LEARNING TO ACT

Oliver Schulte

Simon Fraser University

OUTLINE

- What is Reinforcement Learning?
- Key Definitions
- Key Learning Tasks
- Reinforcement Learning Techniques
- Reinforcement Learning with Neural Nets

OVERVIEW

LEARNING TO ACT

- So far: learning to predict
- Now: learn to **act**
  - In engineering: control theory
  - Economics, operations research: decision and game theory

EXAMPLES

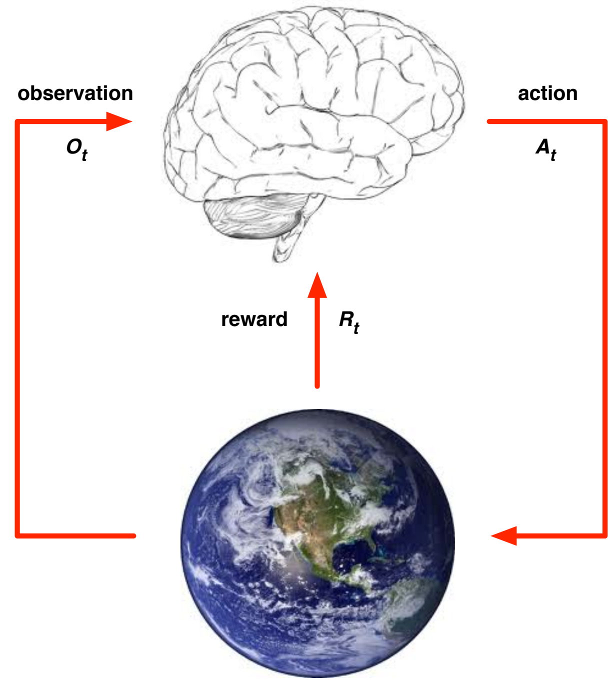
- Fly stunt manoeuvres in a helicopter
- Defeat the world champion at Backgammon, Go
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play Starcraft, Atari games better than humans
- Drive a car
- Play hockey

A NEW KIND OF LEARNING

- There is no supervisor, only a reward signal
  - No labels "wrong choice, right choice"
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

RL FRAMEWORK

- At each step t the **agent:**
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The **environment:**
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}



REMEMBER YOUR PEAS

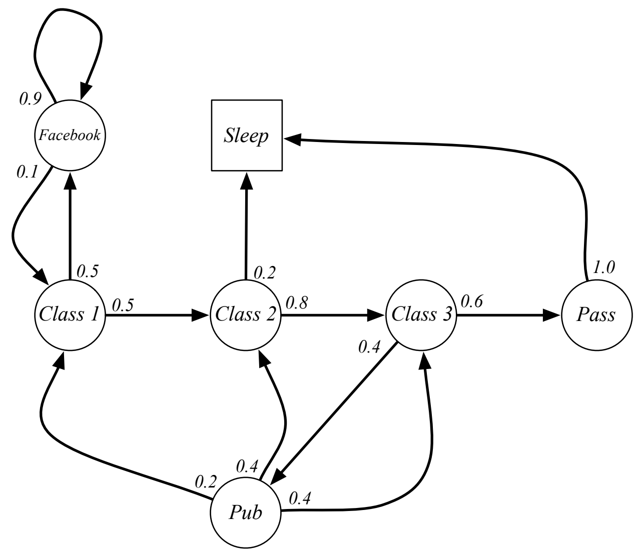
- Performance, Environment, Actuators, Sensors
- Learning to play video games
  - Exercise: What are the PEAS?
- Learn to flip pancakes
  - Exercise: What are the PEAS?
- Autonomous Helicopter
  - An example of **imitation learning**: start by observing human actions
  - Exercise: What are the PEAS?

MARKOV DECISION PROCESSES

MARKOV PROCESS

- Aka Markov Chains
- Think about atomic representation of environment state (Russell and Norvig)
- Like state space in problem search
- A Markov process moves from one state to another with a certain probability
- Transition probability: $P(s_{t+1} = s'|s_t = s)$
- Demo

EXAMPLE: STUDENT LIFE



Sample episodes for Student Markov Chain starting from $S_1 = C1$

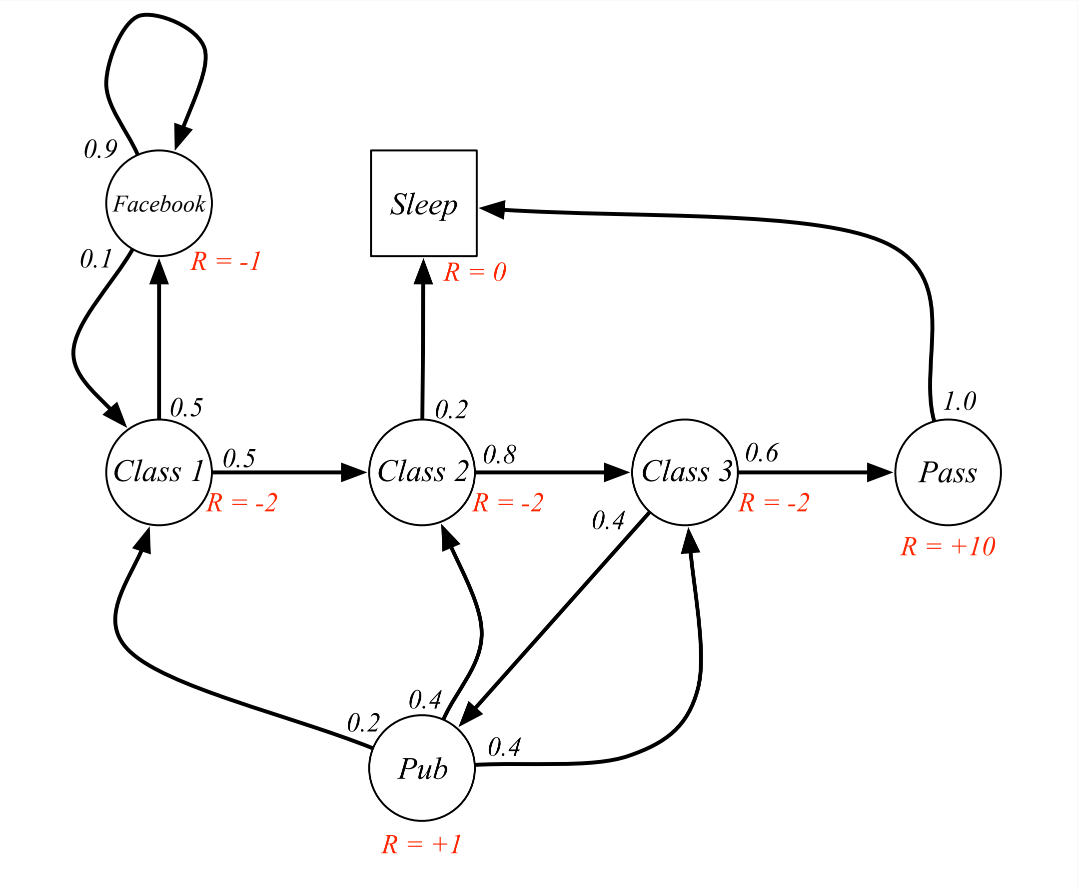
$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Source:
David Silver

