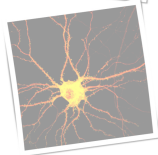
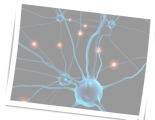
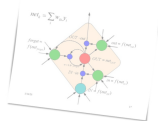
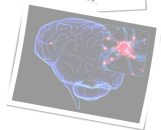


A (gentle) introduction to Reinforcement Learning

Spyros Samothrakis

August 28, 2014

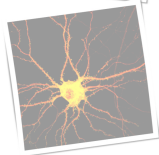
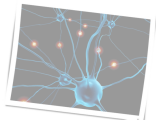
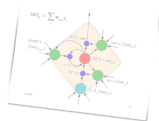
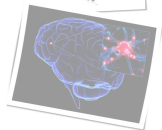


Introduction & Motivation

Markov Decision Process (MDPs)

Model Based Reinforcement Learning

Model Free Reinforcement Learning



What is Reinforcement Learning?

- ▶ *Reinforcement learning is the study of how animals and artificial systems can learn to optimize their behavior in the face of rewards and punishments* – Peter Dyan, Encyclopedia of Cognitive Science
- ▶ **Not** supervised learning - the animal/agent is not provided with examples of optimal behaviour, it has to be discovered!
- ▶ **Not** unsupervised learning either - we have more guidance than just observations

Links to other fields

- ▶ It subsumes most artificial intelligence problems
- ▶ Forms the basis of most modern intelligent agent frameworks
- ▶ Ideas drawn from a wide range of contexts, including psychology (e.g., Skinner's "Operant Conditioning"), philosophy, neuroscience, operations research

Examples of Reinforcement Learning closer to CS

- ▶ Play backgammon/chess/go/poker/any game (at human or superhuman level)
- ▶ Helicopter control
- ▶ Learn how to walk/crawl/swim/cycle
- ▶ Elevator scheduling
- ▶ Optimising a petroleum refinery
- ▶ Optimal drug dosage

The Markov Decision Process

- ▶ The primary abstraction we are going to work with is the Markov Decision Process (MDP).
- ▶ MDPs capture the dynamics of a mini-world/universe/environment
- ▶ An MDP is defined as a tuple $\langle S, A, T, R, \gamma \rangle$ where:
 - ▶ $S, s \in S$ is a set of states
 - ▶ $A, a \in A$ is a set of actions
 - ▶ $R : S \times A, R(s, a)$ is a function that maps state-actions to rewards
 - ▶ $T : S \times S \times A$, with $T(s'|s, a)$ being the probability of an agent landing from state s to state s' after taking a
 - ▶ γ is a discount factor - the impact of time on rewards

The Markov Property and States

- ▶ States represent sufficient statistics.
- ▶ Markov Property ensures that we only care about the present in order to act - we can safely ignore past states
- ▶ Think Tetris - all information are can be captured by a single screen-shot

First DOS Version



Original Tetris



Agents, Actions and Transitions

- ▶ An agent is an entity capable of actions
- ▶ An MDP can capture any environment that is inhabited either by
 - ▶ Exactly one agent
 - ▶ Multiple agents, but only one is adaptive
- ▶ Notice how actions are part of the MDP - notice also how the MDP is a “world model”
- ▶ The agent is just a “brain in a vat”
- ▶ The agent perceives states/rewards and outputs actions
- ▶ Transitions specify the effects of actions in the world (e.g., in Tetris, you push a button, the block spins)

Rewards and the Discount Factor

- ▶ Rewards describe state preferences
- ▶ Agent is happier in some states of the MDP (e.g., in Tetris when the block level is low, a fish in water, pacman with a high score)
- ▶ Punishment is just low/negative reward (e.g., being eaten in pacman)
- ▶ γ , the discount factor,
 - ▶ Describes the impact of time on rewards
 - ▶ “I want it now”, the lower γ is the less important future rewards are
- ▶ There are no “springs/wells of rewards” in the real world
 - ▶ What is “human nature”?

