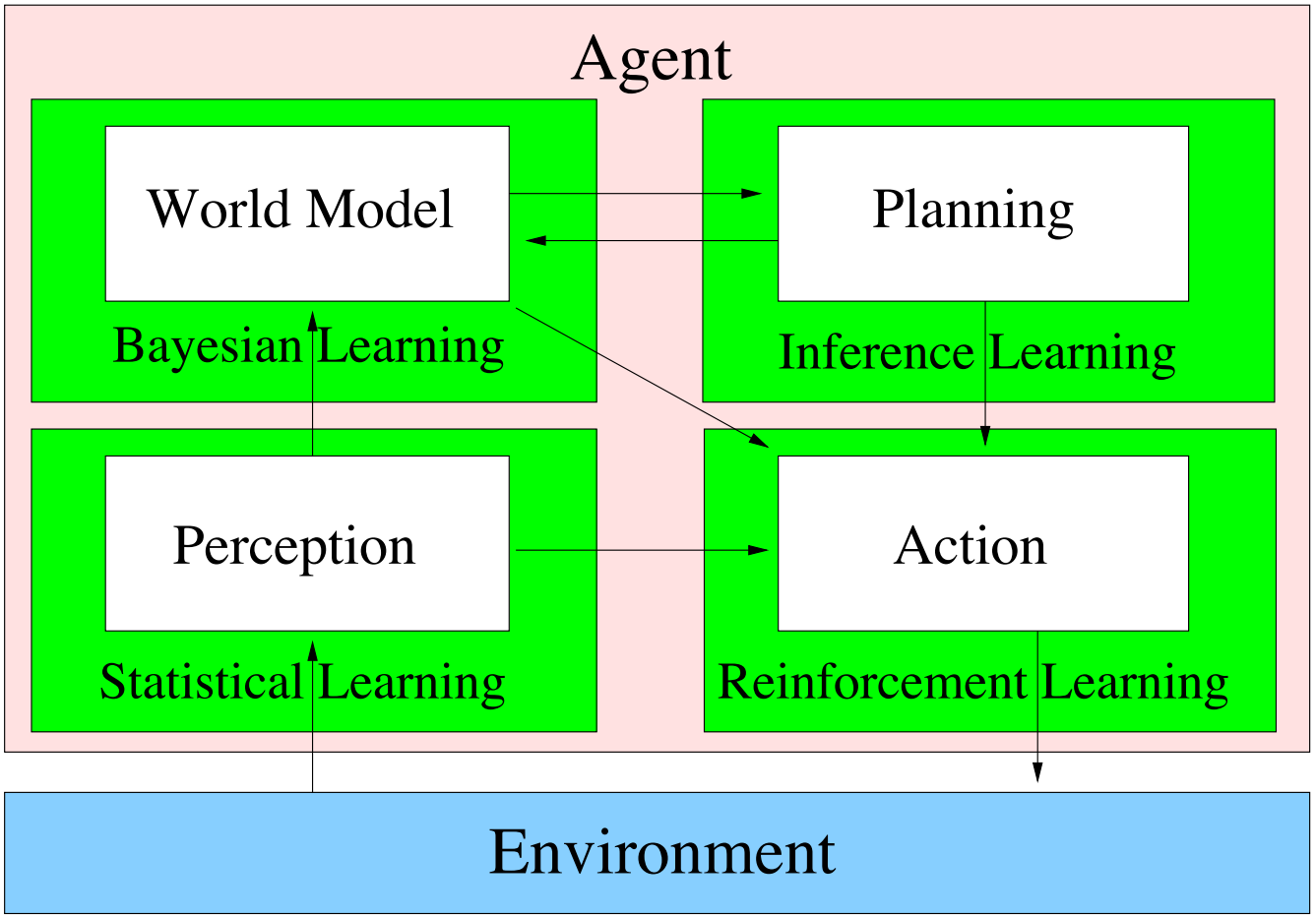
COMP3411/9814: Artificial Intelligence

Learning to Act
Reinforcement Learning

Lecture Overview

- Reinforcement Learning vs Supervised Learning
- Boxes
- Models of Optimality
- Exploration vs Exploitation
- Q-Learning

Learning Agent



Types of Learning

- Supervised Learning
  - Agent is presented with examples of inputs and their target outputs, and must learn a function from inputs to outputs that agrees with the training examples and generalizes to new examples
- Reinforcement Learning
  - Agent is not presented with target outputs for each input, but is periodically given a reward, and must learn to maximize (expected) rewards over time
- Unsupervised Learning
  - Agent is only presented with a series of inputs, and must find and aims to find structure in these inputs