MARKOV REWARD PROCESS

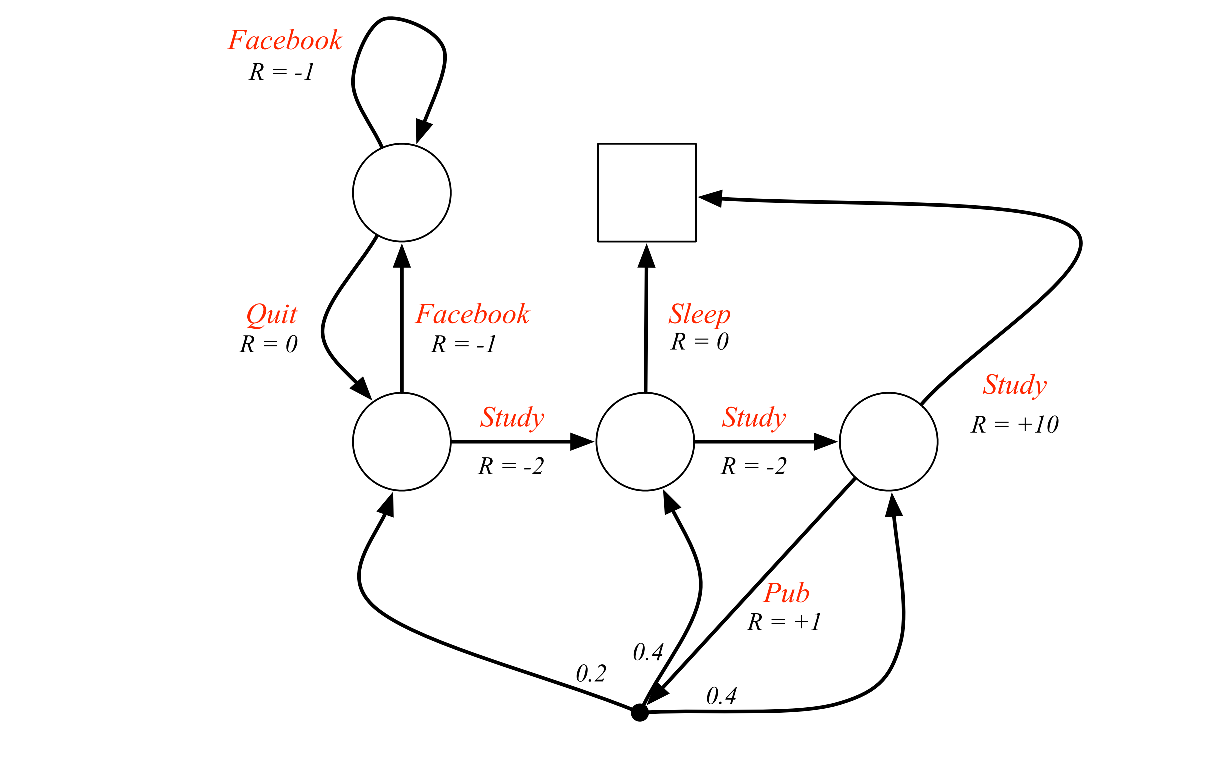
- Markov Process + Reward R_s associated with state
- More generally reward for *transition* R(s,s')



MARKOV DECISION PROCESSES

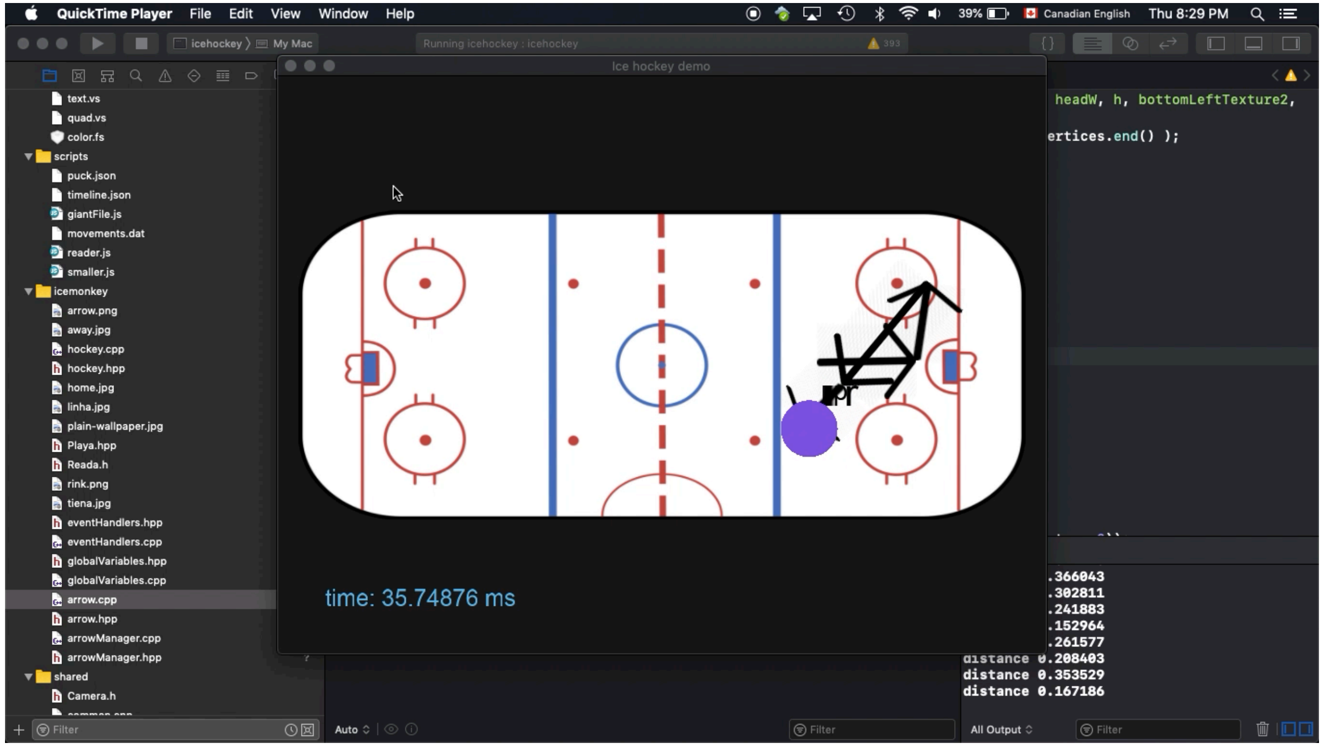
- Markov decision process (MDP) = Markov reward process + **actions**
- Transition probabilities, rewards depend on actions
- Markov game = MDP with actions, rewards for > 1 agent

EXAMPLE: STUDENT MARKOV DECISION PROCESS



HOCKEY EXAMPLE

What are the states?
What are the rewards?



MARKOV CHAINS

Theory and Algorithms

EXERCISES

- Consider a Markov chain like the one shown in this demo
- What is the probability of the sequence AABB?
- What is the longest possible sequence of observations?

MULTI-STEP TRANSITIONS

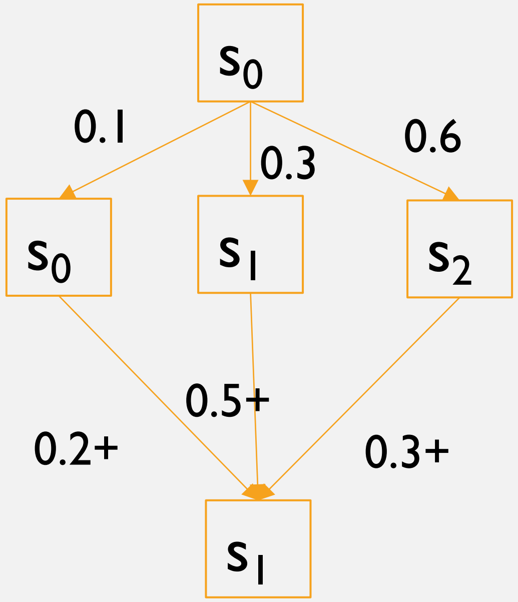
- What is the chance that if we start in state s we will reach state s' after a fixed number of n steps?
  - Think: from initial state, what is the chance of reaching a goal state in n steps?
- E.g. in this demo, what is the chance that we reach state 3 from 0 after 3 steps?
- What we want is a step-d transition matrix – how can we compute this efficiently?
- Notation: $P_d(s'|s)$

DYNAMIC PROGRAMMING

- Think Iterative Deepening: Build up transition matrices for 1, 2,…,d-1, d steps.
- For d = 1: Use given transition matrix $P(s'|s) = P_1(s'|s)$
- For d+1: $P_{d+1}(s'|s)=\sum_{s*} P(s'|s*) \times P_d(s'|s*)$

TREE VISUALIZATION

To compute $P_d(s_1|s_0)$



	s_0	s_1	s_2
s_0	0.1	0.3	0.6

Uses 1-step transition probability $P(s^*|s_0)$

Uses (d-1)-step transition probability $P_{(d-1)}(s_1|s^*)$

INFINITE CHAINS

- What if we let the number of steps (depth) *d* go to infinity?
- It can be shown that under certain conditions on the chain, there is limit transition probability matrix $P_\infty(s'|s)$
- This is the **stationary** transition matrix

PERFORMANCE METRIC FOR MDPS

Policies and Returns

FACTORED STATES

- In practice, RL uses a **factored state representation** (see Russell and Norvig).