Examples of Reward Schemes

- ▶ Scoring in most video games
- ▶ The distance a robot walked for a bipedal robot
- ▶ The amount of food an animal eats
- ▶ Money in modern societies
- ▶ Army Medals (“Gamification”)
- ▶ Vehicle routing
 - ▶ (-Fuel spend on a flight)
 - ▶ (+ Distance Covered)
- ▶ Cold/Hot

Long Term Thinking

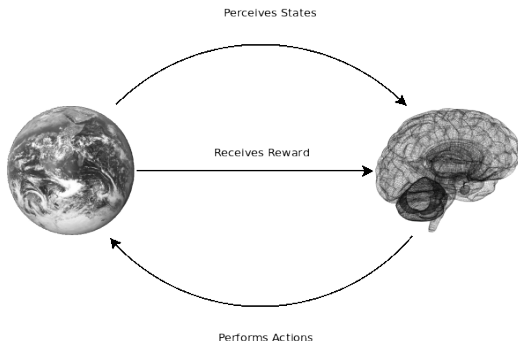
- ▶ It might be better to delay satisfaction
- ▶ Immediate reward is not always the maximum reward
- ▶ In some settings there are no immediate rewards at all (e.g., most solitaire games)
- ▶ MDPs and RL capture this
- ▶ “Not going out tonight, study”
- ▶ Long term investment

Policy

- ▶ The MDP (the world) is populated by an agent (an actor)
- ▶ You can take actions (e.g., move around, move blocks)
- ▶ The type of actions you take under a state is called the *policy*
- ▶ $\pi : S \times A, \pi(s, a) = P(a|s)$, a probabilistic mapping between states and actions
- ▶ Finding an optimal policy is *mostly* what the RL problem is all about

The Full Loop

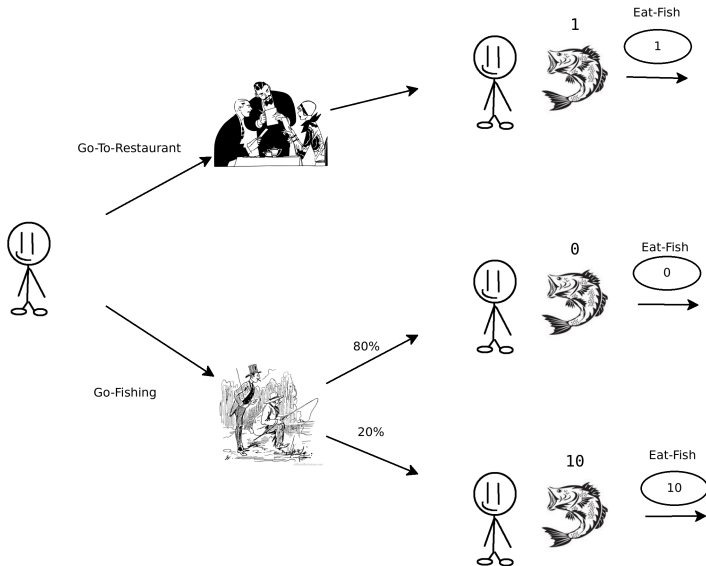
- ▶ See how the universe described by the MDP defines actions, not just states and transitions
- ▶ An agent needs to act upon what it perceives
- ▶ Notice the lack of body - “brain in a vat”. Body is assumed to be part of the world.



Fishing Toon

- ▶ Assume a non-player character (let's call her *toon*)
- ▶ Toon is Hungry!
- ▶ Eating food is rewarding
- ▶ Has to choose between going fishing or going to the restaurant (to eat fish)
 - ▶ Fishing can get you better quality of fish (more reward), but you might also get no fish at all (no reward)!
 - ▶ Going to the restaurant is a low-risk, low-reward alternative

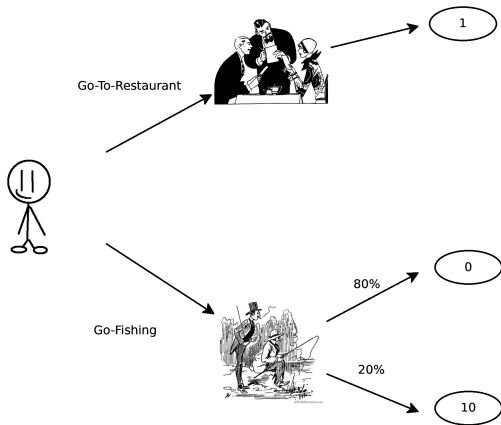
Fishing Toon: Pictorial Depiction



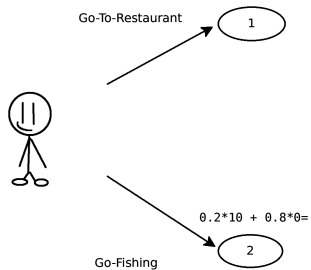
Expected Reward

- ▶ Our toon has to choose between two different actions
- ▶ Go-To-Restaurant or Go-Fishing
- ▶ We assume that toon is interested in maximising *the expected sum* of happiness/reward
- ▶ We can help the toon reason using the tree backwards