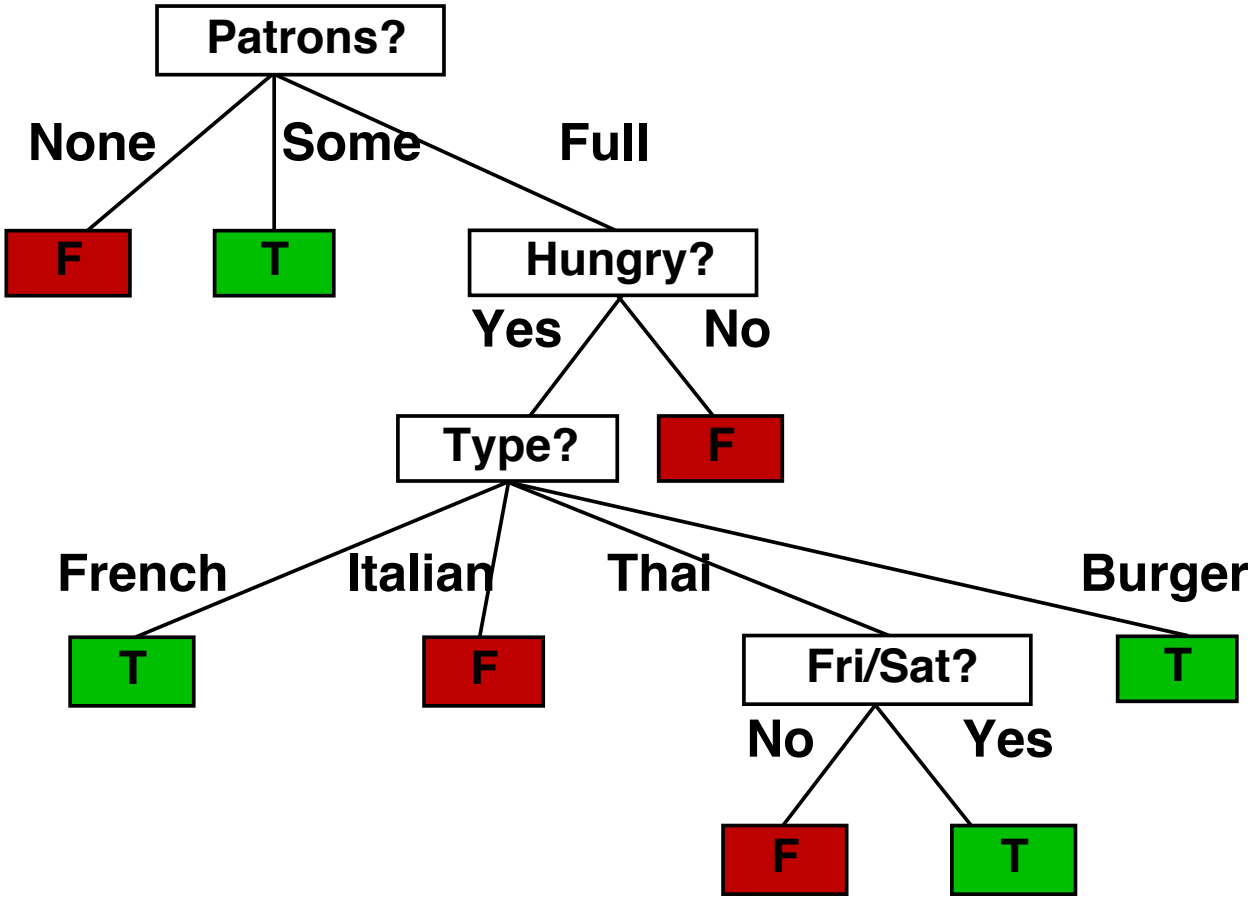
Supervised Learning

- Given a training set and a test set, each consisting of a set of items for each item in the training set, a set of features and a target output
- Learner must learn a model that can predict the target output for any given item (characterized by its set of features)
- Learner is given the input features and target output for each item in the training set
  - Items may be presented all at once (batch) or in sequence (online)
  - Items may be presented at random or in time order (stream)
  - Learner **cannot** use the test set **at all** in defining the model
- Model is evaluated by its performance on predicting the output for each item in the test set

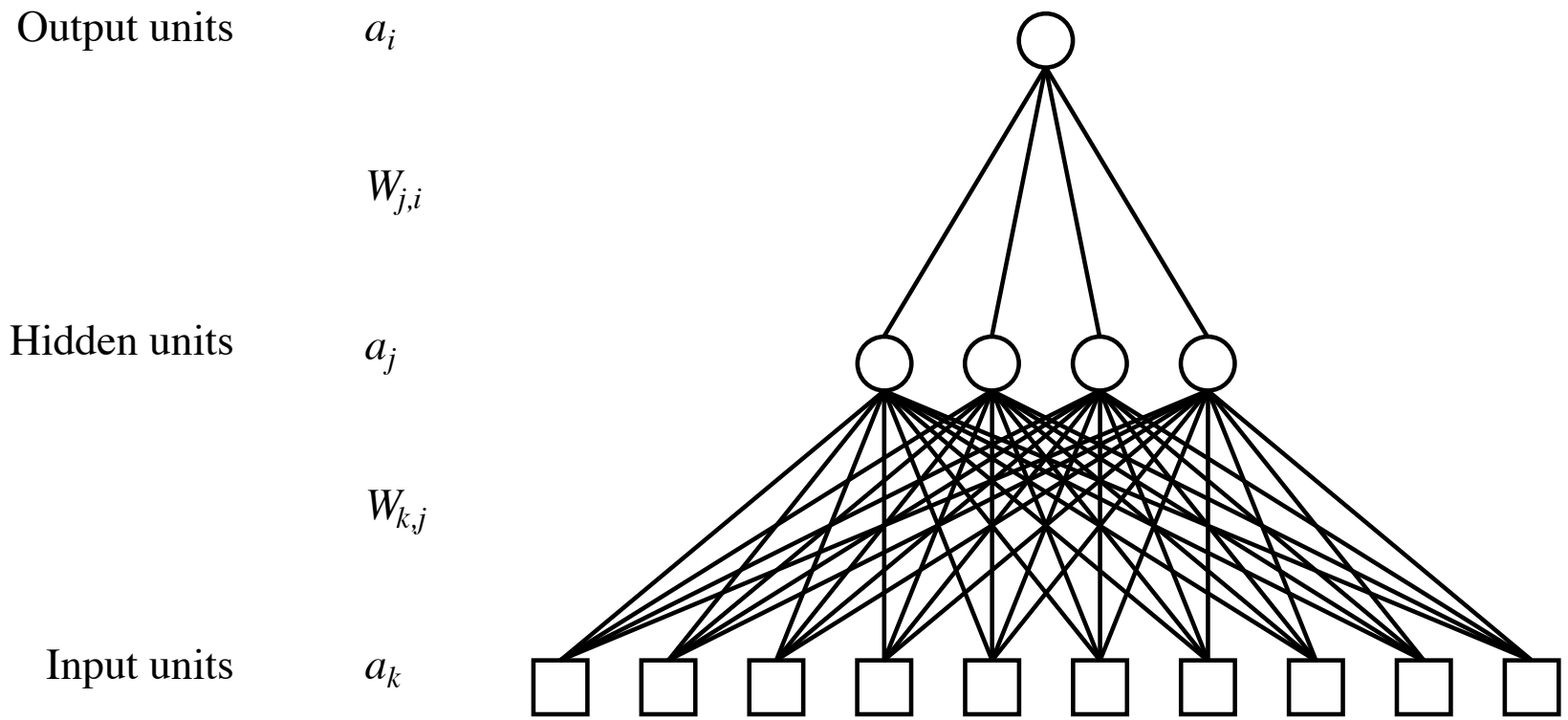
Supervised Learning

- Various learning paradigms are available:
  - Decision Trees
  - Neural Networks
  - .. others ..