➢ The state is defined by a list of values for a set of variables.

  - E.g. in hockey, can include score, game time, locations of players, location of puck

- If we have only 2 integer variables *x* and *y*, we can visualize states in a grid world

GRID WORLD EXAMPLE

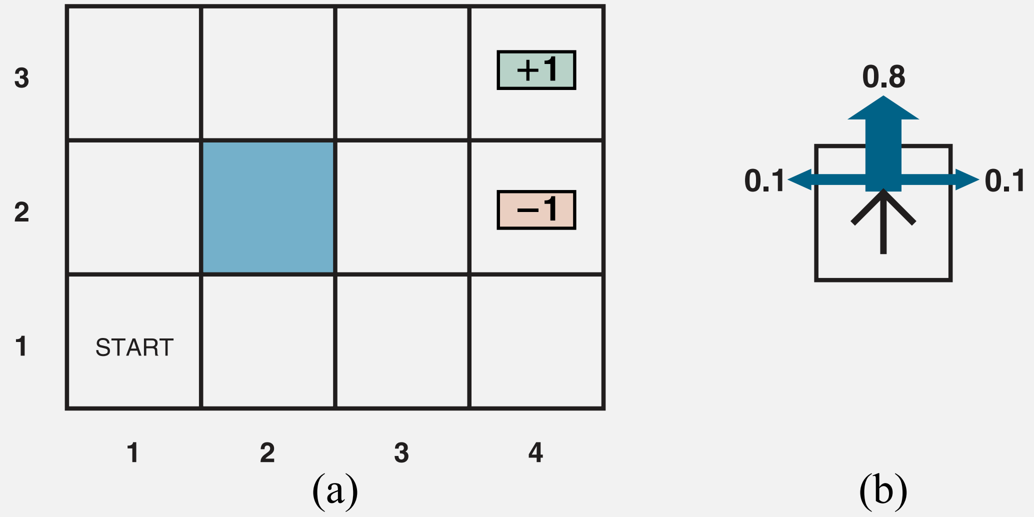
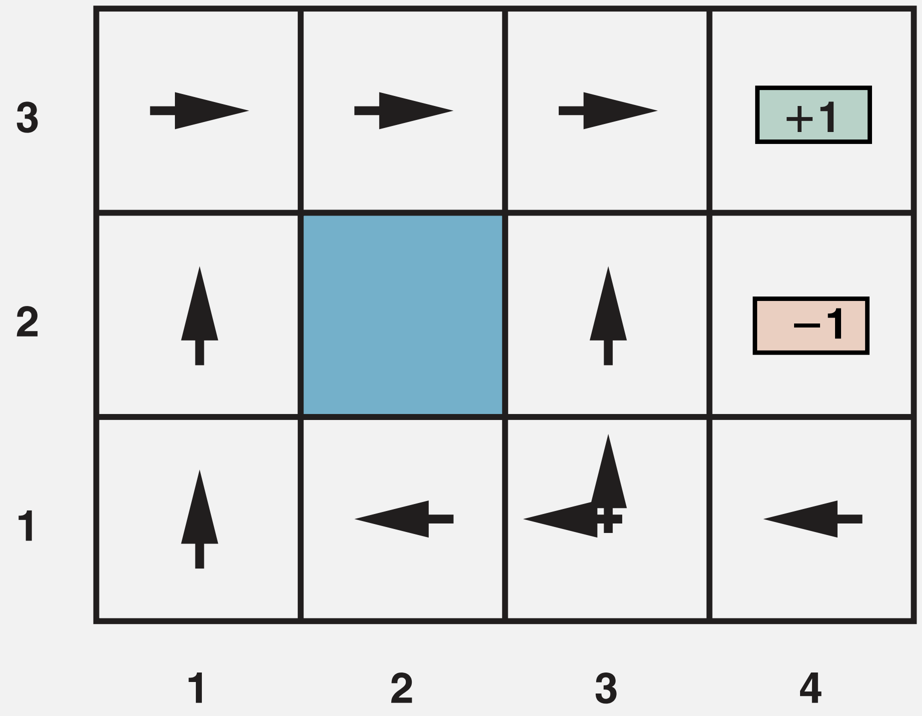


Figure 17.1 (a) A simple, stochastic 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. Transitions into the two terminal states have reward +1 and –1, respectively, and all other transitions have a reward of –0.04.

POLICIES

- A deterministic **policy** π is a function that maps states to actions $\pi(s)=a$
  - i.e. tells us how to act
- Can also be probabilistic $\pi(a|s)$
- Can be implemented using neural nets.
- Represents an agent function
- grid world demo

EXAMPLE



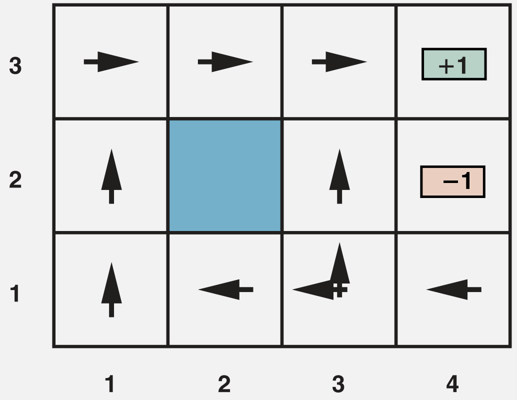
OPTIMAL POLICIES

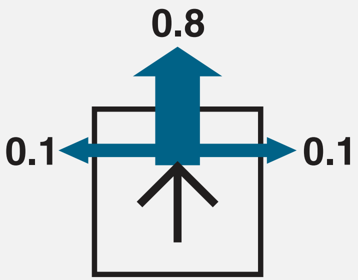
- A policy represents an agent (function)
- As always we want an agent/policy that maximizes expected utility
- How to define the expected utility of a policy in an MDP?
- Basic idea: when we execute the policy in an environment, we get a sequence of rewards.
  - Utility = (discounted) sum of rewards

RETURNS AND DISCOUNTING

- Basic Idea: When the agent executes a policy, they get a sequence of rewards r_0, r_1, r_d
- The <u>return</u> of a trajectory is the total sum of rewards.
- Typically rewards are weighted by a *discount factor* γ between 0 and 1.
- Payoff = Return = $r_0 + \gamma r_1 + \gamma^2 r_d$

EXAMPLE + EXERCISE





$\gamma = 0.5$

What if $\gamma = 1$?

State	Action	Reward	State	Action	State	Reward	Probability	Return
(1,1)	Up	-0.04	(1,2)	Up	(1,3)	-0.04	0.8x0.8	-0.04-0.5x0.04
(1,1)	Up	-0.04	(1,2)	Up	(1,2)	-0.04	?	-0.04-0.5x0.04
(1,1)	Up	-0.04	(1,2)	Right	(1,2)	-0.04	?	-0.04-0.5x0.04

MDPS VS SEARCH

- Solving an MDP is like planning in problem search
- Input: states, transition probabilities, reward function
- Output: optimal policy

Problem Search	Markov Decision Process
State	State
Deterministic Transitions NextState(state,action)	Non-deterministic Transition Probabilities P(state'|state,action)
Edge Cost cost(state,state')	Transition Reward R(state,action,state')
Sum of Edge Costs	Sum of Rewards = Return
Reach Goal State with minimum total cost	Maximize Sum of Rewards = Return
Plan = sequence of actions	Policy: select action in each state

WHY DISCOUNT?