Reasoning Backwards (1)



Reasoning Backwards (2)



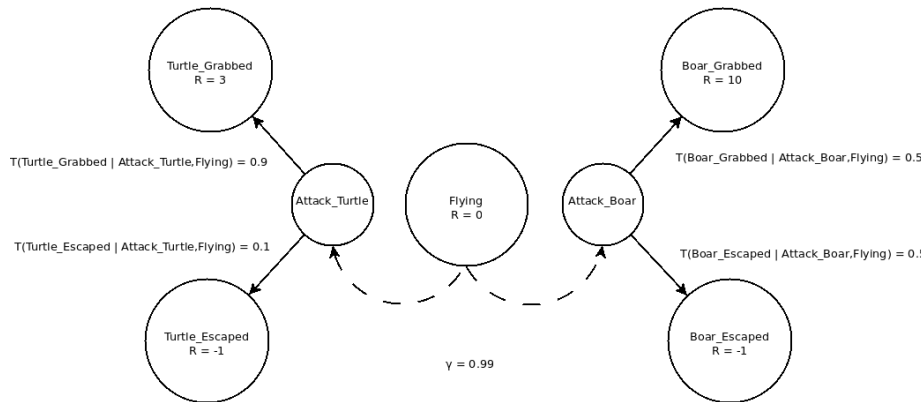
Correct Action

- ▶ Toon should go Go-Fishing
- ▶ **Would you do the same?**
- ▶ **Would a pessimist toon do the same?**
- ▶ We just went through the following equation:

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

- ▶ Looks intimidating - but it's really simple
- ▶ Let's have a look at another example
 - ▶ How about toon goes to the restaurant after failing to fish?
 - ▶ How would that change the reward structure?

Example MDP - EagleWorld



Agent Goals

- ▶ The agent's goal is to maximise its long term reward
$$\mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s^t, a^t) \right]$$
- ▶ Risk Neutral Agent - think of the EagleWorld example
- ▶ Rewards can be anything, but most organisms receive rewards only in a very limited amount of states (e.g., fish in water)
- ▶ What if your reward signal is only money?
 - ▶ Sociopathic, egotistic, greed-is-good Gordon Gekko (*Wall Street*, 1987)
 - ▶ No concept of “externalities” - agents might wreak havoc for marginal reward gains
 - ▶ Same applies to all “compulsive agents” - think Chess

Searching for a good Policy

- ▶ One can possibly search through all combinations of policies until she finds the best
- ▶ Slow, does not work in larger MDPs
- ▶ Exploration/Exploitation dilemma
 - ▶ How much time/effort should be spend exploring for solutions?
 - ▶ How much time should be spend exploiting good solutions?

Model Based Reinforcement Learning

- ▶ ... also known as planning in certain contexts
- ▶ Who was doing the thinking in the previous example (You? The eagle?)
- ▶ An agent has access to model, i.e., has a copy of the MDP (the outside world) in its mind
- ▶ Using that copy, it tries to “think” what is the best route of action
- ▶ It then executes this policy on the real world MDP
- ▶ You can't really copy the world inside your head, but you can copy the dynamics
- ▶ “This and that will happen if I push the chair”
- ▶ Thinking, introspection...