Decision Tree



Neural Network



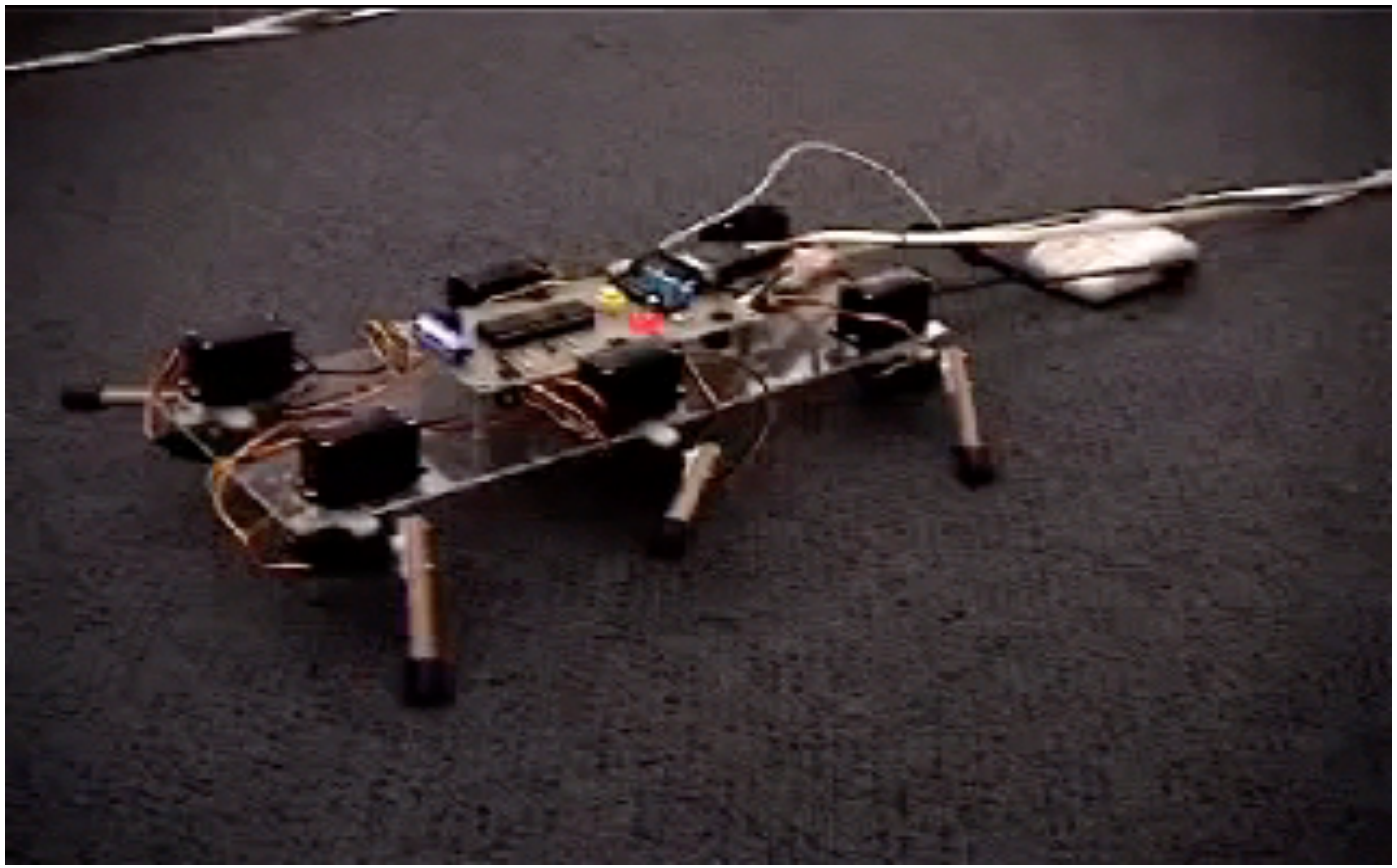
Learning Actions

Supervised Learning can also be used to learn Actions, if we construct a training set of situation-action pairs (called Behavioral Cloning).

However, there are many applications for which it is difficult, inappropriate, or even impossible to provide a “training set”

- Optimal control
  - mobile robots, pole balancing, flying a helicopter
- Resource allocation
  - job shop scheduling, mobile phone channel allocation
- Mix of allocation and control
  - elevator control, backgammon

