COMP3411/9414: Artificial Intelligence

Module 3

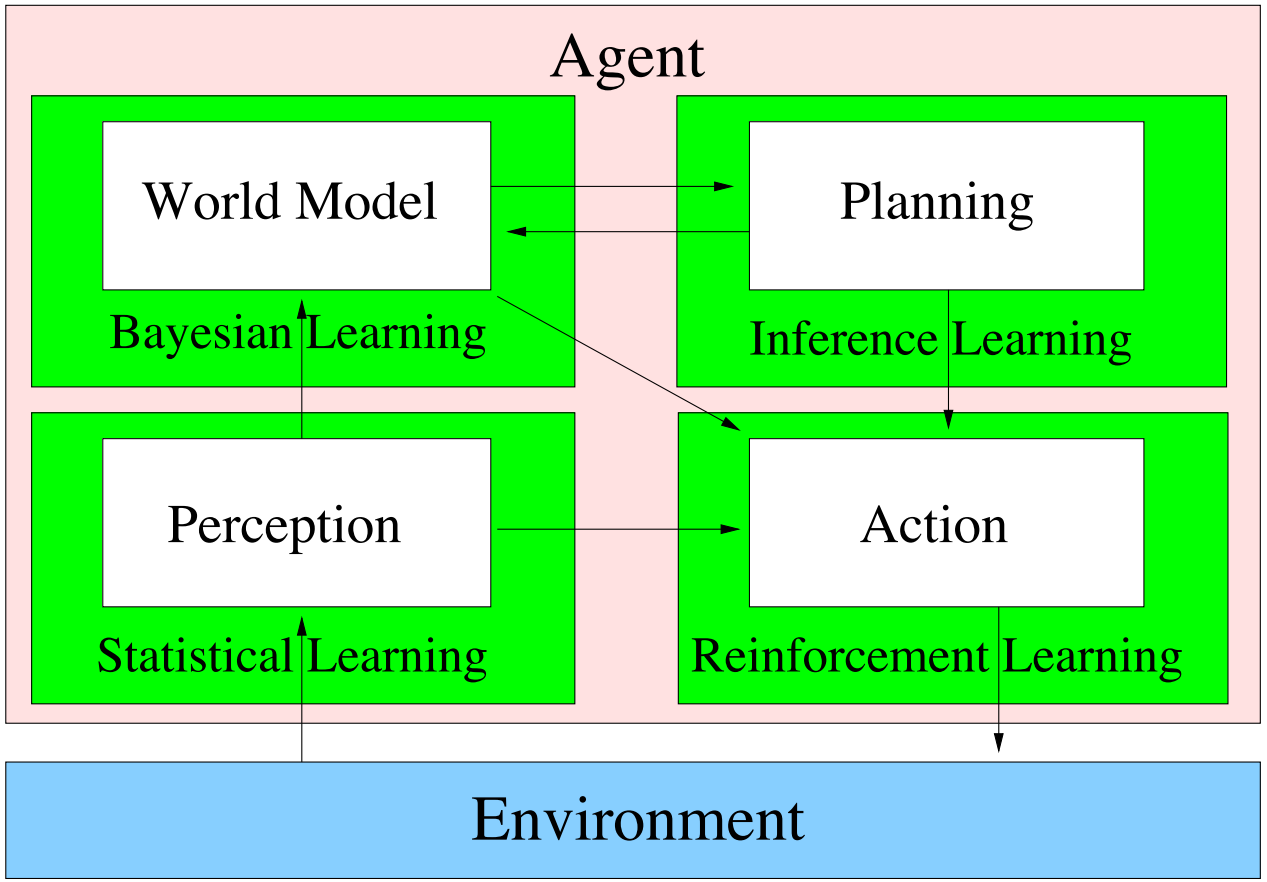
Reinforcement Learning

Russell & Norvig, Chapter 21

Outline

- Reinforcement Learning vs. Supervised Learning
- Models of Optimality
- Exploration vs. Exploitation
- Temporal Difference learning
- Q-Learning