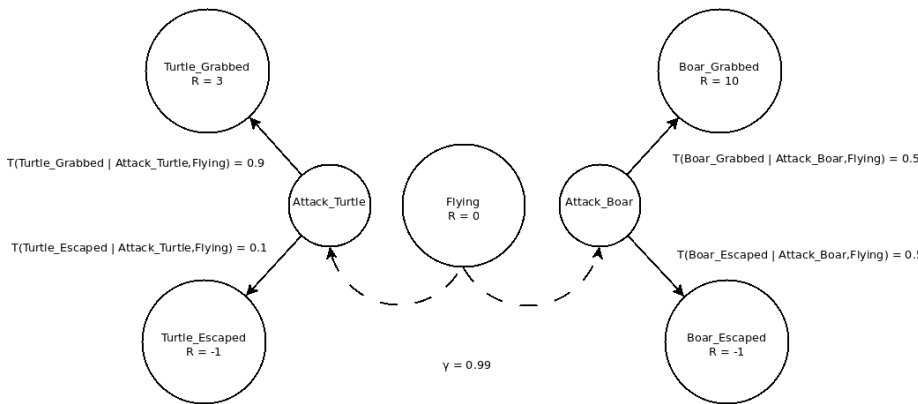
Bellman Expectation Equations / Bellman Backups

- ▶ The two most important equations related to MDP
- ▶ Recursive definitions
- ▶
$$V^\pi(s) = \sum_{a \in A} \pi(s, a) \left(R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V^\pi(s') \right)$$
- ▶
$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) \sum_{a' \in A} \pi(s', a') Q^\pi(s', a')$$
- ▶ **Called V-Value(s) (*state-value function*) and Q-Value(s) (*state-action value function*) respectively**
- ▶ Both calculate the expected rewards under a certain policy

Link between V^π and Q^π

- ▶ V and Q are interrelated
- ▶ $V^\pi(s) = \sum_{a \in A} \pi(s, a) Q^\pi(s, a)$
- ▶ $Q^\pi(s, a) = R(s, a) + \sum_{s' \in S} T(s'|s, a) V^\pi(s')$

Example MDP - EagleWorld - Random Policy



$$\begin{aligned}\pi(\text{Flying}, \text{Attack_Boar}) &= 0.5, \pi(\text{Flying}, \text{Attack_Turtle}) = 0.5 \\ Q(\text{Flying}, \text{Attack_Boar}) &= 0.99 * (10 * 0.5 + 0.5 * -1) = 4.455 \\ Q(\text{Flying}, \text{Attack_Turtle}) &= 0.99 * (0.9 * 3 + 0.1 * -1) = 2.574 \\ V^\pi(\text{Flying}) &= 0.5, Q^\pi(\text{Flying}, \text{Attack_Turtle}) + 0.5, Q(\text{Flying}, \text{Attack_Boar}) = 3.5145\end{aligned}$$

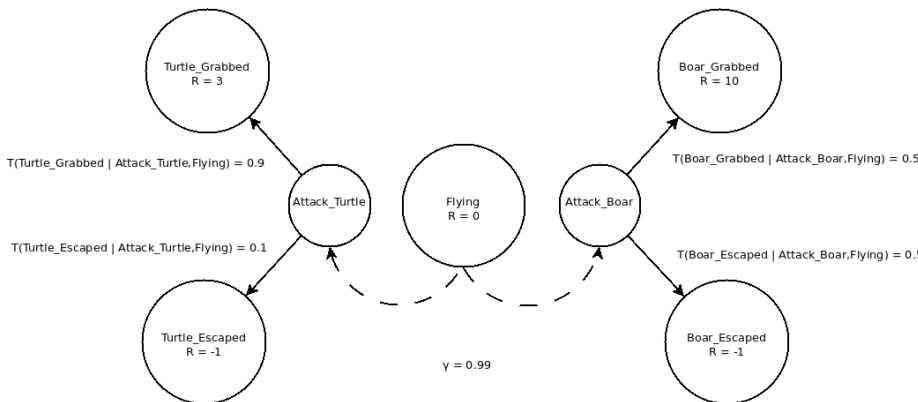
Optimal Policy and the Bellman Optimality Equation

- ▶ An optimal policy can be defined in terms of Q-values
- ▶ It is the policy that maximises Q values
- ▶ $V^*(s) = \max_{a \in A} R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V^*(s')$
- ▶ $Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) \max_{a' \in A} Q^*(s', a')$
- ▶ $\pi^*(s, a) = \begin{cases} 1 & \text{if } a = \arg \max_{a \in A} Q^*(s, a) \\ 0 & \text{otherwise} \end{cases}$