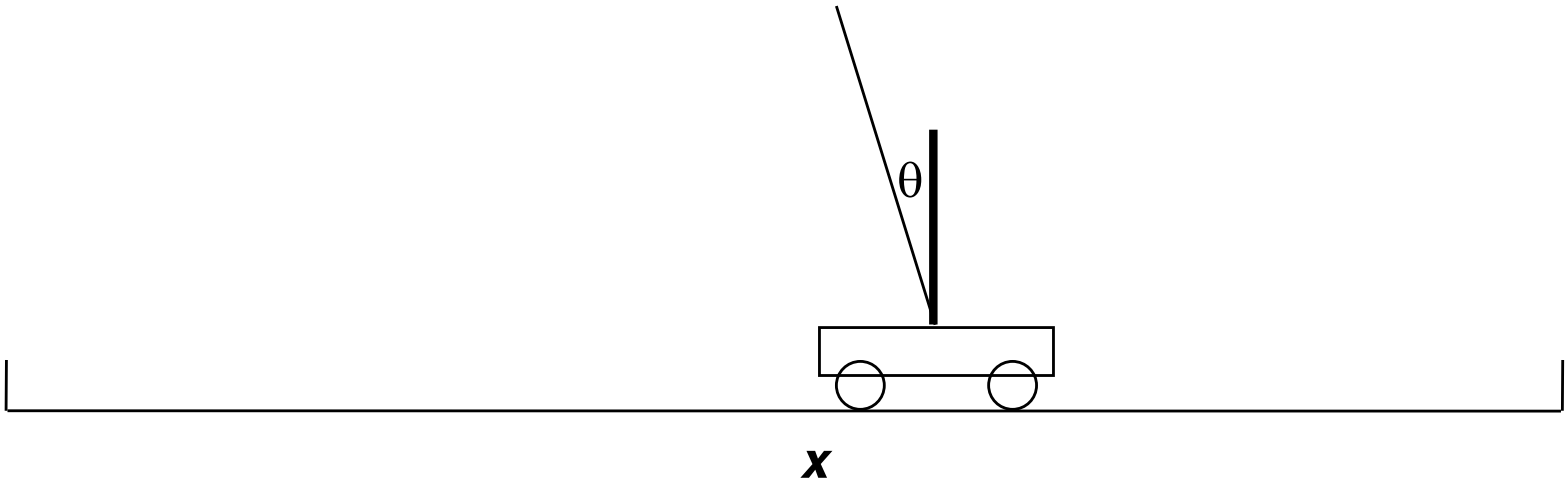
A Simple Learning Robot



A Simple Learning Robot

- "Stumpy" receives a *reward* after each action
  - Did it move forward or not?
- After each move, updates its *policy*
- Continues trying to maximise its reward

Pole Balancing



- Pole balancing can be learned the same way except that reward is only received at the end
  - after falling or hitting the end of the track