Learning Agent

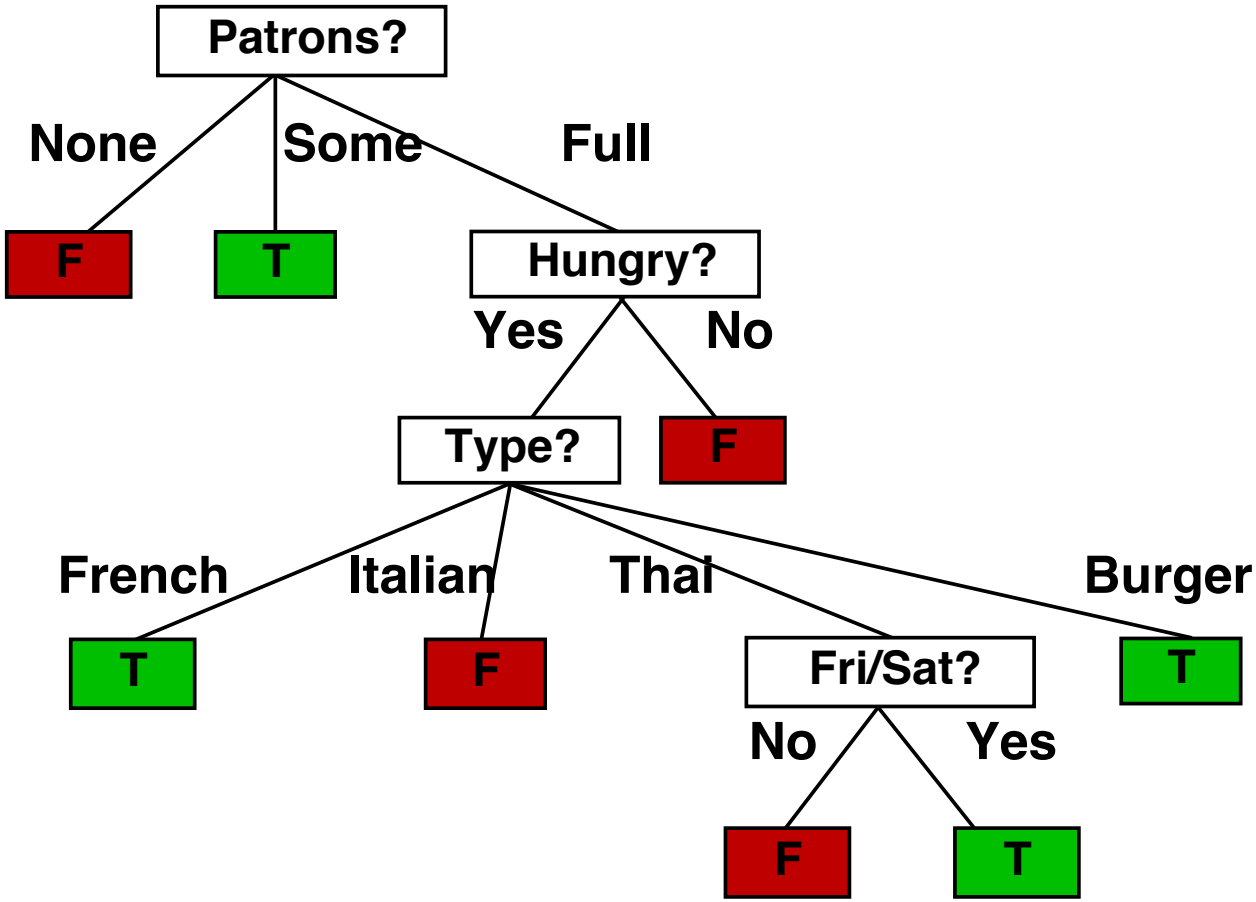


Supervised Learning

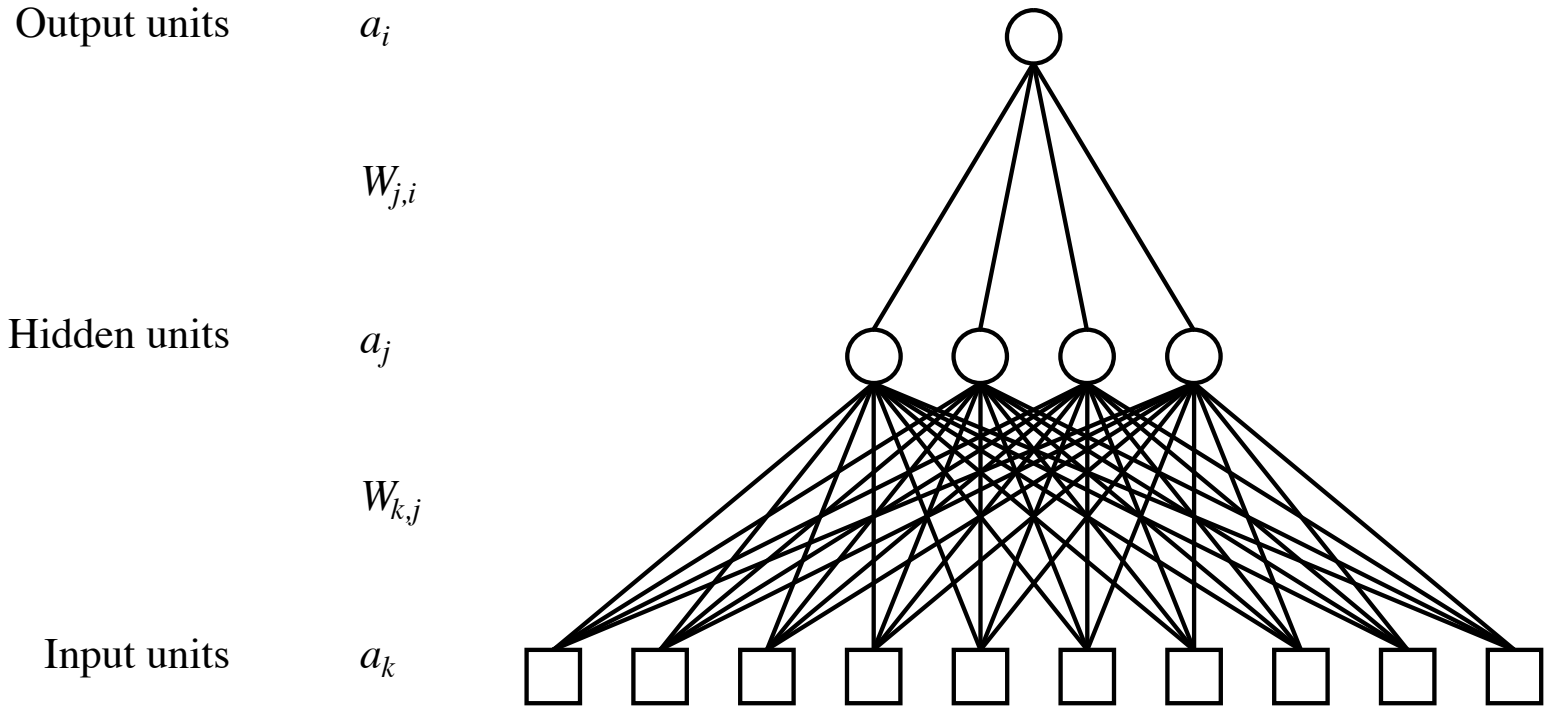
Recall: Supervised Learning

- We have a training set and a test set, each consisting of a set of examples. For each example, a number of input attributes and a target attribute are specified.
- The aim is to predict the target attribute, based on the input attributes.
- Various learning paradigms are available:
  - Decision Trees
  - Neural Networks
  - .. others ..