- Reward in Markov decision processes are discounted. Why?
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Mathematically convenient to discount rewards for infinite trajectories (more below)
- Avoids infinite returns in cyclic Markov processes (like cyclic search)
- Uncertainty about the future may not be fully represented
  - There may be a small probability that process ends
- Animal/human behaviour shows preference for immediate reward

EXPECTED UTILITY FOR POLICIES

- Intuition: expected return (total reward) of policy π from state *s*
  = return from all possible trajectories, weighted by the probability of a trajectory given the policy
  = $\sum_{\text{trajectories } \tau} p(\tau|s,\pi) \times \text{return}(\tau)$
- We write $V^{\pi}_{d}(s)$ for the expected return of policy π from state *s* after *d* steps
  - Like depth *d*
  - In RL, the term "value function" is used instead of "expected utility"

1-STEP EXPECTED UTILITY

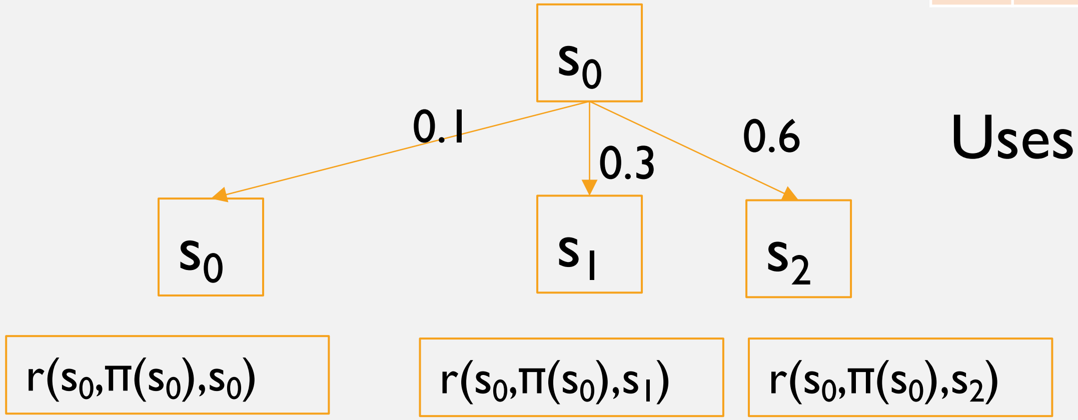
- How can we compute the values $V_1^\pi(s)$ = expected return after 1 step?
- Directly from MDP:
  $V_1^\pi(s)=\sum_{s'} p(s'|\pi(s),s) \times r(s,a,s')$

Probability of next state given current state and policy action

Reward associated with transition

TREE VISUALIZATION

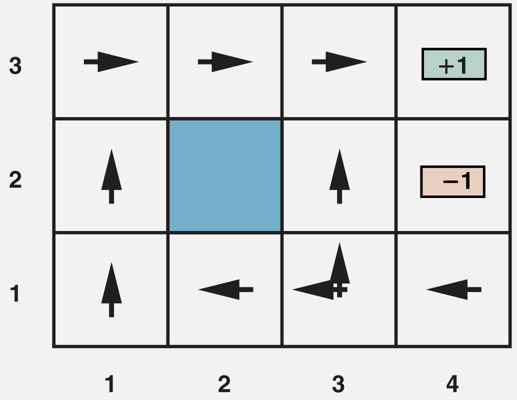
To compute $V_1^{\pi}(s_0)$

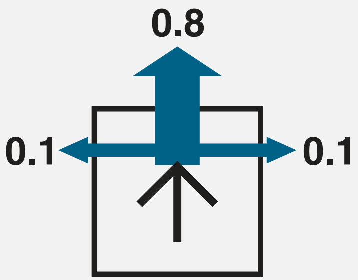


	s_0	s_1	s_2
s_0	0.1	0.3	0.6

Uses 1-step transition probability $P(s^*|s_0)$

EXAMPLE + EXERCISE





- Compute $V_1^{\pi}(1,1)$
- Exercise: what if $\pi(1,1)$=Right?
- So which move is better Up or Right?

Next State	Reward	Probability	XReward	Sum = -0.04
(1,2)	-0.04	0.8	0.8 x -0.04	
(2,1)	-0.04	0.1	0.1 x -0.04	
(1,1)	-0.04	0.1	0.1 x -0.04	

D-STEP EXPECTED UTLITY: THE BELLMAN EQUATION

- Suppose we have computed $V_d^{\pi}(s)$ = expected return after d steps
- How can we extend it to compute $V_{d+1}^{\pi}(s)$?
- $V_{d+1}^{\pi}(s) = \sum_{s'} P(s'|s, \pi(s)) \times [r(s, \pi(s), s') + \gamma V_d^{\pi}(s')]$

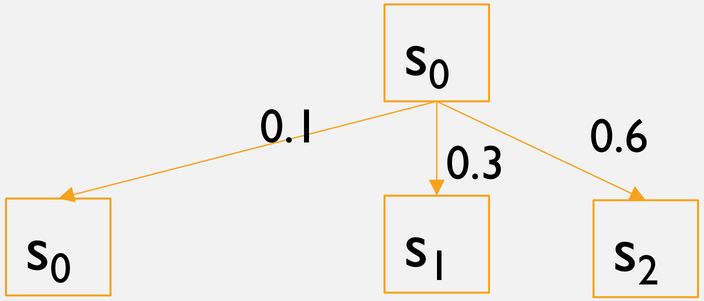
Immediate reward expected future reward

TREE VISUALIZATION

To compute $V_{(d+1)}^{\pi}(s_0)$

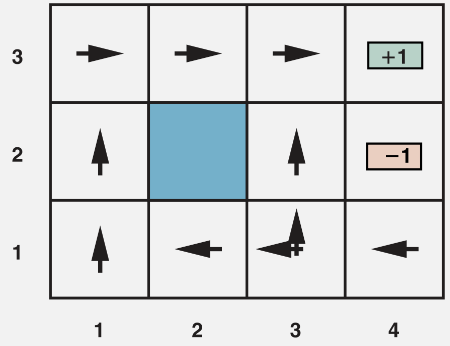
	s_0	s_1	s_2
s_0	0.1	0.3	0.6

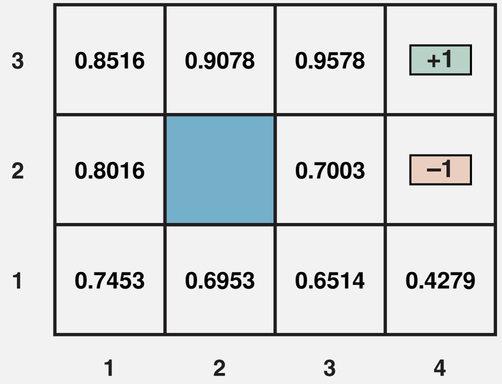
1-step transition probability $P(s^*|s_0)$



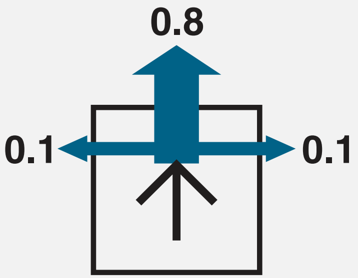
$r(s_0,\pi(s_0),s_0)$ +	$r(s_0,\pi(s_0),s_1)$ +	$r(s_0,\pi(s_0),s_2)$ +	Immediate reward
$\gamma \ x\ V_n^{\pi}(s_0)$	$\gamma \ x\ V_n^{\pi}(s_1)$	$\gamma \ x\ V_n^{\pi}(s_2)$	expected future reward

EXAMPLE + EXERCISE





- Suppose that V_d^π is as shown
- Compute $V_{(d+1)}^\pi(1,1)$
- Assume no discounting



Next State	Reward	Probability	XReward	Future Reward	Sum = ?
(1,2)	-0.04	0.8	0.8 x -0.04		
(2,1)	-0.04	0.1	0.1 x -0.04		
(1,1)	-0.04	0.1	0.1 x -0.04		

COMMENTS ON THE VALUE FUNCTION

- A powerful look-ahead concept.
  - Like searching through an entire search tree for expected success
- Game example: chance of winning, expected total score.
- Dr. Strange looks ahead

COMPUTING THE VALUE FUNCTION

- Computing the expected utility of a policy for each state (= value function) is known as **policy evaluation**
- We can keep applying the Bellman equation to compute state values
  - Known as **value iteration**
  - An instance of dynamic programming

VALUE ITERATION FOR POLICY EVALUATION

- Input: MDP, policy π, depth *d*
- $V^\pi(s) := 0$ for all s
- For *i* = 1 to *d*
  - For all s do
    $V^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) \times [r(s, \pi(s), s') + \gamma V^\pi(s')]$
- End for
- Return V^π

FINDING AN OPTIMAL POLICY: VALUE ITERATION

OPTIMAL POLICIES

- As always our goal is to find an agent that maximizes expected utility.