Link between V^* and Q^*

- ▶ Again, they are interrelated
- ▶ $V(s)^* = \max_{a \in A} Q^*(s, a)$
- ▶ $Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V^*(s')$

Example MDP - EagleWorld - Optimal Policy



$$Q(\text{Flying}, \text{Attack_Boar}) = 0.99 * (10 * 0.5 + 0.5 * -1) = 4.455$$

$$Q(\text{Flying}, \text{Attack_Turtle}) = 0.99 * (0.9 * 3 + 0.1 * -1) = 2.574$$

$$\pi^*(\text{Flying}, \text{Attack_Boar}) = 1, \pi^*(\text{Flying}, \text{Attack_Turtle}) = 0$$

$$V^*(\text{Flying}) = Q(\text{Flying}, \text{Attack_Boar}) = 4.455$$

Agents Revisited

- ▶ An Agent can be composed of a number of things
- ▶ A policy
- ▶ A Q-Value/and or V-Value Function
- ▶ A Model of the environment (the MDP)
- ▶ Inference/Learning Mechanisms
- ▶ ...
- ▶ An agent has to be able to *create a policy* either on the fly or using Q-Values
- ▶ The Model/Q/V-Values serve as intermediate points towards constructing a policy

Simplifying assumptions

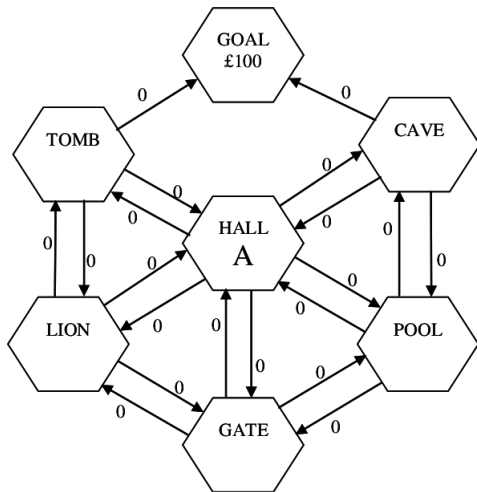
- ▶ Assume deterministic transitions
- ▶ Thus, taking an action on a state will lead only to ONE other possible state for some action a_c
 - ▶ $T(s'|s, a_i) = \begin{cases} 1 & \text{if } a_i = a_c \\ 0 & \text{otherwise} \end{cases}$
 - ▶ $V^*(s) = \max_{a \in A} [R(s, a) + \gamma V^*(s')]$
 - ▶ $Q(s, a) = R(s, a) + \gamma \max_{a' \in A} Q(s', a')$
- ▶ It is easier now to solve for problems that have loops in them
- ▶ We can also attempt to learn Q-Values without a model!
- ▶ All we need in order to find the optimal policy is $Q(s, a)$

Deterministic Q-Learning (1)

- ▶ The policy is deterministic from start to finish
- ▶ We will use $\pi(s) = \arg \max_{a \in A} Q(s, a)$ to denote the optimal policy
- ▶ The algorithm now is:
 - ▶ Initialise all $Q(s, a)$ to low values
 - ▶ Repeat:
 - ▶ Select an action a using an exploration policy
 - ▶ $Q(s, a) \leftarrow R(s, a) + \gamma \max_{a' \in A} Q(s', a')$
 - ▶ $s \leftarrow s'$
 - ▶ Also known as “Dynamic Programming”, “Value Iteration”

An Example (1)

(From Paul Scott's ML lecture notes)



$$R(\text{HALL}, \text{To} - \text{CAVE}) = 0$$

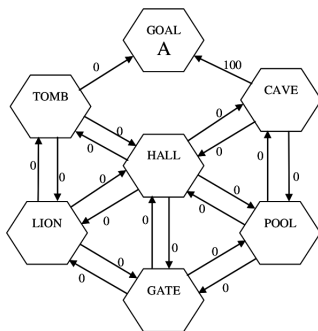
$$Q(\text{CAVE}, a) = 0 \text{ for all actions } a$$

An Example (2)

Next suppose the agent, now in state *CAVE* , selects action *To – GOAL*

$R(CAVE, To - GOAL) = 100$, $Q(GOAL, a) = 0$ for all actions (there are no actions)