Decision Tree



Neural Network



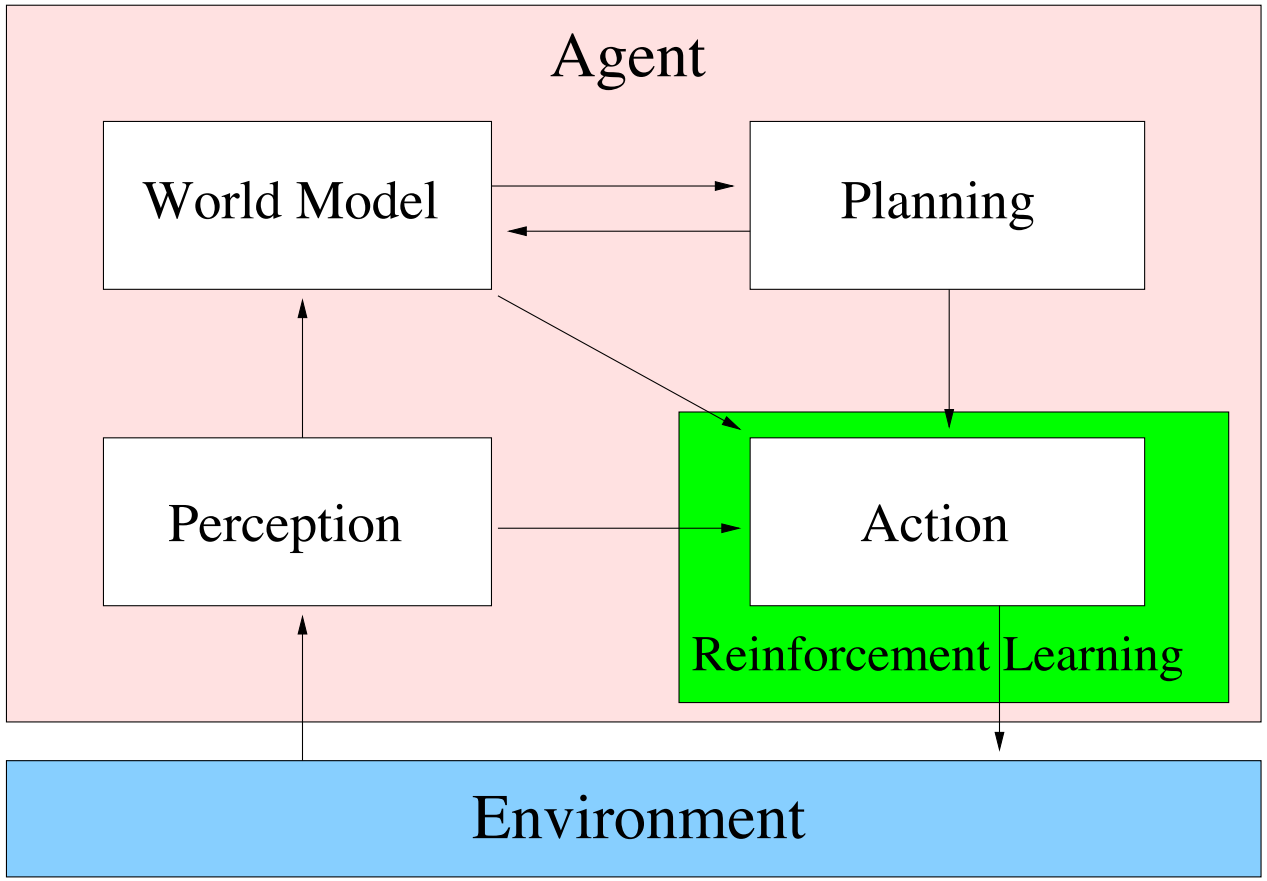
Learning of Actions

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