➢ Want a policy with maximum value

- A policy π^* is **optimal** if for any other policy and for all states s
  $V^{\pi^*}(s) \geq V^{\pi}(s)$
- The value of the optimal policy is written as $V^*(s)$.

OPTIMAL POLICIES: EXAMPLE

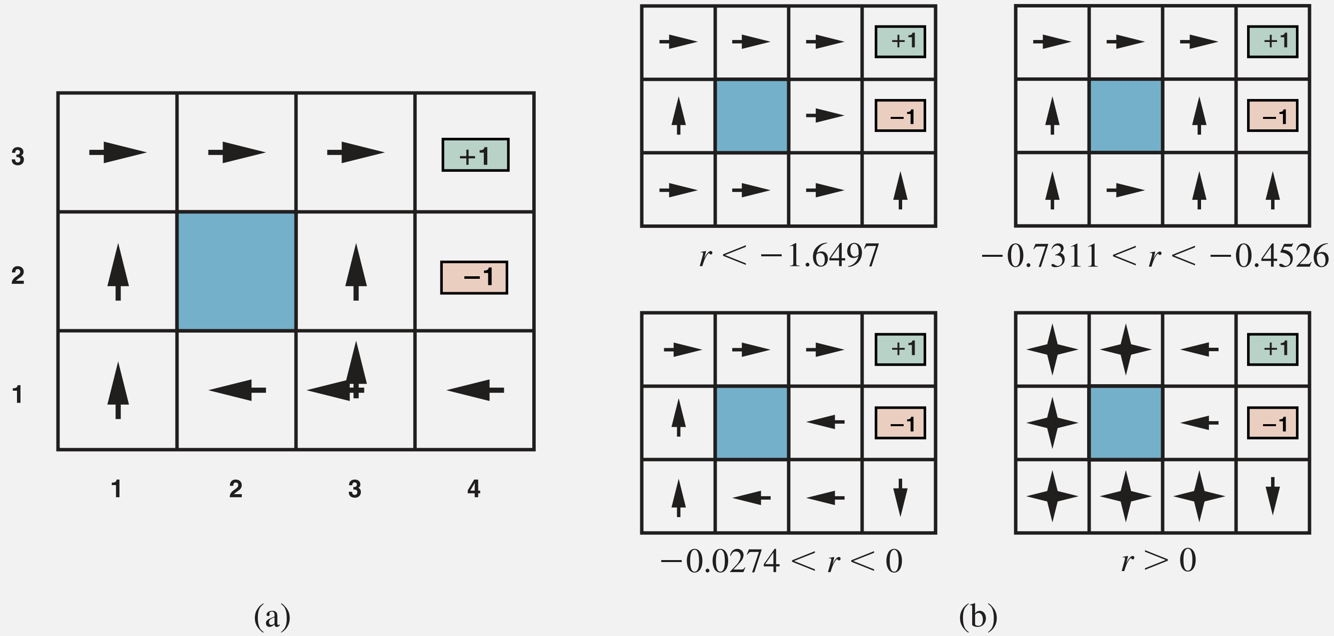


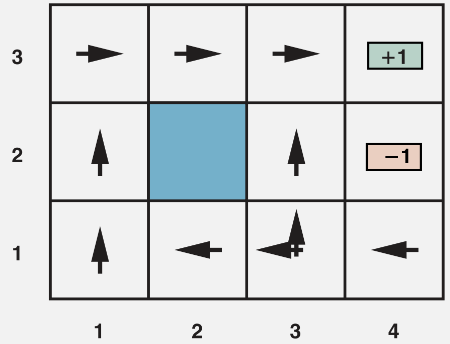
Figure 17.2 (a) The optimal policies for the stochastic environment with $r = -0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal. (b) Optimal policies for four different ranges of r.

OPTIMAL VALUE FUNCTION: EXAMPLE

3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	–1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

Figure 17.3 The utilities of the states in the 4×3 world with $\gamma = 1$ and $r = -0.04$ for transitions to nonterminal states.

EXERCISE



3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	−1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

- Given the value function shown, what is the best move at
  - (1,1)
  - (2,3)?

FROM VALUE TO POLICY

- It is easy to **extract** a policy from a value function:
- At each state, choose an action that maximizes expected future return
- $\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s, a) \times [r(s, a, s') + \gamma V(s')]$
  $= \text{argmax}_a \; Q^*(s,a)$
- $Q^*(s,a)$ is known as the **action-value** function
  - It represents the expected total return if we choose action *a* in state *s*

VALUE ITERATION: OPTIMAL VALUE FUNCTION

- Input: MDP, ~~policy π,~~ depth d
- $V^*(s) := 0$ for all s
- For i = 1 to d
  - For all s do
    $V^*(s) = \underline{\max}_{\underline{a}} \sum_{s'} P(s'|s, a) \times [r(s, a, s') + \gamma V^*(s')]$
- End for
- Return V^*

EXTENSION TO INFINITE HORIZON

- It is often useful to let the process run to any depth
- MDP may run forever ("neverending learning")
- Even if each trajectory is guaranteed to be finite, we may not know a definite upper bound in advance (termination uncertainty)
- Even if we know an upper bound in advance, it can introduce undesirable complications
  - E.g. every video game ends within 10 hours but at the beginning players don't think about the end
- Typically the value function changes very little at a modest depth (e.g. d = 13 for the NHL)

VALUE ITERATION: INFINITE HORIZON

- Input: MDP, ~~policy π, depth *d*~~
- $V^*(s) := 0$ for all s
- Repeat until convergence
  - For all s do
    $V^*(s) = \max_a \sum_{s'} P(s'|s, a) \times [r(s, a,s') + \gamma V^\pi(s')]$
- Return V^*

FINDING AN OPTIMAL POLICY: POLICY ITERATION

IMPROVING POLICIES

- Ultimately we want to find an optimal policy π*
- We *can* find an optimal value function V* and then extract an optimal policy.
- But value iteration for optimal V* is expensive because for every iteration and every state, we need to maximize <u>over all possible actions *a*</u>.
- Value iteration for fixed policy π is much faster because need to consider only the selected action π(s).
- Can we get the best of both worlds?
- Yes. The basic idea: start with initial policy, then improve it.