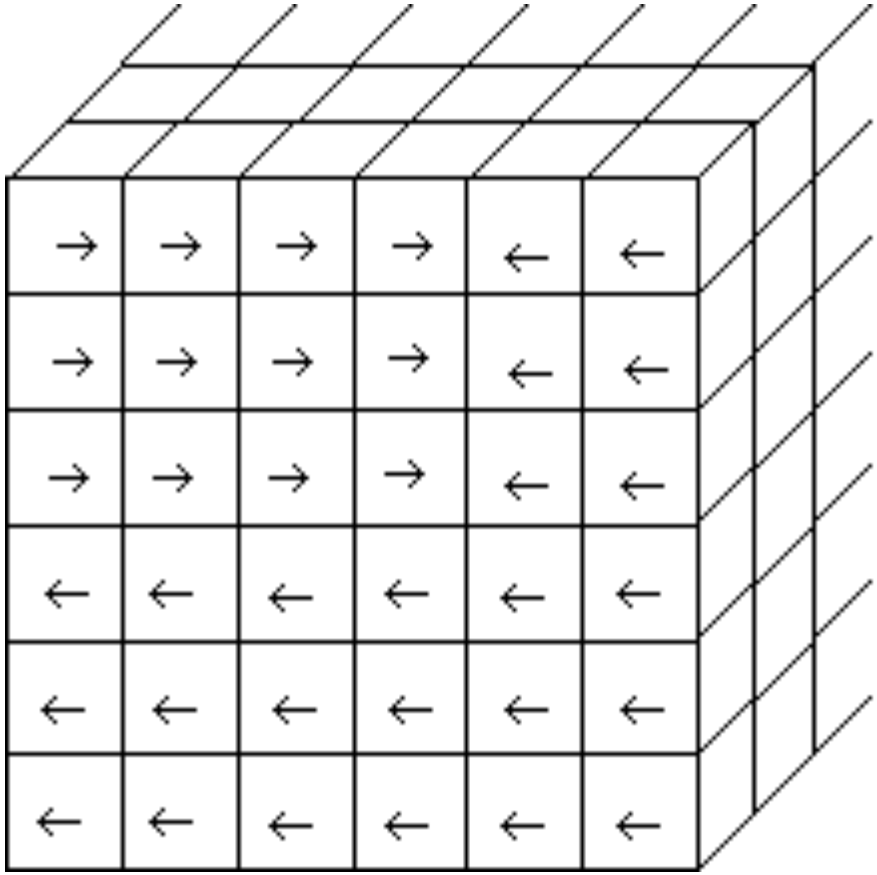
Boxes

State space is discretized

Each “box” represents a subset of state space

When system lands in a box, execute action specified

- left push
- right push



Update Rule

if an action has not been tested

choose that action

else if $\frac{LeftLife}{LeftUsage^k} > \frac{RightLife}{RightUsage^k}$

choose left

else

choose right

k is a bias to force exploration
e.g. k = 1.4

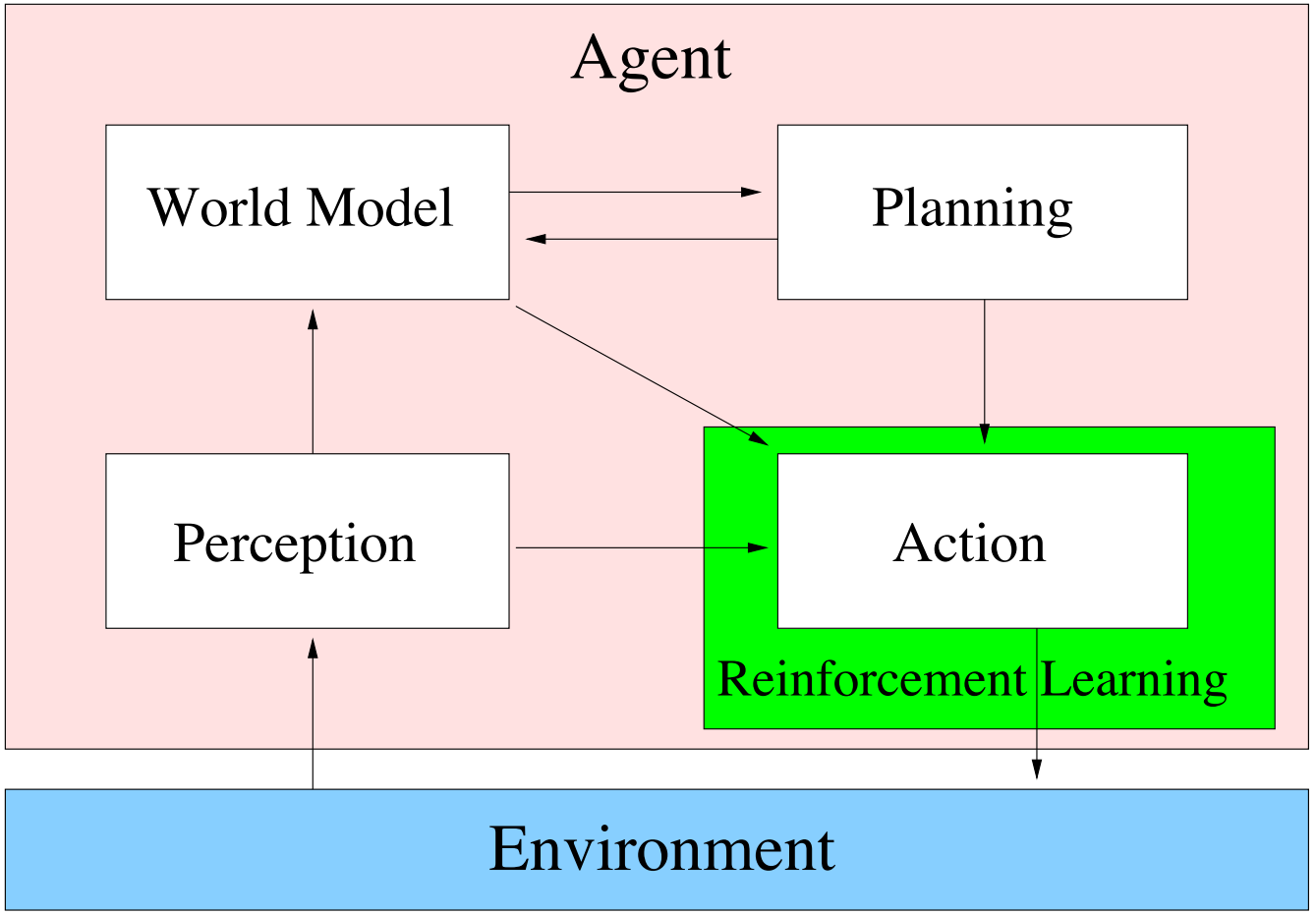
Performance

- BOXES is *much* faster than other suitable ML alg.
  - Only 75 trials, on average, to reach 10,000 time steps
- But only works for *episodic* problems
  - i.e. has a specific termination
- Doesn't work for continuous problems like Stumpy

Reinforcement Learning Examples

- Game - reward winning, punish losing
- Dog - reward obedience, punish destructive behaviour
- Robot - reward task completion, punish dangerous behaviour

Learning to Act - Reinforcement Learning Agent



Learning to Act - Reinforcement Learning Problem

A reinforcement learning (RL)

- Agent acts in an environment, observing its state and receiving rewards.
- From its perceptual and reward information, it must determine what to do.

- Single-agent reinforcement learning for fully observable environment

Reinforcement Learning Framework

An agent interacts with its environment.

There is a set *S* of *states* and a set *A* of *actions*.

At each time step t, the agent is in some state s_t. It must choose an action a_t , whereupon it goes into state

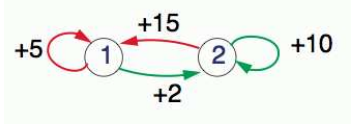
$s_t + 1 = \delta(s_t, a_t)$ and receives reward $r(s_t, a_t)$.

In general, $r()$ and $\delta()$ can be multi-valued, with a random element

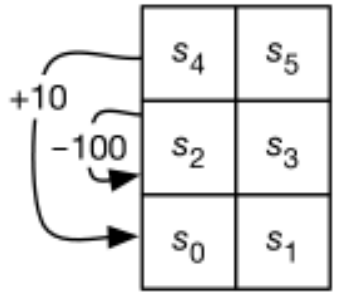
The aim is to find an optimal *policy* $\pi : S \rightarrow A$ which will maximize the cumulative reward.

Markov Decision Process (MDP)

The agent initially only knows the set of possible states and the set of possible actions.



- The dynamics, P(s' | a, s), and the reward function, R(s, a), are not given to the agent.
- After each action, the agent observes the state it is in and receives a reward.



P(s' | a, s), *the probability of the agent transitioning into state s' given that the agent is in state s* and does action a

Assume that current state has all the information needed to decide which action to take

Models of optimality

Is a fast nickel worth a slow dime?