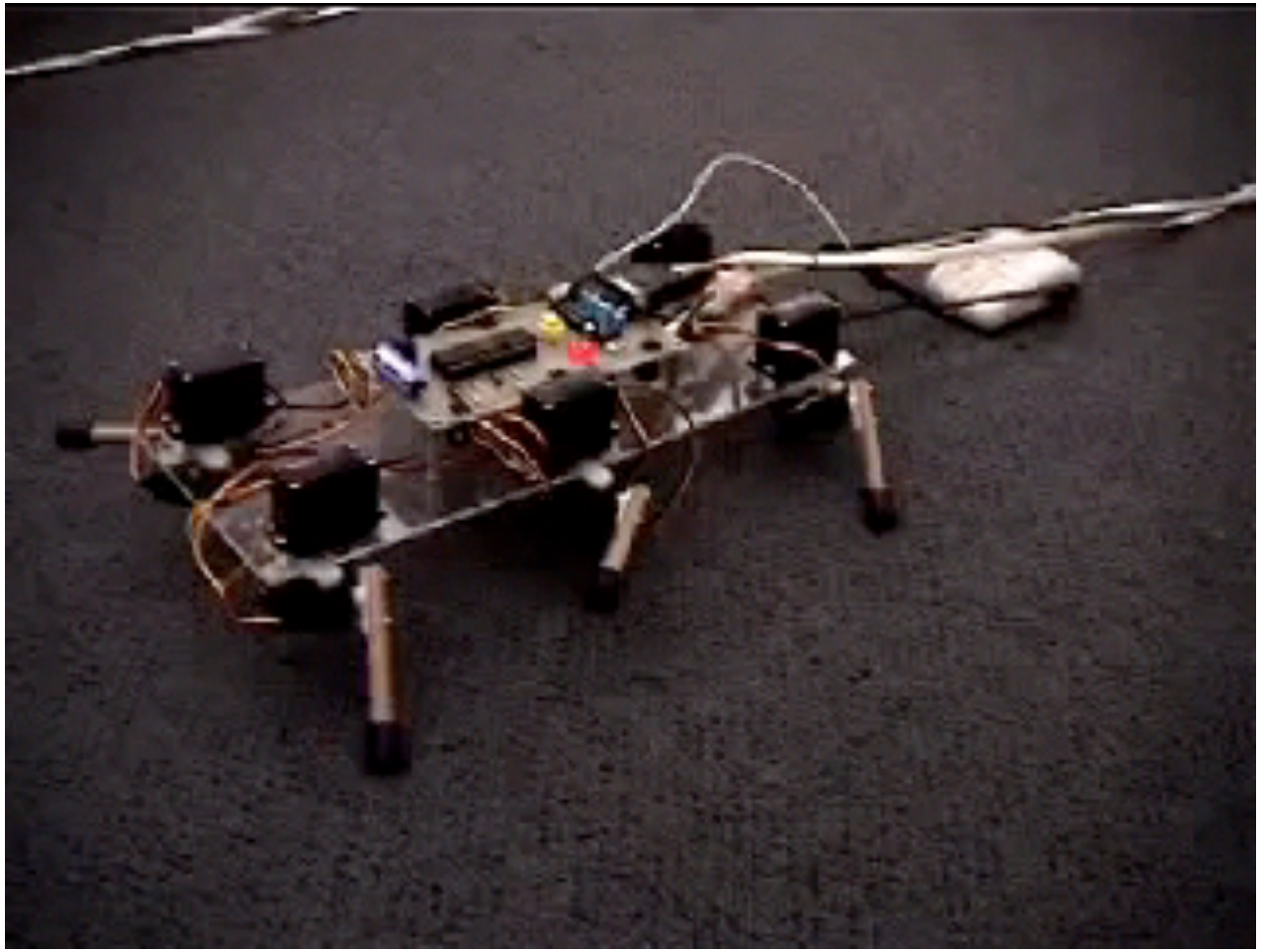
However, there are many applications for which it is difficult, inappropriate, or even impossible to provide a "training set"