Hence $Q(CAVE, To - GOAL) = 100 + \gamma * 0 = 100$



An Example (3)

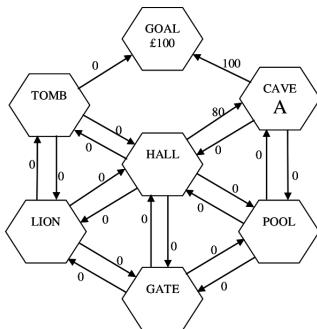
Let's start at hall again and select the same action To-CAVE

$$R(\text{HALL}, \text{To} - \text{CAVE}) = 0, Q(\text{CAVE}, \text{GOAL}) = 100$$

$$Q(\text{CAVE}, a) = 0 \text{ for all other actions } a$$

$$\text{Hence } \max_{a \in A} Q(\text{CAVE}, a) = 100, \text{ if } \gamma = 0.8,$$

$$Q(\text{HALL}, \text{To} - \text{CAVE}) = 0 + \gamma * 100 = 80$$



Exploration / Exploitation

- ▶ How do we best explore?
- ▶ Choose actions at random - but this can be very slow
- ▶ ϵ - greedy is the most common method
- ▶ Act ϵ -greedily
 - ▶ $\pi^\epsilon(s, a) = \begin{cases} 1 - \epsilon + \epsilon/|A| & \text{if } a = \arg \max_{a \in A} Q(s, a) \\ \epsilon/|A| & \text{otherwise} \end{cases}$
 - ▶ ϵ -greedy means acting greedily with probability $1 - \epsilon$, random otherwise
- ▶ When you are done, act greedily $\pi(s) = \arg \max_{a \in A} Q(s, a)$

Algorithms for non-deterministic settings

- ▶ What can we do if the MDP is not deterministic?
- ▶ If we know the model, full blown value iteration
- ▶ Otherwise

- ▶ Q-learning,

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[R(s, a) + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right]$$

- ▶ SARSA(0), $Q(s, a) \leftarrow Q(s, a) + \eta [R(s, a) + \gamma Q(s', a') - Q(s, a)]$
 - ▶ SARSA(1)/MC, $Q(s, a) \leftarrow Q(s, a) + \eta [v_\tau - Q(s, a)]$
 $v_\tau \leftarrow R(s, a) + \gamma R(s', a') + \dots \gamma^2 R(s'', a'') + \gamma^{\tau-1} R(s^\tau, a^\tau)$

- ▶ η is a small learning rate, e.g., $\eta = 0.001$

SARSA vs Q-Learning vs MC

- ▶ MC: updated using the whole chain
 - ▶ Possibly works better when the markov property is violated
- ▶ SARSA: update based on the next action you actually took
 - ▶ On Policy learning
- ▶ Q-Learning: update based on the best possible next action
 - ▶ Will learn optimal policy even if acting off-policy

Monte Carlo Control (1)

- ▶ Remember Q is just a mean/average
- ▶ MC (Naive Version)
 - ▶ Start at any state, initialise $Q_0(s, a)$ as you visit states/actions
 - ▶ Act ϵ -greedily
- ▶ Add all reward you have seen so far to
 $v_\tau^i = R(s') + \gamma R(s'') + \gamma^2 R(s''') + \gamma^{\tau-1} R(s^\tau)$ for episode i
- ▶ $Q_n(s, a) = E_{\pi^\epsilon}[v_\tau^i] = \frac{1}{n} \sum_{i=1}^n v_\tau^i$, where k is the times a state is visited

Monte Carlo Control (2)

- ▶ ϵ -greedy means acting greedily $1 - \epsilon$, random otherwise
- ▶ Better to calculate mean incrementally

$$Q_n(s, a) = E_{\pi_n}[v_\tau^i]$$

$$Q_n(s, a) = \frac{1}{n} \sum_{i=1}^n v_\tau^i$$

$$Q_n(s, a) = \frac{1}{n} \left(v_t^1 + v_\tau^2 \dots v_\tau^{n-1} + v_\tau^n \right)$$

$$Q_n(s, a) = \frac{1}{n} \left(\sum_{i=1}^{n-1} v_\tau^i + v_\tau^n \right)$$

Monte Carlo Control (3)