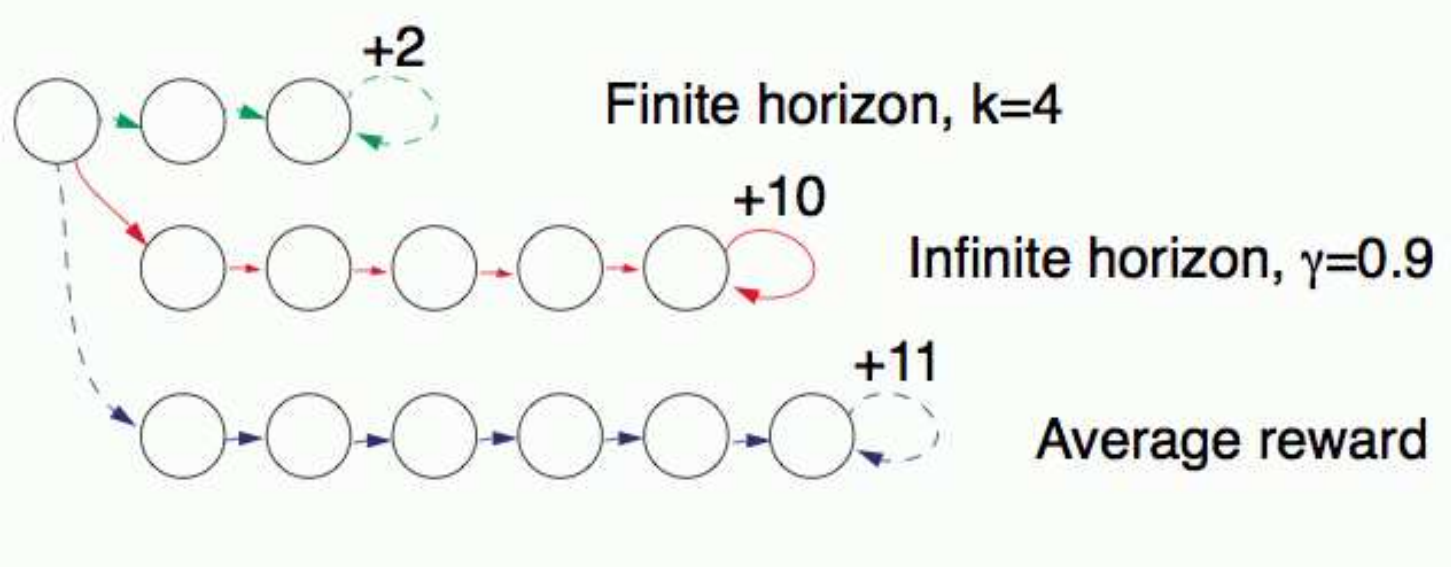
Finite horizon reward - total reward $\sum_{i=0}^{h} r_{t+i}$

Average reward $\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i}$

Infinite discounted reward $\sum_{i=0}^{\infty} \gamma^i r_{t+i}, \quad 0 \leq \gamma < 1$

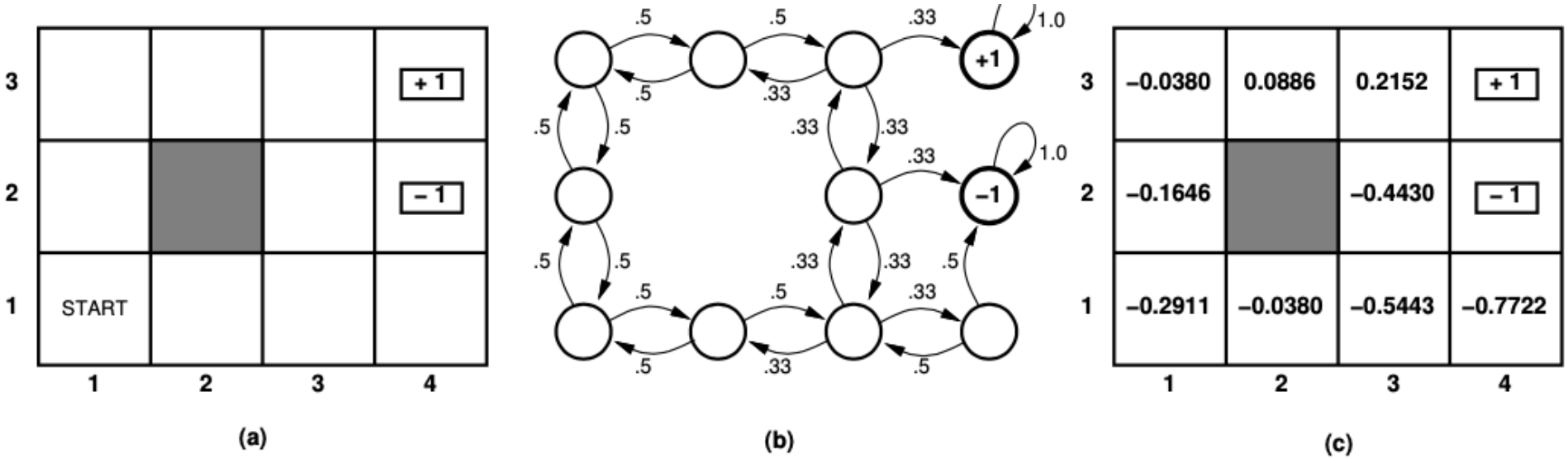
- Finite horizon reward is simple computationally
- Infinite discounted reward is easier for proving theorems
- Average reward is hard to deal with, because can't sensibly choose between small reward soon and large reward very far in the future

Comparing Models of Optimality



Value Function

For each state $s \in S$, let $V^*(s)$ be the maximum discounted reward obtainable from s.