- 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

Reinforcement Learning Agent



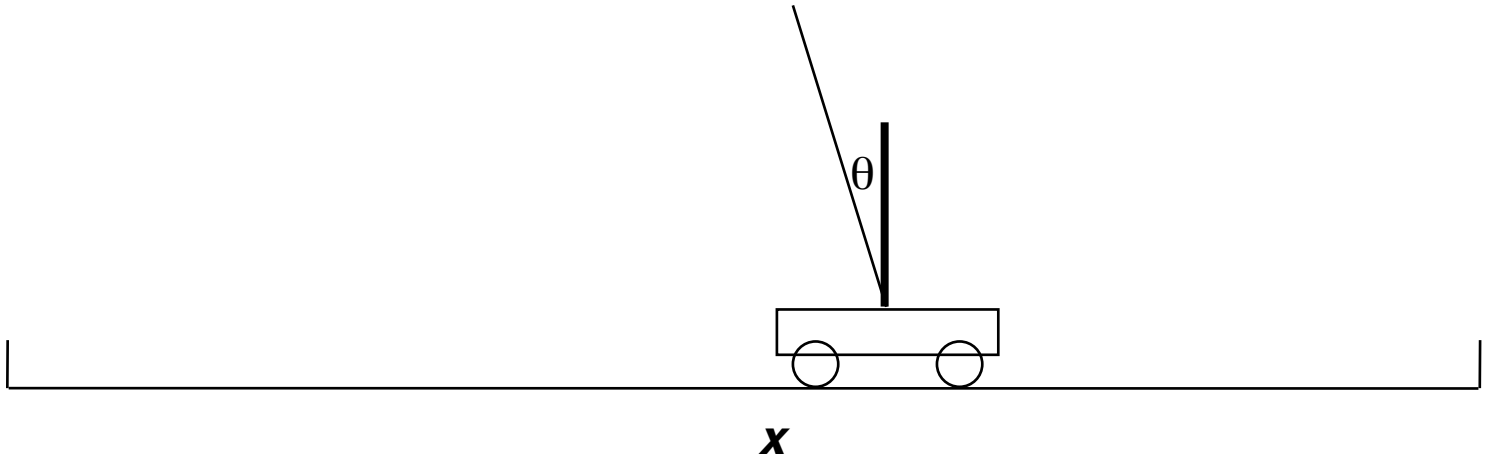
A Simple Learning Robot



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

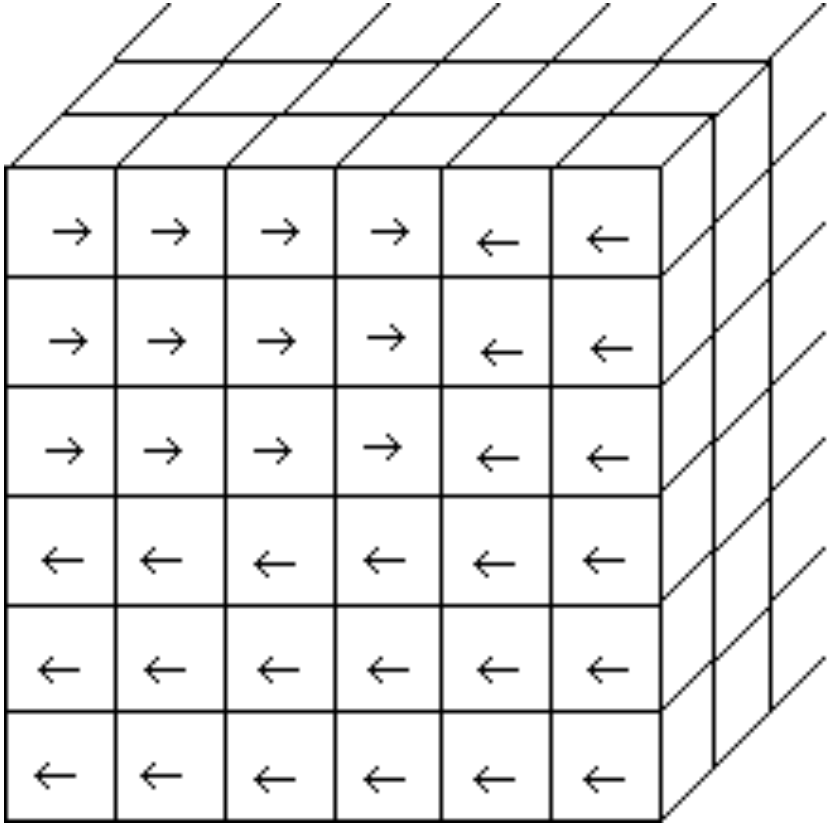
Boxes

State space is discretised

Each “box” represents a subset of state space

When system lands in a box, execute action specified

- left push
- right push



Simulation

$$
\begin{aligned}
x_{t+1} &= x_t + \tau \dot{x}_t \\
\dot{x}_{t+1} &= \dot{x}_t + \tau \ddot{x}_t \\
\theta_{t+1} &= \theta_t + \tau \dot{\theta}_t \\
\dot{\theta}_{t+1} &= \dot{\theta}_t + \tau \ddot{\theta}_t
\end{aligned}
$$

$$
\ddot{x}_t = \frac{F_t + m_p\, l\, [\dot{\theta}^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t]}{m_c + m_p}
$$

$$
\ddot{\theta}_t = \frac{g \sin\theta_t + \cos\theta_t \left[\dfrac{-F_t - m_p\, l\, \dot{\theta}_t^2 \sin\theta_t}{m_c + m_p} \right]}{l \left[\dfrac{4}{3} - \dfrac{m_p \cos^2\theta_t}{m_c + m_p} \right]}
$$

Parameters

$m_c = 1.0$ kg	*mass of cart*
$m_p = 1.0$ kg	*mass of pole*
$l = 0.5$ m	*distance of centre of mass of pole from the pivot*
$g = 9.8$ ms^{-2}	*acceleration due to gravity*
$F_t = \pm 10$ N	*force applied to cart*
$t = 0.02$ s	*time interval of simulation*