$$\text{by definition } Q_{n-1}(s, a) = \frac{1}{n-1} \sum_{i=1}^{n-1} v_{\tau}^i \implies (n-1)Q_{n-1}(s, a) = \sum_{i=1}^{n-1} v_{\tau}^i$$

$$Q_n(s, a) = \frac{1}{n} ((n-1)Q_{n-1}(s, a) + v_{\tau}^n)$$

$$Q_n(s, a) = \frac{1}{n} (Q_{n-1}(s, a)k - Q_{n-1}(s, a) + v_{\tau}^n)$$

$$Q_n(s, a) = \frac{Q_{n-1}(s, a)k}{n} + \frac{-Q_{n-1}(s, a) + v_{\tau}^n}{n}$$

$$Q_n(s, a) = Q_{n-1}(s, a) + \frac{\overbrace{v_{\tau}^n - Q_{n-1}(s, a)}^{\text{MC-Error}}}{n}$$

Monte Carlo Control - Putting it all together

- ▶ But π^n changes continuously, so the distribution of rewards is non-stationary

$$Q_n(s, a) = Q_{n-1}(s, a) + \frac{1}{n} [v_\tau^n - Q_{n-1}(s, a)] \rightarrow \text{Bandit case}$$

$$Q_n(s, a) = Q_{n-1}(s, a) + \eta [v_\tau^n - Q_{n-1}(s, a)] \rightarrow \text{Full MDP case}$$

- ▶ A Bandit is an MDP with a chain of length two (i.e. s, s') - like the initial EagleWorld, η is a learning rate (e.g., 0.001)
- ▶ MC
 - ▶ Start at any state, initialise $Q_0(s, a)$ as you visit states/actions
 - ▶ Act ϵ -greedily
 - ▶ Wait until episode ends, i.e. a terminal state is hit - ϵ set to some low value, e.g., 0.1
 - ▶ Add all reward you have seen so far to $v_\tau^i = R(s) + \gamma R(s') + \dots + \gamma^{T-1} R(s^T)$ for episode i
 - ▶ $Q_n(s, a) = Q_{n-1}(s, a) + \alpha [v_\tau^n - Q_{n-1}(s, a)]$

Function Approximation

- ▶ There is usually some link between states
- ▶ We can train function approximators incrementally to model $Q(s, a)$
- ▶ Examples include Linear Function approximators, Neural Networks, n-tuple networks
- ▶ Not easy to do, few convergence guarantees
 - ▶ But with some effort, this works pretty well

Relationship to the rest of Machine Learning

- ▶ How can one learn a model of the world?
 - ▶ Possibly by breaking it down into smaller, abstract chunks
 - ▶ Unsupervised Learning
 - ▶ ... and learning what effects ones actions have the environment
 - ▶ Supervised Learning
- ▶ RL weaves all fields of Machine Learning (and possibly Artificial Intelligence) into one coherent whole
- ▶ The purpose of all learning is action!
 - ▶ You need to be able to recognise faces so you can create state
 - ▶ ... and act on it

Conclusion

- ▶ RL is a massive topic
- ▶ We have shown the tip of iceberg
- ▶ Rabbit hole goes *deep* - both on the application level and the theory level

Further study (1)

- ▶ **Tom Mitchell, Chapter 13**

- ▶ David Silver's UCL Course:

<http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html>

- ▶ Some ideas in these lecture notes taken from there
- ▶ Probably the best set of notes there is on the subject
- ▶ Online at <http://www.machinelearningtalks.com/tag/rl-course/>

- ▶ Reinforcement Learning, by Richard S. Sutton and Andrew G. Barto

- ▶ Classic book
- ▶ Excellent treatment of most subjects
- ▶ Up to Chapter 5

Further Study (2)

- ▶ Artificial Intelligence: A Modern Approach by Stuart J. Russell and Peter Norvig
 - ▶ The Introductory A.I. Textbook
 - ▶ Chapters 16 and 21
- ▶ Algorithms for Reinforcement Learning by Csaba Szepesvari
 - ▶ Very “Mathematical”, but a good resource that provides a very unified view of the field
- ▶ Reinforcement Learning: State-Of-The-Art by Marco Wiering (Editor), Martijn Van Otterlo (Editor)
 - ▶ Edited Volume
 - ▶ Chapter 1