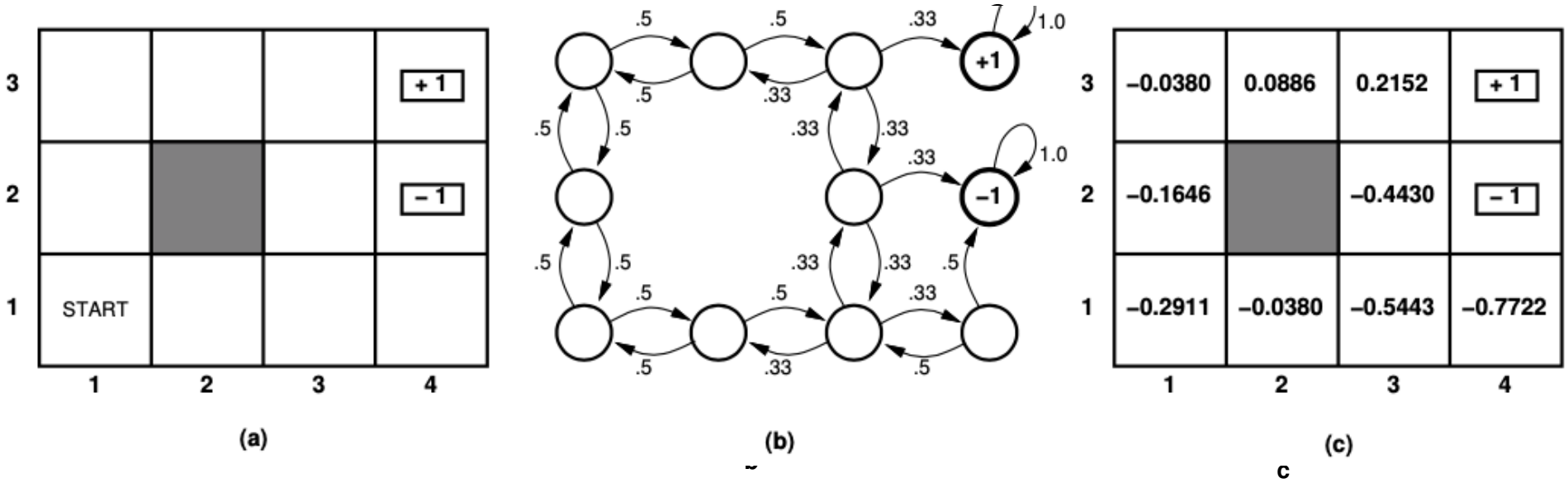


Learning this Value Function can help to determine the optimal strategy.

$V^{\pi}(s)$ is the expected value of following policy π in state s.

Value Function

For each state $s \in S$, let $V^*(s)$ be the maximum discounted reward obtainable from s.



The optimal value function determines the optimal policy.

Unknown environments - random actions

Environment Types

Environments can be:

- passive and stochastic
- active and deterministic (chess)
- active and stochastic (backgammon , robotics)

K-Armed Bandit Problem

The special case of an active, stochastic environment with only one state is called the K-armed Bandit Problem, because it is like being in a room with several (friendly) slot machines, for a limited time, and trying to collect as much money as possible.

Each action (slot machine) provides a different average reward.

Exploration / Exploitation Tradeoff

Most of the time we should choose what we think is the "best" action.

However, in order to ensure convergence to the optimal strategy, we must occasionally choose something different from our preferred action, e.g.

- choose a random action 5% of the time, or
- use a Bolzmann distribution to choose the next action:

$$P(a) = \frac{e^{\hat{V}(a)/T}}{\sum_{b \in A} e^{\hat{V}(b)/T}}$$

Experiences

- We assume there is a sequence of experiences:

  state, action, reward, state, action, reward, ...

- The agent has to choose its action as a function of its history.

- At any time it must decide whether to
  - explore to gain more knowledge
  - exploit knowledge it has already discovered

Exploration / Exploitation Tradeoff

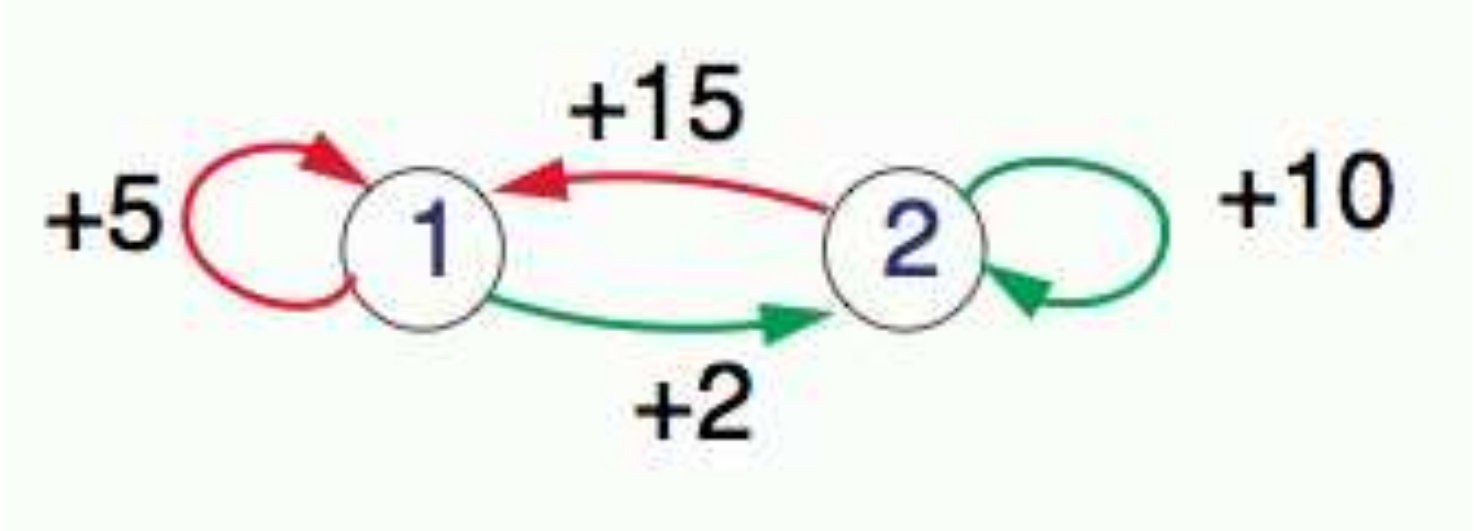
How do you choose an action?

- Random
- Pick the current “best” action
- Combination:
  - most of the time pick the best action
  - occasionally throw in random action

Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
- The long-term effect of an action depend on what the agent will do in the future.
- The explore-exploit dilemma: at each time should the agent be greedy or inquisitive?