Update Rule

if an action has not been tested

choose that action

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

choose left

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

else

choose right

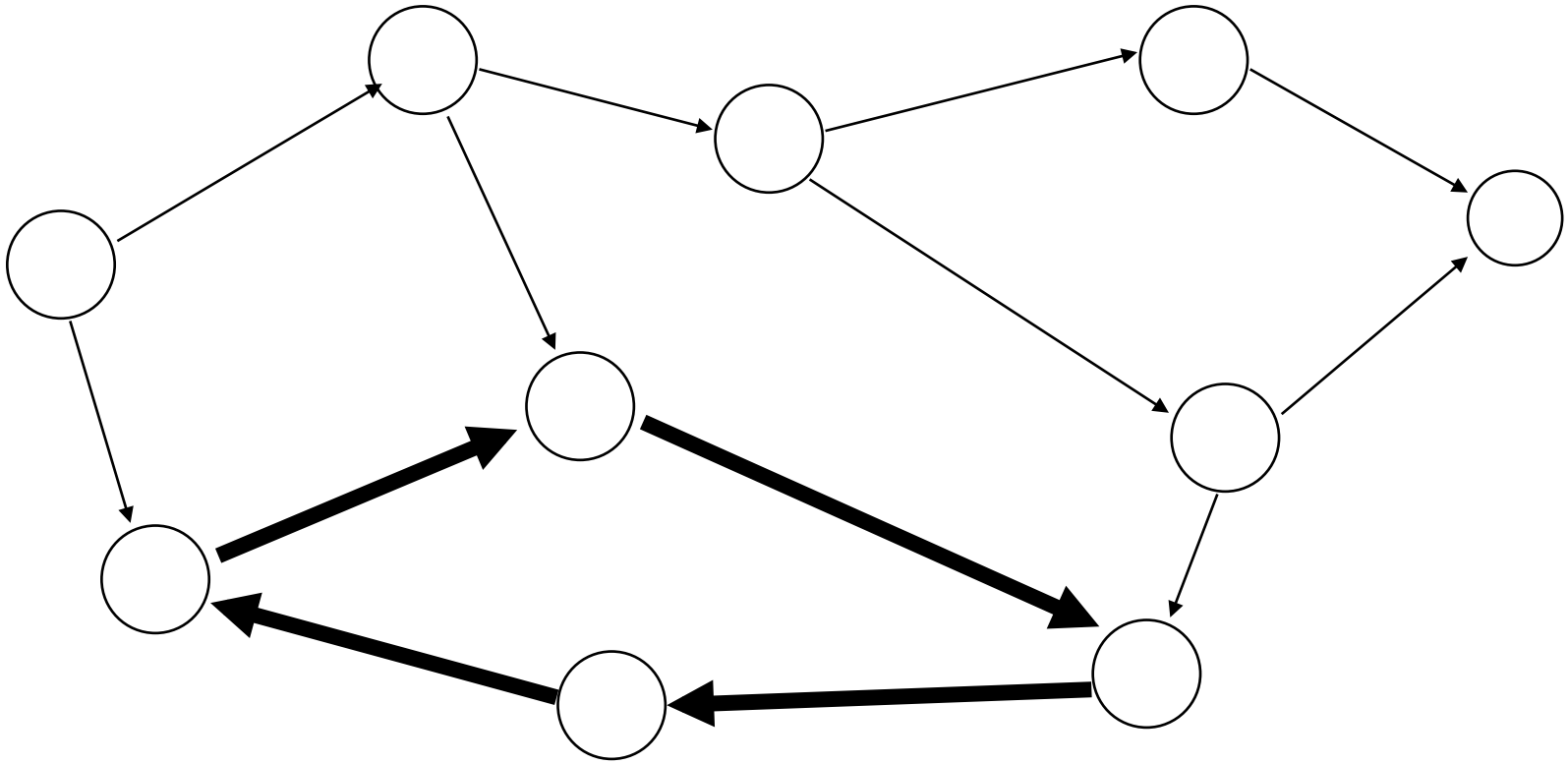
Performance

- BOXES is *much* faster than genetic algorithm
  - 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

The BOXES Algorithm

- Each box contains statistics on performance of controller, which are updated after each failure
  - How many times each action has been performed (usage)
  - The sum of lengths of time the system has survived after taking a particular action (LifeTime)
- Each sum is weighted by a number less than one which places a discount on earlier experience.

State Transition Graph



States and Actions

- Each node is a *state*
- *Actions* cause transitions from one state to another
- A *policy* is the set of transition rules
  - i.e. which action to apply in a given state
- Agent receives a *reward* after each action
- Actions may be non-deterministic
  - Same action may not always produce same state

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 $r(s_t , a_t)$.