POLICY ITERATION

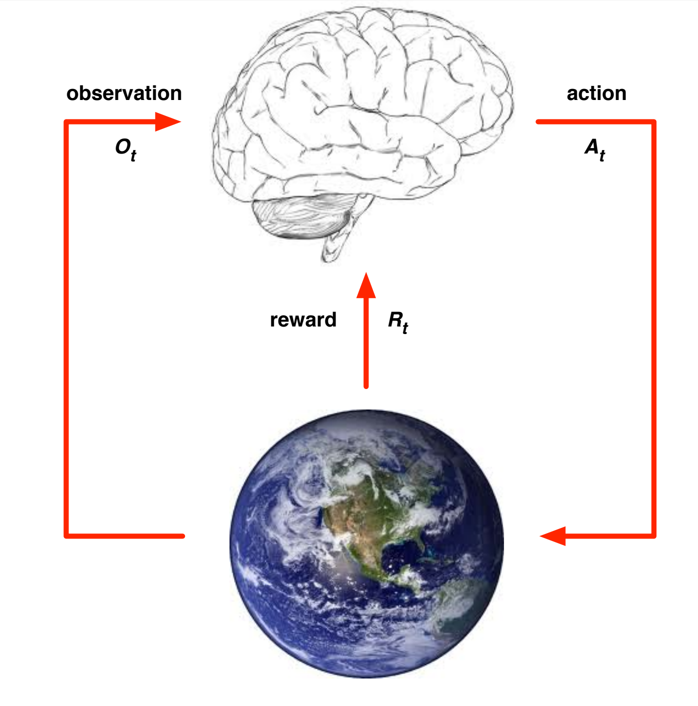
- Input: MDP
- $\pi(s)$:= random action for all s
- Repeat until convergence
  1. Policy evaluation: Compute V^{π} using value iteration
  2. Update policy π via
     $\pi(s) = \text{argmax}_{a} \sum_{s'} P(s'|s, a) \times [r(s, a, s') + \gamma V^{\pi}(s')]$
- Return π

REINFORCEMENT LEARNING

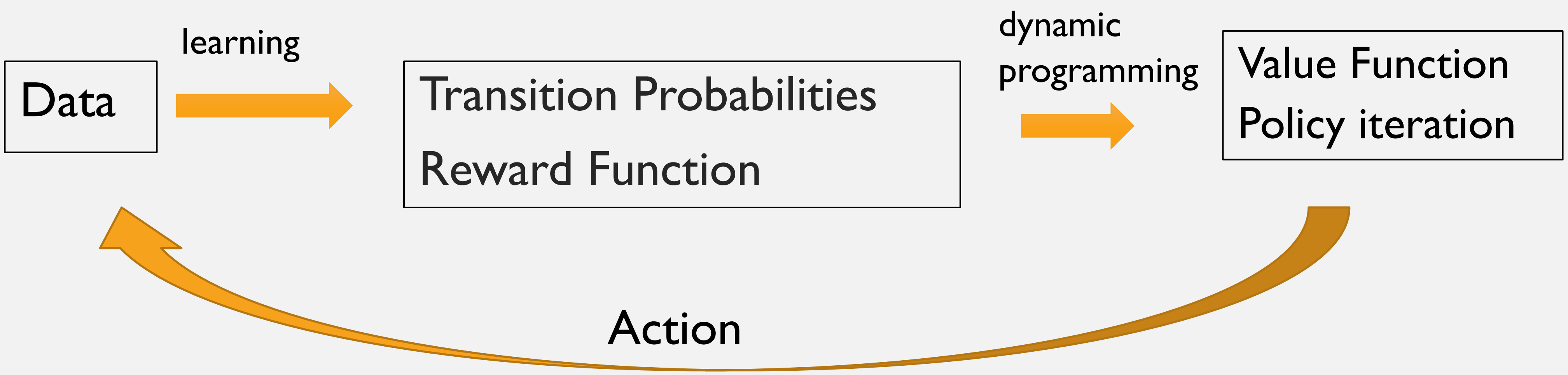
ADDING LEARNING TO MDPS

- If not all aspects of an MDP are known, an agent can try to learn them from observations.
- Typical Reinforcement Learning setup:

Known	Unknown	Learning Target
Possible States	Reward function	Optimal Policy
Possible Observations	Transition Probabilities	Value Function
Possible Actions		



LEARNING VALUE FUNCTIONS



- Can estimate rewards and transition probabilities using event counts
- Like maximum likelihood for Bayesian networks (later)
- Special RL challenge: need to act while learning about the environment

EXPLORATION VS. EXPLOITATION

- An agent needs to both
  - Select actions that seem optimal to keep high rewards
    - “exploit” its current knowledge
  - Select new actions to gather enough data to estimate a value function
    - “explore” the state space
- A simple approach is ε-greedy
  - With probability ε, select a random action (e.g. ε =10% of the time)
  - With probability 1-ε, select an action that is optimal according to the current value function
  - Simple but often effective

EXAMPLES

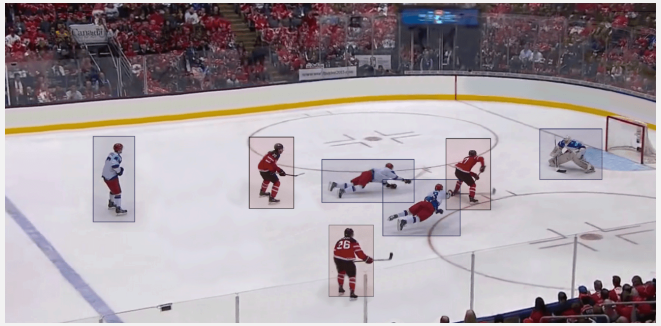
EXAMPLE: ALPHA* GAMES

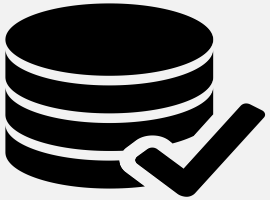
- data generated by self-play
- Neural net outputs 2 quantities
  1. V(s), the win rate from a position
  2. P(a|s): vector of move probabilities
     - more promising moves should have higher probability
     - Like node ordering in tree search
- To play, performs a (Monte Carlo) tree search using the neural net output
- Watch the alphago movie

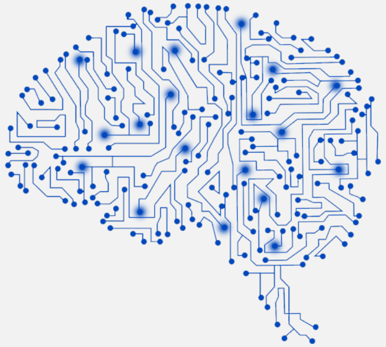
ICE HOCKEY EXAMPLES

- I've done a lot of work applying RL to ice hockey
- Using millions of events from NHL games