Example: Delayed Reinforcement - Rewards



Short term and long term rewards

Calculation

Theorem: In a deterministic environment, for an optimal policy, the value function V^* satisfies the Bellman equations:

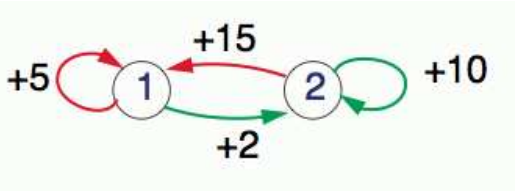
$V^*(s) = r(s, a)+\gamma V^*\delta(s, a)$ where $a = \pi^*(s)$ is the optimal action at state s.

Let $\delta^*(s)$ be the transition function for $\pi^*(s)$ and suppose $\gamma = 0.9$.

1. Suppose $\delta^*(s_1) = s_1$. Then $V^*(s_1) = 5 + 0.9V^*(s_1)$ so $V^*(s_1) = 50$
   Suppose $\delta^*(s_2) = s_2$. Then $V^*(s_2) = 10 + 0.9V^*(s_2)$ so $V^*(s_2) = 100$
2. Suppose $\delta^*(s_1) = s_2$. Then $V^*(s_1) = 2 + 0.9V^*(s_2)$ so $V^*(s_1) = 92$
   Suppose $\delta^*(s_2) = s_2$. Then $V^*(s_2) = 10 + 0.9V^*(s_2)$ so $V^*(s_2) = 100$
3. Suppose $\delta^*(s_1) = s_2$. Then $V^*(s_1) = 2 + 0.9V^*(s_2)$ so $V^*(s_1) = 81.6$
   Suppose $\delta^*(s_2) = s_1$. Then $V^*(s_2) = 15 + 0.9V^*(s_1)$ so $V^*(s_2) = 88.4$

So 2 is the optimal policy.

Calculation



Theorem: In a deterministic environment, for an optimal policy, the value function V^* satisfies the Bellman equations:

$V^*(s) = r(s, a)+\gamma V^*\delta(s, a)$ where $a = \pi^*(s)$ is the optimal action at state s.

Let $\delta^*(s)$ be the transition function for $\pi^*(s)$ and suppose $\gamma = 0.9$.

1. Suppose $\delta^*(s_1) = s_1$. Then $V^*(s_1) = 5 + 0.9V^*(s_1)$ so $V^*(s_1) = 50$
   Suppose $\delta^*(s_2) = s_2$. Then $V^*(s_2) = 10 + 0.9V^*(s_2)$ so $V^*(s_2) = 100$
2. Suppose $\delta^*(s_1) = s_2$. Then $V^*(s_1) = 2 + 0.9V^*(s_2)$ so $V^*(s_1) = 92$
   Suppose $\delta^*(s_2) = s_2$. Then $V^*(s_2) = 10 + 0.9V^*(s_2)$ so $V^*(s_2) = 100$
3. Suppose $\delta^*(s_1) = s_2$. Then $V^*(s_1) = 2 + 0.9V^*(s_2)$ so $V^*(s_1) = 81.6$
   Suppose $\delta^*(s_2) = s_1$. Then $V^*(s_2) = 15 + 0.9V^*(s_1)$ so $V^*(s_2) = 88.4$

So 2 is the optimal policy.

Expected Reward

- Try to maximise expected future reward:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- V is the value of state S under policy π - $V^{\pi}(s)$
- γ is a discount factor (0..1)

Q Function

- How to choose an action in a state?

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

The Q value for an action, a, in a state, s, is the immediate reward for the action plus the discounted value of following the optimal policy after that

- V^* is value obtained by following the optimal policy
- $\delta(s,a)$ is the succeeding state, assuming the optimal policy

Q-learning

- Idea: store Q[State, Action]; update this as in asynchronous value iteration, but using experience (empirical probabilities and rewards).
- Suppose the agent has an experience ⟨s, a, r, s'⟩ This provides one piece of data to update Q[s,a].
- An experience ⟨s, a, r, s'⟩ provides a new estimate for the value of $Q^*(s, a)$:

$$r + \gamma \max_{a'} Q[s', a']$$

Q Learning

initialise $Q(s,a) = 0$ for all s and a

observe current state s

repeat

select an action a and execute it

observe immediate reward r and next state s'

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

$$s \leftarrow s'$$

Theoretical Results

Theorem: Q-learning will eventually converge to the optimal policy, for any deterministic Markov decision process, assuming an appropriately randomized strategy. (Watkins & Dayan 1992)

Limitations of Theoretical Results

- Delayed reinforcement
  - reward resulting from an action may not be received until several time steps later, which also slows down the learning
- Search space must be finite
  - convergence is slow if the search space is large
  - relies on visiting every state infinitely often
- For “real world” problems, we can’t rely on a lookup table
  - need to have some kind of generalisation – Deep Learning

Reinforcement Learning Variants

- There are *many* variations on reinforcement learning to improve search.
- RL was one of the components of alphaGo, which recently beat a Go master

Temporal Difference Learning

TD(0) [also called AHC, or Widrow-Hoff Rule]

$$\hat{V}(s) \leftarrow \hat{V}(s) + \eta\,[\,r(s,a) + \gamma \hat{V}(\delta(s,a)) - \hat{V}(s)\,]$$

$$(\eta = \text{learning rate})$$

The (discounted) value of the next state, plus the immediate reward, is used as the target value for the current state.

A more sophisticated version, called TD(λ), uses a weighted average of future states.

Q-Learning

For each $s \in S$, let $V^*(s)$ be the maximum discounted reward obtainable from s, and let $Q(s, a)$ be the discounted reward available by first doing action a and then acting optimally.

Then the optimal policy is

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

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

then $$V^*(s) = \max_a Q(s,a),$$

so $$Q(s,a) = r(s,a) + \gamma \max_b Q(\delta(s,a),b)$$

which allows us to iteratively approximate Q by

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_b \hat{Q}(\delta(s,a),b)$$

Summary

- Reinforcement Learning is an active area of research
- Mathematical results (more than in other areas of AI)
- Need to have an appropriate representation
- Future algorithms which choose their own representations?
- Many practical applications