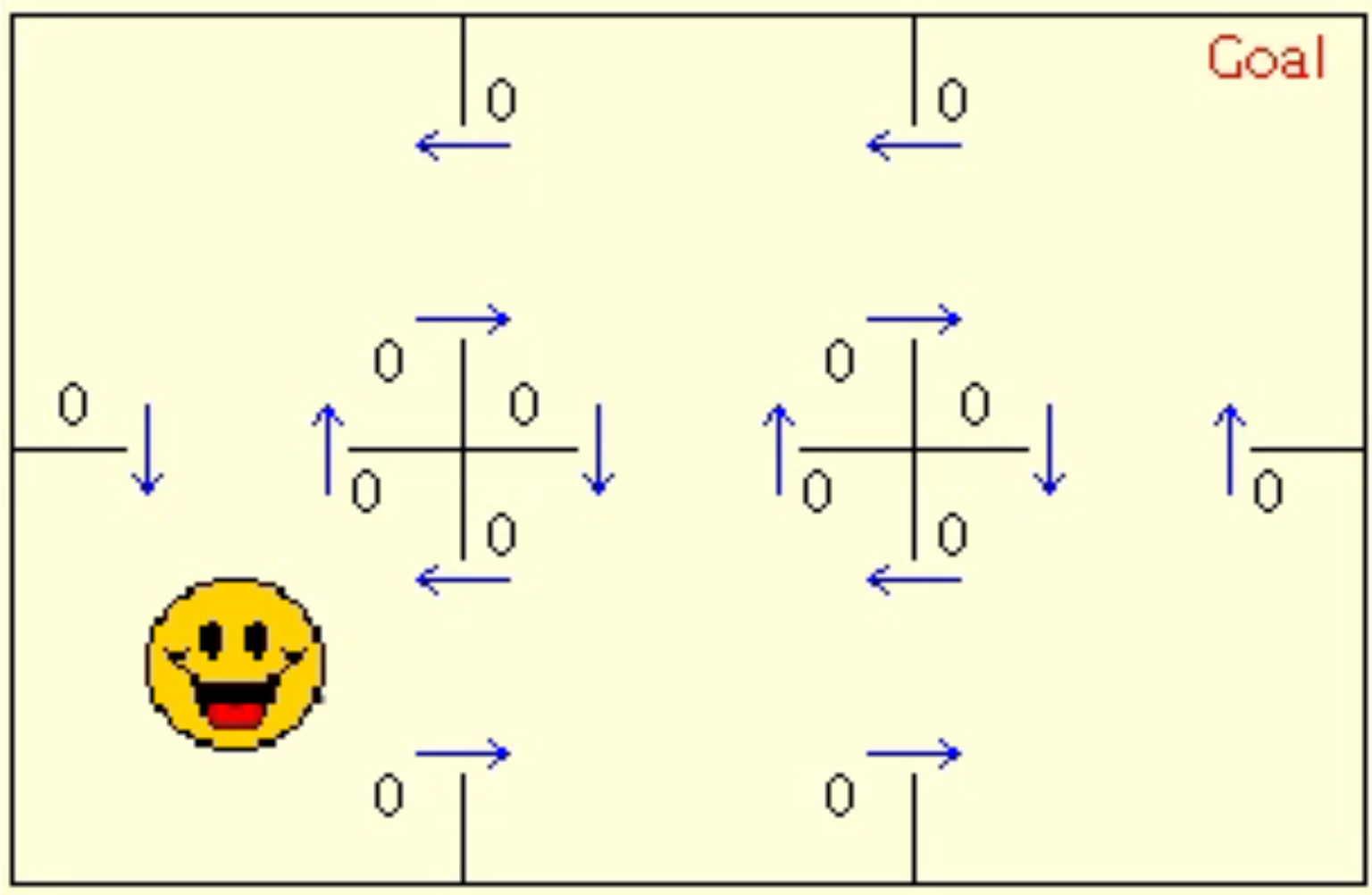
In general, $r()$ and $\delta()$ can be multi-valued, with a random element

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

Markov Decision Process (MDP)

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

Grid World – Q learning



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
- 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

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'

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

$$s \leftarrow s'$$

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

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:

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

Exploration / Exploitation Tradeoff

I was born to try...

But you've got to make choices
Be wrong or right
Sometimes you've got to sacrifice the things you like.

Delta Goodrem

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)

Theorem: TD-learning will also converge, with probability 1.
(Sutton 1988, Dayan 1992, Dayan & Sejnowski 1994)

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 (e.g. TD-Gammon)

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
- Used to learn helicopter aerobatics

Environment Types

Environments can be:

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