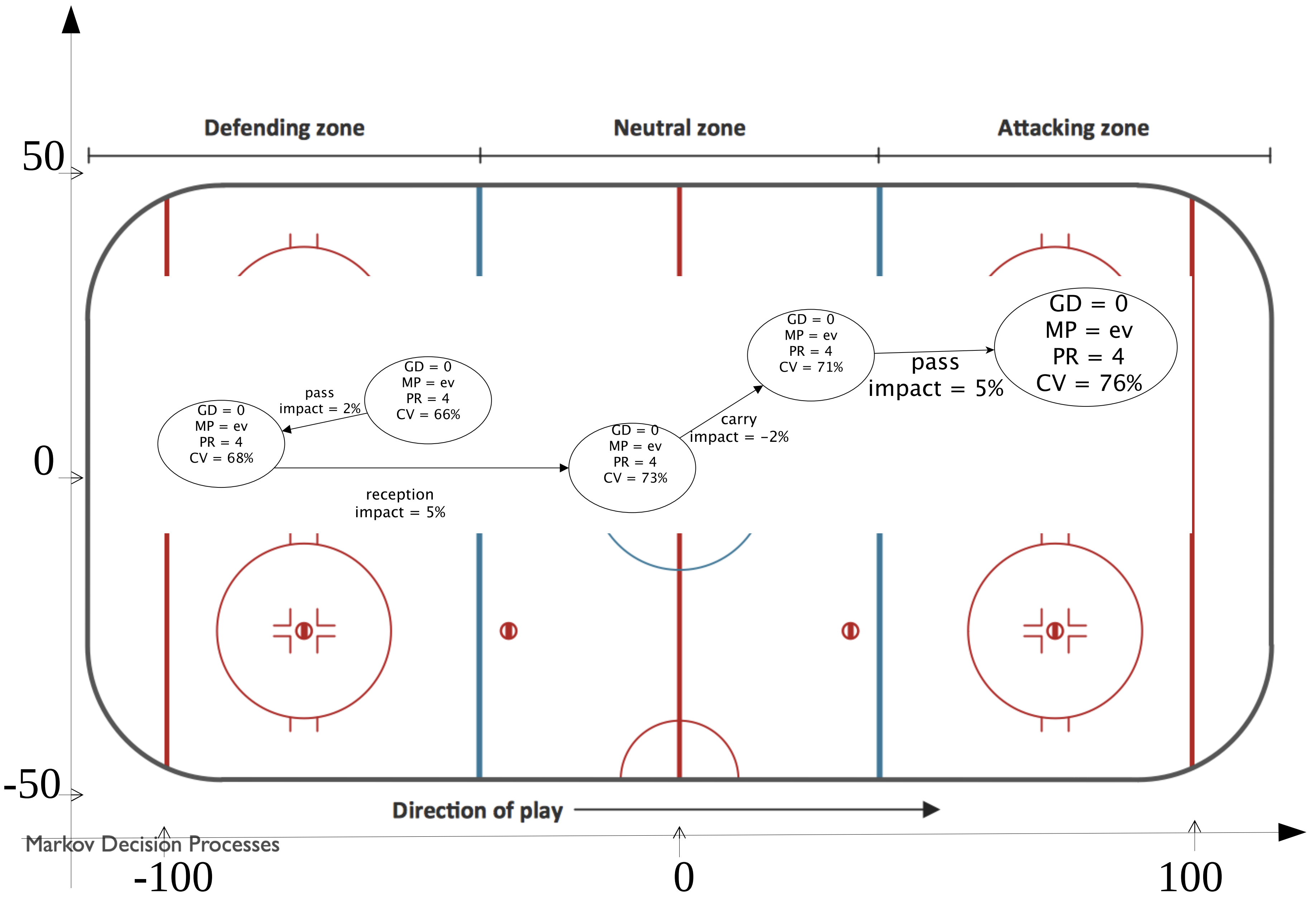
PIPELINE







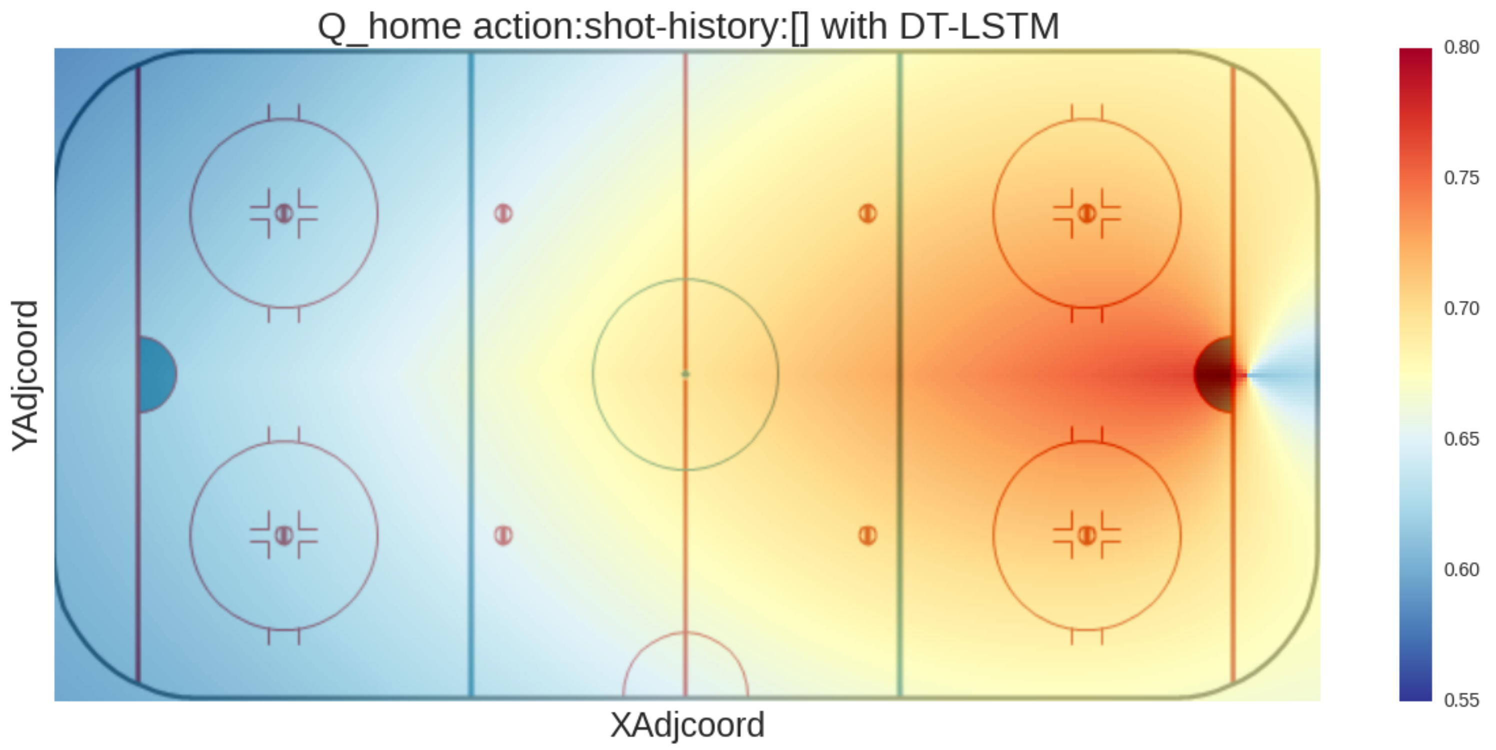
- Computer Vision Techniques: Video tracking
- Play-by-play Dataset
- Large-scale Machine Learning



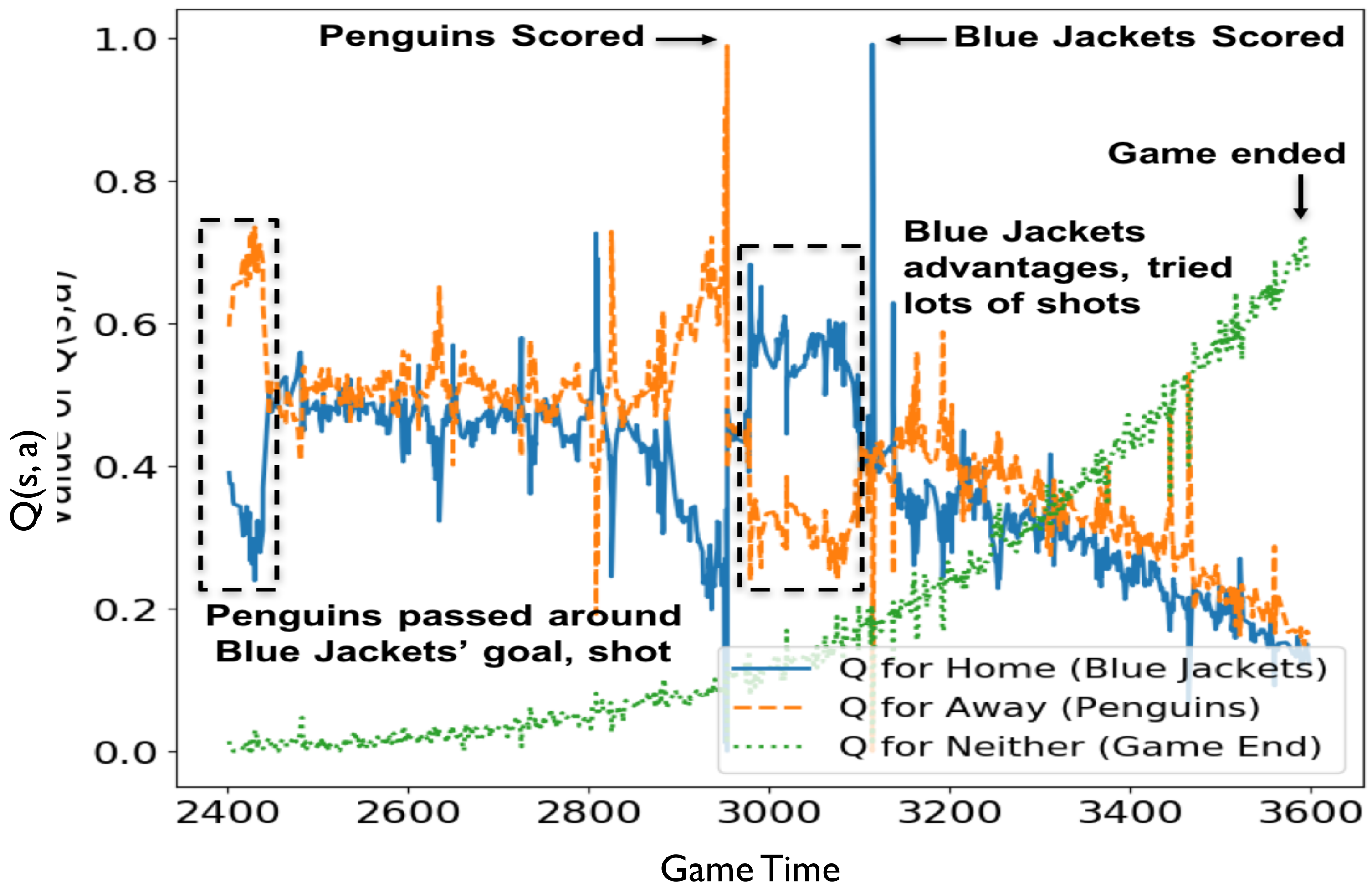
Defending zone
Neutral zone
Attacking zone
50
0
-50
-100
0
100
GD = 0
MP = ev
PR = 4
CV = 68%
pass
impact = 2%
GD = 0
MP = ev
PR = 4
CV = 66%
reception
impact = 5%
GD = 0
MP = ev
PR = 4
CV = 73%
carry
impact = −2%
GD = 0
MP = ev
PR = 4
CV = 71%
pass
impact = 5%
GD = 0
MP = ev
PR = 4
CV = 76%
Direction of play

Spatial Projection

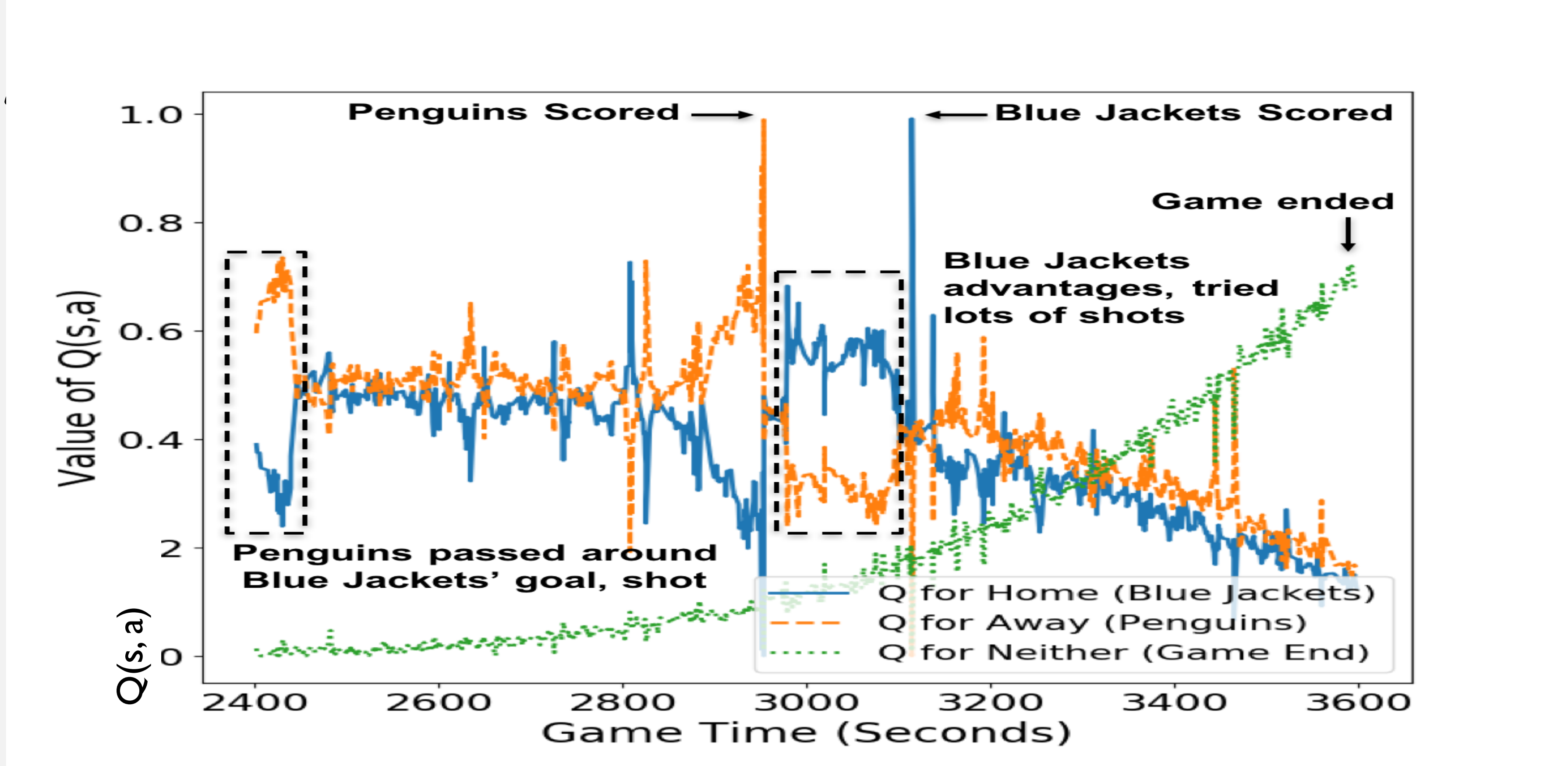
Value for the action “shot” action over the rink.



Value Ticker: Temporal Projection



THE IMPACT OF AN ACTION



PLAYER RANKING

Rank players by the total impact of all their actions

Name	GIM	Assists	Goals	Points	Team	Salary
Taylor Hall	96.40	39	26	65	EDM	$6,000,000
Joe Pavelski	94.56	40	38	78	SJS	$6,000,000
Johnny Gaudreau	94.51	48	30	78	CGY	$925,000
Anze Kopitar	94.10	49	25	74	LAK	$7,700,000
Erik Karlsson	92.41	66	16	82	OTT	$7,000,000
Patrice Bergeron	92.06	36	32	68	BOS	$8,750,000
Mark Scheifele	90.67	32	29	61	WPG	$832,500
Sidney Crosby	90.21	49	36	85	PIT	$12,000,000
Claude Giroux	89.64	45	22	67	PHI	$9,000,000
Dustin Byfuglien	89.46	34	19	53	WPG	$6,000,000
Jamie Benn	88.38	48	41	89	DAL	$5,750,000
Patrick Kane	87.81	60	46	106	CHI	$13,800,000
Mark Stone	86.42	38	23	61	OTT	$2,250,000
Blake Wheeler	85.83	52	26	78	WPG	$5,800,000
Tyler Toffoli	83.25	27	31	58	DAL	$2,600,000
Charlie Coyle	81.50	21	21	42	MIN	$1,900,000
Tyson Barrie	81.46	36	13	49	COL	$3,200,000
Jonathan Toews	80.92	30	28	58	CHI	$13,800,000
Sean Monahan	80.92	36	27	63	CGY	$925,000
Vladimir Tarasenko	80.68	34	40	74	STL	$8,000,000

- Mark Scheifele drew salaries **below** what his GIM rank would suggest.
- Later he received a $5M+ contract in 2016-17 season

SUMMARY

- Reinforcement Learning: learning to act
- Adds *actions* and *rewards* to a temporal Markov model
- Inference/Planning: find optimal policy given fully specified MDP
  - Value iteration: find optimal value function, extract policy
  - Policy iteration: alternate policy evaluation and policy extraction
- Learning problems:
  - Value function: Estimate the expected cumulative reward given a state for a given policy/ an optimal policy
  - Agent discovery: Learn an optimal policy